



TIBCO EBX® Documentation produit

*Version 6.0.3
novembre 2021*

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO and TIBCO EBX are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2006-2021. TIBCO Software Inc. All rights reserved.

Table des matières

Guide utilisateur

Introduction

1. Notions clés de TIBCO EBX®.....	15
2. Interface utilisateur.....	19
3. Glossaire.....	25

Modèles de données

4. Introduction aux modèles de données.....	36

Implémentation des modèles de données

5. Création du modèle de données.....	41
6. Configuration du modèle de données.....	43
7. Modélisation de la structure des données.....	47
8. Propriétés des éléments du modèle de données.....	53
9. Contrôles sur les éléments du modèle de données.....	69
10. Extensions de modèles de données.....	77
11. Actions sur les modèles de données existants.....	89

Publication et gestion de versions des modèles de données

12. Publication du modèle de données.....	93
13. Gestion des versions de modèles de données embarqués.....	95

Espaces de données

14. Introduction aux espaces de données.....	100
15. Création d'un espace de données.....	103
16. Actions sur les espaces de données existants.....	105
17. Images.....	115

Jeux de données

18. Introduction aux jeux de données.....	122
19. Création du jeu de données.....	125
20. Visualisation des données.....	127
21. Edition des données.....	139

Services d'import et d'export

22. Import et export XML.....	143
23. Import et export CSV.....	149
24. Actions sur les jeux de données existants.....	157
25. Héritage entre jeux de données.....	161

Modèles de workflow

26. Introduction aux modèles de workflow.....	166
27. Modélisation du workflow.....	171
28. Configuration du modèle de workflow.....	185
29. Publication d'un modèle de workflow.....	193

Workflows de données

30. Introduction aux workflows de données.....	196
31. Utilisation de l'interface utilisateur de la section Workflow de données.....	199
32. Bons de travail.....	205

Gestion de workflows de données

33. Lancement et monitoring de workflows de données.....	211
34. Administration de workflows de données.....	213

Services de données

35. Introduction aux services de données.....	218
36. Génération de WSDL pour services de données.....	221

Manuel de référence (en anglais)

Intégration

37. Overview of integration and extension.....	227
38. Using TIBCO EBX® as a Web Component.....	229
39. Built-in user services.....	237
40. Supported XPath syntax.....	251

Localisation

41. Labeling and localization.....	260
42. Extending TIBCO EBX® internationalization.....	263

Persistence

43. Overview of persistence.....	266
44. History.....	269
45. Replication.....	277
46. Data model evolutions.....	283

Divers

47. Inheritance and value resolution.....	288
48. Permissions.....	293
49. Criteria editor.....	313
50. Search.....	315
51. Performance guidelines.....	319

Guide d'administration (en anglais)

52. Administration overview.....	328

Installation & configuration

53. Supported environments.....	332
54. Java EE deployment.....	339

Installation notes

55. Installation note for JBoss EAP 7.1.x.....	349
56. Installation note for Tomcat 9.x.....	355
57. Installation note for WebSphere AS 9.....	359
58. Installation note for WebLogic 14c.....	365
59. TIBCO EBX® main configuration file.....	371
60. Initialization and first-launch assistant.....	391
61. Deploying and registering TIBCO EBX® add-ons.....	393

EBX® Container Edition

62. Building the image.....	398
63. Running the image.....	401
64. Customizing the image.....	409

Technical administration

65. Repository administration.....	412
66. UI administration.....	423
67. UI – Workflow launcher.....	439
68. Users and roles directory.....	451
69. Data model administration.....	455
70. Database mapping administration.....	457
71. Workflow management.....	461
72. Task scheduler.....	465
73. Audit trail.....	471
74. Other.....	475

Distributed Data Delivery (D3)

75. Introduction to D3.....	478
76. D3 broadcasts and delivery dataspace.....	483
77. D3 JMS Configuration.....	487
78. D3 administration.....	495

Guide de sécurité (en anglais)

79. Security Best Practices.....506

Guide du développeur (en anglais)

Introduction

80. Packaging TIBCO EBX® modules.....	513
81. Mapping to Java.....	519
82. Tools for Java developers.....	525
83. Terminology changes.....	527

Data model

84. Introduction.....	530
85. Data types.....	533
86. Tables and relationships.....	549
87. Constraints, triggers and functions.....	571
88. Triggers and functions.....	587
89. Labels and messages.....	591
90. Additional properties.....	597
91. Data services.....	605
92. Toolbars.....	607
93. Custom forms.....	609
94. Workflow model.....	647

User interface

95. Interface customization.....	658

User services

96. Overview.....	661
97. Quick start.....	665
98. Implementing a user service.....	669
99. Declaring a user service.....	683
100. Development recommendations.....	689

SOAP data services

101. Introduction.....	694
102. WSDL generation.....	705
103. SOAP operations.....	713

REST data services

104. Introduction.....	748
105. Built-in RESTful services.....	757

JSON Formats

106. Introduction.....	813
107. Extended.....	815
108. Compact.....	841
109. Common.....	847
110. Others.....	857

SQL in EBX®

111. Introduction.....	868
112. Comparison operators.....	876
113. Arithmetic operators and functions.....	880

114. Logical operators.....	884
115. String operators and functions.....	888
116. Date and time functions.....	892
117. EBX® SQL functions.....	895
118. REST Toolkit.....	897

EBX® Scripting

119. Record permission.....	910
120. Function field.....	923

Function field API

121. Unit summary.....	943
122. Unit default.....	945
123. Unit core.complex.....	947
124. Unit core.date.....	949
125. Unit core.datetime.....	954
126. Unit core.list.....	960
127. Unit core.locale.....	967
128. Unit core.log.....	969
129. Unit core.math.....	972
130. Unit core.resource.....	981
131. Unit core.string.....	986
132. Unit core.time.....	991
133. Unit core.uri.....	995

Guide utilisateur

Introduction

CHAPITRE 1

Notions clés de TIBCO EBX®

Ce chapitre contient les sections suivantes :

1. [Concepts et outils associés](#)
2. [Architecture](#)

1.1 Concepts et outils associés

Le Master Data Management (MDM) est un moyen de modéliser, gérer et gouverner les données partagées. Quand des données sont partagées par plusieurs systèmes informatiques, ainsi que des équipes professionnelles différentes, l'existence d'une seule version gouvernée des données de référence est essentielle.

Avec EBX®, les utilisateurs métier et les équipes informatiques peuvent collaborer sur une solution unifiée, afin de concevoir des modèles de données et de gérer le contenu des données de référence.

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

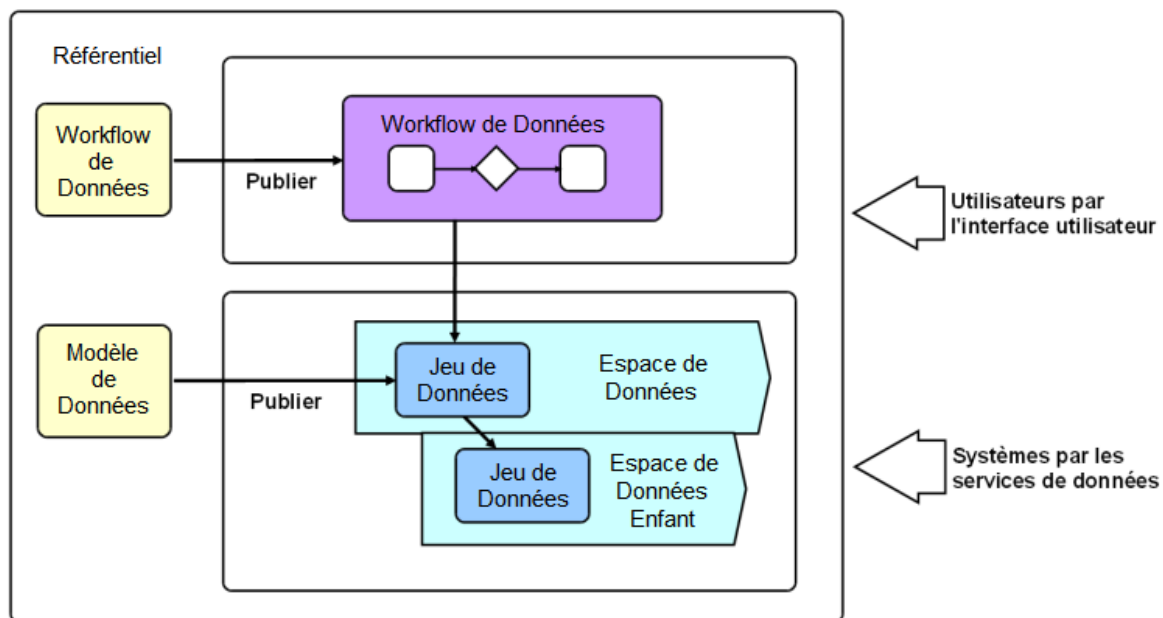
Un projet MDM basé sur EBX® commence par la création d'un *modèle de données*. Celui-ci définit les tables, les champs, les liens et les règles métiers permettant de décrire les données de référence. De bons exemples sont les catalogues de produits, les hiérarchies financières, les listes de fournisseurs ou simplement les tables de référence.

Ce modèle de données peut ensuite être publié en tant que *jeu de données*, stockant le contenu des données de référence. Les jeux de données sont organisés dans des *espaces de données*. Un espace de données est un conteneur qui permet d'isoler toutes les mises à jour effectuées. Cela permet de travailler sur plusieurs versions parallèles des données.

Les *workflows* sont indispensables aux processus de modification et d'approbation sur les données. Un workflow permet de modéliser un processus étape par étape, comprenant la participation de plusieurs utilisateurs humains et automatisés.

Les *modèles de workflow* définissent les tâches à effectuer, ainsi que les participants associés à chaque tâche. Dès qu'un modèle de workflow est publié, il peut être exécuté en tant que *workflow de données*. Les workflows de données permettent d'envoyer aux utilisateurs des notifications concernant les événements pertinents et les tâches à accomplir, le tout dans un contexte collaboratif.

Les *services de données* aident à intégrer EBX® à des systèmes tiers ("middleware"), en leur permettant d'accéder aux données, ou de gérer des espaces de données et des workflows.



Voir aussi

[Modèle de données](#) [p 26]

[Jeu de données](#) [p 28]

[Espace de données](#) [p 30]

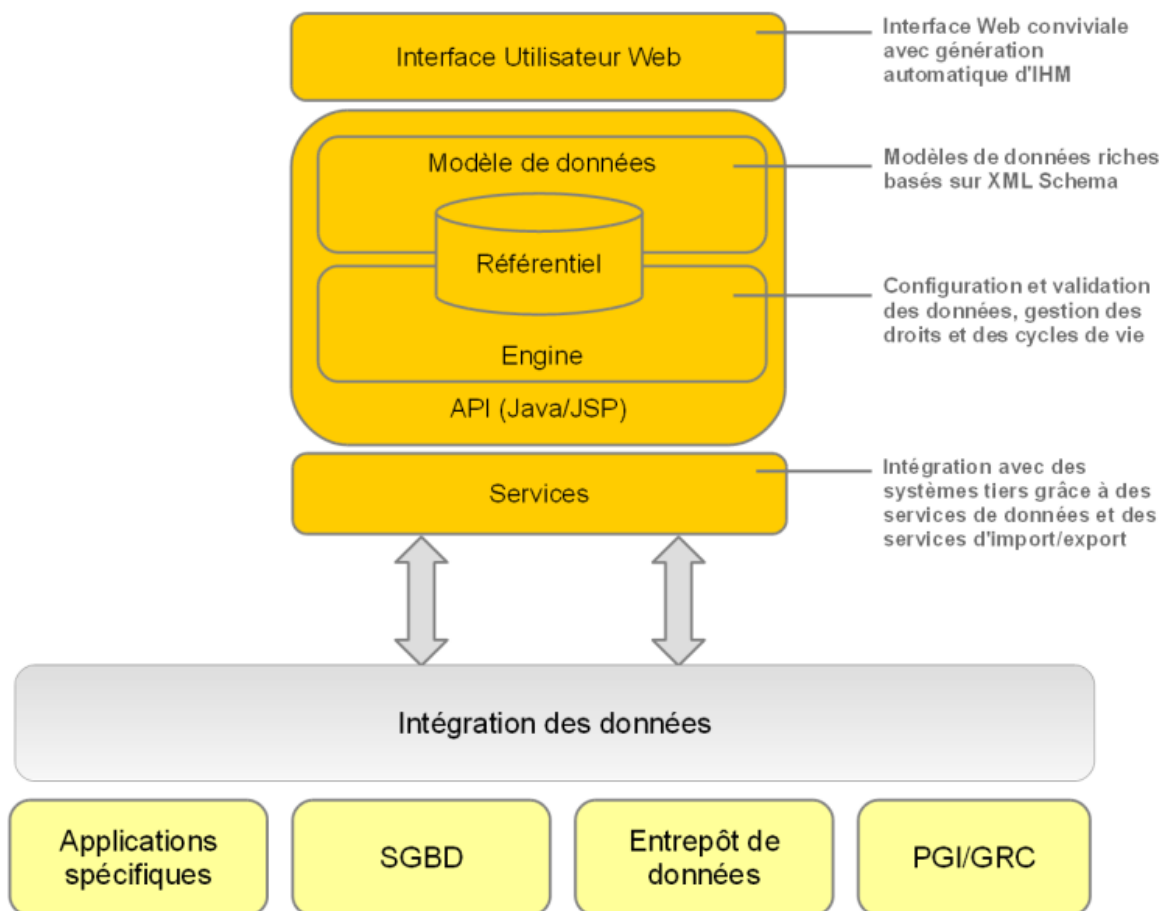
[Modèle de workflow](#) [p 31]

[Workflow de données](#) [p 32]

[Service de données](#) [p 33]

1.2 Architecture

Le schéma suivant présente l'architecture de EBX®.



CHAPITRE 2

Interface utilisateur

Ce chapitre contient les sections suivantes :

1. [Présentation](#)
2. [Perspective avancée](#)
3. [Perspectives](#)
4. [Panneau utilisateur](#)
5. [Fonctionnalités de l'interface utilisateur](#)
6. [Où trouver de l'aide sur EBX®](#)

2.1 Présentation

La présentation générale des espaces de travail sur TIBCO EBX® peut être entièrement personnalisée par un administrateur.

Lorsque plusieurs perspectives personnalisées ont été créées, elles peuvent être sélectionnées via l'icône 'Perspective' dans l'en-tête de l'écran.

La perspective avancée est accessible par défaut.

Voir aussi [UI administration](#) [p 423]

2.2 Perspective avancée

Par défaut, la perspective avancée d'EBX® est accessible à tous les utilisateurs. Cependant, son accès peut être restreint à certains profils uniquement. Cette vue est divisée en plusieurs zones principales, référencées dans la documentation sous les termes suivants:

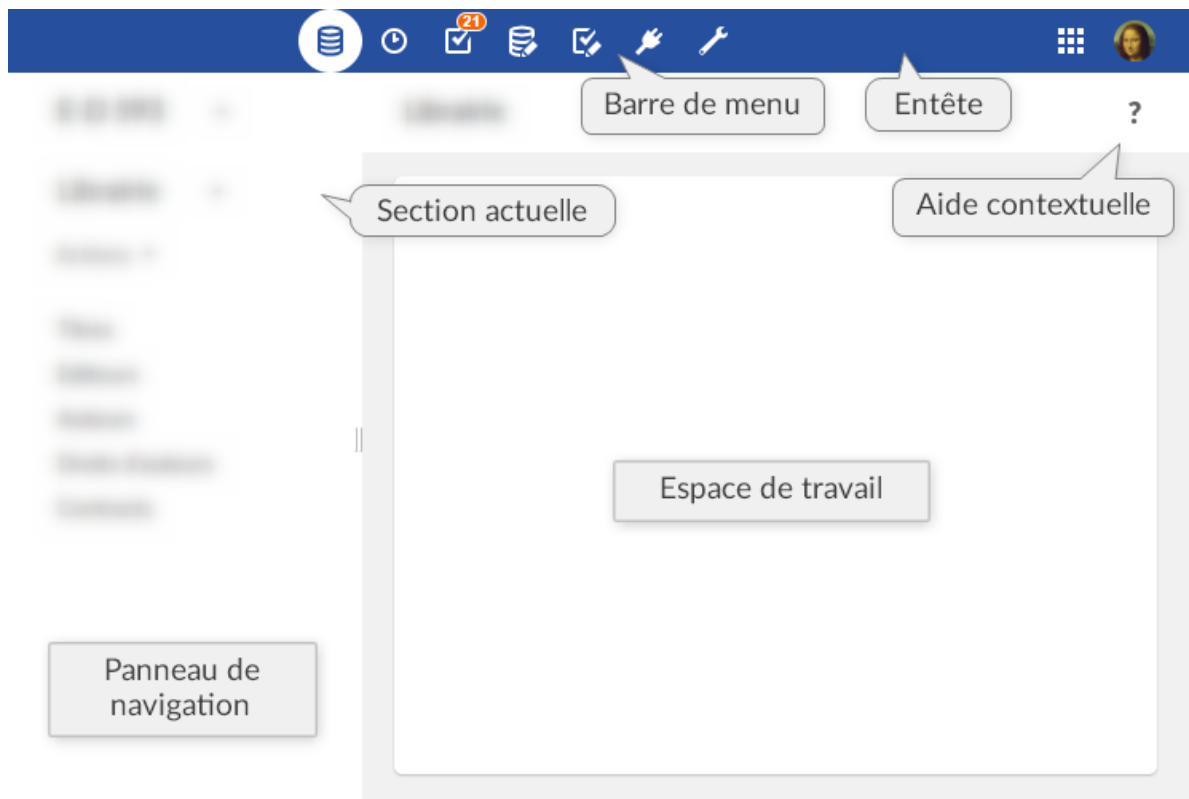
Note

La perspective avancée est toujours accessible aux utilisateurs via une sélection explicite (via un composant Web par exemple). Contrairement aux autres perspectives, elle peut seulement être « cachée » dans l'interface afin que l'utilisateur ne puisse pas l'appliquer lui-même.

- **En-tête** : l'avatar de l'utilisateur actuel s'affiche dans cette zone, ainsi que l'icône de sélection des perspectives. En cliquant sur l'avatar de l'utilisateur, le panneau utilisateur s'ouvre.

- **Barre de menu** : cette zone comprend toutes les fonctionnalités accessibles à l'utilisateur actuel 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** : zone de travail principale dépendant du contexte. 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*.



2.3 Perspectives

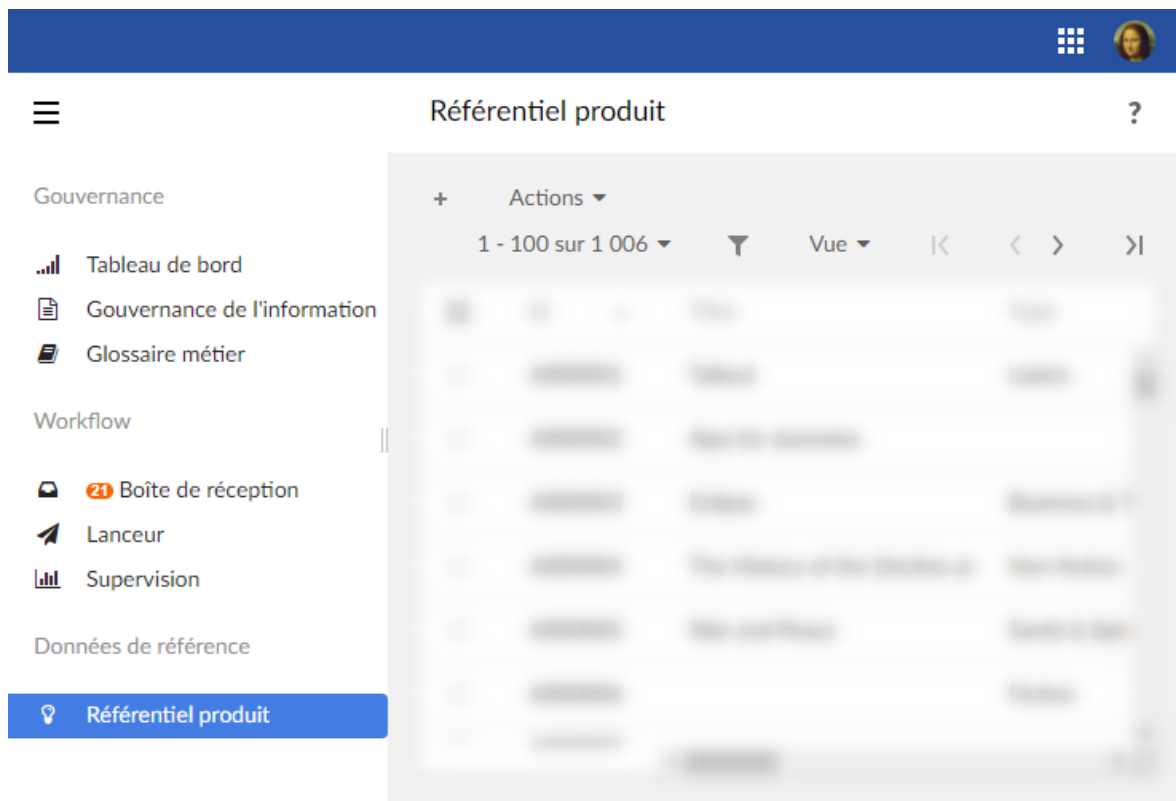
Les perspectives dans EBX® sont des vues configurables avec une audience définie. Les perspectives permettent aux utilisateurs métier de bénéficier d'une interface simplifiée. Une perspective peut être affectée à un ou plusieurs profils. Cette vue est divisée en plusieurs zones principales, référencées dans la documentation sous les termes suivants :

- **En-tête** : l'avatar de l'utilisateur actuel s'affiche dans cette zone, ainsi que l'icône de sélection des perspectives (lorsque plusieurs sont disponibles). En cliquant sur l'avatar de l'utilisateur, le panneau utilisateur s'ouvre.
- **Panneau de navigation** : cette zone affiche le menu hiérarchique tel qu'il a été configuré par l'administrateur de perspectives. Ce panneau peut être développé ou réduit et permet d'accéder aux entités et services correspondant à l'activité de l'utilisateur.

- **Espace de travail** : zone de travail principale dépendant du contexte.

Une perspective est configurée par un utilisateur ayant les autorisations nécessaires. Pour plus d'informations sur la configuration d'une perspective, voir [perspective administration \(en anglais\)](#) [p 424].

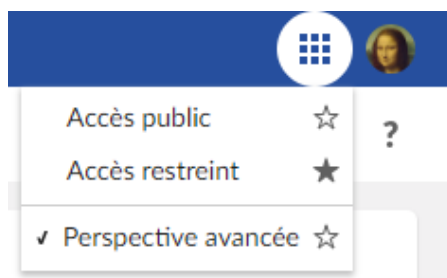
Exemple de menu hiérarchique :



Perspectives favorites

Un utilisateur, lorsque plusieurs perspectives ont été définies pour son profil, peut définir une perspective favorite afin que cette dernière soit sélectionnée par défaut à sa connexion. Pour ce faire, une icône est présente à côté de chaque perspective dans le sélecteur de perspectives :

- Une étoile pleine indique la perspective favorite. Un clic sur cette icône désélectionne la perspective favorite.
- Une étoile vide indique que la perspective associée n'est pas la favorite. Un clic sur cette icône va définir cette perspective comme favorite.



Voir aussi [Recommended perspectives](#) [p 436]

2.4 Panneau utilisateur

Les fonctionnalités générales d'EBX® sont regroupées dans le panneau utilisateur. Pour y accéder, cliquer sur l'avatar (ou les initiales) de l'utilisateur actuel, dans l'en-tête de chaque page.

Le panneau utilisateur affiche alors l'avatar de l'utilisateur et donne accès à la configuration du profil (selon les droits de l'utilisateur), à la sélection de la langue et de la densité d'affichage ; et à la documentation en ligne.

Attention

Le bouton de déconnexion est situé sur le panneau utilisateur.

Avatar

Un avatar peut être défini pour chaque utilisateur. L'avatar est constitué d'une image, définie via une URL ; ou de deux lettres (par défaut les initiales de l'utilisateur). La couleur de fond est attribuée automatiquement et ne peut pas être modifiée. L'image utilisée doit impérativement être au format carré mais n'est pas limitée en termes de taille.

Note

Les avatars s'affichent au niveau du panneau utilisateur, de l'historique et des interfaces du workflow.

La fonctionnalité est également disponible via la méthode Java `UIComponentWriter.addUserAvatarAPI`. L'affichage de l'avatar peut être personnalisé à partir de la section 'Ergonomie et disposition' dans 'Administration'. Il est possible d'afficher l'avatar, le nom de l'utilisateur, ou les deux à la fois.

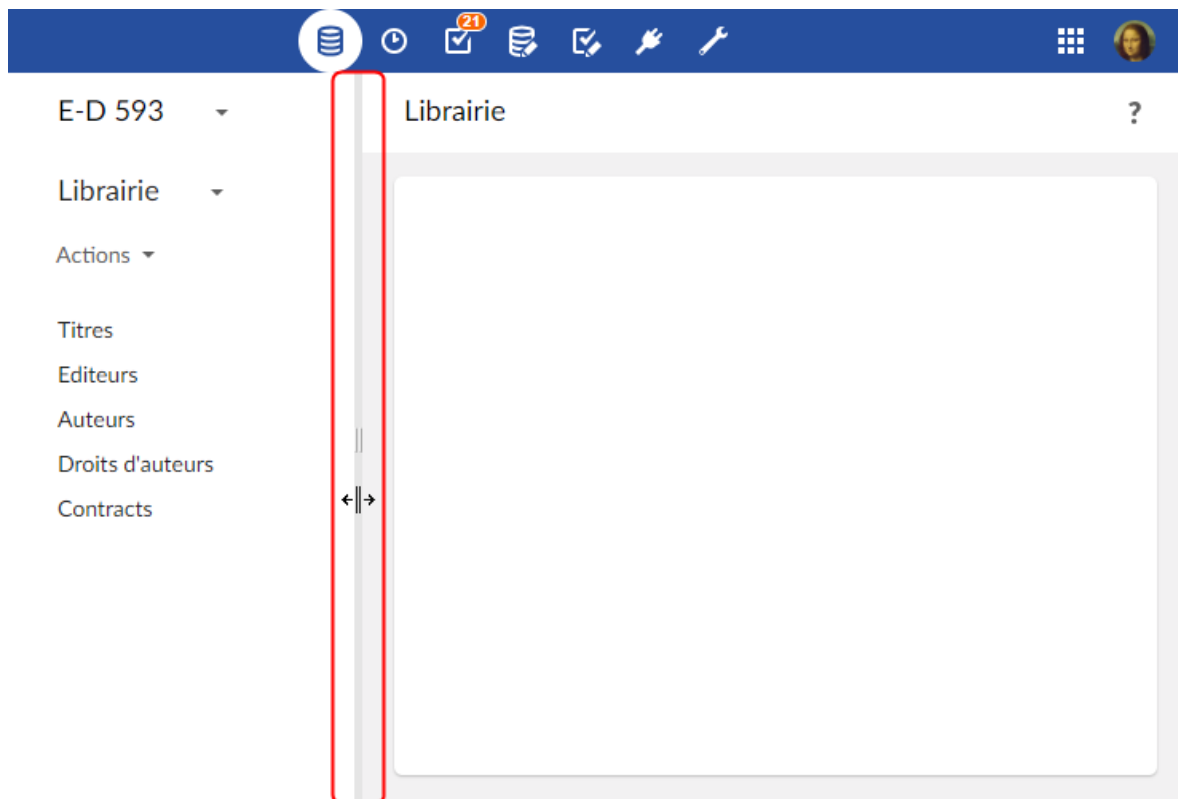
Densité

La densité d'affichage de la police peut être personnalisée : 'Compacte' ou 'Confortable'. Le mode d'affichage peut être modifié via le panneau utilisateur.

2.5 Fonctionnalités de l'interface utilisateur

Réinitialiser la largeur du panneau de navigation

Si la largeur du panneau de navigation a été modifiée, elle peut être réinitialisée en double-cliquant sur la bordure.



2.6 Où trouver de l'aide sur EBX®

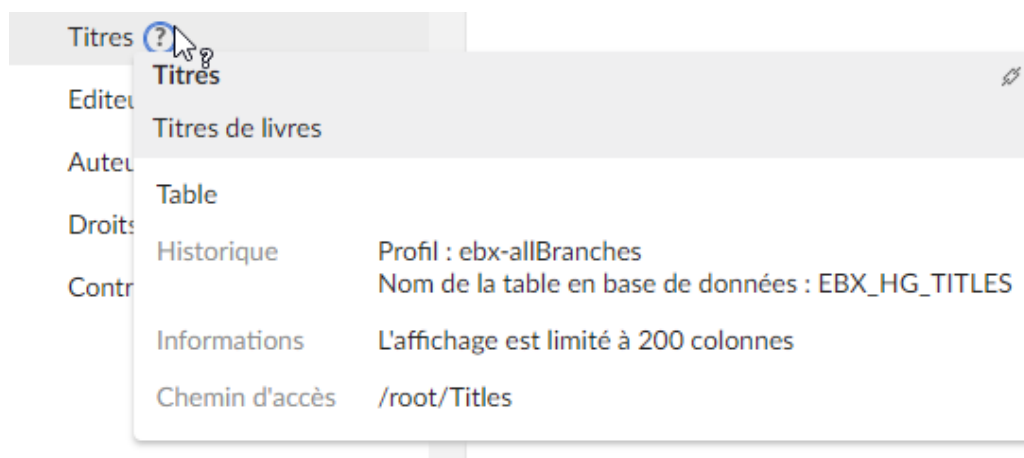
En plus de la documentation complète du produit accessible via le [panneau utilisateur](#) [p 22], l'aide est accessible de plusieurs façons dans l'interface.

Aide contextuelle

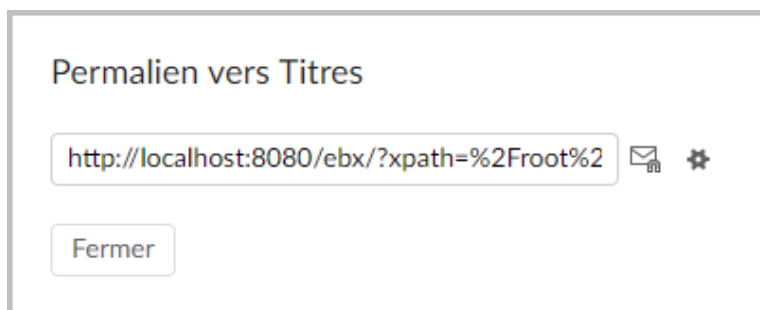
Dans n'importe quel espace de travail d'EBX®, une aide spécifique au contexte actuel est disponible en cliquant sur le point d'interrogation situé à droite du second bandeau. Le chapitre correspondant de la documentation produit sera alors affiché.

Aide contextuelle sur éléments

Lorsque la souris survole un élément pour lequel une aide spécifique a été définie, un point d'interrogation apparaît. En cliquant sur le point d'interrogation, un panneau de documentation affiche les informations associées.



Un bouton dans le coin supérieur droit du panneau permet de récupérer un permalien vers l'élément. Ce bouton n'est pas disponible pour tous les types d'éléments.



CHAPITRE 3

Glossaire

Ce chapitre contient les sections suivantes :

1. [Gouvernance](#)
2. [Modélisation de données](#)
3. [Gestion des données](#)
4. [Cycle de vie des données](#)
5. [Historique](#)
6. [Modélisation de workflow](#)
7. [Workflows de données](#)
8. [Services de données](#)
9. [Transverse](#)

3.1 Gouvernance

référentiel

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

Voir [espace de données](#) [p 30].

profil

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

Voir [utilisateur](#) [p 25], [rôle](#) [p 26].

API Java : Profile^{API}.

utilisateur

Entité créée dans le référentiel afin de permettre à des personnes physiques ou des systèmes externes de s'authentifier et d'accéder à EBX®. Un utilisateur peut avoir un ou plusieurs rôles et posséder diverses informations de compte telles que nom, prénom, login, e-mail, etc.

Voir [annuaire des utilisateurs et des rôles](#) [p 26], [profil](#) [p 25].

Concept apparenté : [User and roles directory](#) [p 451].

API Java : `UserReference`^{API}.

rôle

Classification d'utilisateur utilisée pour les règles de permission et les workflows de données. Chaque utilisateur peut appartenir à plusieurs rôles.

Dès qu'un profil de type rôle est configuré dans EBX®, 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 pour le(s) destinataire(s) d'un bon de travail.

Voir [annuaire des utilisateurs et des rôles](#) [p 26], [profil](#) [p 25].

Concept apparenté : [User and roles directory](#) [p 451].

API Java : `Role`^{API}.

administrateur

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

annuaire des utilisateurs et des rôles

Annuaire définissant les méthodes disponibles pour l'authentification d'accès au référentiel, ainsi que les rôles disponibles et les utilisateurs autorisés à accéder au référentiel, avec leurs rôles respectifs.

Voir [utilisateur](#) [p 25], [rôle](#) [p 26].

Concept apparenté : [User and roles directory](#) [p 451].

API Java : `Directory`^{API}, `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](#) [p 451].

API Java : `Session`^{API}.

3.2 Modélisation de données

Section de la documentation [Modèles de données](#) [p 36]

modèle de données

Définition structurée des données à gérer dans le référentiel EBX®. Un modèle de données décrit la structure des données en termes d'organisation, de type 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 contenant les données gérées par le référentiel.

Voir [jeu de données](#) [p 28].

Concept apparenté : [Modèles de données](#) [p 36].

champ

Élément de base du modèle de données défini par un nom et un type de données simple. 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 maximale supérieure à 1. Chaque élément de la liste ainsi définie sera du même type que le champ initial. Les champs peuvent être regroupés pour faciliter l'organisation du modèle de données.

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

Voir [enregistrement](#) [p 28], [groupe](#) [p 27], [table \(modèle de données\)](#) [p 27], [règle de validation](#) [p 28], [héritage](#) [p 29].

Concept apparenté : [Propriétés des éléments de structure](#) [p 53], [Contrôles sur les champs de données](#) [p 69].

API Java : `SchemaNodeAPI`.

L'ancien nom (avant la version 5) de "champ" était "attribute".

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](#) [p 549].

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](#) [p 27].

Concept apparenté : [Clé étrangère](#) [p 554].

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](#) [p 28], [clé primaire](#) [p 27], [type réutilisable](#) [p 28].

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](#) [p 28].

API Java : 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.

L'ancien nom (avant la version 5) de "règle de validation" était "constraint".

assistant de modélisation de données (DMA)

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

Voir [Modèles de données](#) [p 36].

3.3 Gestion des données

Section de la documentation [Jeux de données](#) [p 122]

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.

Voir [table \(jeu de données\)](#) [p 28], [clé primaire](#) [p 27].

L'ancien nom (avant la version 5) pour "enregistrement" était "occurrence".

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](#) [p 28], [clé primaire](#) [p 27].

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 \(jeu de données\)](#) [p 28], [champ](#) [p 27], [groupe](#) [p 27], [vues](#) [p 29].

Les jeux de données sont représentés par l'icône .

Concept apparenté : [Jeux de données](#) [p 122].

L'ancien nom (avant la version 5) de "jeu de données" était "adaptation instance".

héritage

Mécanisme par lequel une donnée d'une entité peut être valorisée par défaut à partir d'une autre entité. Dans EBX®, 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. Il est possible de surcharger dans les jeux de données enfants les valeurs héritées du parent. 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 s'applique entre champs d'un même jeu de données. Le champ hérité prend pour valeur par défaut la valeur du champ source.

Les champs hérités sont représentés par l'icône .

Concept apparenté : [Inheritance and value resolution](#) [p 288].

vues

Configuration d'affichage applicable sur une table. Une vue peut être créée pour un utilisateur ou un rôle et permet de spécifier notamment sous quel mode seront affichés les enregistrements : hiérarchique ou tabulaire ; ainsi que de définir des critères de filtrage et de tri.

Le mode de vue hiérarchique présente les données d'une table sous forme d'arbre. Cette vue est utilisée 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 aussi

[Vues](#) [p 131]

[Hiérarchies](#) [p 133]

vue recommandée

Une vue recommandée peut être définie par le propriétaire d'un jeu de données pour chaque profil cible. Quand un utilisateur se connecte sans spécifier de vue, la vue recommandée, si elle existe, s'applique. Sinon, la vue par défaut s'affiche.

L'action 'Gérer les vues recommandées' permet de définir les règles d'attribution des vues recommandées par utilisateur et par rôle.

Concept apparenté : [Vues recommandées](#) [p 136].

vue favorite

Lors de la consultation d'une table, l'utilisateur peut choisir de définir la vue actuelle comme sa vue favorite via la barre d'outils du menu 'Vue'.

Une fois définie comme favorite, la vue s'appliquera automatiquement pour cet utilisateur lors de chaque accès à la table.

Concept apparenté : [Barre d'outils du menu 'Vue'](#) [p 137].

3.4 Cycle de vie des données

Section de la documentation [Espaces de données](#) [p 100]

espace de données

Contient les 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 avec son parent. A tout moment, une comparaison avec d'autres espaces de données est possible.

Voir [héritage](#) [p 29], [référentiel](#) [p 25], [fusion](#) [p 30].

Concept apparenté : [Dataspaces](#) [p 100].

L'ancien nom (avant la version 5) pour "espace de données" était "branch" ou "snapshot".

espace de données de référence

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

Voir [espace de données](#) [p 30], [fusion](#) [p 30], [référentiel](#) [p 25].

fusion

Intégration, dans l'espace de données parent, des changements réalisés dans un espace de données enfant depuis sa création. L'espace de données enfant est fermé après une fusion 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 modification des 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](#) [p 109].

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](#) [p 115]

L'ancien nom (avant la version 5) pour "image" était "version" ou "home".

3.5 Historique

Section de la documentation [History](#) [p 269]

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 telles que l'export, la comparaison et les filtres restent 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](#) [p 31], [vue historique des transactions](#) [p 31], [profil d'historisation](#) [p 31].

profil d'historisation

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

Voir [profil d'historisation](#) [p 31].

vue historique de table

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

Référence technique apparentée [History](#) [p 269].

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 plusieurs tables dans un ou plusieurs jeux de données, cette vue montre toutes les opérations qui ont été effectuées dans le périmètre en question pour chaque transaction.

Référence technique apparentée [History](#) [p 269].

3.6 Modélisation de workflow

Section de la documentation [Modèles de workflow](#) [p 166]

modèle de workflow

Définition de la succession d'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 termes d'états ou d'actions associées à effectuer par des utilisateurs et des tâches automatiques.

Concept apparenté : [Modèles de workflow](#) [p 166].

L'ancien nom (avant la version 5) de "modèle de workflow" était "workflow definition".

Les modèles de workflows sont représentés par l'icône .

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](#) [p 31].

tâche utilisateur

Tâche de workflow composée d'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](#) [p 31].

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 .

appel à des sous-workflows

Etape qui met le workflow de données courant en attente et qui lance un ou plusieurs autres workflows de données. Si une telle étape lance plusieurs sous-workflows, les sous-workflows sont exécutés en parallèle.

tâche d'attente

Etape d'un workflow de données qui met en pause le workflow en cours en attendant un événement donné. Lorsque l'événement est reçu, le workflow est réveillé et va automatiquement à l'étape suivante.

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.

3.7 Workflows de données

Section de la documentation [Workflows de données](#) [p 196]

publication de workflow

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

L'ancien nom (avant la version 5) de "publication de workflow" était "workflow".

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](#) [p 31].

Concept apparenté : [Workflows de données](#) [p 196].

L'ancien nom (avant la version 5) de "workflow de données" était "workflow instance".

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](#) [p 33].

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](#) [p 32].

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.

3.8 Services de données

Section de la documentation [Services de données](#) [p 218]

service de données

EBX® partage les données de référence conformément à l' [Architecture Orientée Service](#) en utilisant la technologie XML web service. Tous les services de données sont générés directement à partir des modèles ou services built-in. Ils peuvent être utilisés pour accéder à une partie des fonctionnalités disponibles via l'interface utilisateur.

Les services de données d'EBX® proposent :

- Un générateur WSDL pour les modèles de données ainsi que pour les services built-in. Le fichier WSDL peut-être produit indifféremment au travers de l'interface utilisateur ou du connecteur HTTP(S) pour un logiciel d'intégration ou une application cliente. Les opérations de services sont invoquées par des messages XML communiqués au point d'entrée d'EBX®.
- Un connecteur SOAP, ou point d'entrée pour les messages SOAP, permet aux systèmes externes d'interagir avec le contenu du référentiel. Ce connecteur répond aux demandes issues des WSDL produits par EBX®. Après authentification, il accepte les messages XML et les exécute selon les permissions de l'utilisateur authentifié.
- Un connecteur RESTful, ou point d'entrée pour les opérations de sélection, permet aux systèmes externes d'interroger le contenu du référentiel EBX®. Après authentification, il accepte la requête définie dans l'URL et l'exécute selon les permissions de l'utilisateur authentifié.

lignage

Mécanisme de mise en place de profils de droits d'accès pour des services de données. Les profils de droits d'accès ainsi définis sont utilisés pour accéder aux données via des interfaces WSDL.

Concept apparenté : [Générer un WSDL pour un lignage](#) [p 223].

3.9 Transverse

noeud

Un noeud est un élément d'une arborescence ou d'un graphe. Dans EBX®, 'Noeud' peut faire référence à différents concepts selon le contexte d'utilisation :

- Dans le contexte du [modèle de workflow](#) [p 31], un noeud est une étape du workflow ou une condition.
- Dans le contexte du [modèle de données](#) [p 26], un noeud est un groupe, une table ou un champ.
- Dans le contexte des [hiérarchies](#) [p 29], un noeud représente une valeur d'une dimension.
- Dans un arbre d'[héritage de jeux de données](#) [p 29], un noeud est un jeu de données.
- Dans un [jeu de données](#) [p 28], un noeud est le noeud du modèle de données évalué dans le contexte du jeu de données ou de l'enregistrement.

Modèles de données

CHAPITRE 4

Introduction aux modèles de données

Ce chapitre contient les sections suivantes :

1. [Présentation](#)
2. [Utilisation de l'interface utilisateur de la section Modèles de données](#)

4.1 Présentation

La fonction d'un modèle de données

La première étape de toute gouvernance de données dans TIBCO EBX® est le développement d'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'organisation, de types de données, et de relations sémantiques. Une fois que le modèle de données a été défini et publié, il devient possible de créer des jeux de données à partir de celui-ci.

Afin de définir un modèle de données dans le référentiel, créez d'abord un nouveau modèle de données, puis définissez sa structure et les propriétés de ses éléments (tables, champs et groupes). Le modèle de données ainsi défini doit être publié pour devenir disponible. Les utilisateurs pourront créer des jeux de données à partir de cette publication qui contiendront les données gérées par le référentiel EBX®.

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](#) [p 27]
- [clé primaire](#) [p 27]
- [clé étrangère](#) [p 27]
- [table](#) [p 27]
- [groupe](#) [p 27]
- [type réutilisable](#) [p 28]
- [règle de validation](#) [p 28]

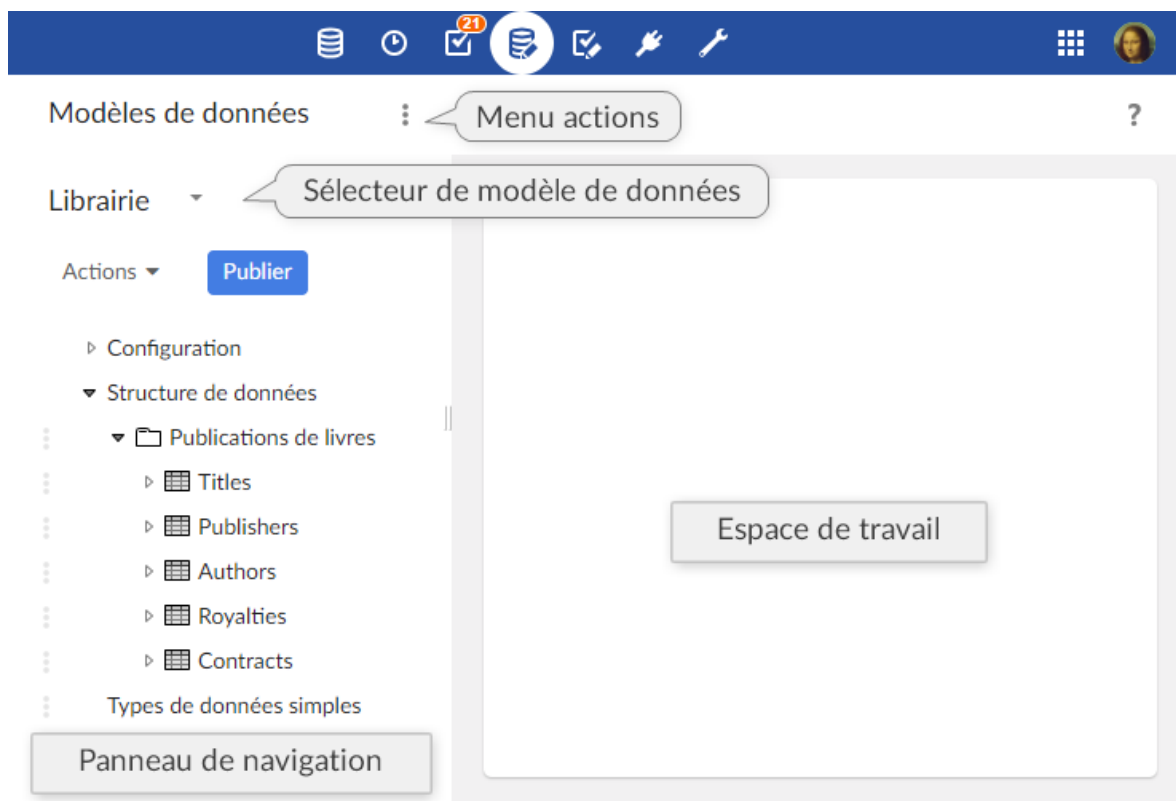
4.2 Utilisation de l'interface utilisateur de la section Modèles de données

Navigation dans le Data Model Assistant

Les modèles de données peuvent être importés, édités, et publiés dans la section **Modèles de données**. L'assistant fourni dans EBX® permet d'élaborer facilement des modèles de données.

Note

Seuls les utilisateurs autorisés peuvent accéder à cet écran via la 'Perspective avancée'.



Le panneau de navigation est organisé selon les sections suivantes :

Configuration	La configuration technique du modèle de données.
Propriétés globales	Définit les propriétés techniques du modèle.
Modèles de données inclus	Définit les modèles de données inclus dans le modèles courant. Tous les types définis par les modèles de données inclus pourront être utilisés dans le modèle courant.
Librairie de composants	Définit les composants Java disponibles dans le modèle. Ces composants représentent l'ensemble des fonctionnalités programmatiques qui seront disponibles dans le modèle (contraintes programmatiques, fonctions, bean, etc.).
Add-ons	Définit les add-ons utilisés par le modèle de données. Ces add-ons pourront enrichir le modèle de données après la publication en ajoutant des propriétés et contraintes aux éléments du modèle.
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.
Extensions	Extensions disponibles dans le modèle de données courant.
Barres d'outils	Les barres d'outils disponibles dans le modèle de données.
Services utilisateurs	Déclare les services utilisateurs utilisant l'API disponible avant la version 5.8.0. A partir de la version 5.8.0, il est recommandé d'utiliser la nouvelle API UserService (ces services étant directement enregistrés via l'API Java, il

n'est plus nécessaire de les déclarer dans le Data Model Assistant).

Services de données	Cette table définit les suffixes des opérations WSDL qui peuvent être utilisés dans le modèle de données. Ces suffixes permettent, dans le cadre d'une opération d'un service de données, de faire référence à une table en utilisant un nom unique au lieu de son chemin dans le modèle de données.
Réplifications	Cette table définit les unités de réplication du modèle de données. Une unité de réplication permet la réplication d'une table source dans la base de données relationnelle, de sorte que les systèmes externes peuvent accéder à ces données par le biais de requêtes et de vues SQL.
Composants Ajax	Définit les composants Ajax disponibles dans le modèle.
'Bindings' du modèle	Les bindings du modèle d'adaptation spécifient quels sont les types Java à générer à partir du modèle.

Voir aussi

[Modélisation de la structure des données](#) [p 47]

[Configuration du modèle de données](#) [p 43]

[Types réutilisables](#) [p 49]

[Extensions de modèles de données](#) [p 77]

Icônes des éléments du modèle de données

 [champ](#) [p 27]

 [clé primaire](#) [p 27]

 [clé étrangère](#) [p 27]

 [table](#) [p 27]

 [groupe](#) [p 27]

Concepts apparentés

[Espaces de données](#) [p 100]

[Jeux de données](#) [p 122]

CHAPITRE 5

Création du modèle de données

Ce chapitre contient les sections suivantes :

1. [Création d'un modèle de données](#)

5.1 Création d'un modèle de données

Pour créer un modèle de données, cliquez sur le bouton **Créer** dans le sélecteur, puis suivez les instructions de l'assistant de création de modèle de données.

CHAPITRE 6

Configuration du modèle de données

Ce chapitre contient les sections suivantes :

1. [Informations associées au modèle de données](#)
2. [Permissions](#)
3. [Propriétés du modèle de données](#)
4. [Modèles de données inclus](#)
5. [Add-ons utilisés par le modèle de données](#)

6.1 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](#) [p 37] dans le panneau de navigation.

Note

Seuls les utilisateurs autorisés peuvent accéder à cet écran via la 'Perspective avancée'.

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.

6.2 Permissions

Pour définir les permissions d'accès au modèle de données, sélectionnez "Permissions" dans le menu ["Actions" du modèle de données](#) [p 37] dans le panneau de navigation.

La définition des permissions d'un modèle de données s'effectue de la même manière que pour les jeux de données. Les détails se trouvent dans la section [Permissions](#) [p 158].

6.3 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. Ce nom de module désigne également le module cible lors de 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	Emplacement des 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	Les deux modes possibles sont : la publication du modèle dans un document XML Schema au sein d'un module ou en tant que modèle de données embarqué dans le référentiel TIBCO EBX®. Les modèles de données embarqués permettent d'avoir des fonctionnalités additionnelles, comme la gestion des versions et la restauration des versions précédentes du modèle. Voir Modes de publication [p 93] pour plus d'informations. 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 par "/".
Héritage des données	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. Voir héritage [p 161] pour plus d'informations.
Documentation	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. Voir Dynamic labels and descriptions [p 592] pour plus d'informations.

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 [p 589] pour plus d'informations.
Activer services utilisateurs (ancienne API)	Indique si des services utilisateurs utilisant l'API disponible avant la version 5.8.0 peuvent être déclarés. Si 'Non', la section "Configuration > Services utilisateurs" n'est pas affichée (sauf si cette section déclare déjà au moins un service). A partir de la version 5.8.0, il est recommandé d'utiliser l'API UserService (ces services étant directement enregistrés via l'API Java, il n'est plus nécessaire de les déclarer dans le Data Model Assistant). Voir <code>UserServiceDeclaration^{API}</code> pour plus d'informations.

6.4 Modèles de données inclus

Les types de données définis dans un autre modèle de données peuvent être utilisés 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 à l'enregistrement du modèle inclus depuis cette table, des informations techniques liées à ce modèle sont consultables sous l'onglet **Information**. Cet onglet contient aussi le rapport de validation du modèle de données inclus.

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

Les types de données doivent être uniques à la fois dans le modèle courant et dans tous les modèles inclus. Il est impossible d'inclure un modèle contenant des types de données déjà existant dans le modèle courant ou dans les autres modèles de données inclus.

Voir aussi [Including external data models](#) [p 546]

6.5 Add-ons utilisés par le modèle de données

Dans tout type de modèle de données, il est possible de définir les *add-ons* utilisés dans le modèle de données courant. Ces add-ons pourront enrichir le modèle de données après la publication en ajoutant des propriétés et contraintes aux éléments du modèle.

Pour définir un add-on en utilisant l'interface utilisateur, créer un nouvel enregistrement dans la table 'Add-ons' située dans la section 'Configuration du modèle de données' dans le panneau de navigation. Un enregistrement de cette table définit les propriétés suivantes :

Nom de l'add-on	Description de l'add-on.
Version de l'add-on.	Indique la version de l'add-on.
Activé	Indique si l'add-on est activé. Un add-on doit être activé pour être utilisé.

CHAPITRE 7

Modélisation de la structure des données

Pour définir la structure du modèle de données, sélectionnez le modèle de données avec lequel vous voulez travailler dans le panneau de navigation.

La structure du modèle de données est accessible depuis 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.

Ce chapitre contient les sections suivantes :

1. [Actions et propriétés communes](#)
2. [Types réutilisables](#)
3. [Détails de la création des éléments du modèle de données](#)
4. [Modification des éléments existants](#)

7.1 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

- associations

Ajoutez un de ces éléments sous un élément existant en cliquant sur la flèche ▼ située à la droite de l'élément existant, puis en sélectionnant une option de création parmi les options présentées dans le menu. Suivez ensuite l'assistant de création pour 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. S'il faut renommer cet élément, il peut être supprimé et recréé avec un nom différent.

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 peut plus être modifié.

Il est possible 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, TIBCO EBX® affichera le libellé et la description localisés de l'élément.

Supprimer des éléments

Tous les éléments du modèle de données 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. Spécifiez 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, en dernière position. Lorsqu'un élément contenant d'autres éléments est dupliqué, tous les sous-éléments sont dupliqués avec leurs noms originaux.

Note

En cas de duplication d'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

Pour déplacer un élément, cliquez sur la flèche ▼ située à la droite de l'élément à déplacer et sélectionnez "Déplacer". Sélectionnez ensuite la flèche qui correspond à l'élément *avant lequel* positionner l'élément actuel.

Note

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

7.2 Types réutilisables

Les types réutilisables sont des éléments partagés qui, après leur création, peuvent être réutilisés par différents éléments du modèle de données.

Note

En modifiant la définition d'un type réutilisable, la structure de tous les éléments basés sur ce type réutilisable est aussi modifiée. L'arborescence "Structure de données" affiche, en lecture seule, la structure d'un groupe ou d'une table qui utilise un type réutilisable. Pour éditer la structure du type réutilisable associé, accédez 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 ▼ des sections "Types de données simples" ou "Types de données complexes" dans le panneau de navigation, il est possible de créer des types simples et des types complexes réutilisables qui seront disponibles pour créer d'autres éléments avec la même structure et les mêmes propriétés. Il est également possible de 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 n'est utilisé par aucun élément, il peut être supprimé 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 à la création d'un élément. L'élément créé utilisera la structure et les propriétés du type réutilisable.

Inclusion des types de données définis dans d'autres modèles de données

Les types réutilisables peuvent aussi être partagés entre plusieurs modèles de données. En configurant l'inclusion d'un modèle de données externe, il est possible d'utiliser les types de données inclus pour

créer des éléments dans la structure de données, de la même manière que pour les types réutilisables définis en local.

Note

Les types de données devant être uniques pour tous les types définis en local et inclus, il n'est pas possible de créer un type réutilisable portant le même nom qu'un type de données dans un modèle de données inclus. De la même manière, il n'est pas possible d'inclure un modèle de données externe qui définit un type de données portant le même nom qu'un type réutilisable défini en local ou dans un autre modèle de données inclus.

Les types de données inclus apparaissent dans les sections "Types de données simples inclus" et "Types de données complexes inclus" dans le panneau de navigation. Les détails de ces types réutilisables sont consultables, mais ils ne sont éditables que dans leurs modèles de données d'origine.

Voir [Modèles de données inclus](#) [p 45] pour plus d'informations.

7.3 Détails de la création des éléments du modèle de données

Création de champs

A la création d'un champ, un type de données doit être sélectionné. Il 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.

Durant la création d'un champ, il est également possible de le désigner comme clé étrangère, champ obligatoire, ou comme clé primaire si le champ est créé sous une table.

Création de tables

Lors de la création d'une nouvelle table, un type réutilisable existant peut être utilisé pour définir la structure et les propriétés de cette nouvelle table. Voir [Types réutilisables](#) [p 49] pour plus d'informations.

Chaque table nécessite la désignation d'au moins un champ clé primaire, qui peut être créé comme un élément enfant de la table dans la section "Structure de données" du panneau de navigation.

Création de groupes

Lors de la création d'un groupe, il est possible d'utiliser un type réutilisable existant pour définir la structure et les propriétés du nouveau groupe. Voir [Types réutilisables](#) [p 49] pour plus d'informations.

Création de clés primaires

Pour chaque table, il est nécessaire de définir une clé primaire. Pour cela, ajoutez un nouvel élément enfant 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, sur l'onglet "Clé primaire" dans les "Propriétés avancées" de la table.

Création ou définition de clés étrangères

Les champs associés à une clé étrangère sont de type "Chaîne de caractères". Pour créer une clé étrangère sur une table, ajoutez un nouvel élément enfant à partir du menu d'actions disponible sur la table dans la section "Structure de données" du panneau de navigation. Il est également possible de 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.

Il faut toujours spécifier la table référencée par une clé étrangère.

Création d'associations

Une association permet de définir un lien sémantique entre deux tables. Une association peut être définie en créant un élément dans une table de la section "Structure de données" du panneau de navigation en sélectionnant la propriété "association" dans le formulaire de création. Une association peut uniquement être créée dans une table et il n'est pas possible de convertir un champ existant en association.

Lors de la création d'une association, spécifiez son type. Pour cela, différentes options sont disponibles :

- Relation inverse d'une *clé étrangère*. Dans ce cas, l'association est définie dans une *table source* et fait référence à une *table cible*. Ce type d'association est l'inverse d'une clé étrangère qui est définie dans la table cible et qui référence la table source. Définissez la clé étrangère qui référence la table contenant l'association. Pour cela, des assistants de saisie sont disponibles lors de la création de l'association.
- Utilisation d'une *table de liens*. Dans ce cas, l'association est définie dans une *table source* et fait référence à une *table cible* qui est inférée à partir d'une *table de liens*. Cette table de liens doit définir deux clés étrangères : une clé étrangère référençant la table source et une autre référençant la table cible. La clé primaire de la table de liens doit aussi être composée de champs auto-incrémentés et/ou de clés étrangères vers la table source ou cible de l'association. Définissez la table de liens et ces deux clés étrangères. Pour cela, des assistants de saisie sont disponibles à la création de l'association.
- Utilisation d'un *prédicat XPath*. Dans ce cas, l'association est définie dans une *table source* et fait référence à une *table cible* spécifié par un chemin *XPath*. Une *expression XPath* doit aussi être définie afin de spécifier les critères qui permettent d'associer un enregistrement de la table courante avec des enregistrements de la table cible.

Pour ces types d'association, nous appelons *enregistrements associés* les enregistrements de la table cible sémantiquement liés aux enregistrements de la table source.

Une fois l'association créée, il est possible de définir d'autres propriétés :

- filtrer les enregistrements associés en définissant un filtre *XPath*. Il est uniquement possible d'utiliser les champs de la table source et de la table cible lors de la définition du filtre *XPath*. Il n'est donc pas possible d'utiliser les champs d'une table de liens dans un filtre *XPath*. Un assistant de saisie est disponible pour définir les champs à utiliser dans un filtre *XPath*.
- configurer une vue tabulaire pour définir les champs de la table cible devant être affichés dans les formulaires des jeux de données. Il n'est pas possible de configurer ou de modifier une vue tabulaire si la table cible n'est pas définie ou n'existe pas. Par défaut, tous les champs de la table

cible qu'un utilisateur a le droit de voir seront affichés dans les jeux de données si la vue tabulaire n'est pas définie.

- définir comment doivent être présentés les enregistrements associés dans les formulaires des jeux de données. Indiquez si les enregistrements associés doivent être inclus dans le formulaire ou dans un onglet spécifique. Par défaut, les enregistrements associés seront inclus dans le formulaire au niveau de l'association dans le modèle de données.
- inclure / exclure les enregistrements associés dans les opérations de sélection de Data Service. Par défaut, les enregistrements ne sont pas inclus dans les opérations de sélection des Data Service.
- spécifier les nombres minimum et maximum d'enregistrements associés requis. Dans les jeux de données associés, un message de validation est ajouté lorsque les nombres minimum ou maximum d'enregistrements associés ne correspondent pas à ces critères. Par défaut, les nombres minimum et maximum d'enregistrements associés requis ne sont pas restreints.
- définir une contrainte de validation en utilisant un prédicat XPath pour restreindre les enregistrements associés. Il est uniquement possible d'utiliser les champs de la table source et de la table cible lors de la définition du prédicat XPath. Il n'est donc pas possible d'utiliser les champs d'une table de liens dans un prédicat XPath. Un assistant de saisie permet de sélectionner les champs à utiliser dans un prédicat XPath. Dans les jeux de données associés, un message de validation sera ajouté pour tout enregistrement associé ne vérifiant pas cette contrainte.

7.4 Modification des éléments existants

Suppression d'un champ de la clé primaire

Tout champ appartenant à la clé primaire peut être supprimé de la clé primaire d'une table sur l'onglet "Clé primaire" dans les "Propriétés avancées" de la table.

Voir [clé primaire](#) [p 27] dans le glossaire.

CHAPITRE 8

Propriétés des éléments du modèle de données

Après avoir créé un élément, il est possible de définir des propriétés supplémentaires pour compléter sa définition.

Voir aussi [Contrôles sur les éléments du modèle de données](#) [p 69]

Ce chapitre contient les sections suivantes :

1. [Propriétés basiques des éléments](#)
2. [Propriétés avancées des éléments](#)

8.1 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	Nombre minimum de valeurs pour cet élément. 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 est automatiquement défini à "0".
Nombre maximum de valeurs	Nombre maximum de valeurs pour cet élément. Si cette propriété est supérieure à "1", l'élément est considéré comme multi-valué. Les clés primaires ne pouvant être multi-valuées, cette propriété doit être égale à "1" ou être "non définie". Pour une table, le nombre maximum d'éléments est défini à "unbounded" par défaut lors de sa création. 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.
Règles de validation	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 des types complexes réutilisables et les noeuds de sélection. Cette propriété est utilisée pour définir des règles de validation riches et complexes à l'aide d'un éditeur de prédicats XPath 1.0. Voir éditeur de prédicats [p 313]. Une règle de validation peut être utile lorsque la validation de la valeur dépend de critères complexes ou des valeurs d'autres champs. Il est aussi possible d'indiquer qu'une règle de validation définit des conditions sous lesquelles la valeur d'un champ est obligatoire. La valeur du champ est obligatoire si les critères de la règle ne sont pas respectés. Voir Contraintes sur valeurs "null" [p 579] pour plus d'informations. En utilisant l'assistant associé, il est possible de définir des libellés localisés pour la règle de validation, ainsi qu'un

message localisé avec la sévérité qui s'affichera si le critère n'est pas satisfait.

Il est possible d'indiquer si la règle doit toujours être valide ou non lorsqu'un utilisateur soumet un formulaire. Cette option est uniquement disponible sur les règles de validation définies par un champ et lorsque la sévérité du message est définie à "erreur". Si les erreurs de validation ne sont pas autorisées, alors toutes les saisies qui rendraient la règle invalide seront rejetées et les données seront non modifiées. Si les erreurs de validation sont autorisées alors toutes les saisies qui rendraient la règle invalide seront acceptées et les données seront modifiées. Si cette propriété n'est pas spécifiée, alors la règle bloquera uniquement les erreurs lors de la soumission d'un formulaire.

Si une règle de validation est définie sur un champ de type table, alors cette règle sera considérée comme une "contrainte sur table" et sera exécutée sur chaque enregistrement de la table. Voir [Contraintes sur table](#) [p 580] pour plus d'informations.

Propriétés basiques des champs

Les propriétés basiques suivantes sont spécifiques aux champs simples :

Valeur par défaut	Définit une valeur par défaut pour ce champ. Cette valeur sera insérée automatiquement dans le champ de saisie du formulaire de création des nouveaux enregistrements. Le type de la valeur par défaut doit être compatible avec le type du champ courant. Voir Valeur par défaut [p 597] pour plus d'informations.
Message d'erreur de conversion	Messages d'erreur internationalisés affichés aux utilisateurs lors de la saisie d'une valeur qui n'est pas compatible avec le type du champ courant.
Règle de calcul	Cette propriété est disponible pour les champs des tables, excepté 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. Voir Editeur de prédicats [p 313] pour plus d'informations. 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. Les limitations suivantes existent pour les règles de calcul : • Les règles de calcul s'utilisent uniquement avec les champs simples d'une table. • Les règles de calcul ne peuvent pas être définies sur des champs du type OResource ou Password. • Les règles de calcul ne peuvent pas être définies sur les noeuds de sélection ni les champs clés primaires. • Les règles de calcul ne peuvent pas être définies en accédant à l'élément depuis le rapport de validation.

8.2 Propriétés avancées des éléments

Propriétés avancées communes

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

Affichage par défaut et outils > Visibilité

Cette section permet d'indiquer si cet élément doit être affiché dans la vue par défaut d'un jeu de données, dans l'outil de recherche textuelle d'un jeu de données, ou dans l'opération "select" d'un service de données. Ces propriétés sont ignorées si elles sont définies sur un élément qui n'est pas dans une table.

- Vue dirigée par le modèle

Indique si l'élément courant doit être affiché ou non dans la vue par défaut d'un jeu de données, la vue tabulaire par défaut d'une table et le formulaire par défaut associé aux enregistrements d'une table. La vue par défaut d'un jeu de données, la vue tabulaire par défaut d'une table, et le formulaire par défaut associé aux enregistrements d'une table respectent la structure du jeu de données ou de la table définie dans le modèle de données. Lorsque la propriété "Caché" est sélectionnée, l'élément courant ne sera pas affiché dans la vue par défaut d'un jeu de données si l'élément courant est une table. Lorsque la propriété "Caché" est sélectionnée, l'élément courant ne sera pas affiché dans la vue tabulaire par défaut et le formulaire par défaut associé aux enregistrements d'une table si l'élément courant est dans une table. L'élément courant sera cependant affiché dans l'assistant de création de vues tabulaires et hiérarchiques. Il sera donc possible de créer une vue personnalisée qui affichera l'élément courant. Si cette propriété n'est pas renseignée, alors l'élément courant sera affiché par défaut. Cette propriété est ignorée si elle est définie sur un champ qui n'est ni une table, ni dans une table.

- Toutes les vues

Lorsque la propriété 'Caché dans les vues' est sélectionnée, l'élément courant ne sera pas affiché dans les vues par défaut, tabulaires (vue par défaut incluse) ou hiérarchiques, d'une table d'un jeu de données. L'élément courant ne sera pas affiché dans l'assistant de création de vues tabulaires et hiérarchiques. Il ne sera donc pas possible de créer une vue personnalisée qui affichera l'élément courant. Cette propriété concerne uniquement les vues. L'élément courant sera donc affiché dans le formulaire d'un enregistrement si la

propriété 'Caché dans les vues' est sélectionnée. Cette propriété est ignorée si elle est définie sur un champ hors d'une table.

- Recherche structurée

Lorsque la propriété "Caché dans la recherche structurée" est sélectionnée, l'élément courant ne sera pas affiché uniquement dans l'outil de recherche typée d'un jeu de données. Cet élément sera tout de même pris en compte dans l'outil de recherche rapide. Si ces propriétés concernant les recherches ne sont pas renseignées, alors l'élément courant sera affiché par défaut dans l'outil de recherche typée d'un jeu de données. Cette propriété est ignorée si elle est définie sur un champ hors d'une table.

- Services de données

Lorsque la propriété "Exclu dans les Data Services" est sélectionnée, l'élément courant ne sera pas inclus lors de l'opération "select" d'un data service. Cette propriété est ignorée si elle est définie sur un champ hors d'une table.

Voir [Default view](#) [p 599] dans le Guide du développeur.

Affichage par défaut et outils > Widget

Définit le widget à utiliser. Un widget est un composant de saisie qui est affiché dans les formulaires contenus dans les jeux de données associés. Si aucun widget n'est défini, alors un widget par défaut sera affiché dans les jeux de données associés. Ce widget par défaut est défini à partir du type et des propriétés de l'élément courant. Il est possible d'utiliser un widget natif ou un widget personnalisé. Un widget personnalisé est défini en utilisant une API Java. Cette API Java permet de développer des composants de saisie riche pour des champs ou des groupes. Les widgets natifs et personnalisés ne peuvent pas être définis sur une table et une association. Il est interdit de définir en même temps un widget personnalisé et un UI bean. Sur un champ de type clé étrangère, il est interdit de définir en même temps un widget personnalisé et un sélecteur de combo box.

Voir `UIWidgetFactoryAPI` pour plus d'informations.

Affichage par défaut et outils > Sélecteur combo box

Définit le nom de la vue publiée utilisée dans le menu de sélection de la clé étrangère. Un bouton de sélection sera affiché en bas à droite de la liste déroulante du menu de sélection. Lors de la définition d'une clé étrangère, cette fonctionnalité permet d'accéder à une vue de sélection avancée en cliquant sur le bouton 'Sélecteur' qui ouvre la vue de sélection avancée, permettant ainsi d'utiliser les options de tri et de recherche. Si aucune vue publiée n'a été définie,

la vue de sélection avancée sera désactivée. Si le chemin de la table référencée est absolu, alors seules les vues publiées correspondant à cette table seront affichées. Si le chemin de la table référencée est relatif, alors toutes les vues associées au modèle de données contenant la table cible seront affichées. Cette propriété ne peut pas être utilisée si un widget personnalisé est défini.

Voir [Defining a view for the combo box selector of a foreign key](#) [p 601] dans le Guide du développeur.

UI bean

Attention

Depuis la version 5.8.0 de TIBCO EBX®, il est recommandé d'utiliser les widgets personnalisés à la place des UI beans. Les widgets personnalisés proposent plus de fonctionnalités que les UI beans. Plus d'évolutions seront apportées aux UI Beans. Voir [_widget](#) [p 58] pour plus d'informations.

Cette propriété est disponible pour tous les types d'éléments, excepté 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`^{API} dans l'API Java.

Transformation à l'export

Cette propriété est disponible pour les champs et pour les groupes qui sont des noeuds terminaux.

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.

Voir `NodeDataTransformer`^{API} pour plus d'informations.

Propriétés d'accès

Définit le mode d'accès pour l'élément courant, à savoir : s'il peut être écrit et/ou lu.

- "Lecture & Ecriture" correspond au mode `RW` dans le XSD du modèle de données.
- "Lecture seule" correspond au mode `R-` dans le XSD du modèle de données.
- "Elément hors d'un jeu de données" correspond au mode `CC` dans le XSD du modèle de données.
- "Noeud non terminal" correspond au mode `--` dans le XSD du modèle de données.

Voir [Access properties](#) [p 597] dans le Guide du développeur.

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'enregistrement de tables, à l'exception de la catégorie "Hidden".

Voir [Categories](#) [p 603] dans le Guide du développeur.

Mode de comparaison

Définit un mode de comparaison pour l'élément. Ce mode permet de restreindre la comparaison des données associées à l'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'import 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](#) [p 602] dans le Guide du développeur.

Règle d'application des dernières modifications

Indique si cet élément doit être inclus ou non dans le service permettant d'appliquer à d'autres enregistrements de la même table les dernières modifications effectuées.

- "Défaut" indique que la dernière modification effectuée sur cet élément peut être appliquée à d'autres enregistrements.
- "Ignoré" indique que la dernière modification effectuée sur cet élément ne peut pas être appliquée à d'autres enregistrements. Cet élément ne sera pas visible dans le service permettant d'appliquer les dernières modifications.

Voir [Apply last modifications policy](#) [p 602] dans le Guide du développeur.

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 obligatoires est effectuée, non pas 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 égal ou supérieur 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.

Voir [Constraints, triggers and functions](#) [p 584] dans le Guide utilisateur.

Supprimer espaces

Supprimer espaces

Implémente la propriété `osd:trim`. Cette propriété permet d'indiquer si les espaces avant ou après une valeur doivent être supprimés lors de la saisie utilisateur. Si cette propriété n'est pas définie, alors les espaces avant ou après une valeur seront supprimés par défaut lors de la saisie utilisateur.

Voir [Whitespace handling upon user input](#) [p 585] dans le Guide utilisateur.

UI bean

Voir [Propriétés avancées communes](#) [p 59].

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`^{API}.

Désactiver validation

Indique si les contraintes définies sur le champ doivent être désactivées. Cette propriété peut uniquement être définie sur des champs calculés. Si 'Oui', les contraintes simples, avancées et de cardinalité ne seront pas exécutées lors de la validation des jeux de données associés au modèle de données.

Transformation à l'export

Voir [Propriétés avancées communes](#) [p 59].

Propriétés d'accès

Voir [Propriétés avancées communes](#) [p 59].

Auto-incrément

Cette propriété est disponible uniquement pour les champs de type "entier" 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 s'il faut désactiver 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.

Les valeurs auto-incrémentées ont le comportement suivant :

- Le calcul et l'allocation de la valeur de ce champ sont effectués dès 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 différente de 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 à tous les jeux de données basés sur le modèle de données, et également à tous les espaces de données. Ce dernier cas de figure permet de fusionner un espace de données à son parent avec une garantie raisonnable d'absence de conflit 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 [Auto-incremented values](#) [p 589] dans le Guide du développeur.

Affichage par défaut

Voir [Propriétés avancées communes](#) [p 57].

Catégorie du noeud

Voir [Propriétés avancées communes](#) [p 60].

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](#) [p 29] dans le glossaire.

Pour plus d'informations, voir aussi [Inherited fields](#) [p 290].

Propriétés avancées des tables

Les propriétés avancées suivantes sont spécifiques aux tables.

Table

Clé primaire	Liste des champs composant la clé primaire de la table. Il est possible d'ajouter ou de supprimer des champs de la clé primaire. Chaque champ de la clé est identifié par un chemin XPath absolu qui débute sous la table. Si la clé est composite, la liste des champs doit être séparée par des espaces. Par exemple, <code>"/name /startDate"</code>.
Présentation	Spécifie la manière dont les enregistrements de cette table sont affichés dans l'interface utilisateur dans les jeux de données associés.
Labellisation des enregistrements	Définit les champs composant le libellé par défaut et les libellés localisés pour les enregistrements de la table. Peut aussi définir une classe Java pour affecter le libellé programmatiquement, ou définir le libellé dans une hiérarchie. Cette classe Java doit implémenter l'interface <code>UILabelRenderer^{API}</code> ou <code>UILabelRendererForHierarchy^{API}</code> de l'API Java. Attention : Les droits d'accès définis dans les jeux de données associés ne sont pas appliqués lors de l'affichage des libellés. Les champs habituellement cachés à cause de droits d'accès seront affichés dans les libellés des enregistrements.
Présentation par défaut des groupes dans un formulaire	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. Voir Record form: rendering mode for nodes [p 431] dans le Guide d'administration. Présentation activée des groupes : spécifie un mode de présentation autorisé en complément de "Développé" et "Réduit", qui sont toujours disponibles. Les liens doivent être activés sur la 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". Cela permet d'éviter les complexités d'affichage qui se posent quand les liens et les onglets sont dans la même interface utilisateur.

Présentation spécifique d'un formulaire

Définit une présentation spécifique pour personnaliser le formulaire d'un enregistrement dans les jeux de données associés.

Voir `UIFormAPI` et `UserServiceRecordFormFactoryAPI` pour plus d'informations.

Barres d'outils

Définit les barres d'outils à afficher dans la table.

Les barres d'outils peuvent être consultées et modifiées dans la section *Configuration > Barre d'outils*.

Entête de table: Définit la barre d'outils à afficher en entête de la vue table par défaut.

Ligne de table: Définit la barre d'outils à afficher pour chaque ligne de la vue table par défaut.

Entête de formulaire: Définit la barre d'outils à afficher en entête du formulaire d'un enregistrement.

Entête de hiérarchie: Définit la barre d'outils à afficher en entête de la hiérarchie par défaut de la table.

Voir [Barre d'outils](#) [p 79] pour plus d'informations.

Historisation

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**.

Voir [Configuration de l'historique dans le référentiel](#) [p 269] pour plus d'informations.

Filtres spécifiques

Définit des filtres pour afficher uniquement les enregistrements correspondant à certains critères.

Actions

Indique les actions autorisées ou interdites dans le contexte d'un jeu de données. Toutes les actions sont autorisées par défaut, cependant des droits d'accès peuvent être définis dans les jeux de données associés pour restreindre ces actions.

Contraintes d'unicité

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

Triggers à la mise à jour

Spécifie des classes Java définissant des méthodes exécutées automatiquement 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](#) [p 588] dans le Guide du développeur.

Propriétés d'accès

Voir [Propriétés avancées communes](#) [p 59].

Affichage par défaut

Voir [Propriétés avancées communes](#) [p 57].

Catégorie du noeud

Voir [Propriétés avancées communes](#) [p 60].

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](#) [p 59].

Transformation à l'export

Voir [Propriétés avancées communes](#) [p 59].

Propriétés d'accès

Voir [Propriétés avancées communes](#) [p 59].

Affichage par défaut

Visibilité	Voir Propriétés avancées communes [p 57].
Présentation	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. Propriétés de l'onglet 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 en fonction de la position du groupe dans le modèle de données.

Catégorie du noeud

Voir [Propriétés avancées communes](#) [p 60].

Concepts apparentés [Contrôles sur les éléments du modèle de données](#) [p 69]

CHAPITRE 9

Contrôles sur les éléments du modèle de données

Après la création d'un élément, complétez sa définition avec des contrôles supplémentaires.

Voir aussi [Propriétés des éléments du modèle de données](#) [p 53]

Ce chapitre contient les sections suivantes :

1. [Contrôles simples](#)
2. [Contrôles avancés](#)

9.1 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	Nombre exact de caractères requis pour ce champ.
Longueur minimale	Nombre minimum de caractères requis pour ce champ.
Longueur maximale	Nombre maximum de caractères autorisés pour ce champ.
Pattern	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 autorisés dans la partie décimale de la valeur d'un champ de type décimal.
Nombre maximum de chiffres	Nombre maximum de chiffres composant la valeur d'un champ de type entier ou décimal.
Énumération	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 données, alors une énumération représentant l'intersection entre ces deux énumérations sera utilisée 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](#) [p 571].

9.2 Contrôles avancés

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

Voir aussi [Dynamic constraints](#) [p 575]

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 d'un autre jeu de données du même espace de données, ou un jeu de données d'un autre espace de données.
Mode	Emplacement de la table ciblée par la clé étrangère. "Défaut" : modèle courant. "Autre jeu de données" : jeu de données qui se trouve dans le même espace de données. "Autre espace de données" : jeu de données appartenant à un espace de données différent.
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. Nom unique de l'espace de données contenant la table référencée.
Libellé	Définit les champs composant un libellé par défaut et des libellés localisés pour présenter les enregistrements de la table cible. 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 <code>TableRefDisplay^{API}</code> de l'API Java. Attention : Les droits d'accès définis dans les jeux de données associés ne sont pas appliqués lors de l'affichage des libellés. Les champs habituellement cachés à cause de droits d'accès seront affichés dans les libellés des enregistrements.
Filtre	Définit un filtre de clé étrangère en utilisant une expression XPath. Peut aussi spécifier une classe Java qui implémente l'interface <code>TableRefFilter^{API}</code> de l'API Java.

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 maximale non autorisée pour ce champ. Valeur minimale exclue : La valeur minimale non autorisée pour ce champ.
Contrainte spécifique (composant)	Spécifie une ou plusieurs classes Java qui implémentent l'interface <code>Constraint^{API}</code> de l'API Java. Voir Programmatic constraints [p 578] pour plus d'informations.
Énumération spécifique (composant)	Spécifie une classe Java pour définir une énumération. La classe doit définir une liste ordonnée de valeurs, en implémentant l'interface <code>ConstraintEnumeration^{API}</code> de l'API Java.
Énumération 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.
Configuration des espaces de données	Définit quels sont les espaces de données pouvant être référencés par un champ de type Identifiant d'espaces de données (osd:dataspaceKey). • Inclusions Définit les espaces de données qui peuvent être référencés par ce champ. Pattern:Définit une expression régulière à laquelle le nom d'un espace de données doit être conforme.

Type:Définit le type d'espaces de données (branches ou versions) qui peuvent être référencés par ce champ. Si non définie, cette restriction s'appliquera aux espaces de données de type branche.

Inclure les descendants:Définit si les espaces de données enfants ou descendants sont inclus dans cet ensemble. Si non définie, cette restriction ne s'appliquera pas aux espaces de données enfants. Si "Aucun" alors les enfants et descendants des espaces de données qui vérifient le pattern défini ne sont pas inclus. Si "Tous les descendants" alors toutes les branches et versions descendantes des espaces de données qui vérifient le pattern défini sont incluses. Si "Toutes les branches descendantes" alors toutes les branches descendantes des espaces de données qui vérifient le pattern défini sont incluses. Si "Toutes les versions descendantes" alors toutes les versions descendantes des espaces de données qui vérifient le pattern défini sont incluses. Si "Branches enfants" alors seules les branches directes des espaces de données qui vérifient le pattern défini sont incluses. Si l'espace de données qui vérifie le pattern est une version, alors les branches qui sont des enfants directs de cette version sont incluses. Si l'espace de données qui vérifie le pattern est une branche, alors sont incluses les branches qui sont des enfants directs des versions qui sont des enfants directs de cette branche. Si "Versions enfants" alors seules les versions directes des espaces de données qui vérifient le pattern défini sont incluses. Si l'espace de données qui vérifie le pattern est une branche, alors les versions qui sont des enfants directs de cette branche sont incluses. Si l'espace de données qui vérifie le pattern est une version, alors sont incluses les versions qui sont des enfants directs des branches qui sont des enfants directs de cette version.

- Exclusions

Définit les espaces de données qui ne peuvent pas être référencés par ce champ. Les exclusions sont ignorées si la configuration ne définit pas d'inclusion.

Pattern: Définit une expression régulière à laquelle le nom d'un espace de données doit être conforme.

Type: Définit le type d'espaces de données (branches ou versions) qui peuvent être référencés par ce champ. Si non définie, cette restriction s'appliquera aux espaces de données de type branche.

Inclure les descendants: Définit si les espaces de données enfants ou descendants sont inclus dans cet ensemble. Si non définie, cette restriction ne

s'appliquera pas aux espaces de données enfants. Si "Aucun" alors les enfants et descendants des espaces de données qui vérifient le pattern défini ne sont pas inclus. Si "Tous les descendants" alors toutes les branches et versions descendantes des espaces de données qui vérifient le pattern défini sont incluses. Si "Toutes les branches descendantes" alors toutes les branches descendantes des espaces de données qui vérifient le pattern défini sont incluses. Si "Toutes les versions descendantes" alors toutes les versions descendantes des espaces de données qui vérifient le pattern défini sont incluses. Si "Branches enfants" alors seules les branches directes des espaces de données qui vérifient le pattern défini sont incluses. Si l'espace de données qui vérifie le pattern est une version, alors les branches qui sont des enfants directs de cette version sont incluses. Si l'espace de données qui vérifie le pattern est une branche, alors sont incluses les branches qui sont des enfants directs des versions qui sont des enfants directs de cette branche. Si "Versions enfants" alors seules les versions directes des espaces de données qui vérifient le pattern défini sont incluses. Si l'espace de données qui vérifie le pattern est une branche, alors les versions qui sont des enfants directs de cette branche sont incluses. Si l'espace de données qui vérifie le pattern est une version, alors sont incluses les versions qui sont des enfants directs des branches qui sont des enfants directs de cette version.

- Filtre

Définit un filtre pour accepter ou refuser des espaces de données dans le contexte d'un jeu de données ou d'un enregistrement. Ce filtre est uniquement utilisé par le composant de saisie dédié à ce type de champ. Ce filtre n'est pas utilisé lors de la validation de ce champ. Des contrôles additionnels peuvent être définis en utilisant une contrainte spécifique (composant). Un filtre est défini par une classe Java qui implémente l'interface `DataspaceSetFilterAPI`.

Configuration des jeux de données

Définit quels sont les jeux de données pouvant être référencés par un champ de type Identifiant de jeux de données (osd:datasetName).

- Inclusions

Définit les jeux de données qui peuvent être référencés par ce champ.

Pattern: Définit une expression régulière à laquelle le nom d'un jeu de données doit être conforme.

Inclure les descendants: Définit si les jeux de données enfants ou descendants sont inclus dans cet ensemble. Si non définie, cette restriction ne s'appliquera pas aux jeux de données enfants.

- Exclusions

Définit les jeux de données qui ne peuvent pas être référencés par ce champ. Les exclusions sont ignorées si la configuration ne définit pas d'inclusion.

Pattern: Définit une expression régulière à laquelle le nom d'un jeu de données doit être conforme.

Inclure les descendants: Définit si les jeux de données enfants ou descendants sont inclus dans cet ensemble. Si non définie, cette restriction ne s'appliquera pas aux jeux de données enfants.

- Filtre

Définit un filtre pour accepter ou refuser des jeux de données dans le contexte d'un jeu de données ou d'un enregistrement. Ce filtre est uniquement utilisé par le composant de saisie dédié à ce type de champ. Ce filtre n'est pas utilisé lors de la validation de ce champ. Des contrôles additionnels peuvent être définis en utilisant une contrainte spécifique (composant). Un filtre est défini par une classe Java qui implémente l'interface `DatasetSetFilter`^{API}.

Propriétés 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 ils 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".
Gestion des erreurs	Spécifie le comportement de la contrainte en cas d'erreur de validation. Il est possible d'indiquer que la contrainte doit toujours être valide pour toute opération (mise à jour d'un jeu de données, création, mise à jour et suppression d'un enregistrement), ou lorsqu'un utilisateur soumet un formulaire. Dans ce cas, toutes les saisies ou opérations qui rendraient la contrainte invalide seront rejetées et les données seront non modifiées. Si cette propriété n'est pas spécifiée, alors la contrainte bloquera uniquement les erreurs lors de la soumission d'un formulaire, sauf dans le cas de modèles relationnels où les contraintes de clés étrangères bloquent par défaut toutes les erreurs. Cette propriété est uniquement disponible sur les contrôles statiques, valeurs exclues, plage exclue de valeurs et contraintes de clés étrangères. Le caractère bloquant pour toutes les opérations ne s'applique pas sur les filtres de clé étrangère. Une contrainte de clé étrangère n'est donc pas bloquante si un enregistrement référencé existe mais ne répond pas aux critères de filtrage. Il n'est pas possible de définir cette propriété sur les contraintes structurelles définies dans les modèles relationnels. Si cette propriété est définie sur les contraintes "Longueur fixe", "Longueur maximum", "Nombre total de chiffres" et "Partie décimale" dans des modèles relationnels, une erreur sera levée lors de la compilation du modèle. Le caractère bloquant d'une contrainte est ignoré lors de l'import d'archives. Toutes les contraintes bloquantes, sauf les contraintes dites structurelles, sont désactivées lors de l'import d'archives.
Message	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](#) [p 53]

CHAPITRE 10

Extensions de modèles de données

Ce chapitre contient les sections suivantes :

1. [Extensions utilisées par un modèle de données](#)
2. [Indexation et stratégies de recherche](#)
3. [Barres d'outils](#)
4. [Services de données](#)
5. [Réplication des données vers tables relationnelles](#)

10.1 Extensions utilisées par un modèle de données

Une extension offre la possibilité de définir des fonctionnalités additionnelles au modèle de données. Les extensions d'un modèle de données sont affichées dans le panneau de navigation sous la section 'Extensions'.

Les extensions suivantes sont par défaut activées dans les nouveaux modèles de données :

- [Barres d'outils](#) [p 78]
- [Services de données](#) [p 85]
- [Répliques](#) [p 85]
- [Indexation et stratégies de recherche](#) [p 78]
- [Formulaires personnalisés](#) [p 609]
- [Fonctions](#) [p 923]
- [Permissions sur l'enregistrement](#) [p 910]

Certaines extensions sont optionnelles et sont définies dans des jeux de données dédiés. Ces extensions peuvent être désactivées ou activées à partir de l'interface utilisateur.

Une extension peut être désactivée en sélectionnant l'action 'Désactiver l'extension' à partir du menu ▼ située à gauche de l'extension à désactiver.

Note

Lorsqu'une extension est désactivée le dataset associé est supprimé et il n'est pas possible de récupérer son contenu après sa suppression.

A partir du menu ▼ situé à gauche de 'Extensions dans le panneau de navigation, il est possible d'activer une extension en sélectionnant l'action 'Activer une extension'. Cette option est affichée uniquement si des extensions sont disponibles. Après avoir sélectionné cette action un formulaire présente les extensions disponibles et pouvant être activées.

Après avoir activé une extension celle-ci devient accessible dans le panneau de navigation sous l'entrée 'Extensions' et peut être éditée.

10.2 Indexation et stratégies de recherche

Cette fonctionnalité est documentée dans le chapitre [Search](#) [p 315].

10.3 Barres d'outils

Une barre d'outils permet de personnaliser les boutons et menus affichés lors de la consultation de tables ou d'enregistrements dans un jeu de données. Les barres d'outils sont configurables dans le modèle de données via la section 'Configuration'.

Ajoutez une barre d'outils à partir de la section *Barres d'outils* du panneau de navigation, en cliquant sur la flèche ▼ située à droite de l'élément [*Tous les éléments*], puis en sélectionnant l'option *Créer barre d'outils*. Suivez ensuite l'assistant de création pour créer une barre d'outils. Une barre d'outils définit les informations suivantes :

Nom	Nom de la barre d'outils. Le nom de la barre d'outils doit être unique dans le modèle de données, il n'est pas possible de créer plusieurs barres d'outils avec le même nom. Les tables et associations pouvant utiliser des barres d'outils, utilisent ce nom pour définir la barre d'outils à utiliser.
Libellé et description	Libellés et descriptions internationalisés affichés à l'utilisateur.
Modèle de barre d'outils	Permet de créer une barre d'outils à partir d'une barre d'outils par défaut.
Emplacements	Définit les emplacements pour lesquels la barre d'outils peut être utilisée dans les jeux de données associés.

Définir la structure d'une barre d'outils

Une barre d'outils peut définir les éléments suivants :

- [Bouton action](#) [p 80]
- [Bouton menu](#) [p 81]
- [Séparateur](#) [p 81]
- [Groupe de menu](#) [p 82]
- [Action de menu](#) [p 83]
- [Sous-menu](#) [p 83]

Ajoutez un de ces éléments sous une barre d'outils ou à un élément existant en cliquant sur la flèche

▼ située à droite de l'élément existant, puis en sélectionnant une option de création parmi celles présentées dans le menu. Suivez ensuite l'assistant de création pour créer un élément.

Bouton action

Ce type d'élément permet d'associer une action à un bouton dans une barre d'outils. L'action est exécutée lorsque l'utilisateur clique sur le bouton associé dans une barre d'outils. Un élément de type *Bouton action* définit les informations suivantes :

Service	Définit le service à exécuter lorsque l'utilisateur clique sur le bouton. Il est possible de sélectionner un service fourni, ou un service utilisateur défini dans un module ou dans le modèle de données courant. Si la cible 'Composant web' est sélectionnée, le service devra avoir été déclaré comme utilisable en tant que composant web dans les barres d'outils.
Libellé et description	Libellés et descriptions internationalisés affichés à l'utilisateur.
Rendu	Définit la manière d'afficher cet élément dans les jeux de données utilisant la barre d'outils. Il est possible d'afficher : l'icône seule, le texte seul, le texte avec l'icône sur la gauche ou le texte avec l'icône sur la droite.
Icône	Icône à afficher. Il est possible d'utiliser une icône parmi un ensemble d'icônes proposées, ou de faire référence à une icône en utilisant une URL.
Relief	Définit l'apparence du bouton. Le bouton peut être affiché en relief ou avec un design plat.
Est en surbrillance	Indique si le bouton doit être par défaut en surbrillance.

Note

Un élément de type *Bouton action* peut uniquement être créé sous un élément de type *barre d'outils*.

Bouton menu

Ce type d'élément permet de définir un menu qui est affiché lorsque l'utilisateur clique sur le bouton associé dans une barre d'outils. Un élément de type *Bouton menu* définit les informations suivantes :

Libellé et description	Libellés et descriptions internationalisés affichés à l'utilisateur.
Rendu	Définit la manière d'afficher cet élément dans les jeux de données utilisant la barre d'outils. Il est possible d'afficher : l'icône seule, le texte seul, le texte avec l'icône sur la gauche ou le texte avec l'icône sur la droite.
Icône	Icône à afficher. Il est possible d'utiliser une icône parmi un ensemble d'icônes proposées, ou de faire référence à une icône en utilisant une URL.
Relief	Définit l'apparence du bouton. Le bouton peut être affiché en relief ou avec un design plat.
Est en surbrillance	Indique si le bouton doit être par défaut en surbrillance.

Note

Un élément de type *Bouton menu* peut uniquement être créé sous un élément de type *barre d'outils*.

Séparateur

Ce type d'élément permet d'insérer un séparateur sous la forme d'espacements entre deux éléments d'une barre d'outils.

Note

Un élément de type *Séparateur* peut uniquement être créé sous un élément de type *barre d'outils*.

Groupe de menu

Ce type d'élément permet de définir un groupe d'éléments dans un menu. Un élément de type *Groupe de menu* définit les informations suivantes :

Libellé et description	Libellés et descriptions internationalisés affichés à l'utilisateur.
Type de groupe	Indique le type de groupe de menu à créer : - 'Local' permet de créer un groupe de menu vide. - 'Groupe de services' permet d'assigner un groupe de services existant à ce groupe de menu. - 'Menu builder' permet d'assigner un menu prédéfini à ce groupe de menu. Une fois créé, il n'est pas possible de changer le type de ce groupe de menu.
Nom du groupe de service	Définit un groupe de services existant à réutiliser. Un groupe de services est déclaré dans un module et peut inclure d'autres groupes de services. Tous les services contenus dans ce groupe seront affichés dans les jeux de données associés.
Nom du Menu builder	Indique le menu prédéfini à assigner à ce groupe de menu: - 'Menu par défaut "Actions"' correspond au menu 'Actions' de la barre d'outil par défaut. Les services standards et utilisateurs y sont présentés sans distinction. - 'Menu par défaut "Actions"' (avec séparateur) a le même contenu que le menu 'Actions' de la barre d'outil par défaut, mais les services standards et utilisateurs y sont présentés séparément (d'abord les services standards, puis les services utilisateurs).
Services exclus	Définit les services à exclure du groupe de services à réutiliser. Les services exclus ne seront pas affichés dans les jeux de données associés.
Groupes de services exclus	Définit les groupes à exclure du groupe de services à réutiliser. Les services contenus dans les groupes à exclure ne seront pas affichés dans les jeux de données associés.
Politique d'affichage	Si "Filtrage intelligent", les services configurés en accès direct, c'est à dire via un bouton action ou une action de menu, seront retirés de la génération automatique de ce groupe.

Note

Un élément de type *Groupe de menu* peut uniquement être créé sous les éléments suivants :

- Bouton menu
- Sous-menu

Action de menu

Ce type d'élément permet d'associer une action à une entrée d'un menu dans une barre d'outils. L'action est exécutée lorsque l'utilisateur clique sur l'entrée correspondante dans un menu. Un élément de type *Action de menu* définit les informations suivantes :

Libellé et description	Libellés et descriptions internationalisés affichés à l'utilisateur.
Service	Définit le service à exécuter lorsque l'utilisateur clique sur le bouton. Il est possible de sélectionner un service fourni, ou un service utilisateur défini dans un module ou dans le modèle de données courant. Si la cible 'Composant web' est sélectionnée, le service devra avoir été déclaré comme utilisable en tant que composant web dans les barres d'outils.

Note

Un élément de type *Action de menu* peut uniquement être créé sous un élément de type *Groupe de menu*.

Sous-menu

Ce type d'élément permet d'ajouter un sous-menu dans un menu d'une barre d'outils. Un *Sous-menu* définit les informations suivantes :

Libellé et description	Libellés et descriptions internationalisés affichés à l'utilisateur.

Note

Un élément de type *Sous-menu* peut uniquement être créé sous un élément de type *Groupe de menu*.

Supprimer des éléments

Tous les éléments d'une barre d'outils peuvent être supprimés de celle-ci en utilisant la flèche ▼ située à droite de l'élément à supprimer.

Si un élément contenant d'autres éléments est supprimé, alors la suppression est effectuée récursivement sur tous les éléments situés sous l'élément supprimé.

Dupliquer des éléments existants

Pour dupliquer un élément, cliquez sur la flèche ▼ située à droite de l'élément à dupliquer. Spécifiez le nom et les propriétés de l'élément dupliqué. Toutes les propriétés de l'élément source sont dupliquées. L'élément dupliqué est ajouté au même niveau que l'élément d'origine, en dernière position. Lorsqu'un élément contenant d'autres éléments est dupliqué, tous les sous-éléments sont dupliqués avec leurs propriétés.

Déplacer des éléments

Pour déplacer un élément, cliquez sur la flèche ▼ et sélectionnez l'option de déplacement souhaitée.

Associer avec des tables existantes

Pour associer une barre d'outils avec des tables existantes, cliquez sur la flèche ▼ et sélectionnez l'option *Associer à des tables*. Ce service permet de définir une barre d'outils en tant que barre d'outils par défaut d'un ensemble de tables. Pour cela, définissez les positions cibles de la barre d'outils et sélectionnez les tables ou types de données complexes, définissant les propriétés de la table, à associer à la barre d'outils. La barre d'outils sera définie par défaut aux positions définies sur les tables et types sélectionnées.

Exporter les barres d'outils

Il est possible d'exporter les barres d'outils définies dans le modèle dans un document XML. Pour cela, sélectionner l'option *Export XML* disponible dans le menu *Actions* de la section 'Barres d'outils'. Suivez ensuite l'assistant pour exporter les barres d'outils.

Note

Une sélection de barres d'outils peut être exportée en sélectionnant dans la section 'Barres d'outils' les barres d'outils à exporter, puis en sélectionnant l'option *Export XML* disponible dans le menu *Actions*. Les barres d'outils peuvent aussi être exportées en utilisant le service d'export des modèles de données accessible depuis le menu '[Actions](#)' [du modèle de données](#) [p 37] dans le panneau de navigation.

Voir aussi [Import et export de modèles de données](#). [p 89]

Importer des barres d'outils

Il est possible d'importer des barres d'outils existantes à partir d'un document XML. Pour cela, sélectionner l'option *Import XML* disponible dans le menu *Actions* de la section Barres d'outils. Suivez ensuite l'assistant pour importer les barres d'outils.

Note

Les barres d'outils peuvent aussi être importées en utilisant le service d'import de modèles de données accessible depuis le menu '[Actions](#)' [du modèle de données](#) [p 37] dans le panneau de navigation.

Voir aussi [Import et export de modèles de données](#). [p 89]

Voir aussi [Utilisation des barres d'outils](#) [p 65]

10.4 Services de données

Il est possible de faire référence, dans une opération d'un service de données, à une table en utilisant un nom d'entité unique au lieu de son chemin dans le modèle de données en définissant des suffixes pour les opérations WSDL. Un suffixe WSDL est l'association entre le chemin d'une table et un nom. Pour définir un suffixe WSDL en utilisant l'interface utilisateur, créer un nouvel enregistrement dans la table 'Services de données' située dans la section 'Configuration du modèle de données' dans le panneau de navigation. Un enregistrement de cette table définit les propriétés suivantes :

Chemin de la table	Indique le chemin de la table dans le modèle de données qui doit être associée à ce nom d'entité.
Suffixe d'opération WSDL	Ce nom est utilisé pour suffixer tous les noms des opérations qui concernent la table spécifiée. Si cette propriété n'est pas définie pour une table donnée, alors le dernier élément du chemin de la table est utilisé. Ce nom doit être unique dans le modèle de données.

Voir aussi [Data services](#) [p 605]

10.5 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 du 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 "Répliquations" 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

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.
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	Définit le mode de synchronisation. Les différents modes de synchronisation sont les suivants : • 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	Indique les tables du modèle de données qui doivent être répliquées dans la base de données. Chemin de la table : Indique le chemin de la table dans le modèle de données qui doit être répliquée dans la base de données. 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.
Listes agrégées	Définit les propriétés des listes agrégées contenues dans la table. Chemin de l'élément : Indique le chemin de la liste agrégée dans la table qui doit être répliquée dans la base de données. Nom dans la base de données : Indique le nom de la table dans la base de données qui contiendra les données répliquées de la liste agrégée. Ce nom doit être unique par rapport à toutes les unités de réplication.

Voir aussi [Replication](#) [p 277]

CHAPITRE 11

Actions sur les modèles de données existants

Lorsque le modèle de données est créé, des actions sont disponibles dans le menu **"Actions" du modèle de données** [p 37] dans le panneau de navigation.

Ce chapitre contient les sections suivantes :

1. [Validation du modèle de données](#)
2. [Import et export de modèles de données.](#)
3. [Duplication d'un modèle de données](#)
4. [Suppression d'un modèle de données](#)

11.1 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, cliquez sur le bouton **Revalider** pour mettre à jour ce rapport, ou cliquez 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](#) [p 322] pour obtenir plus d'informations concernant la validation incrémentale de données.

11.2 Import et export de modèles de données.

TIBCO EBX® fournit des services intégrés pour importer et exporter des fichiers XML Schema Document (XSD) ou des archives (zip). Les services d'import et d'export sont accessibles depuis le menu **"Actions" du modèle de données** [p 37] 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 correspondant au modèle à importer. Lors d'un export, le modèle de données complet est exporté dans le document XML Schema ou l'archive cible.

Pour importer un modèle de données, le fichier XSD correspondant doit être valide et conforme aux règles de validation du référentiel EBX®. Si le document déclare des 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, le fichier XSD ne pourra pas être importé. Voir [Propriétés du modèle de données](#) [p 44] pour plus d'informations sur la déclaration des modules.

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

Un document XML Schema (XSD) ou une archive (zip) peut être importé à partir du système de fichier local. Pour cela, sélectionnez *Importer à partir d'un document local* :

- **Modèle de données** : chemin du document XSD ou archive (zip) à importer dans le système de fichiers local.

Note

Lorsqu'une archive est importée, celle-ci doit contenir un seul document XML Schema (XSD) à sa racine. Les documents XML correspondants aux extensions associées au modèle à importer doivent aussi se situer à la racine de l'archive. Ces documents XML sont automatiquement importés si ils suivent la convention de nommage définie par les extensions supportées et enregistrées dans EBX®.

Un modèle de données XSD peut aussi être importé 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	Codes sources utilisés pour configurer les "Règles et objets métier". Si le chemin est relatif, il sera résolu à partir du "Chemin du module". Cette propriété est obligatoire si le modèle définit des éléments programmatiques.
Modèle	Modèle de données à importer.

Note

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

11.3 Duplication d'un modèle de données

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

11.4 Suppression d'un modèle de données

Pour supprimer un modèle de données, sélectionnez "Supprimer" dans le menu [Actions du modèle de données](#) [p 37]. Quand un modèle de données est supprimé, toutes les publications associées au modèle restent disponibles. Si un nouveau modèle de données est recréé avec le même nom que celui qui

a été supprimé, le nouveau modèle de données sera réassocié avec toutes les publications existant dans le référentiel. Au moment de la publication, il sera possible 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](#) [p 93] pour plus d'informations sur le processus de publication.

CHAPITRE 12

Publication du modèle de données

Ce chapitre contient les sections suivantes :

1. [A propos des publications](#)
2. [Modes de publication](#)
3. [Mode de publication "Embarqué"](#)

12.1 A propos des publications

Chaque jeu de données dans le référentiel TIBCO EBX® 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 le **Data Model Assistant**. Une publication est créée la première fois qu'un modèle de données est publié en utilisant le bouton **Publier** du 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](#) [p 43] pour plus d'informations.

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

12.2 Modes de publication

Un modèle de données peut être publié en mode "Embarqué" ou "Dans un module". Le mode de publication "Embarqué" génère une publication gérée et persistée dans le référentiel EBX® 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 EBX®.

Selon la configuration du modèle de données, EBX® détermine automatiquement le processus de publication à utiliser lorsque l'on clique 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.

12.3 Mode de publication "Embarqué"

La première fois qu'un modèle de données est publié 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, sélectionnez celle à mettre à jour.

Voir [Consultation et création des publications](#) [p 94] pour plus d'informations sur l'utilisation de différentes publications.

Pendant le processus de publication, si le modèle de données a déjà été publié, il est possible de visualiser dans une interface de comparaison les différences structurelles introduites par la nouvelle publication.

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 précédent état 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 des versions du modèle de données embarqué](#) [p 95] 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](#) [p 37] dans le panneau de navigation. Vous pourrez consulter les détails des publications et en créer de nouvelles.

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 pour les utilisateurs ayant le rôle "Administrateur".

Pour créer une nouvelle publication, sélectionnez "Gérer les publications" dans le menu ["Actions" du modèle de données](#) [p 37] dans le panneau de navigation, puis cliquez sur le bouton **Créer une publication**. Le nom donné 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 être mise à jour en publiant le modèle de données.

CHAPITRE 13

Gestion des versions de modèles de données

Ce chapitre contient les sections suivantes :

1. [A propos des versions](#)
2. [Accès aux versions](#)
3. [Actions sur les versions](#)
4. [Limitations connues sur la gestion de versions du modèle de données](#)

13.1 A propos des versions

Il est possible de créer des *versions* pour les modèles de données, leur permettant ainsi d'évoluer 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.

13.2 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](#) [p 37].

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

13.3 Actions sur les versions

Dans l'espace de travail, en utilisant le menu avec la flèche vers le bas ▼ à côté de chaque version, accédez aux 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 les contenus évoluent indépendamment.
Définir en tant que version par défaut	Définit la version par défaut sélectionnée lorsqu'un utilisateur accède au modèle de données.
Exporter une archive	Exporte le modèle de données sélectionné dans une archive contenant les données de la version, ses permissions et ses informations. L'archive est exportée vers le répertoire d'archives, accessible par les administrateurs du référentiel. Voir Dossier archives [p 416] pour plus d'informations sur le répertoire d'archives.
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 sa droite. 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 leur case 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 gauche et la plus récente à droite.

13.4 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 jour 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 et utilisées par un modèle de données embarqué ne sont pas versionnées lorsqu'une version est créée. En effet, seules les références des ressources sont sauvegardées, et les développeurs doivent s'assurer que les ressources référencées restent compatibles avec les différentes versions.

Espaces de données

CHAPITRE 14

Introduction aux espaces de données

Ce chapitre contient les sections suivantes :

1. [Présentation](#)
2. [Utilisation de l'interface utilisateur de la section Espace de données](#)

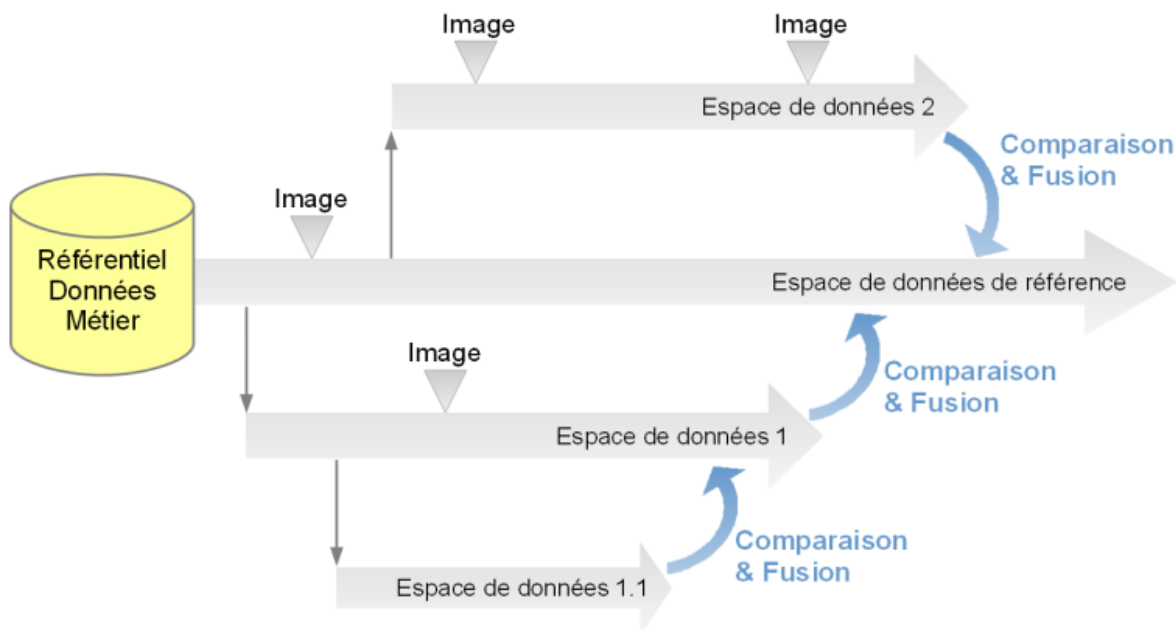
14.1 Présentation

Fonction d'un espace de données

Le cycle de vie des données est souvent complexe. Il est parfois nécessaire d'entretenir une version courante des données tout en travaillant sur des évolutions futures. De plus, il faut conserver une trace des états intermédiaires. TIBCO EBX® rend ces démarches possibles grâce aux espaces de données et aux images.

Un espace de données est un conteneur qui isole différentes versions de jeux de données et les organise. Des espaces de données enfants peuvent être créés à partir d'un espace de données. Un espace de données enfant est initialisé avec le même état que son parent au moment de sa création. Ainsi, des modifications peuvent être effectuées isolément dans l'espace de données enfant, sans impacter l'espace de données parent ni d'autres espaces de données. Lorsque les modifications dans l'espace de données enfant sont terminées, cet espace de données peut être fusionné avec son parent.

Une image est une copie statique d'un espace de données qui capture son état et tout son contenu à un moment donné. Les images peuvent être consultées, exportées, et comparées à d'autres espaces de données.



Concepts de base liés aux espaces de données

La compréhension des termes suivants est recommandée pour l'utilisation des espaces de données :

- [espace de données](#) [p 30]
- [image](#) [p 30]
- [jeu de données](#) [p 28]
- [fusion](#) [p 30]
- [espace de données de référence](#) [p 30]

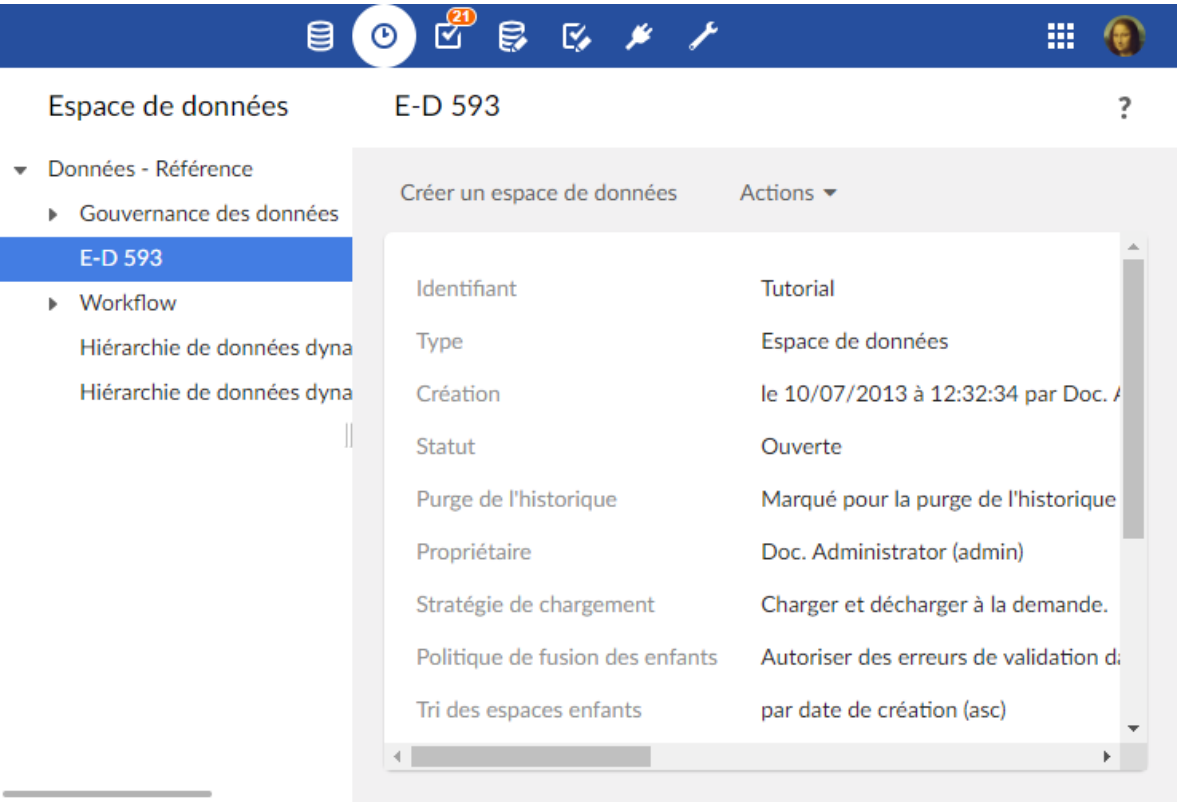
14.2 Utilisation de l'interface utilisateur de la section Espace de données

Les espaces de données sont créés, consultés et modifiés dans la section **Espaces de données**.

Note

Seuls les utilisateurs autorisés peuvent accéder à cet écran via la 'Perspective avancée'.

Le panneau de navigation affiche l'organisation hiérarchique des espaces de données existants, tandis que l'espace de travail affiche les informations concernant l'espace de données sélectionné et liste ses images.



Voir aussi

[Création d'un espace de données](#) [p 103]

[Images](#) [p 115]

Concepts apparentés [Jeux de données](#) [p 122]

CHAPITRE 15

Création d'un espace de données

Ce chapitre contient les sections suivantes :

1. [Création d'un espace de données](#)
2. [Propriétés](#)

15.1 Création d'un espace de données

Pour créer un nouvel espace de données, sélectionnez un espace de données existant sur lequel il se basera, puis cliquez sur le bouton **Créer un espace de données** situé dans l'espace de travail.

Note

Seuls les utilisateurs autorisés peuvent accéder à cet écran via la 'Perspective avancée'.

L'espace de données nouvellement créé devient un espace de données enfant de celui qui était sélectionné. Son contenu est automatiquement initialisé avec le contenu de son parent au moment de la création. En outre, une image initiale représentant cet état est aussi créée.

Excepté l'espace de données de référence, qui est la racine de tous les espaces de données du référentiel, les espaces de données sont toujours l'enfant d'un autre espace de données.

15.2 Propriétés

Les informations suivantes sont requises lors de la création d'un nouvel espace de données :

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

CHAPITRE 16

Actions sur les espaces de données existants

Ce chapitre contient les sections suivantes :

1. [Informations de l'espace de données](#)
2. [Permissions sur un espace de données](#)
3. [Fusion d'un espace de données](#)
4. [Comparaison d'un espace de données](#)
5. [Validation d'un espace de données](#)
6. [Archives d'espaces de données](#)
7. [Fermeture d'un espace de données](#)

16.1 Informations de l'espace de données

Certaines propriétés associées à un espace de données peuvent être modifiées en sélectionnant **Actions** > **Information** dans l'espace de travail de la section Espaces de Données.

Documentation	Libellé et description localisés associés à l'espace de données.
Stratégie de chargement	<i>Seul l'administrateur a le droit de modifier cette option.</i> • Chargement et déchargement sur demande : le principal avantage de cette stratégie est de permettre de libérer de la mémoire si nécessaire. L'inconvénient est que cela implique un coût de chargement quand on accède à une ressource pour la première fois depuis le démarrage du serveur, ou si elle a été déchargée entre-temps. Ce mode est utilisé par défaut. • Chargement forcé : ce mode est particulièrement recommandé pour les espaces de données et images utilisés de façon intensive, ou exigeants en termes de temps de réponse. • Chargement forcé et pré-validation : ce mode est particulièrement recommandé pour les espaces de données et images utilisés de façon intensive ou exigeants en termes de temps de réponse, et pour lesquels le processus de validation peut être long. Note : Toute modification de la stratégie de chargement requiert un redémarrage du serveur.
Politique de fusion	La politique de fusion des espaces de données enfants ne s'applique qu'aux fusions initiées par un utilisateur. Elle ne s'applique pas aux fusions programmatiques, telles que celles initiées par les scripts 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 quelle que soit la validité du résultat de cette fusion. • Fusion pré-validante : un espace de données enfant ne peut être fusionné que si le résultat de la fusion est valide.
Propriétaire	L'utilisateur possédant l'espace de données et qui est autorisé à en modifier les informations et les permissions.

Le propriétaire d'un espace de données n'est pas obligatoirement son créateur.

Tri des espaces enfants	Définit l'ordre d'affichage des espaces de données enfants dans l'arbre des espaces de données. Si la valeur n'est pas définie, l'ordre défini par le parent est pris en compte. La valeur par défaut est "par libellé".
Changement de propriétaire	Définit si le propriétaire de l'espace de données a le droit de modifier l'attribut "Propriétaire". Si la valeur est "Non habilité", seul l'administrateur a le droit d'effectuer cette modification.
Changement des permissions	Définit si le propriétaire de l'espace de données a le droit de modifier les permissions de l'espace de données. Si la valeur est "Non habilité", seul l'administrateur a le droit d'effectuer cette modification.

16.2 Permissions sur un espace de données

Permissions générales

Identifiant de l'espace de données	Indique l'espace de données auquel les permissions vont être appliquées.
Sélection du profil	Indique le profil concerné par la permission définie.
Restriction d'accès	Indique si la permission définie restreint les permissions affectées à un utilisateur donné par des politiques définies pour d'autres profils. Voir Restriction d'accès [p 304].
Permissions d'accès à l'espace de données	Indique la permission globale d'accès à l'espace de données. Lecture seule • Permet de visualiser l'espace de données et son image, et de voir les espaces de données enfants 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 en autorisent l'accès. Ecriture • Permet de visualiser l'espace de données et son image, et de voir les espaces de données enfants selon les droits s'appliquant à chacun d'eux. • Permet de modifier le contenu de l'espace de données, à condition que les droits s'appliquant au contenu en autorisent l'accès. Non visible • Ni l'espace de données, ni ses images ne peuvent être vus. • A partir d'un espace de données enfant, l'espace de données courant est visible sans toutefois pouvoir être sélectionné. • Le contenu de l'espace de données n'est pas accessible. • Aucune action ne peut être effectuée sur l'espace de données.

Actions possibles

Les utilisateurs peuvent être autorisés à effectuer les actions suivantes :

Créer un espace de données enfant	Indique si le profil peut créer un espace de données enfant.
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 exporter l'espace de données.
Importer une archive	Indique si le profil peut importer dans l'espace de données.
Fermer un espace de données	Indique si le profil peut fermer l'espace de données.
Fermer une image	Indique si le profil peut fermer les images de l'espace de données.
Droits sur les services	Spécifie les permissions d'accès aux services.
Permissions des espaces de données enfants à la création	Spécifie les permissions d'accès aux espaces de données enfants créés à partir de l'espace de données courant.

16.3 Fusion d'un espace de données

Quand le travail dans un espace de données est terminé, il est possible d'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, excepté l'utilisateur qui a initié la fusion ainsi que les utilisateurs administrateurs. Les verrous sont maintenus pendant la durée de la fusion. Ainsi, les contenus des deux espaces de données peuvent être lus, mais ne peuvent pas être modifiés.

Note : en plus de s'appliquer aux modifications directes sur l'espace de données, cette restriction s'applique également aux autres fusions des autres espaces de données enfants. Ainsi, il est impossible de fusionner d'autres espaces de données enfants tant que la fusion en cours n'est pas terminée.

2. les changements qui ont été effectués dans l'espace de données 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 la fusion d'un espace de données avec 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, sélectionnez les changements survenus dans l'espace de données enfant (source) qui doivent être fusionnés dans l'espace de données parent (cible).

Note

Cette étape de revue n'existe pas pour une fusion faite à travers un service de données ou 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 devant 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 la fusion. Pour exclure un changement de la fusion, désélectionnez-le. Les changements relatifs aux différents éléments sont visibles en les sélectionnant dans le panneau de navigation.

Afin de détecter les conflits, la fusion implique l'espace de données courant, l'image initiale et l'espace de données parent. En effet, les données peuvent avoir 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 aux tables. 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 **Marquer les différences comme revues** pour indiquer que vous avez vérifié les changements dans le périmètre actuel. Tous les changements doivent être revus pour effectuer la fusion.

Types de modifications

Le processus de fusion considère les opérations suivantes comme des 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 détectés. Des conflits peuvent survenir quand une entité contient des modifications dans l'espace de données source et l'espace de données cible.

Les conflits sont catégorisés comme suit :

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

Finalisation d'une fusion

Lorsque tous les changements ont été vérifiés et que ceux à inclure dans le résultat de la fusion ont été choisis, 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. En revanche, l'administrateur de l'espace de données parent peut configurer la politique de fusion pour que les fusions de ses espaces de données enfants soient finalisées uniquement si le résultat ne contient pas d'erreur de validation.

Si la politique de fusion pré-validante est utilisée, un espace de données dédié est d'abord créé pour accueillir 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 seront accessibles 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>". 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 restitue l'espace de données enfant dans son état précédant la fusion.
- **Poursuivre la fusion** : cette action permet de tenter de nouveau la 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, il est possible, à travers l'interface utilisateur, d'interrompre la finalisation des fusions de ses espaces de données enfants si le résultat contient des erreurs de validation. Pour ce faire, 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 enfants avec l'espace de données parent ainsi paramétré.

Note

Si la fusion est effectuée à travers un 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. Elle est donc ignorée pour les tâches automatiques des workflows de données.

Voir aussi [Politique de fusion](#) [p 111]

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, cliquez sur le bouton **Annuler** afin d'abandonner l'opération.

Si vous naviguez vers une autre page pendant une fusion, celle-ci sera abandonnée mais les verrous sur l'espace de données parent et l'espace de données enfant seront maintenus. 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 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, lui seul sera déverrouillé, vous devrez donc aussi effectuer un déverrouillage de l'espace de données enfant.

16.4 Comparaison d'un espace de données

Il est possible de comparer le contenu d'un espace de données à celui d'un autre espace de données ou à une image dans le référentiel. Pour effectuer une comparaison, ouvrez l'espace de données dans le panneau de navigation, puis sélectionnez **Actions > Comparer** dans l'espace de travail. L'assistant permet de choisir un autre espace de données ou une image à comparer avec l'espace de données courant.

Pour une comparaison plus rapide, qui ignore les champs avec une valeur héritée ou calculée, sélectionnez le filtre "Valeurs persistantes seulement".

Voir aussi [Compare contents](#) [p 247]

16.5 Validation d'un espace de données

Le contenu d'un espace de données peut être validé globalement en utilisant le service de validation au niveau de l'espace de données. Ce service est accessible en sélectionnant **Actions > Validation** dans l'espace de travail.

Note

Ce service est proposé uniquement si l'utilisateur a la permission de valider tous les jeux de données contenus dans l'espace de données.

16.6 Archives d'espaces de données

Export d'une archive

Le contenu d'un espace de données peut être exporté dans une archive en sélectionnant **Actions > Export** dans le panneau de navigation. L'archive est sauvegardée sur le système de fichiers du serveur, où seul un administrateur peut la récupérer.

Note

Voir [Archives directory](#) [p 416] dans le Guide d'administration.

Pour réaliser un export, les informations suivantes sont requises:

Nom du fichier archive	Nom de l'archive exportée.
Type d'export	Obligatoire. Le type d'export par défaut est "Contenu complet de l'espace de données". Il permet d'exporter l'ensemble des données sélectionnées dans l'archive. Il peut être utile d'inclure uniquement les différences entre l'espace de données et son image initiale dans l'archive. Il existe deux types d'export qui incluent un delta : "Mises à jour avec leur contenu complet" et "Mises à jour seules". Le premier exporte l'état actuel de l'espace de données ainsi que le delta qui contient les différences entre l'état actuel de l'espace de données et l'image initiale. Le second exporte uniquement le delta qui contient les différences entre l'état actuel de l'espace de données et l'image initiale. Les deux options affichent une page de comparaison, sur laquelle il est possible de sélectionner les différences à inclure dans le delta. Les différences sont détectées au niveau table.
Jeu de données à exporter	Jeux de données de cet espace de données à exporter. Pour chaque jeu de données, il est possible de spécifier si les données, les permissions et les informations doivent être exportées.

Importer une archive

Le contenu d'une archive peut être importé dans un espace de données en sélectionnant **Actions > Import**.

Si l'archive sélectionnée ne contient pas de delta, l'état actuel de l'espace de données sera remplacé par le contenu de l'archive.

Si l'archive sélectionnée se compose d'un contenu complet et d'un delta, vous pouvez choisir d'appliquer le delta afin de fusionner les différences incluses. Un écran de comparaison sera affiché, depuis lequel les différences à fusionner peuvent être sélectionnées.

Si l'archive sélectionnée contient uniquement un delta, il est possible de sélectionner les différences à fusionner dans un écran de comparaison.

16.7 Fermeture d'un espace de données

Si un espace de données n'est plus utilisé, il peut être fermé. Dès qu'il est fermé, l'espace de données n'apparaîtra plus dans la section **Espaces de Données** de l'interface utilisateur, et ne sera plus accessible.

Un administrateur peut rouvrir un espace de données fermé, tant que celui-ci n'a pas encore été purgé du référentiel.

Pour fermer un espace de données, sélectionnez **Actions > Fermer l'espace de données**.

Voir aussi [*Closing unused dataspace and snapshots*](#) [p 417]

CHAPITRE 17

Images

Ce chapitre contient les sections suivantes :

1. [Présentation des images](#)
2. [Création d'une image](#)
3. [Visualisation de contenu d'une image](#)
4. [Informations de l'image](#)
5. [Comparaison d'une image](#)
6. [Validation d'une image](#)
7. [Export d'une image](#)
8. [Fermeture d'une image](#)

17.1 Présentation des images

Une image est une copie en lecture seule d'un espace de données. Une image sert de référence de l'état et du contenu d'un espace de données à un instant donné.

Voir aussi [image](#) [p 30]

17.2 Création d'une image

Une image est créée en sélectionnant un espace de données dans le panneau de navigation, puis en cliquant sur le bouton 'Créer une image' sous le menu 'Actions'.

Les informations suivantes sont requises :

Identifiant	Identifiant unique de l'image. Le modèle suivant doit être respecté: [a-zA-Z0-9_:\-\\]*.
Libellé	Libellé et description localisés associés à l'image.

17.3 Visualisation de contenu d'une image

Pour visualiser le contenu d'une image, ouvrir l'image, puis sélectionner *Actions > Voir ou éditer les jeux de données* dans l'espace de travail.

17.4 Informations de l'image

Certaines propriétés associées à une image peuvent être modifiées en sélectionnant l'image, puis *Actions > Informations* dans l'espace de travail de la section *Espaces de données*.

Documentation	Libellé et description localisés associés à l'image.
Mode relationnel	Indique que les tables des jeux de données présents dans cet espace de données sont contenues dans une base de données relationnelle. Il ne sera pas possible de créer des images et des espaces de données enfants.
Stratégie de chargement	Toute modification de ce champ requiert de redémarrer le serveur. Pour un espace de données en mode sémantique, la stratégie par défaut, 'Charger et décharger à la demande', permet de libérer la mémoire si nécessaire. En contrepartie, cela implique un coût de chargement quand la ressource accédée l'est pour la première fois depuis le redémarrage du serveur ou quand elle a été déchargée depuis. La stratégie 'Chargement forcé' est recommandée pour les espaces de données et images à vie longue et qui sont intensivement utilisées. Pour un espace de données en mode relationnel, les stratégies sont 'Aucune' (défaut) ou 'Prévalidation'. Seul l'administrateur a le droit de modifier cette option. Voir Stratégie de chargement [p 106].
Politique de fusion des enfants	Un espace de données peut définir une politique qui définit la manière de fusionner un espace de données enfant. La politique par défaut est 'Autoriser des erreurs de validation dans le résultat'. Une autre politique, la 'Fusion pré-validante', permet de s'assurer que le résultat de la fusion est valide avant d'effectuer définitivement la fusion. La politique de fusion des enfants est uniquement appliquée en utilisant les interfaces utilisateur. Ce paramétrage est ignoré pour une fusion programmatique (incluant les tâches automatiques des workflows de données).
Propriétaire	L'utilisateur possédant l'image, et qui est autorisé à en modifier les informations et les permissions. Le propriétaire d'une image n'est pas obligatoirement son créateur.
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'.

Changer le propriétaire	Définit si le propriétaire de l'image a le droit de modifier l'attribut "Propriétaire". Si la valeur est "Non habilité", seul l'administrateur a le droit d'effectuer cette modification.
Changer les permissions	Spécifie si un utilisateur qui est propriétaire de l'espace de données a le droit de modifier les permissions de cet espace de données. Si ce n'est pas le cas, seul un administrateur ou un 'super propriétaire' de l'espace de données a le droit de modifier les permissions.

17.5 Comparaison d'une image

Il est possible de comparer le contenu d'une image à celui d'une autre image ou d'un espace de données dans le référentiel. Pour effectuer une comparaison, ouvrir l'image, puis sélectionner *Actions* > *Comparer* dans l'espace de travail. L'assistant permet de choisir une autre image ou un espace de données à comparer avec l'image courante.

Pour une comparaison plus rapide, qui ignore les champs avec une valeur héritée ou calculée, sélectionner le filtre 'Valeurs persistantes seulement'.

Voir aussi [Compare contents](#) [p 247]

17.6 Validation d'une image

Le contenu d'une image peut être validé globalement en utilisant le service de validation au niveau de l'image. Ce service est accessible en sélectionnant *Actions* > *Valider* dans l'espace de travail.

Note

Pour utiliser ce service, l'utilisateur doit avoir la permission de valider tous les jeux de données contenus dans l'image.

17.7 Export d'une image

Le contenu d'une image peut être exporté dans une archive en sélectionnant *Actions* > *Exporter* dans l'espace de travail. L'archive est sauvegardée sur le système de fichiers du serveur, où seul un administrateur peut la récupérer.

Note

Voir [Archives directory](#) [p 416] dans le Guide d'administration.

Pour réaliser un export, les informations suivantes sont requises :

Nom du fichier archive à créer	Le nom de l'archive exportée.
Type d'export	Le contenu complet de l'espace de données, Mises à jour avec leur contenu complet (*), Mises à jour seules (*). (*) : les mises à jour à exporter sont sélectionnées dans les pages suivantes.
Jeu de données (ou arbre de jeux de données) à exporter	Les jeux de données de cette image à exporter. Pour chaque jeu de données, il est possible de choisir si les données, les permissions et les informations doivent être exportées.

17.8 Fermeture d'une image

Si une image n'est plus utilisée, elle peut être fermée. Dès qu'elle est fermée, l'image n'apparaît plus dans la section 'Espaces de données' de l'interface utilisateur et n'est plus accessible.

Un administrateur peut rouvrir une image fermée, tant qu'elle n'a pas été purgée du référentiel.

Pour fermer une image, sélectionner *Actions > Fermer l'image*.

Voir aussi [Closing unused dataspace and snapshots](#) [p 417]

Jeux de données

CHAPITRE 18

Introduction aux jeux de données

Ce chapitre contient les sections suivantes :

1. [Présentation](#)
2. [Utilisation de l'interface utilisateur de la section Données](#)

18.1 Présentation

Fonction d'un jeu de données

Un jeu de données est un conteneur de données qui se base sur les définitions de structure fournies par le modèle de données qu'il implémente. Lors de la publication d'un modèle de données, il est possible de créer des jeux de données basés sur sa définition. Par la suite, si ce modèle est modifié et republié, tous ses jeux de données associés sont mis à jour automatiquement.

Dans un jeu de données, les valeurs de données sont consultables et modifiables. A l'aide des vues, il est possible d'afficher les tables d'une manière adaptée à la nature des données et du mode d'accès. Les recherches et les filtres peuvent aussi être utilisés pour restreindre l'affichage ou rechercher des données.

Des permissions peuvent aussi être affectées à différents rôles pour contrôler l'accès au niveau du jeu de données. Ainsi, en appliquant des permissions spécifiques, on peut permettre à certains utilisateurs d'afficher ou de modifier des données tout en les cachant à d'autres.

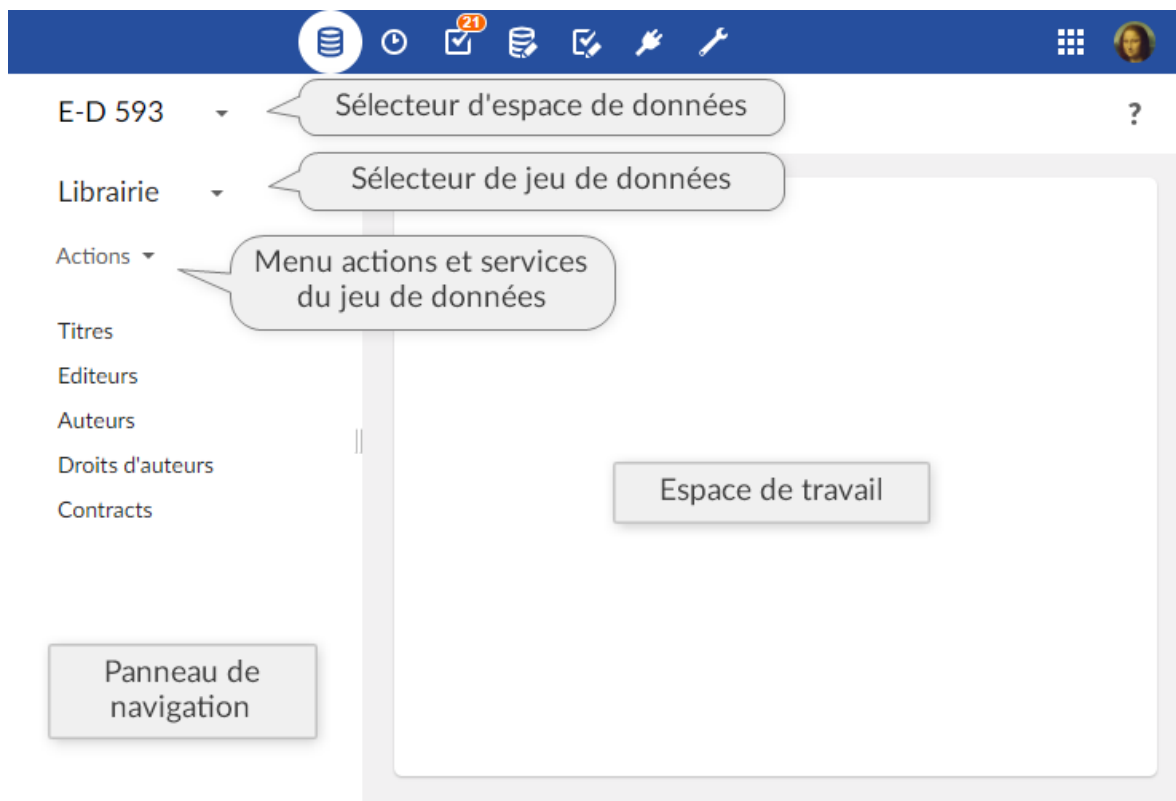
Concepts de base liés aux jeux de données

La compréhension des termes suivants est recommandée pour utiliser les jeux de données :

- [espace de données](#) [p 30]
- [jeu de données](#) [p 28]
- [enregistrement](#) [p 28]
- [champ](#) [p 27]
- [clé primaire](#) [p 27]
- [clé étrangère](#) [p 27]
- [table \(jeu de données\)](#) [p 28]
- [groupe](#) [p 27]

18.2 Utilisation de l'interface utilisateur de la section Données

Dans le cadre de l'utilisation de la [Perspective avancée](#) [p 19], ou d'une perspective spécifique, les jeux de données sont créés, consultés et modifiés dans la section 'Données'. Seuls les utilisateurs autorisés peuvent accéder à ces interfaces spécifiques.



Pour sélectionner ou créer un jeu de données, cliquer sur 'Sélectionner le jeu de données' dans le panneau de navigation. La structure de données du jeu de données s'affichera dans le panneau de navigation et les formulaires d'enregistrement ainsi que les vues de table s'afficheront dans l'espace de travail.

Lors de la visualisation d'une table du jeu de données dans l'espace de travail, le bouton  permet d'afficher les recherches et filtres disponibles pour restreindre l'affichage des enregistrements.

Les actions applicables au jeu de données sont disponibles dans le menu **Actions** du panneau de navigation (les services sont accessibles en bas de la liste).

Voir aussi

[Création du jeu de données](#) [p 125]

[Recherche rapide](#) [p 128]

[Actions sur les enregistrements dans l'interface utilisateur](#) [p 139]

[Héritage](#) [p 29]

Concepts apparentés

[Modèle de données](#) [p 36]

[Espace de données](#) [p 100]

CHAPITRE 19

Création du jeu de données

Ce chapitre contient les sections suivantes :

1. [Création d'un jeu de données racine](#)
2. [Création d'un jeu de données enfant utilisant l'héritage](#)

19.1 Création d'un jeu de données racine

Afin de créer un jeu de données racine, qui n'hérite pas d'un jeu de données parent, il faut sélectionner le menu '[Sélectionner le jeu de données](#) [p 123]' ▼ dans le panneau de navigation, cliquer sur le bouton 'Créer un jeu de données' dans la fenêtre contextuelle, et suivre l'assistant.

Note

Seuls les utilisateurs autorisés peuvent accéder à cet écran via la 'Perspective avancée' ou via une perspective spécifique.

L'assistant vous permet de sélectionner un des trois types de modèles de données, sur lequel le nouveau jeu de données sera basé : packagé, embarqué ou externe.

- Un *modèle de données packagé* est un modèle de données défini à l'intérieur d'un module (application web).
- Un *modèle de données embarqué* est un modèle de données créé et publié en utilisant l'assistant de modèles de données. Il est entièrement géré à l'intérieur du référentiel TIBCO EBX®. Un modèle de données doit avoir été préalablement publié pour que cette fonctionnalité soit disponible.
- Un *modèle de données externe* est un modèle de données référencé au moyen d'une URI.

Après avoir choisi le modèle de données sur lequel le nouveau jeu de données sera basé, vous devez fournir un nom unique, sans espaces ni caractères spéciaux. Facultativement, vous pouvez donner des libellés localisés pour le jeu de données, qui seront affichés aux utilisateurs dans l'interface en fonction de leurs préférences de langue.

Attention

Le contenu des tables n'est pas copié lors de la duplication d'un jeu de données.

19.2 Création d'un jeu de données enfant utilisant l'héritage

Le mécanisme d'héritage permet d'utiliser les relations parent-enfant, grâce auxquelles les jeux de données enfants héritent par défaut des valeurs de leurs jeux de données ancêtres. Pour pouvoir créer des jeux de données enfants depuis un jeu de données parent, il faut activer l'héritage entre jeux de données dans le modèle de données associé.

Pour créer un jeu de données enfant, cliquer sur [Créer ou sélectionner un jeu de données](#) [p 123] ▼ dans le panneau de navigation, puis sur le bouton + correspondant au jeu de données parent.

Le jeu de données enfant sera basé sur le même modèle de données que celui de son parent, et donc les seules informations à spécifier sont son nom unique, et éventuellement les libellés localisés.

Voir aussi [Héritage entre jeux de données](#) [p 161]

CHAPITRE 20

Visualisation des données

TIBCO EBX® propose différentes manières de lister les enregistrements. Ce chapitre présente comment trier, rechercher, et afficher les enregistrements de différentes manières, et selon différents profils d'utilisateurs, grâce au concept de 'Vues'.

Ce chapitre contient les sections suivantes :

1. [Menu 'Vue'](#)
2. [Tri des données](#)
3. [Recherche rapide](#)
4. [Recherche et filtrage des données](#)
5. [Vues](#)
6. [Gestion des vues](#)
7. [Édition tabulaire](#)
8. [Historique](#)

20.1 Menu 'Vue'

Le menu déroulant 'Vue' permet d'accéder facilement aux différentes vues disponibles et aux fonctionnalités de gestion des vues.

Les vues sont gérées directement via une barre d'outils, disponible au survol des vues affichées dans le menu : [Barre d'outils des vues](#) [p 137].

Les vues peuvent également être regroupées. L'administrateur doit avoir, au préalable, défini des groupes dans la table 'Groupes de vues' de la section 'Configuration des vues'. L'utilisateur peut ainsi définir une vue comme appartenant à un groupe dans le champ 'Groupe de la vue' lors de la création ou de la modification d'une vue. Voir [Description d'une vue](#) [p 132] pour plus d'informations.

20.2 Tri des données

Les critères de tri contrôlent l'ordre dans lequel les enregistrements sont présentés.

Utilisez le bouton 'Sélection et tri' en haut à gauche de la table pour définir des critères de tri spécifiques.

Il y a deux types de tri :

- le tri par pertinence, utilisé lors de la [recherche rapide](#) [p 128],

- le tri par colonne.

Tri par colonne

La boîte de dialogue 'Critères de tri', propose :

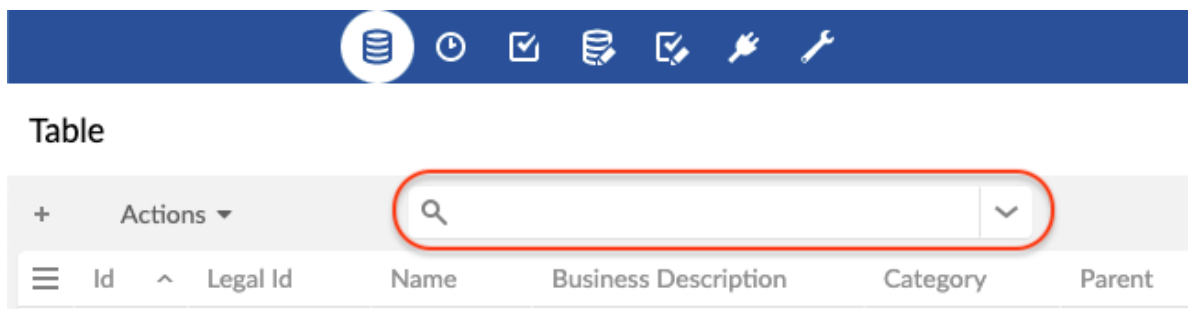
- sur la gauche, la liste des colonnes triées,
- sur la droite, la liste des colonnes non triées.

Utilisez les boutons flèches ← → pour faire basculer les colonnes d'une liste à une autre.

Utilisez les boutons flèches ↑ ↓ pour changer l'ordre de priorité de la colonne.

Pour changer l'ordre de tri d'une colonne, utilisez le bouton 'ASC' (croissant) ou 'DESC' (décroissant) qui apparaît au survol.

20.3 Recherche rapide



La recherche rapide est utilisée pour trouver facilement un résultat dans une vue tabulaire ou hiérarchique.

Elle ne fait pas de différence entre les majuscules et les minuscules. Elle permet de rechercher en une seule fois plusieurs termes (séparés par des espaces). Par défaut, les enregistrements trouvés sont triés par pertinence.

Voir aussi [Search](#) [p 315]

Caractères spéciaux

La recherche rapide propose des caractères spéciaux pour préciser votre recherche.

+... *obligatoire*

Rend obligatoire la présence du mot dans le résultat. Interdit au moteur de recherche d'écarter ce mot de la recherche.

Note

Fonctionne avec +"phrase" ou +(groupe).

Exemple :

```
+flute +bach
```

↳ Trouve des résultats avec obligatoirement *flute* et obligatoirement *bach*. Les résultats avec seulement *flute* ou seulement *bach* seront ignorés.

-... *sauf*

Exclut le mot du résultat.

Note

Fonctionne avec -"phrase" ou -(groupe).

Exemple :

```
bach -flute
```

↳ Trouve des résultats avec *bach*, mais sans *flute*.

...~ *approx*

Précise que la recherche peut changer 2 caractères du mot pour le trouver.

Pour permettre le changement d'un seul caractère, utiliser ~1.

Note

Fonctionne avec "phrase"~, pour changer 2 mots de la phrase.

Exemple :

```
handel~
```

↳ Trouve des résultats avec un mot qui est *handel*, à 2 lettres près.

? *joker simple*

Remplace un caractère inconnu.

Exemple :

```
?uttle
```

↳ Trouve des résultats avec un mot commençant par n'importe quel caractère, et finissant par *utlle*.

*** *joker étendu***

Remplace plusieurs caractères inconnus.

Exemple :

```
rachmanino*
```

↳ Trouve des résultats avec un mot commençant par *rachmanino*.

"..." *phrase*

Recherche la correspondance exacte de la phrase.

Note

Peut être entourée de + - ~ .

Exemple :

```
"Johann Sebastian Bach"
```

↳ Trouve des résultats contenant exactement *Johann Sebastian Bach*.

(...) *groupe*

Permet de grouper des mots pour leur appliquer un caractère spécial + ou -.

Note

Les groupes de groupes sont autorisés.

Exemple :

```
bach +(flute piano)
```

↳ Trouve des résultats avec éventuellement *bach*, et obligatoirement *flute* ou *piano*.

Note

Ces caractères spéciaux peuvent également être utilisés dans le moteur de recherche de la documentation.

20.4 Recherche et filtrage des données

Le panneau de recherche est caché par défaut, et accessible par l'icône  située à droite de la recherche rapide dans la barre d'outils de la table ou de la vue hiérarchique.

La recherche rapide et les lignes des critères se cumulent pour préciser la recherche. Cela donne des résultats de plus en plus restreints.

Il est possible de désactiver une ligne de critère en la décochant. Les critères désactivés ne sont pas conservés lors d'une sauvegarde.

Le bouton 'poubelle'  à la fin de la ligne de chaque critère supprime définitivement le critère.

Pour sauvegarder le filtre appliqué à une recherche, utiliser le bouton 'Sauvegarder'. La sauvegarde prend en compte la recherche rapide et tous les critères actifs.

Pour rappeler un filtre sauvegardé, utiliser le bouton 'Charger'. Le chargement remplace la recherche rapide et tout le panneau de critères. Cliquez sur le bouton 'Appliquer' pour lancer la nouvelle recherche.

Lorsqu'une vue est appliquée, elle garantit que son affichage soit conforme à sa configuration. Tous les critères existants du panneau de recherche sont donc retirés. La vue peut contenir un ensemble de critères de recherche, qui sont appliqués en même temps que la vue.

Certains opérateurs (tel que 'correspond à') permettent d'utiliser les expressions régulières Lucene. Voir [spécifications techniques des expressions régulières de Lucene](#) pour plus d'informations.

Recherche sur un champ

Tous les champs cherchables sont disponibles.

Recherche sur validation

Dans la sélection du critère, les critères de validation affichent les enregistrements dans la dernière validation effectuée.

Note

Cette recherche s'applique seulement aux enregistrements de tables qui ont déjà été validés en sélectionnant *Actions > Valider* au niveau de la table dans l'espace de travail, ou au niveau du jeu de données dans le panneau de navigation.

Pour filtrer sur le niveau de sévérité (indépendant du message), utiliser le critère de validation Sévérité. Les niveaux de sévérité disponibles sont 'Erreurs', 'Avertissements' et 'Informations'.

Pour filtrer sur le contenu du message (indépendant du niveau de sévérité), utiliser le critère de validation Message.

Recherches spécifiques sur tables

Afin d'assurer la rétro-compatibilité pour les filtres programmatiques, la fonctionnalité de recherche et filtrage des enregistrements reste opérationnelle et accessible par l'icône  de l'espace de travail. L'icône et la fonctionnalité sont disponibles uniquement s'il existe au moins un filtre programmatique.

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

Voir aussi [Customizing table filter](#) [p 660]

20.5 Vues

Il est possible de personnaliser l'affichage des tables dans EBX® en fonction de l'utilisateur cible. Il existe deux types de vues : [tabulaire](#) [p 133] et [hiérarchique](#) [p 133].

Une vue est créée en sélectionnant *Vue > Créer une nouvelle vue* dans l'espace de travail. Pour appliquer une vue, la sélectionner dans *Vue > nom de la vue*.

Deux modes avancés de visualisation sont disponibles à la création d'une nouvelle vue :

- 'Vue tabulaire simple' : une vue tabulaire qui permet de trier et filtrer les enregistrements affichés ;

- 'Vue hiérarchique' : une arborescence qui lie les données de différentes tables en utilisant leurs relations.

Description d'une vue

Lors de la création ou de la modification d'une vue, la première page permet de définir les informations générales concernant la vue.

Documentation	Libellé et description localisés associés à la vue.
Propriétaire	Nom du propriétaire de la vue, pouvant à ce titre la gérer et la modifier. (Seulement disponible pour les administrateurs et le propriétaire du jeu de données)
Partager avec	Autres profils pouvant utiliser cette vue depuis le menu 'Vue'. Note Requiert une permission, voir Permissions des vues [p 438].
Mode de vue	Vue tabulaire simple ou vue hiérarchique.
Groupe de la vue	Groupe d'appartenance de cette vue (le cas échéant).

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.

Colonnes affichées	Spécifie, à l'aide de flèches, les colonnes de la table à afficher dans la vue.
Colonnes triées	Spécifie l'ordre d'affichage des colonnes et indique si les enregistrements de chaque colonne sont triés par ordre croissant ou décroissant. Voir Tri des données [p 127].
Filtre	Définit les critères utilisés pour filtrer les enregistrements. Voir Editeur de critères [p 313].
Limite de pagination	Force une limite au nombre d'enregistrements visibles.
Édition tabulaire	Si activée, les utilisateurs de cette vue pourront basculer en édition tabulaire afin d'éditer des enregistrements directement depuis la vue tabulaire.
Désactiver la création et la duplication	Si 'Oui', les utilisateurs de cette vue ne pourront ni créer ni dupliquer d'enregistrement depuis l'édition tabulaire.

Vues hiérarchiques

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

Dimension d'une hiérarchie

Une dimension définit une dépendance dans la hiérarchie. Par exemple, une dimension pourrait être précisée pour afficher des produits par catégorie. Plusieurs dimensions peuvent être définies pour une vue hiérarchique.

Options de configuration d'une vue hiérarchique

Ce formulaire permet de configurer les options d'une vue hiérarchique.

Afficher les enregistrements dans une nouvelle fenêtre	Si 'Oui', une nouvelle fenêtre sera ouverte pour afficher l'enregistrement. Sinon, il sera affiché dans une nouvelle page de la fenêtre courante.
Hiérarchie élaguée	Si 'Oui', les noeuds de la hiérarchie qui n'ont pas d'enfants et qui n'appartiennent pas à la table cible ne seront pas affichés.
Afficher les orphelins	Si 'Oui', les noeuds de la hiérarchie qui n'ont pas de parent seront affichés.
Afficher le noeud racine	Si 'Non', le noeud racine de la hiérarchie sera exclu de la vue.
Libellé du noeud racine	Libellé localisé du noeud racine de la hiérarchie.
Barre d'outils au dessus de la hiérarchie	Permet de positionner la barre d'outils en haut de la vue hiérarchique.
Afficher les enfants des noeuds recherchés	Dans le cas récursif, quand un filtre de recherche est appliqué, définit si doivent être affichés les noeuds enfants des noeuds vérifiant le filtre.
Retirer les noeuds racines récursifs sans enfants	Dans le cas récursif, quand un filtre de recherche est appliqué ou lorsque le mode est 'élagué', permet de ne pas afficher les noeuds racines sans enfants.
Détecter les cycles	Dans le cas récursif, autorise la détection et l'affichage des noeuds appartenant à un cycle, en choisissant comme racine le plus ancien noeud du cycle. Limite : ne fonctionne pas en mode recherche ou élagué.
Détecter les feuilles	Détecter si un membre est une feuille ou non. La détection des feuilles est très coûteuse pour les gros volumes de données. Il est donc recommandé de désactiver cette option quand la requête prend beaucoup de temps pour afficher la vue hiérarchique. Cette propriété est toujours désactivée pour les membres parents d'orphelins.

Libellés

Pour chaque niveau de dimension faisant référence à une autre table, il est possible de définir les libellés localisés pour les noeuds correspondants dans la hiérarchie. Utiliser l'assistant pour sélectionner les champs utilisés dans les définitions des libellés.

Filtre

L'éditeur de critères permet de définir un filtre d'enregistrement pour la vue.

Voir aussi [Editeur de critères](#) [p 313]

Stratégie de tri

Pour chaque niveau de dimension, il est possible de choisir l'une des stratégies de tri suivantes :

Par défaut	Les noeuds sont triés par libellé par ordre alphabétique
Tri par colonne	Les noeuds sont triés par colonne(s) sélectionnée(s). L'ordre de tri (croissant/décroissant) peut être défini pour chaque colonne.
Tri par champ d'ordonnancement	Les noeuds sont triés par ordre de champ numérique masqué, ce qui permet à l'utilisateur de modifier dynamiquement l'ordre des noeuds frères dans la vue hiérarchique. Cette stratégie est uniquement disponible si il y a au moins un champ numérique 'Masqué' dans la table. Afin de pouvoir déplacer les noeuds dans la vue hiérarchique, il est nécessaire de désigner un champ d'ordonnancement éligible, défini dans la table sur laquelle la vue hiérarchique s'applique. Un champ d'ordonnancement doit avoir le type de données 'Entier' et doit avoir une visibilité par défaut à 'Masqué' dans les propriétés avancées dans la définition du modèle de données. Des actions de positionnement sur chaque noeud sont alors possibles, à moins que le champ d'ordonnancement ne soit en lecture seule ou qu'un filtre ne soit défini sur la hiérarchie.

Attention

Ne pas définir un champ qui a pour objet de contenir les données comme noeud d'ordonnancement, puisque les données seront écrasées dans la vue hiérarchique.

Actions sur un noeud de hiérarchie

Chaque noeud d'une vue hiérarchique possède un menu correspondant ▼ qui offre des actions contextuelles.

Les noeuds terminaux peuvent être détachés de leur parent en sélectionnant l'option 'Détacher du parent'. L'enregistrement devient ainsi un noeud orphelin dans l'arborescence, rangé sous un conteneur portant le nom 'non défini'.

Les noeuds terminaux peuvent aussi changer de noeud parent, grâce à l'option 'Attacher à un autre parent'. Si, selon le modèle de données, un noeud peut avoir plusieurs parents, le noeud sera à la fois

sous le parent d'origine et rajouté également sous le nouveau parent. Sinon, le noeud terminal sera déplacé sous le nouveau noeud parent.

Partage de vues

Les utilisateurs disposant de la permission 'Partager des vues' sur une vue ont la possibilité de définir quels utilisateurs peuvent sélectionner cette vue depuis leur menu 'Vue'.

Pour cela, il suffit d'ajouter des profils dans le champ 'Partager avec' de l'écran de configuration de la vue.

Publication de vue

Les utilisateurs disposant de la permission 'Publier des vues' peuvent publier les vues qui apparaissent dans leur menu 'Vue'.

Une vue publiée devient accessible à tous les utilisateurs via les composants Web, les tâches utilisateur du workflow, les services de données et les perspectives. Pour publier une vue, cliquez sur le bouton 'Editer' disponible au survol d'une vue présente dans le menu 'Vue' et remplissez le champ 'Nom de publication'.

20.6 Gestion des vues

Gérer les vues recommandées

Le propriétaire d'un jeu de données peut définir des vues recommandées pour chaque profil cible.

Quand un utilisateur se connecte sans spécifier de vue, la vue recommandée — si elle existe — s'applique. Sinon, la vue par défaut s'affiche. L'action 'Gérer les vues recommandées' permet de définir les règles d'attribution des vues recommandées par utilisateur et par rôle.

Les actions disponibles sur les vues recommandées sont les suivantes : changer l'ordre d'attribution des règles, ajouter une règle, éditer une règle, supprimer une règle existante.

Pour un utilisateur donné, les vues recommandées sont évaluées en fonction du profil de l'utilisateur : la règle appliquée sera la première de la liste qui correspond au profil de l'utilisateur.

Note

La fonctionnalité 'Gérer les vues recommandées' est uniquement accessible au propriétaire du jeu de données.

Barre d'outils des vues

La barre d'outils présente dans le menu 'Vue' permet d'accéder aux actions suivantes :

Editer	Cliquez sur le bouton 'Editer' dans la barre d'outils présente sur la ligne de la vue ciblée pour accéder au formulaire d'édition de la vue.
Dupliquer	Pour dupliquer la vue, cliquez sur le bouton 'Dupliquer' dans la barre d'outils présente sur la ligne de la vue ciblée. Un formulaire apparaît, pré-rempli à partir des valeurs de la vue dupliquée.
Supprimer	Cliquez sur le bouton 'Supprimer' dans la barre d'outils présente sur la ligne de la vue ciblée pour supprimer la vue.
Définir cette vue comme ma favorite	Cliquez sur le bouton 'Définir cette vue comme ma favorite' dans la barre d'outils présente sur la ligne de la vue ciblée. La vue favorite sera automatiquement appliquée lors de l'accès à la table. En cliquant une nouvelle fois sur le bouton, la vue courante ne sera plus la vue favorite de l'utilisateur.

20.7 Édition tabulaire

La fonctionnalité d'édition tabulaire permet de modifier les données dans une vue table. Elle est accessible en cliquant sur le bouton .

L'accès à l'édition tabulaire à partir d'une vue table nécessite que la fonctionnalité ait été préalablement activée dans la configuration de la vue.

Voir aussi [Édition tabulaire](#) [p 133]

Copier/coller

Le copier/coller d'une cellule vers une autre cellule de la même table s'effectue grâce au menu *Édition*. Il est aussi possible d'utiliser le raccourci clavier associé *Ctrl+C* et *Ctrl+V*.

Ce système n'utilise pas le presse-papier natif du système d'exploitation mais un mécanisme interne. Copier une cellule et la coller dans un fichier externe ne fonctionnera donc pas. Inversement, coller une valeur dans une cellule de la table ne fonctionnera pas non plus.

Tous les champs de type simple utilisant les composants intégrés sont supportés, sauf :

- les clés étrangères pointant vers des champs qui ne sont pas des chaînes de caractères ;
- les énumérations qui ne sont pas des chaînes de caractères.

20.8 Historique

La fonctionnalité d'historique permet le suivi des modifications de données.

La visualisation de l'historique de données nécessite que la fonctionnalité ait été préalablement activée au niveau des tables dans le modèle de données. Voir [Propriétés avancées des tables](#) [p 63] pour plus d'informations.

Pour visualiser l'historique d'un jeu de données, sélectionner *Actions* > *Historique* dans le panneau de navigation.

Pour visualiser l'historique d'une table ou d'une sélection d'enregistrements, sélectionner *Actions* > *Voir l'historique* dans l'espace de travail.

Il existe différents modes d'historique qui permettent de visualiser l'historique des données selon différentes perspectives :

Historique dans l'espace de données en cours	La vue d'historique de table affiche les opérations sur la branche en cours. C'est le mode par défaut.
Historique dans l'espace de données courant et ses ancêtres	La vue d'historique de table affiche les opérations sur la branche en cours ainsi que tous ses ancêtres.
Historique dans l'espace de données courant et ses enfants fusionnés	La vue d'historique de table affiche les opérations sur la branche en cours ainsi que toutes ses branches filles fusionnées.
Historique dans tous les espaces de données	La vue d'historique de table affiche les opérations sur toute la hiérarchie de la branche en cours.

Dans la vue d'historique, utiliser le menu *View* pour passer à un autre mode de l'historique.

Voir aussi [Historique](#) [p 269]

CHAPITRE 21

Edition des données

Ce chapitre contient les sections suivantes :

1. [Actions sur les enregistrements dans l'interface utilisateur](#)
2. [Import et export de données](#)
3. [Restauration depuis l'historique](#)

21.1 Actions sur les enregistrements dans l'interface utilisateur

L'édition des enregistrements s'effectue dans l'espace de travail de l'interface utilisateur.

Note

Seuls les utilisateurs autorisés peuvent accéder à cet écran via la 'Perspective avancée' ou via une perspective spécifique.

Création d'un enregistrement

Dans la vue tabulaire, un nouvel enregistrement peut être créé à l'aide du bouton **+** situé en haut à gauche de la table.

Dans une vue hiérarchique, sélectionnez "Créer un enregistrement" dans le menu du noeud parent du nouvel enregistrement.

Dans les deux cas, un formulaire s'affiche, permettant d'entrer des données. Les données obligatoires sont repérées par une astérisque rouge.

Modification d'un enregistrement

Un enregistrement peut être édité par double clic. Le formulaire qui s'affiche permet d'éditer l'enregistrement, tandis que le bouton *Rétablir* permet de recharger le formulaire sans soumettre aucun des changements effectués.

Dupliquer un enregistrement

Pour dupliquer un enregistrement, sélectionnez-le, puis sélectionnez *Actions > Dupliquer*.

Un formulaire apparaît, pré-rempli à partir des valeurs de l'enregistrement copié. La clé primaire doit ensuite être modifiée pour pouvoir créer ce nouvel enregistrement, à moins qu'elle ne soit générée automatiquement (à l'exemple d'une valeur auto-incrémentée).

Supprimer

Pour supprimer un ou plusieurs enregistrements sélectionnés, sélectionnez *Actions > Supprimer*.

Comparer

Deux enregistrements sélectionnés peuvent être comparés en sélectionnant *Actions > Comparer*.

Note

La comparaison ne prend pas en compte le contenu des noeuds terminaux complexes, comme les listes agrégées ou les attributs utilisateurs. Toutes les différences portant sur de tels noeuds seront ignorées.

21.2 Import et export de données

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

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

Voir aussi

[Services CSV](#) [p 149]

[Services XML](#) [p 143]

21.3 Restauration depuis l'historique

Dans une table où l'historique est activé, il est possible de restaurer un enregistrement à un état antérieur en se basant sur son historique. Dans le cas où l'enregistrement (identifié par sa clé primaire) existe encore dans la table, il sera mis à jour par les valeurs historisées à restaurer. Dans le cas contraire, il sera créé.

Pour restituer un enregistrement à un état antérieur, sélectionner un enregistrement dans la vue d'historique, puis sélectionner *Actions > Restaurer depuis l'historique* dans l'espace de travail. Un écran de résumé s'affiche avec les détails de la mise à jour ou de la création à effectuer.

La fonctionnalité de restauration est uniquement applicable sur un enregistrement à la fois.

Si un trigger de table doit avoir un comportement spécifique pour la restauration, c'est à dire différent de celui des opérations classiques de création et de mise à jour, le développeur peut utiliser la méthode `TableTriggerExecutionContext.isHistoryRestoreAPI`.

Note

Il existe des limitations liées à la fonctionnalité de l'historique :

- La restauration depuis l'historique n'est pas disponible pour les tables contenant des listes non supportées par l'historique. Voir [Limitations du modèle de données](#) [p 274].
- Les types de champ suivants ne sont pas historisés : les valeurs calculées, les champs cryptés, ainsi que les champs dont l'historique est désactivé. Ces champs seront donc ignorés lors de la restauration d'un enregistrement depuis l'historique.

Voir aussi [Historique](#) [p 269]

CHAPITRE 22

Import et export XML

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Imports](#)
3. [Exports](#)
4. [Gestion des valeurs de champ](#)
5. [Limitations connues](#)

22.1 Introduction

L'import et l'export XML des tables s'effectuent via l'interface utilisateur à partir du menu 'Actions' de l'espace de travail.

L'import et l'export sont réalisés dans le contexte d'un jeu de données.

L'import et l'export peuvent aussi être réalisés programmatiquement.

Les valeurs par défaut des options peuvent être définies dans 'Administration', sous *Interface utilisateur > Configuration de l'interface graphique > Valeurs par défaut des options > Import / Export*.

22.2 Imports

Attention

Les documents XML importés doivent être encodés selon la norme UTF-8 et leur structure doit respecter le modèle de données du jeu de données cible.

Mode d'import

Lors de l'import d'un fichier XML, il est nécessaire de spécifier un des modes d'import suivants, qui déterminera la façon dont la procédure d'import gère les enregistrements source.

Insertion seulement	Seule la création d'enregistrement est autorisée. Si un enregistrement avec la même clé primaire existe dans la table, une erreur se produit et l'opération est annulée.
Mise à jour seulement	Seule la mise à jour d'enregistrement est autorisée. Si un enregistrement avec la même clé primaire n'existe pas dans la table, une erreur se produit et l'opération est annulée.
Mise à jour ou insertion	Si un enregistrement avec la même clé primaire existe dans la table, il est mis à jour ; sinon, il est créé.
Remplacement (synchronisation)	Si un enregistrement avec la même clé primaire existe dans la table cible, celui-ci est mis à jour ; sinon, un nouvel enregistrement est créé. D'autre part, si un enregistrement n'est plus présent dans la source, il est supprimé.

Opérations d'insertion et de mise à jour

Le mode '*by delta*' permet d'ignorer les éléments du modèle de données qui manquent dans le document XML source. Ce mode peut être activé via les services de données ou l'API Java. Le tableau suivant résume le comportement des opérations d'insertion et de mise à jour lorsque les éléments sont absents du document source.

Voir les opérations des services de données [mise à jour](#) [p 721] et [insertion](#) [p 723], ainsi que `ImportSpec.setByDeltaAPI` dans l'API Java pour plus d'informations.

État dans le document XML source	Comportement
L'élément n'existe pas dans le document source	Si le mode 'by delta' est désactivé (par défaut) : Le champ cible prend une des valeurs suivantes : • Si l'élément définit une valeur par défaut, le champ cible prend cette valeur par défaut. • Si l'élément est d'un type autre que chaîne de caractères ou liste, le champ cible prend la valeur <code>null</code>. • Si l'élément est une liste agrégée, la valeur du champ cible prend la valeur d'une liste vide. • Si l'élément est une chaîne qui différencie <code>null</code> d'une chaîne de caractères vide, la valeur du champ cible prend la valeur <code>null</code>. S'il s'agit d'une chaîne qui ne fait pas la différence entre les deux, une chaîne vide. • Si l'élément (simple ou complexe) est caché dans services de données, la valeur cible reste inchangée. Voir aussi Hiding a field in Data Services [p 600] Note : L'utilisateur qui exécute l'import doit avoir les permissions nécessaires pour créer ou modifier la valeur du champ cible. Autrement, la valeur restera inchangée. Si le mode 'by delta' a été activé au travers des services de données ou de l'API Java : • Pour l'opération <code>update</code>, la valeur de champ reste inchangée. • Pour l'opération <code>insert</code>, le comportement est le même que lorsque le mode <code>byDelta</code> est désactivé.
L'élément existe tout en étant vide (par exemple, <code><fieldA/></code>)	• Pour des nœuds de type <code>xs:string</code> (ou un de ses sous-types), le champ cible prend la valeur <code>null</code> s'il distingue <code>null</code> d'une chaîne vide. Autrement, la valeur est une chaîne vide. • Pour les types de nœuds non <code>xs:string</code>, une exception est lancée conformément au XML Schema. Voir aussi TIBCO EBX® whitespace management for data types [p 584]
L'élément est présent et de valeur <code>null</code> (par exemple, <code><fieldA xsi:nil="true"/></code>)	Le champ cible prend toujours la valeur <code>null</code> sauf dans le cas des listes, pour lesquelles il n'est pas supporté. Afin d'utiliser l'attribut <code>xsi:nil="true"</code>, il est nécessaire d'ajouter la déclaration du namespace <code>xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"</code>.

Met les valeurs manquantes à nul

Lors d'une mise à jour d'enregistrement existant, si un nœud est absent ou vide dans le fichier XML : si cette option est à "oui", il sera considéré comme nul. Si cette option est à "non", il ne sera pas modifié.

Ignorer les colonnes supplémentaires

Il peut arriver que le document XML contienne des éléments qui n'existent pas dans le modèle de données cible. Par défaut, dans ce genre de cas, la procédure d'import échouera. Cependant, il est possible d'autoriser les utilisateurs à lancer des procédures d'import qui ignoreront les colonnes supplémentaires définies dans les fichiers XML. Cela peut se définir dans les paramètres de configuration de l'assistant d'import XML. La valeur par défaut de ce paramètre peut être modifiée dans la configuration de l'Interface utilisateur dans l'espace 'Administration'.

Verrouillage optimiste

Si l'attribut technique `ebxd:lastTime` existe dans le fichier XML source, le mécanisme d'import réalise une vérification afin d'empêcher une opération de mise à jour sur un enregistrement qui pourrait avoir changé depuis la dernière lecture. Afin d'utiliser l'attribut `ebxd:lastTime`, il est nécessaire d'ajouter la déclaration du namespace `xmlns:ebxd="urn:ebx-schemas:deployment_1.0"`. L'horodatage associé à l'enregistrement courant sera comparé à cet horodatage. S'ils sont différents, la mise à jour est rejetée.

22.3 Exports

Note

Les documents XML exportés sont toujours encodés en UTF-8.

Lors d'un export au format XML, si des filtres sont appliqués à la table, seuls les enregistrements correspondant au filtre seront inclus.

Les options d'export XML sont les suivantes :

Nom du fichier de téléchargement	Spécifie le nom du fichier XML à exporter. Ce champ est pré-rempli avec le nom de la table source des enregistrements.
Mode convivial	Indique si les valeurs doivent être présentées de façon conviviale pour l'utilisateur ou sous leur forme brute (format XML standard). En mode convivial, les dates et les nombres sont formatés selon la région de l'utilisateur, les clés étrangères et valeurs énumérées présentent les libellés associés, etc. Note: Si cette option est sélectionnée, le fichier exporté ne pourra pas être ré-importé.
Inclure les données techniques	Indique si des données techniques internes seront incluses dans l'export. Note: Si cette option est sélectionnée, le fichier exporté ne pourra pas être ré-importé.
Indenté	Spécifie si le fichier doit être indenté pour améliorer sa lisibilité par un humain.
Enlever le commentaire XML	Spécifie si le commentaire XML généré qui décrit la localisation des données et la date d'export doit être enlevé.

22.4 Gestion des valeurs de champ

Date, heure & format dateTime

Les formats de date et d'heure suivants sont supportés :

Type	Format	Exemple
xs:date	aaaa-MM-jj	2007-12-31
xs:time	HH:mm:ss ou HH:mm:ss.SSS	11:55:00
xs:dateTime	aaaa-MM-jjTHH:mm:ss ou aaaa-MM-jjTHH:mm:ss.SSS	2007-12-31T11:55:00

22.5 Limitations connues

Champs d'association

Les services d'import et d'export XML ne supportent pas les valeurs d'association.

L'export de ces champs ne causera aucune erreur, cependant, aucune valeur ne sera exportée.

L'import de ces champs causera une erreur et la procédure d'import échouera.

Nœuds de sélection

Les services d'import et d'export XML ne supportent pas les valeurs de sélection.

L'export de ces champs ne causera aucune erreur, cependant, aucune valeur ne sera exportée.

L'import de ces champs causera une erreur et la procédure d'import échouera.

CHAPITRE 23

Import et export CSV

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Exports](#)
3. [Imports](#)
4. [Gestion des valeurs de champ](#)
5. [Limitations connues](#)

23.1 Introduction

L'import et l'export CSV peuvent être réalisés sur des tables via l'interface utilisateur en utilisant le menu 'Actions' de l'espace de travail.

L'import et l'export sont réalisés dans le contexte d'un jeu de données.

L'import et l'export peuvent aussi être réalisés programmatiquement.

Les valeurs par défaut des options peuvent être définies dans 'Administration', sous *Interface utilisateur > Configuration de l'interface graphique > Valeurs par défaut des options > Import / Export*.

Voir aussi [Default option values](#) [p 432]

23.2 Exports

Lors d'un export au format CSV, si des filtres sont appliqués à la table, seuls les enregistrements correspondant au filtre seront inclus.

Les options d'export CSV sont :

Nom du fichier de téléchargement	Spécifie le nom du fichier CSV à exporter. Ce champ est pré-rempli avec le nom de la table source des enregistrements.
Jeu de caractères	Spécifie le jeu de caractère à utiliser pour le fichier exporté. La valeur par défaut est UTF-8.
Activer l'héritage	Afin de prendre en compte l'héritage [p 29] lors d'un export CSV, l'option doit être spécifiée au préalable dans le modèle. Pour plus d'informations sur l'héritage, voir Inheritance and value resolution [p 288]. Spécifie si l'héritage est pris en compte durant l'export CSV. Si l'héritage est activé, les valeurs de champs résolues sont exportées avec les données techniques qui définissent le mode d'héritage potentiel de l'enregistrement ou du champ. Si l'héritage est désactivé, les valeurs de champs résolues sont exportées et les enregistrements occultés sont ignorés. Par défaut, cette option est désactivée. Note : L'héritage est toujours ignoré lorsque le jeu de données de la table n'a pas de parent ou si la table n'a pas de champ hérité.
Mode convivial	Indique si les valeurs doivent être présentées de façon conviviale pour l'utilisateur ou sous leur forme brute (format XML standard). En mode convivial, les dates et les nombres sont formatés selon la région de l'utilisateur, les clés étrangères et valeurs énumérées présentent les libellés associés, etc. Note : Si cette option est sélectionnée, le fichier exporté ne pourra pas être ré-importé.
Inclure les données techniques	Indique si des données techniques internes seront incluses dans l'export. Note : Si cette option est sélectionnée, le fichier exporté ne pourra pas être ré-importé.
En-têtes de colonne	Spécifie s'il faut ou non inclure les en-têtes de colonne dans le fichier CSV. • Pas d'en-tête • Libellé : Une ligne est ajoutée au début du fichier CSV et contient dans chaque colonne son libellé correspondant. Chaque libellé est localisé conformément à la préférence de langue de la session

active. Si aucun libellé n'est défini pour un nœud, le nom technique du nœud est utilisé.

- **XPath:** Une ligne est ajoutée au début du fichier CSV et contient dans chaque colonne le chemin d'accès correspondant.

Séparateur de champ

Spécifie le séparateur de champ à utiliser lors des exports. Le séparateur par défaut est la virgule, il peut être redéfini dans *Administration > Interface utilisateur*.

Séparateur de liste

Spécifie le séparateur à utiliser pour les listes de valeurs. Le séparateur par défaut est le retour à la ligne, il peut être redéfini dans *Administration > Interface utilisateur*.

Les exports CSV programmatiques sont réalisés en utilisant les classes `ExportSpecAPI` et `ExportImportCSVSpecAPI` dans l'API Java.

23.3 Imports

Nom du fichier de téléchargement	Spécifie le nom du fichier CSV à importer.
Mode d'import	Lors de l'import d'un fichier CSV, il est nécessaire de spécifier un des modes suivants, qui contrôlera l'intégrité des opérations entre la source et la table cible. • Insertion seulement : Seule la création d'enregistrement est autorisée. Si un enregistrement avec la même clé primaire existe dans la table, une erreur se produit et l'opération est annulée. • Mise à jour seulement : Seule la mise à jour d'enregistrement est autorisée. Si un enregistrement avec la même clé primaire n'existe pas dans la table, une erreur se produit et l'opération est annulée. • Mise à jour ou insertion : Si un enregistrement avec la même clé primaire existe dans la table, il est mis à jour ; sinon, il est créé. • Remplacement (synchronisation) : Si un enregistrement avec la même clé primaire existe dans la table cible, celui-ci est mis à jour ; sinon, un nouvel enregistrement est créé. D'autre part, si un enregistrement n'est plus présent dans la source, il est supprimé.
Jeu de caractères	Spécifie le jeu de caractère à utiliser pour le fichier importé. La valeur par défaut est UTF-8.
En-têtes de colonne	Spécifie s'il faut ou non inclure les en-têtes de colonne dans le fichier CSV. • Pas d'en-tête • Libellé : Une ligne est ajoutée au début du fichier CSV et contient dans chaque colonne son libellé correspondant. Chaque libellé est localisé conformément à la préférence de langue de la session active. Si aucun libellé n'est défini pour un nœud, le nom technique du nœud est utilisé. • XPath: Une ligne est ajoutée au début du fichier CSV et contient dans chaque colonne le chemin d'accès correspondant.

Séparateur de champ	Spécifie le séparateur de champ à utiliser lors des imports. Le séparateur par défaut est la virgule, il peut être redéfini dans <i>Administration > Interface utilisateur</i> .
Séparateur de liste	Spécifie le séparateur à utiliser pour les listes de valeurs. Le séparateur par défaut est le retour à la ligne, il peut être redéfini dans <i>Administration > Interface utilisateur</i> .
Activer l'héritage	Afin de prendre en compte l'héritage [p 29] lors d'un export CSV, l'option doit être spécifiée au préalable dans le modèle. Pour plus d'informations sur l'héritage, voir Inheritance and value resolution [p 288] et <code>ExportImportCSVSpec.setInheritanceEnabled^{API}</code>. Spécifie si l'héritage est pris en compte pendant un import CSV. Si les données techniques dans le fichier CSV définissent un mode d'héritage, les champs ou les enregistrements correspondants sont obligatoirement hérités. Si des données techniques définissent un mode occulté, les enregistrements correspondants sont obligatoirement occultés. Autrement, les champs sont écrasés au profit de valeurs issues du fichier CSV. Par défaut, cette option est désactivée. Note : L'héritage est toujours ignoré lorsque le jeu de données de la table n'a pas de parent ou si la table n'a pas de champ hérité.

Les imports CSV programmatiques sont réalisés en utilisant les classes `ImportSpecAPI` et `ExportImportCSVSpecAPI` dans l'API Java.

23.4 Gestion des valeurs de champ

Listes agrégées

Les services d'import et d'export CSV supportent les champs à valeurs multiples, à savoir les listes agrégées. Seules les listes simples telles que les listes de `string`, `date`, ou `int` et les clés étrangères sont supportées. Si une clé étrangère est liée à un enregistrement via une clé primaire composée, chaque champ de la clé étrangère est une chaîne formatée, par exemple, `"true|99"`. Les listes agrégées des groupes ne sont pas exportées.

Lors d'un export, les éléments de la liste sont séparés par le séparateur de ligne. Dans les cas où le champ exporté contient déjà un séparateur de ligne, par exemple dans un `osd:html` ou un `osd:text`, le code `_crnl_` est inséré à la place du séparateur de ligne du champ de valeur. Le même formatage est attendu lors de l'import, l'ensemble des valeurs de champ sont alors entourées de guillemets.

Champs cachés

Les champs cachés sont exportés en tant que chaînes `ebx-csv:hidden`. Une chaîne cachée importée ne modifiera pas le contenu d'un champ.

Valeur de chaîne 'Null'

En utilisant les services d'import et d'export CSV, une chaîne dont la valeur est `null` est exportée en tant que chaîne vide. Par conséquent, une boucle de procédure d'export-import finira par remplacer les valeurs de chaîne `null` par des chaînes vides.

En utilisant les services programmatiques, la valeur spécifique `ebx-csv:nil` peut être assignée à des chaînes dont la valeur est `null`. Dans ce cas, les valeurs de chaîne `null` ne seront pas remplacées par des chaînes vides lors de boucles de procédure export-import. Voir `ExportImportCSVSpec.setNullStringEncodedAPI` dans l'API Java pour plus d'informations.

Formats de date, heure et dateTime

Les formats de date et d'heure suivants sont supportés :

Type	Format	Exemple
xs:date	aaaa-MM-jj	2007-12-31
xs:time	HH:mm:ss ou HH:mm:ss.SSS	11:55:00
xs:dateTime	aaaa-MM-jjTHH:mm:ss ou aaaa-MM-jjTHH:mm:ss.SSS	2007-12-31T11:55:00

23.5 Limitations connues

Listes agrégées de groupes

Les services d'import et d'export CSV ne supportent pas les groupes à valeurs multiples, à savoir, des listes agrégées d'éléments de type complexe. L'export de ces nœuds ne causera aucune erreur, cependant, aucune valeur ne sera exportée.

Groupes terminaux

Dans un fichier CSV, il est impossible de différencier un groupe terminal créé, contenant uniquement des champs vides, d'un groupe terminal non créé.

Par conséquent, quelques différences peuvent apparaître lors de la comparaison après avoir réalisé une boucle de procédure d'export-import. Afin de s'assurer de la symétrie de l'import et de l'export, il est préférable d'utiliser la fonction import et export XML. Voir [Import et export XML](#) [p 143].

En-têtes de libellé de colonne

Si deux colonnes partagent le même libellé, l'export de la table peut être réalisé avec succès, mais les données exportées ne pourront pas être ré-importées par la suite.

Champs d'association

Les services d'import et d'export CSV ne supportent pas les valeurs d'association, c'est à dire les enregistrements associés.

L'export de ces champs ne causera aucune erreur, cependant, aucune valeur ne sera exportée.

L'import de ces champs causera une erreur et la procédure d'import échouera.

Nœuds de sélection

Les services d'import et d'export CSV ne supportent pas les valeurs de sélection, c'est à dire les enregistrements sélectionnés.

L'export de ces champs ne causera aucune erreur, cependant, aucune valeur ne sera exportée.

L'import de ces champs causera une erreur et la procédure d'import échouera.

CHAPITRE 24

Actions sur les jeux de données existants

Ce chapitre contient les sections suivantes :

1. [Validation du jeu de données](#)
2. [Duplication d'un jeu de données](#)
3. [Désactivation d'un jeu de données](#)
4. [Gestion des permissions de jeux de données](#)

24.1 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, cliquer sur le bouton *Revalider* pour mettre à jour ce rapport. Pour supprimer tous les messages de validation actuellement associés au jeu de données, et pouvoir relancer une validation complète, cliquer sur le bouton *Réinitialiser le rapport de validation*.

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

Voir [Validation](#) [p 322] pour obtenir plus d'informations concernant la validation incrémentale de données.

24.2 Duplication d'un jeu de données

Pour dupliquer un jeu de données existant, sélectionnez-le dans le menu "[Sélectionner le jeu de données](#) [p 123]" ▼ dans le panneau de navigation, puis sélectionnez *Actions > Dupliquer*.

24.3 Désactivation d'un jeu de données

Si un jeu de données est activé, il sera sujet à la validation. Tous les éléments obligatoires doivent être définis pour que le jeu de données soit valide. Si un jeu de données est activé et valide, on considère qu'il peut être exporté de façon sécurisée vers des systèmes externes (ou utilisé par d'autres applications Java).

Il est possible de désactiver un jeu de données dont les éléments obligatoires ne sont pas définis, en spécifiant "Non" pour la propriété "Activé" dans *Actions > Informations*.

24.4 Gestion des permissions de jeux de données

Les permissions de jeux de données sont accessibles en sélectionnant *Actions* > *Permissions* dans le panneau de navigation.

Les permissions sont définies en créant des *profils*. Pour créer un nouveau profil de permissions, créez un nouvel enregistrement dans la table "Droits d'accès par profil".

Voir aussi [Profil](#) [p 25]

Profil	Indique le profil concerné par la permission définie.
Restriction d'accès	Indique si la permission définie restreint celles affectées à un utilisateur donné par des politiques définies pour d'autres profils. Voir Restriction d'accès [p 304].
Actions sur les jeux de données	Cette section spécifie les permissions des actions sur les jeux de données.
Créer un le jeu de données enfant	Indique si le profil peut créer un jeu de données enfants. L'héritage doit aussi être activé dans le modèle de données.
Dupliquer le jeu de données	Indique si le profil peut dupliquer le jeu de données.
Supprimer le jeu de données	Indique si le profil peut supprimer le jeu de données.
Activer/désactiver le jeu de données	Indique si le profil peut modifier la propriété "Activé" dans les informations du jeu de données. Voir Désactivation d'un jeu de données [p 157].
Créer une vue	Indique si le profil peut créer des vues et des hiérarchies.
Droits sur tables	Spécifie les permissions par défaut pour toutes les tables. Des permissions spécifiques peuvent être appliquées à une ou plusieurs tables en cliquant sur le bouton '+' sous Droits spécifiques par table.
Droits par défaut >Créer un nouvel enregistrement	Indique si le profil peut créer des enregistrements dans une table.
Droits par défaut >Surcharger des enregistrements	Indique si le profil peut remplacer des enregistrements hérités dans une table. Cette permission est utile quand on utilise l'héritage de jeu de données.
Droits par défaut >Occulter des enregistrements	Indique si le profil peut occulter des enregistrements hérités dans une table. Cette permission est utile quand on utilise l'héritage de jeu de données.
Droits par défaut >Supprimer un enregistrement	Indique si le profil peut supprimer des enregistrements dans une table.

Droits sur valeurs

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 affectée à un élément.

Le sélecteur de droits spécifiques permet d'attribuer des permissions d'accès spécifiques à un élément. Les liens "Lecture", "Ecriture" et "Non visible" déterminent les permissions d'accès correspondant à l'élément sélectionné.

Il est possible de retirer une permission d'accès spécifique en utilisant le lien "(par défaut)".

Droits sur les services

Spécifie les permissions d'accès sur les services. Un service barré n'est pas accessible à un profil.

CHAPITRE 25

Héritage entre jeux de données

En utilisant le concept d'héritage entre jeux de données, vous pouvez créer des jeux de données additionnels, à partir d'un jeu de données racine. Ces jeux de données enfants héritent des propriétés et des valeurs de leur parents, qui peuvent être surchargées si nécessaire. Plusieurs niveaux d'héritage peuvent être créés.

L'héritage peut être utilisé pour adapter des données de référence à divers contextes. Par exemple, il serait possible de définir des valeurs globales dans un jeu de données parent, et de créer des jeux de données enfants par zones géographiques. Ceci permettra à ces derniers d'hériter des valeurs de leur parent, et de les surcharger si besoin.

Note

Le comportement standard est d'interdire l'héritage de jeux de données. Il est donc nécessaire d'activer explicitement cette fonction au niveau du modèle de données.

Voir aussi [Configuration du modèle de données](#) [p 44]

Ce chapitre contient les sections suivantes :

1. [Structure de l'héritage entre jeux de données](#)
2. [Héritage de valeurs](#)

25.1 Structure de l'héritage entre jeux de données

Une fois le jeu de données racine créé, un jeu de données enfant peut être créé à l'aide du bouton **+**, situé dans l'écran de sélection des jeux de données du panneau de navigation.

Note

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

25.2 Héritage de valeurs

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

Dans une vue tabulaire, les valeurs héritées sont signalées par un repère dans le coin en haut à gauche de la cellule.

Le bouton  permet de surcharger une valeur.

Héritage d'enregistrement

Une table dans un jeu de données enfant hérite des enregistrements des tables de ses jeux de données ancêtres. La table dans le jeu de données enfant peut rajouter, éditer ou supprimer des enregistrements. Des états sont définis pour différencier les types d'enregistrement.

Racine	Un enregistrement racine est un enregistrement créé dans le jeu de données courant, qui n'existe pas dans les jeux de données ancêtres. Il sera hérité par les jeux de données enfants.
Hérité	Un enregistrement hérité est défini dans un des jeux de données ancêtres 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 enfants.
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 enfants.

Quand le bouton  est activé, la valeur de l'enregistrement est héritée du jeu de données parent. Ce bouton peut être désactivé, afin de surcharger l'enregistrement ou la valeur. Pour un enregistrement occulté, l'activation de ce bouton restaure l'état hérité.

La table suivante résume le comportement des enregistrements lorsque l'on crée, modifie, ou supprime un enregistrement, selon son état initial.

Etat	Création	Edition	Suppression
Racine	Création normale d'un enregistrement. L'enregistrement nouvellement créé sera hérité par ses jeux de données enfant.	Edition normale d'un enregistrement. Les nouvelles valeurs seront héritées par les jeux de données enfants.	Suppression normale d'un enregistrement. L'enregistrement va disparaître du jeu de données courant ainsi que des jeux de données enfants.
Hérité	Si un enregistrement est créé à l'aide de la clé primaire d'un enregistrement hérité existant, l'état de l'enregistrement devient surchargé, et sa valeur sera celle soumise à sa création.	Un enregistrement hérité doit être déclaré comme surchargé pour que ses valeurs soient modifiables.	Supprimer un enregistrement hérité change son état à "occulté".
Surchargé	Non applicable. Il est impossible de créer un nouvel enregistrement si la clé primaire est déjà utilisée.	Un enregistrement surchargé peut être remis à l'état "hérité", mais sa valeur spécifique sera perdue. Les valeurs de l'enregistrement surchargé peuvent être héritées ou modifiées.	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 devient "surchargé" et sa valeur sera celle soumise à la création.	Non applicable. Un enregistrement occulté ne peut plus être édité.	Non applicable. Un enregistrement occulté est déjà considéré comme supprimé, et ne peut donc pas être supprimé une deuxième fois.

Voir aussi [Record lookup mechanism](#) [p 290]

Modèles de workflow

CHAPITRE 26

Introduction aux modèles de workflow

Ce chapitre contient les sections suivantes :

1. [Présentation](#)
2. [Utilisation de l'interface de modélisation de workflow](#)
3. [Modèles de message génériques](#)
4. [Limitations de workflows](#)

26.1 Présentation

Définition d'un modèle de workflow

Dans TIBCO EBX®, les workflows facilitent la gestion collaborative des données dans le référentiel. Un workflow peut contenir des actions utilisateurs sur les données ainsi que des tâches automatiques, tout en émettant des notifications sur différents événements.

La première étape pour réaliser un workflow est de créer un *modèle de workflow* qui définit la succession d'étapes, les implications des utilisateurs, ainsi que le comportement du workflow.

Une fois qu'un modèle de workflow est défini, il peut être validé et publié comme *publication de workflow*. Ensuite, les workflows de données peuvent être lancés à partir de la publication de workflow pour exécuter les étapes définies dans le modèle de workflow.

Voir aussi

[Modèle de workflow \(glossaire\)](#) [p 31]

[Workflow de données \(glossaire\)](#) [p 33]

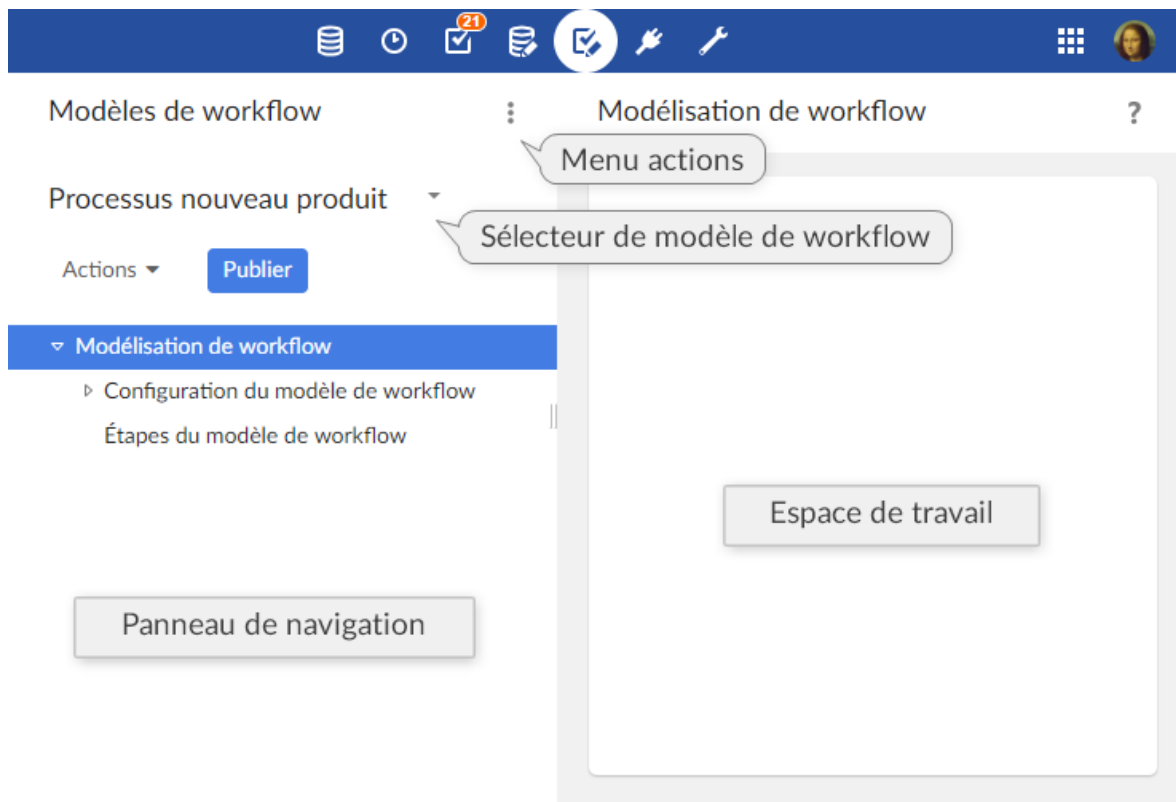
Concepts de base utilisés dans la modélisation des workflows

Une compréhension des termes suivants est nécessaire pour réaliser la création de modèles de workflows :

- [tâche automatique](#) [p 32]
- [tâche utilisateur](#) [p 32]
- [bon de travail](#) [p 33]

- [condition de workflow](#) [p 32]
- [appel à des sous-workflows](#) [p 32]
- [tâche d'attente](#) [p 32]
- [contexte des données](#) [p 32]

26.2 Utilisation de l'interface de modélisation de workflow



Note

Seuls les utilisateurs autorisés peuvent accéder à cet écran via la 'Perspective avancée'.
Seuls les utilisateurs autorisés peuvent accéder à ces interfaces spécifiques.

26.3 Modèles de message génériques

Des emails de notification peuvent être envoyés pour notifier les utilisateurs d'événements spécifiques pendant l'exécution d'un workflow.

Les modèles de message peuvent être définis et réutilisés dans n'importe quel modèle de workflow dans le référentiel. Pour modifier les modèles de message génériques, sélectionnez "Modèles de message" dans le menu 'Actions' de la section Modèles de Workflow.

Ces modèles, qui sont partagés par tous les modèles de workflows, 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.

A noter également que, lors d'un archivage, si l'on souhaite sauvegarder ces modèles de messages, le jeu de données "configuration" doit être sélectionné car celui-ci contient ces modèles.

A la création d'un modèle de message générique, deux champs sont obligatoires :

- 'Libellé & Description' : spécifie les libellés et descriptions associés à ce modèle de message, localisé.
- 'Message' : spécifie l'objet de l'email et son corps de texte, localisés.

Le 'Type du message' est une donnée facultative.

Le message peut inclure des variables du contexte de données sous la forme `${nom.variable}`, qui seront évaluées lorsque le message sera envoyé. De plus, les variables système suivantes peuvent être incluses :

system.time	Heure système du référentiel.
system.date	Date système du référentiel.
workflow.lastComment	Dernier commentaire sur la tâche utilisateur précédente. (Note : cette variable concerne la dernière tâche utilisateur, et non la courante. Est considérée courante la tâche sur laquelle est positionné le workflow, incluant également la notification de complétion de tâche utilisateur).
workflow.lastDecision	Dernières décisions sur la tâche utilisateur précédente. (Note : cette variable concerne la dernière tâche utilisateur, et non la courante. Est considérée courante la tâche sur laquelle est positionné le workflow, incluant également la notification de complétion de tâche utilisateur).
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	Lien d'accès au bon de travail courant dans la corbeille, au moyen de l'API du composant web. Pour que ce lien soit calculé, il est nécessaire qu'un bon de travail courant soit défini et

que l'URL soit configurée dans Workflow-executions, dans la configuration de mail.

workflow.workItem.link.allocateAndStart	Lien d'accès au bon de travail courant dans la corbeille, au moyen de l'API du composant web. Si le bon de travail cible n'est pas encore démarré, il sera automatiquement alloué à l'utilisateur qui a cliqué sur le lien, puis démarré. Pour que ce lien soit calculé, il est nécessaire qu'un bon de travail courant soit défini et que l'URL soit configurée dans Workflow-executions, dans la configuration de mail.
workflow.currentStep.label	Libellé de l'étape courante.
workflow.currentStep.description	Description de l'étape courante.

Exemple

Modèles de message générique :

Aujourd'hui à \${system.time}, un nouveau bon de travail vous a été proposé.

Email résultant :

Aujourd'hui à 15:19, un nouveau bon de travail vous a été proposé.

26.4 Limitations de workflows

Les fonctionnalités suivantes ne sont pas supportées :

- **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.
- La compatibilité du workflow n'est pas garantie pour des **noms d'utilisateurs dépassant les 64 caractères**.

Concepts apparentés [Workflows de données](#) [p 196]

CHAPITRE 27

Modélisation du workflow

Ce chapitre contient les sections suivantes :

1. [Création d'un modèle de workflow](#)
2. [Implémentation des étapes](#)
3. [Tâche utilisateur](#)
4. [Tâche autonome](#)
5. [Conditions](#)
6. [Appels à des sous-workflows](#)
7. [Tâches d'attente](#)
8. [Visualisation du diagramme de workflow](#)

27.1 Création d'un modèle de workflow

Un modèle de workflow peut être créé à partir de la section 'Modèle de workflow'. La seule information requise à la création est un nom de modèle unique dans le référentiel.

Les étapes du modèle de workflow sont initialisées avec une transition initiale. Pour implémenter le modèle de workflow, il faut définir la séquence des étapes qui suivent cette transition initiale.

27.2 Implémentation des étapes

Un modèle de workflow définit des étapes correspondant à des opérations et des conditions. Les étapes possibles sont les suivantes :

- Tâche utilisateur
- Tâche automatique
- Condition
- Appel à des sous-workflows
- Tâche d'attente

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

Stratégie d'avancement de l'étape suivante

Pour chaque type d'étape (sauf les tâches utilisateurs à multi bons de travail), une propriété est disponible pour préciser la stratégie d'avancement pour l'étape suivante. A la terminaison de l'étape, cette stratégie est évaluée pour conditionner la navigation lors de l'exécution du workflow. Par défaut, la stratégie d'avancement est 'Afficher la table des bons de travail'. Dans ce cas, après l'exécution de l'étape, la table des bons de travail est automatiquement affichée pour sélectionner le prochain bon de travail à ouvrir.

Une autre stratégie est disponible : 'Ouvrir automatiquement l'étape suivante'. Cette stratégie permet à l'utilisateur de garder la main sur ce workflow et d'exécuter directement l'étape suivante. Si, à la suite de cette exécution, un bon de travail est atteint et que l'utilisateur connecté peut le démarrer, le bon de travail est automatiquement ouvert (si plusieurs bons de travail sont atteints, le premier créé est ouvert). Sinon, la stratégie d'avancement de l'étape suivante est évaluée. Si aucun bon de travail n'est finalement atteint, la table des bons de travail est affichée.

Cette stratégie est préconisée pour exécuter plusieurs étapes à la suite sans repasser par la corbeille de tâches.

Il existe actuellement plusieurs limitations qui font que cette stratégie peut être ignorée. Dans ce cas, la table des bons de travail est automatiquement affichée. Le cas où cette propriété est ignorée est le suivant : si l'étape courante est une tâche utilisateur avec plus d'un bon de travail.

Dans le cas des conditions, deux autres stratégies sont proposées : 'Si vrai, ouvrir automatiquement l'étape suivante' et 'Si faux, ouvrir automatiquement l'étape suivante'. Ces stratégies permettent de conditionner la stratégie à appliquer en fonction du résultat de la condition.

Dans le cas des appels sous-workflows, une propriété dédiée "Sous-workflow de premier plan" est disponible pour la stratégie d'avancement. Pour la stratégie d'avancement "Ouvrir automatiquement le prochain bon de travail", un sous-workflow de premier plan doit être sélectionné. Seulement les tâches de ce sous-workflow de premier plan et leurs stratégies d'avancement seront évaluées. Veuillez noter les règles spécifiques de la propriété sous-workflow de premier plan.

- Si un seul sous-workflow est déployé, il sera automatiquement considéré comme premier plan,
- La propriété de sous-workflow de premier plan sera ignoré si l'étape précédente n'a pas sa stratégie d'avancement définie sur "Ouvrir automatiquement le prochain bon de travail",
- When all the sub-workflow are completed, and if the last completed sub-workflow is the foreground one, then the progress strategy defined on the sub-workflow invocation step is evaluated: if the progress strategy is « automatically open the next step », the next step can be opened without displaying the inbox. In all cases, the progress strategy of the last step of the sub-workflow is always ignored. Quand tous les sous-workflows sont terminés et que le dernier complété est celui de premier plan, dans ce cas, la stratégie d'avancement de l'appel sous-workflow sera évaluée: si la stratégie est "Ouvrir automatiquement le prochain bon de travail", la prochaine étape sera ouverte au lieu d'ouvrir la corbeille. Dans tous les cas, la stratégie d'avancement de la dernière étape du sous-workflow sera ignorée.

Masquée dans la vue graphique

Pour chaque type d'état, une propriété est disponible pour définir quelles étapes doivent être masquées dans la vue graphique de workflow par défaut.

Si cette propriété est activée, l'étape sera automatiquement masquée dans la vue graphique de workflow pour les utilisateurs non-administrateurs (ni administrateurs built-in, ni administrateurs de workflow).

27.3 Tâche utilisateur

Une tâche utilisateur est une étape qui nécessite une action de la part d'un utilisateur humain. Son libellé et sa description peuvent être localisés.

Mode

Pour des raisons de compatibilité ascendante, deux modes de tâche utilisateur sont disponibles : le mode par défaut et le mode de compatibilité.

Dans le mode par défaut, une tâche utilisateur génère un seul bon de travail. Ce mode propose d'avantage de fonctionnalités, comme proposer un bon de travail à une liste de profils, ou afficher les avatars directement dans la vue graphique de workflow.

Dans le mode de compatibilité, une tâche utilisateur peut générer plusieurs bons de travail.

Par défaut, le service de création de tâche utilisateur est masqué en mode compatibilité. Pour l'afficher, il faut activer une propriété dans le fichier `ebx.properties`. Pour plus d'informations, voir [Disabling user task legacy mode](#) [p 384].

Liste de profils

La définition des profils d'une tâche utilisateur varie en fonction du mode de la tâche utilisateur.

[Mode par défaut] Proposée aux profils suivants

Les profils définis sont les rôles ou les utilisateurs auxquels la tâche utilisateur est proposée. Lors de l'exécution de la tâche utilisateur, un seul bon de travail est généré. Si un seul utilisateur est défini, le bon de travail est automatiquement alloué à cet utilisateur. Si un rôle est défini, le bon de travail est proposé aux membres du rôle. Si plusieurs utilisateurs et rôles sont définis, le bon de travail est proposé à la fois à ces utilisateurs et aux membres de ces rôles.

[Mode de compatibilité] Participants

Les participants sont les rôles ou les utilisateurs auxquels la tâche utilisateur est destinée. Par défaut, lors de l'exécution de la tâche utilisateur, un bon de travail est généré par profil. Si un profil est un utilisateur, le bon de travail est automatiquement alloué à cet utilisateur. Si un profil est un rôle, le bon de travail est proposé aux membres du rôle.

For more information

Voir aussi [Extension](#) [p 175]

Service

TIBCO EBX® propose les services prédéfinis suivants :

- Accéder à des données
- Accéder à l'interface de fusion des espace de données
- Accéder à un espace de données
- Comparer deux contenus
- Créer un nouvel enregistrement

- Dupliquer un enregistrement
- Exporter les données depuis une table au format CSV
- Exporter les données depuis une table au format XML
- Fusionner un espace de données
- Importer les données dans une table depuis un fichier CSV
- Importer les données dans une table depuis un fichier XML
- Valider un espace de données, une image ou un jeu de données

Voir aussi [Services prédéfinis de EBX®](#) [p 237]

Configuration

Options générales > Rejet activé

Par défaut, seulement l'action *accepter* est proposée à l'utilisateur lorsqu'il enregistre sa décision.

Il est possible d'activer l'action 'rejeter' en positionnant ce champ à 'Oui'.

Options générales > 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 'Non'.

Options générales > Activation des commentaires

Par défaut, les commentaires sont activés. Lorsqu'un bon de travail est ouvert, un bouton 'Commentaires' est affiché et permet à l'utilisateur de saisir un commentaire.

Il est possible de masquer ce bouton 'Commentaires' en positionnant cette propriété à *Non*.

Options générales > Caractère obligatoire du commentaire

Par défaut, le commentaire associé à un bon de travail est optionnel.

Il est possible de forcer l'utilisateur à saisir un commentaire avant d'enregistrer sa décision en positionnant ce champ sur le comportement attendu : 'toujours obligatoire', 'obligatoire seulement si le bon de travail a été accepté' ou 'obligatoire seulement si le bon de travail a été rejeté'.

Options générales > 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 certaines tâches 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.

[Mode de compatibilité] 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 de l'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.handleWorkItemCompletionAPI` dans la JavaDoc pour plus d'informations.

[Mode de compatibilité] Terminaison > Tolérance de 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 suivants tolèrent automatiquement tous les rejets :

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

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, pour la création du bon de travail ou pour terminer la tâche de l'utilisateur.

La règle spécifiée est une classe Java Bean qui doit étendre la classe `UserTaskAPI`.

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 génériques](#) [p 167]

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 atteinte et si les bons de travail associés n'ont pas été terminés, un courrier électronique spécifique est envoyé aux utilisateurs concernés. Ce courrier électronique va alors être renvoyé 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.

27.4 Tâche autonome

Il existe deux types de tâches automatiques :

Script de la bibliothèque	EBX® fournit des scripts de la bibliothèque prédéfinis, qui peuvent être utilisés directement. Les scripts de la bibliothèque spécifiques doivent être déclarés dans un fichier <code>module.xml</code>. Un script de la bibliothèque spécifique doit définir son libellé, sa description et ses paramètres. Quand un utilisateur sélectionne un script de la bibliothèque dans une étape d'un modèle de données, les paramètres associés sont affichés dynamiquement.
Script spécifique	Spécifie une classe Java qui exécute des actions spécifiques. La classe associée doit être dans le même module que celui associé au modèle de workflow. Ses libellés et ses descriptions ne sont pas affichés dynamiquement aux utilisateurs dans un modèle de workflow.

[Packaging TIBCO EBX® modules](#) [p 513]

Script de la bibliothèque

EBX® inclut les scripts de la bibliothèque prédéfinis suivants :

- Créer un espace de données
- Créer une image
- Fusionner un espace de données
- Importer une archive
- Fermer un espace de données
- Modifier une variable du contexte de données

- Envoyer un courrier électronique
- Supprimer des enregistrements (Note : ce script peut supprimer plusieurs enregistrements à la fois)

Les scripts de la bibliothèque sont des classes qui étendent la classe `ScriptTaskBeanAPI`. En complément des scripts de la bibliothèque prédéfinis, vous pouvez définir des scripts de la bibliothèque additionnels pour les utiliser dans les modèles de workflows. Leurs libellés et descriptions peuvent être localisés.

La méthode `ScriptTaskBean.executeScriptAPI` est appelée lorsque le workflow de données atteint l'étape correspondante.

Attention

La méthode `ScriptTaskBean.executeScriptAPI` ne doit créer 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 ne garantit donc pas sa terminaison avant le passage à l'étape suivante.

Voir [l'exemple](#) [p 648].

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

Note

Certains scripts de bibliothèque prédéfinis sont marqués comme "obsolètes" car ils ne sont pas compatibles avec internationalisation. Il est recommandé d'utiliser les nouvelles tâches qui sont compatibles avec internationalisation.

Scripts spécifiques

Les scripts spécifiques ont la particularité d'étendre la classe [Sample of ScriptTask](#) [p 648].

La méthode `ScriptTask.executeScriptAPI` est appelée lorsque le workflow de données atteint l'étape correspondante.

Attention

La méthode `ScriptTask.executeScriptAPI` ne doit créer 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 ne garantit donc pas sa terminaison avant le passage à l'étape suivante.

Voir [l'exemple](#) [p 648].

Il n'est pas possible de paramétrer dynamiquement les variables d'un script spécifique. Toutefois, le contexte des données est accessible depuis le Java Bean.

27.5 Conditions

Les conditions sont des étapes décisionnelles du workflow.

Il existe deux types de conditions qui, une fois définies, peuvent être utilisées dans les étapes de modélisation du workflow :

Condition de la bibliothèque	EBX® fournit des conditions de la bibliothèque prédéfinies, qui peuvent être utilisées directement. Les conditions de la bibliothèque spécifiques doivent être déclarées dans un fichier <code>module.xml</code>. Une condition de la bibliothèque spécifique doit définir son libellé, sa description et ses paramètres. Quand un utilisateur sélectionne une condition de la bibliothèque dans une étape d'un modèle de données, les paramètres associés sont affichés dynamiquement.
Condition spécifique	Spécifie une classe Java qui implémente une condition spécifique. La classe associée doit être dans le même module que celui associé au modèle de workflow. Ses libellés et ses descriptions ne sont pas affichés dynamiquement aux utilisateurs dans un modèle de workflow.

[Packaging TIBCO EBX® modules](#) [p 513]

Condition de la bibliothèque

EBX® inclut les conditions de la bibliothèque prédéfinies suivantes :

- Espace de données modifié ?
- Espace de données valide ?
- Dernière tâche utilisateur acceptée ?
- Valeur nulle ou vide ?
- Valeurs égales ?

Les conditions de la bibliothèque sont des classes qui étendent la classe `ConditionBeanAPI`. En complément aux conditions de la bibliothèque prédéfinies, vous pouvez définir des conditions de la bibliothèque additionnelles pour les utiliser dans des modèles de données. Leurs libellés et descriptions peuvent être localisés.

Voir [l'exemple](#) [p 651].

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

Conditions spécifiques

Les conditions spécifiques ont la particularité d'étendre la classe `ConditionAPI`.

Voir [l'exemple](#) [p 650] dans l'API Java.

Il n'est pas possible de paramétrer dynamiquement les variables d'une condition spécifique. Toutefois, le contexte des données est accessible depuis le Java bean.

27.6 Appels à des sous-workflows

Les étapes du type appel à des sous-workflows positionnent le workflow de données courant dans l'état "en attente" et lancent un ou plusieurs sous-workflows.

Il est possible d'inclure un autre modèle de workflow dans le modèle de workflow courant en définissant un appel unique à ce sous-workflow dans une étape.

Si plusieurs sous-workflows sont appelés par une étape, ils sont exécutés simultanément, en parallèle. Tous les sous-workflows doivent être terminés pour que le workflow parent passe à l'étape suivante. Le libellé et la description de chaque sous-workflow peuvent être localisés.

The property « Foreground sub-workflow » is useful to precise the progress strategy when several sub-workflows are launched. If the previous step progress strategy is « directly open the next step », this property defines which sub-workflow should be considered in the foreground (only one sub-workflow can hold this behavior). Only the work items which have been generated in this foreground sub-workflow will be able to be opened automatically without passing by the inbox. If only one sub-workflow is defined, this property is ignored (the single sub-workflow is automatically considered in the foreground). La propriété "Sous-workflow de premier plan" est utile pour déterminer la stratégie d'avancement quand plusieurs sous-workflows sont lancés. Si la stratégie d'avancement de l'étape précédente est "Ouvrir automatiquement le prochain bon de travail", cette propriété définit quel sous-workflow sera considéré de premier plan (seulement un peu avoir ce comportement). Seul les bon de travail généré dans ce sous-workflow de premier plan pourront être ouvert automatiquement sans passer par la corbeille. Pour plus d'information, veuillez vous référer à la stratégie d'avancement : [Stratégie d'avancement de l'étape suivante](#) [p 172]

Deux types d'appels à des sous-workflows existent :

Statique	Définit un ou plusieurs sous-workflows à lancer chaque fois que cette étape est exécutée dans un workflow de données. Pour chaque sous-workflow, il est possible de spécifier ses libellés et ses descriptions, ainsi que les mappings d'entrée et de sortie dans son contexte de données. Ce mode est utile quand on sait à l'avance quels sous-workflows doivent être lancés, et comment le mapping de sortie doit être réalisé.
Dynamique	Spécifie une classe Java qui implémente un appel spécifique à des sous-workflows. Tous les workflows qui peuvent éventuellement être appelés comme sous-workflows par le code doivent être déclarés comme dépendances. Le contexte de données est directement accessible depuis le Java bean. Les appels de sous-workflows dynamiques doivent être déclarés dans un fichier <code>module.xml</code>. Ce mode est utile quand le lancement des sous-workflows est conditionnel (par exemple, lorsqu'il dépend d'une variable du contexte de données), ou quand le mapping de sortie dépend de l'exécution des différents sous-workflows.

27.7 Tâches d'attente

Une étape d'attente dans un modèle de workflow met le workflow en pause en attendant la réception d'un événement donné.

Lorsqu'une tâche d'attente est appelée, le moteur de workflow génère un identifiant unique de réveil lié à la tâche d'attente. Cet identifiant est requis pour réveiller la tâche d'attente et, par conséquent, le workflow associé.

Une tâche d'attente spécifie quel profil est autorisé à réveiller la tâche d'attente ; et la classe Java qui implémente un bean de tâche d'attente : `waitTaskBean`^{API}.

Le contexte de données du workflow est directement accessible à partir du Java bean.

Les beans de tâche d'attente doivent être déclarés dans un fichier `module.xml`.

Le bean de tâche d'attente est d'abord appelé lorsque le workflow est mis en pause. L'identifiant de réveil est alors disponible pour appeler un web service, par exemple. Puis, le bean de tâche d'attente est appelé lorsque la tâche d'attente est réveillée. De cette façon, le contexte de données peut être mis à jour en fonction des paramètres reçus.

Note

L'administrateur built-in est toujours autorisé à réveiller un workflow.

27.8 Visualisation du diagramme de workflow

A propos

Lorsqu'un modèle de workflow est défini, il est possible de le visualiser sous forme d'un diagramme qui se rapproche de la norme BPMN.

Ce service est accessible en vue par défaut dans l'éditeur de modèles de workflow et permet de faire l'intégralité des actions de création, d'édition et de suppression.

Notez également que ce diagramme s'inspire des normes BPMN mais n'en est pas une stricte implémentation, puisque les concepts du workflow d'EBX® sont légèrement différents.

Sauvegarde de la mise en page

Il est possible de sauvegarder la mise en page du diagramme et de le sauvegarder. Notez toutefois que la sauvegarde est globale et non locale ; elle est donc partagée par tous les utilisateurs.

Actions

Exporter en PNG	Crée une image PNG du modèle.
Exporter en SVG	Crée une image SVG du modèle.
Exporter en PDF	Créer un document PDF basé sur le modèle workflow. Vous pourrez configurer l'orientation des pages du pdf (Paysage, Portrait), la taille des pages (A3, Lettre, etc ...). Ainsi que deux propriétés qui pourraient donner plus de détail au modèle: • Afficher les identifiants (valeur par défaut est vrai): Même comportement que "Afficher les identifiants d'étapes" dans la barre d'outil du diagramme de workflow mais restreint pour l'export pdf. Les identifiants de chaque étapes seront affichées. • Ajouter l'index (valeur par défaut est vrai): A la fin du pdf, un index listant chaque étapes du diagramme sera ajouté. Cela pourra être utile dans le cas où le titre de plusieurs étapes seraient trop long pour pouvoir être affiché dans leur intégralité. <i>Pour cette propriété, il est recommandé d'afficher les identifiants des étapes puisqu'ils seront utiles pour savoir quelle ligne de l'index correspond à quelle étape.</i>

Vue

Mise en page > Mise en page par défaut	Applique la mise en page par défaut.
Affichage > Afficher/Masquer la grille	Affiche la grille lorsque celle-ci est invisible, sinon masque la grille.
Affichage > Zoomer	Fait un zoom sur le diagramme. Pour aller plus vite, vous pouvez utiliser Ctrl + molette vers le haut avec votre souris se trouvant dans la zone du diagramme.
Affichage > Dézoomer	Fait un dézoom sur le diagramme. Pour aller plus vite, vous pouvez utiliser Ctrl + molette vers le bas avec votre souris se trouvant dans la zone du diagramme, voir Vue [p 182]
Affichage > Afficher/Masquer les identifiants d'étapes	Affiche ou masque une étiquette contenant l'identifiant d'une étape workflow.
Vues de plan > Hiérarchie	Si elle est activée, affiche la vue hiérarchique du modèle de workflow.

Boutons

Sauvegarder	Sauvegarde la mise en page.
Sauvegarder et fermer	Sauvegarde la mise en page et ferme le service.
Recharger	Annule les modifications de la mise en page en cours et recharge la mise en page précédemment sauvegardée.
Fermer	Ferme le service.

Edit

Création	Au survol ou à la sélection d'un lien, un bouton '+' apparaît. Ce dernier sert à la création d'une étape.
Suppression	SUPPR Le raccourci clavier "SUPPR" supprime le noeud sélectionné. NB: cette action est irréversible.
Barre d'outils	Au survol ou à la sélection d'un noeud, une barre d'outils s'affiche. Elle sert à : • Éditer une étape. • Supprimer une étape. • Dupliquer une étape. • Ajouter une redirection vers une étape existante. • Afficher ou cacher dans la vue d'exécution.
Éditer un noeud	Le double-clic permet d'éditer rapidement une étape.

Fonctionnalités

La vue graphique offre également des fonctionnalités additionnelles

Annuler l'action précédente	CTRL + Z Les actions ci-dessous ne seront pas affectées par ce raccourci : • Création/Suppression/Edition/Duplication d'une étape • Déplacement de liens • Afficher/Cacher dans la vue d'exécution • Recharger
Zoomer/Dezoomer	Bouton du milieu de la souris puis roulette / CTRL puis roulette
Sélection multiple	cliquer sur le noeud ou lien à sélectionner tout en appuyant sur la touche CTRL / Dessiner un rectangle de sélection (Il faut attendre 1 seconde après le clic pour activer la zone de sélection)
Personnaliser le tracé des liens	Lorsqu'on clique sur un lien, on peut déplacer chaque segment grâce aux carrés de sélection qui apparaissent sur les coins de chaque arête, ou fractionner le segment en déplaçant le cercle qui apparaît au milieu de chaque segment
Aperçu	Un panneau est maintenant disponible avec un diagramme de workflow miniaturisé qui pourra être utilisé pour la navigation. Il sera possible de le réduire, l'agrandir et le déplacer dans la surface correspondant à la vue du diagramme de workflow.

CHAPITRE 28

Configuration du modèle de workflow

Ce chapitre contient les sections suivantes :

1. [Informations](#)
2. [Propriétés du modèle de workflow](#)
3. [Contexte de données](#)
4. [Personnalisation des vues d'exécution de workflow](#)
5. [Permissions sur les workflows de données associés](#)
6. [Historique des images du modèle de workflow](#)
7. [Suppression d'un modèle de workflow](#)

28.1 Informations

Pour visualiser et éditer les informations concernant le propriétaire et la documentation du modèle de workflow, sélectionnez "Informations" dans le menu ["Actions" du modèle de workflow](#) [p 167] dans le panneau de navigation.

Propriétaire	Personne pouvant éditer les informations du modèle de workflow et définir des règles de permission.
Documentation locale	Libellé et description associés au modèle de workflow localisés.
Activation	<i>Cette propriété est obsolète.</i> Spécifie si un modèle de workflow est activé. Un modèle de workflow doit être activé pour pouvoir être publié.

28.2 Propriétés du modèle de workflow

La configuration d'un modèle de workflow est accessible depuis le panneau de navigation.

Module	Module contenant les ressources Java spécifiques (extension de tâche utilisateur, script et conditions spécifiques).
Notification de démarrage	Liste des profils qui doivent recevoir une notification (choisie dans la liste des modèles de messages) quand un workflow démarre. Voir Modèles de message génériques [p 167].
Notification de fin	Liste des profils qui doivent recevoir une notification (choisie dans la liste des modèles de messages), quand un workflow se termine. La notification n'est envoyée que si le workflow a été terminé "normalement" (pas par une action d'administration). Voir Modèles de message génériques [p 167].
Notification d'erreur	Liste des profils qui recevront une notification (choisie dans la liste des modèles de messages) quand un workflow est mis en d'erreur. Voir Modèles de message génériques [p 167].
Priorité	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 [p 209] pour plus d'informations. Note : Seuls les utilisateurs définis en tant qu'administrateurs de workflows seront autorisés à modifier la priorité des workflows de données associés manuellement.
Activer le lancement direct	Par défaut, lorsqu'un workflow est lancé, il est proposé à l'utilisateur de saisir une documentation pour le nouveau workflow dans un formulaire intermédiaire. Cette documentation est facultative. Positionner la propriété "Activer le lancement direct" à "Oui" permet d'éviter cette étape de documentation et de lancer directement le workflow.

Ouvrir automatiquement la première étape	Permet de conditionner la navigation, suite à un lancement de workflow. Par défaut, une fois le workflow lancé, la table courante (lanceurs de workflow ou monitoring > publications) est automatiquement affichée. Activer cette propriété permet au créateur du workflow de garder la main sur le workflow lancé. Si, suite à l'exécution de la première étape du workflow, un bon de travail est atteint, et que ce bon de travail peut être démarré par le lanceur de workflow, le bon de travail est automatiquement ouvert (si plusieurs bons de travail sont atteints, le premier créé est ouvert). Cela évite au lanceur de sélectionner le bon de travail correspondant dans la corbeille de tâches. Si aucun bon de travail n'est atteint, la stratégie d'avancement de l'étape suivante est évaluée. Si aucun bon de travail n'est finalement ouvert, la table à partir de laquelle le workflow a été lancé est affichée. Limitation : Cette propriété est ignorée si la première étape est un appel de sous-workflow.
Trigger de workflow	Composant interceptant les événements principaux d'un workflow. Ce bean doit être déclaré dans un fichier <code>module.xml</code>. Voir l'exemple [p 654].
Permissions	Définit les permissions pour les actions liées au workflows de données associés au modèle de workflow. Ce bean doit être déclaré dans un fichier <code>module.xml</code>. Voir l'exemple [p 653].
Permissions programmatiques	Définit le composant gérant les permissions du workflow. S'il est défini, il remplace l'ensemble des permissions définies dans la propriété 'Permissions'.

28.3 Contexte de données

La configuration du contexte de données est accessible depuis le panneau de navigation.

Chaque workflow possède un contexte de données qui lui est propre, lui permettant ainsi de disposer d'un espace de données local durant son exécution. Cela permet de stocker et faire varier des valeurs qui orienteront l'exécution du workflow.

Le contexte de données est défini par une liste de variables. Chaque variable possède les propriétés suivantes :

Nom	Identifiant de la variable.
Valeur par défaut	Si définie, la variable sera initialisée avec cette valeur par défaut.
Paramètre d'entrée	'Oui' doit être coché pour définir cette variable comme un paramètre d'entrée.
Paramètre de sortie	'Oui' doit être coché pour définir cette variable comme un paramètre de sortie. Sinon, la variable ne sera pas affichée dans la liste des paramètres de sortie, dans l'interface de définition d'une tâche.

28.4 Personnalisation des vues d'exécution de workflow

La personnalisation des vues d'exécution de workflow est accessible depuis le panneau de navigation. Cette personnalisation permet de configurer les colonnes spécifiques des vues de bons de travail et de workflows (corbeille, monitoring des bons de travail, monitoring des workflows actifs et workflows terminés). Pour chaque colonne spécifique, il est possible d'associer une expression qui peut contenir des variables du contexte de données qui seront évaluées lors de l'affichage du workflow.

28.5 Permissions sur les workflows de données associés

Administration de workflow	Définit le profil autorisé à exécuter des tâches d'administration sur les workflows. Les actions d'administration sont : rejouer une étape, réveiller un workflow, terminer un workflow, désactiver une publication et dépublier. Pour pouvoir exécuter ces actions, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". L'administrateur built-in est toujours autorisé à administrer les workflows.
Administration de workflow > Rejouer une étape	Définit le profil autorisé à rejouer une étape de workflow. Pour pouvoir exécuter cette action, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". Pour rejouer une étape, un bouton est disponible dans la section "Monitoring > Workflows actifs". Un profil qui a la permission "Administration de workflow" est automatiquement autorisé à effectuer cette action spécifique. L'administrateur built-in est toujours autorisé à rejouer une étape de workflow.
Administration de workflow > Terminer un workflow	Définit le profil autorisé à terminer et supprimer un workflow. Pour pouvoir exécuter cette action, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". Pour terminer et supprimer un workflow en cours, un bouton est disponible dans la section "Monitoring > Workflows actifs". Pour supprimer un workflow terminé, un bouton est disponible dans la section "Workflows terminés". Un profil qui a la permission "Administration de workflow" est automatiquement autorisé à effectuer cette action spécifique. L'administrateur built-in est toujours autorisé à terminer un workflow.
Administration de workflow > Forcer le réveil d'un workflow	Définit le profil autorisé à forcer le réveil d'un workflow en attente. Pour pouvoir exécuter cette action, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". Pour réveiller un workflow, un bouton est disponible dans la section "Monitoring > Workflows actifs". Un profil qui a la permission "Administration de workflow" est automatiquement autorisé à effectuer cette action spécifique. L'administrateur built-in est toujours autorisé à réveiller un workflow.
Administration de workflow > Désactiver une publication	Définit le profil autorisé à désactiver une publication de workflow. Pour pouvoir exécuter cette action, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". Pour désactiver, un bouton est disponible

dans la section "Monitoring > Publications" uniquement pour les publications actives. Un profil qui a la permission "Administration de workflow" est automatiquement autorisé à effectuer cette action spécifique. L'administrateur built-in est toujours autorisé à désactiver une publication.

Administration de workflow > Dépublier

Définit le profil autorisé à dépublier une publication de workflow. Pour pouvoir exécuter cette action, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". Pour dépublier, un bouton est disponible dans la section "Monitoring > Publications" pour les publications désactivées uniquement. Un profil qui a la permission "Administration de workflow" est automatiquement autorisé à effectuer cette action spécifique. L'administrateur built-in est toujours autorisé à dépublier.

Gestion de l'allocation

Définit le profil autorisé à gérer l'allocation des bons de travail. Les actions d'allocation sont : allouer les bons de travail, réallouer les bons de travail et désallouer les bons de travail. Pour pouvoir exécuter ces actions, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". L'administrateur built-in est toujours autorisé à gérer l'allocation des workflows.

Gestion de l'allocation > Allouer les bons de travail

Définit le profil autorisé à allouer les bons de travail. Pour pouvoir exécuter cette action, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". Pour allouer un bon de travail, un bouton est disponible dans la section "Monitoring > Bons de travail" uniquement pour les bons de travail proposés. Un profil qui a la permission "Gestion de l'allocation" est automatiquement autorisé à effectuer cette action spécifique. L'administrateur built-in est toujours autorisé à allouer les bons de travail.

Gestion de l'allocation > Réallouer les bons de travail

Définit le profil autorisé à réallouer les bons de travail. Pour pouvoir exécuter cette action, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". Pour réallouer, un bouton est disponible dans la section "Monitoring > Bons de travail" uniquement pour les bons de travail alloués. Un profil qui a la permission "Gestion de l'allocation" est automatiquement autorisé à effectuer cette action spécifique. L'administrateur built-in est toujours autorisé à réallouer les bons de travail.

Gestion de l'allocation > Désallouer les bons de travail

Définit le profil autorisé à désallouer les bons de travail. Pour pouvoir exécuter cette action, ce profil bénéficie automatiquement de la permission "Visualiser les workflows". Pour désallouer, un bouton est disponible dans

la section "Monitoring > Bons de travail" uniquement pour les bons de travail alloués. Un profil qui a la permission "Gestion de l'allocation" est automatiquement autorisé à effectuer cette action spécifique. L'administrateur built-in est toujours autorisé à désallouer les bons de travail.

Lancer les workflows	Définit le profil autorisé à lancer manuellement de nouveaux workflows. Cette permission permet de lancer des workflows à partir des publications actives dans la section "Lanceurs de workflow". L'administrateur built-in est toujours autorisé à lancer les workflows.
Visualiser les workflows	Définit le profil autorisé à visualiser les workflows. Par défaut, un utilisateur final peut uniquement voir les bons de travail qui lui sont proposés ou alloués dans la section "Boîte de réception". Cette permission permet en plus de visualiser les publications, workflows et bons de travail associés à ce modèle de workflow dans les sections "Monitoring" et "Workflows terminés". Ce profil bénéficie automatiquement de la permission "Visualiser les workflows terminés". L'administrateur built-in est toujours autorisé à visualiser les workflows.
Visualiser les workflows > Le créateur d'un workflow peut le visualiser	Si activé, le créateur d'un workflow a la permission de visualiser les workflows qu'il a lancés. Cette permission restreinte lui donne accès aux workflows qu'il a lancés et aux bons de travail associés dans les sections "Monitoring > Workflows actifs", "Monitoring > Bons de travail" et "Workflows terminés". La valeur par défaut est "Non".
Visualiser les workflows > Visualiser les workflows terminés	Définit le profil autorisé à visualiser les workflows terminés. Cette permission permet de visualiser les workflows terminés dans la section "Workflows terminés" et d'accéder à leur historique. Un profil qui a la permission "Visualiser les workflows" est automatiquement autorisé à effectuer cette action. L'administrateur built-in est toujours autorisé à visualiser les workflows terminés.

Note

Un utilisateur n'ayant aucun privilège particulier pourra voir un bon de travail uniquement si celui-ci lui est proposé ou personnellement alloué.

Voir aussi [Administration d'un workflow](#) [p 213]

28.6 Historique des images du modèle de workflow

L'historique des images d'un modèle de workflow peut être géré sous **Actions > Afficher l'historique des publications**.

La table d'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**.

28.7 Suppression d'un modèle de workflow

Un modèle de workflow peut être supprimé. Cependant, toute version publiée auparavant reste accessible dans la section Workflows de Données. D'autre part, si un modèle de workflow est recréé avec un nom identique, lors d'une publication, un message demandera la confirmation de remplacement de la publication précédente.

Voir aussi [Publication d'un modèle de workflow](#) [p 193]

CHAPITRE 29

Publication d'un modèle de workflow

Ce chapitre contient les sections suivantes :

1. [A propos des publications de workflow](#)
2. [Publication et images de modèle de données](#)
3. [Sous-workflows dans les publications](#)

29.1 A propos des publications de workflow

Dès qu'un modèle de workflow est défini, il doit être publié afin d'autoriser les utilisateurs à lancer des workflows de données associés. Une publication est réalisée en cliquant sur le bouton 'Publier' dans le panneau de navigation.

Si aucune étape d'appel à des sous-workflows n'est incluse dans le modèle courant, l'option de publier d'autres modèles en même temps vous sera proposée sur la page de publication. Si le modèle de workflow actuel contient des étapes d'appel à des sous-workflows, il doit être publié seul.

Les modèles de workflow peuvent être publiés plusieurs fois. Une publication est identifiée par son nom de publication.

29.2 Publication et images de modèle de données

Durant la publication d'un modèle de workflow, une image est enregistrée de son état actuel. Vous pouvez préciser un libellé et une description de l'image. Le libellé par défaut pour l'image est l'heure et la date au moment de publication. La description par défaut indique l'utilisateur qui a publié le modèle de workflow.

Pour chaque modèle de workflow publié, le nom de la publication doit être unique. Si un modèle de workflow a déjà été publié, il est possible de mettre à jour une publication existante en réutilisant le même nom de publication. Les noms des publications de workflow existantes sont proposés dans un menu. Dans le cas d'une mise à jour de publication, l'ancienne version ne sera plus disponible pour lancer des workflows de données ; mais elle permettra aux workflows existants de finir de s'exécuter. Le contenu des différentes images peut être consulté dans l'historique des images de modèle de workflow.

Voir aussi [Historique des images du modèle de workflow](#) [p 191]

29.3 Sous-workflows dans les publications

Quand un modèle de workflow, contenant un appel à des sous-workflows, est publié, il n'est pas nécessaire de publier séparément les sous-workflows appelés. D'un point de vue d'administration, le modèle du workflow principal (celui actuellement publié par l'utilisateur) et les modèles de sous-workflows sont publiés comme une entité unique.

Le système établit les dépendances avec les modèles de workflows utilisés comme sous-workflows et crée automatiquement une publication par modèle dépendant. Ces publications techniques sont exclusivement dédiées au moteur de workflow pour le lancement des sous-workflows et ne sont donc pas disponibles dans la section Workflows de données.

La publication multiple n'est pas disponible pour un modèle de workflow contenant un appel à des sous-workflows. C'est pourquoi la première étape de publication (sélection des modèles de workflow à publier) n'est pas proposée dans ce cas.

La republication du modèle de workflow principal met automatiquement à jour les modèles de sous-workflows appelés.

Bien qu'un sous-workflow puisse être publié séparément comme modèle de workflow principal, cela ne modifiera pas la version utilisée par un autre modèle de workflow principal publié qui utilise ce sous-workflow.

Workflows de données

CHAPITRE 30

Introduction aux workflows de données

Ce chapitre contient les sections suivantes :

1. [Présentation](#)

30.1 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](#) [p 205]

[Lancement et monitoring de workflows de données](#) [p 211]

[Administration de workflows de données](#) [p 213]

[Permissions sur les workflows de données associés](#) [p 189]

Concepts apparentés [*Modèles de workflow*](#) [p 166]

CHAPITRE 31

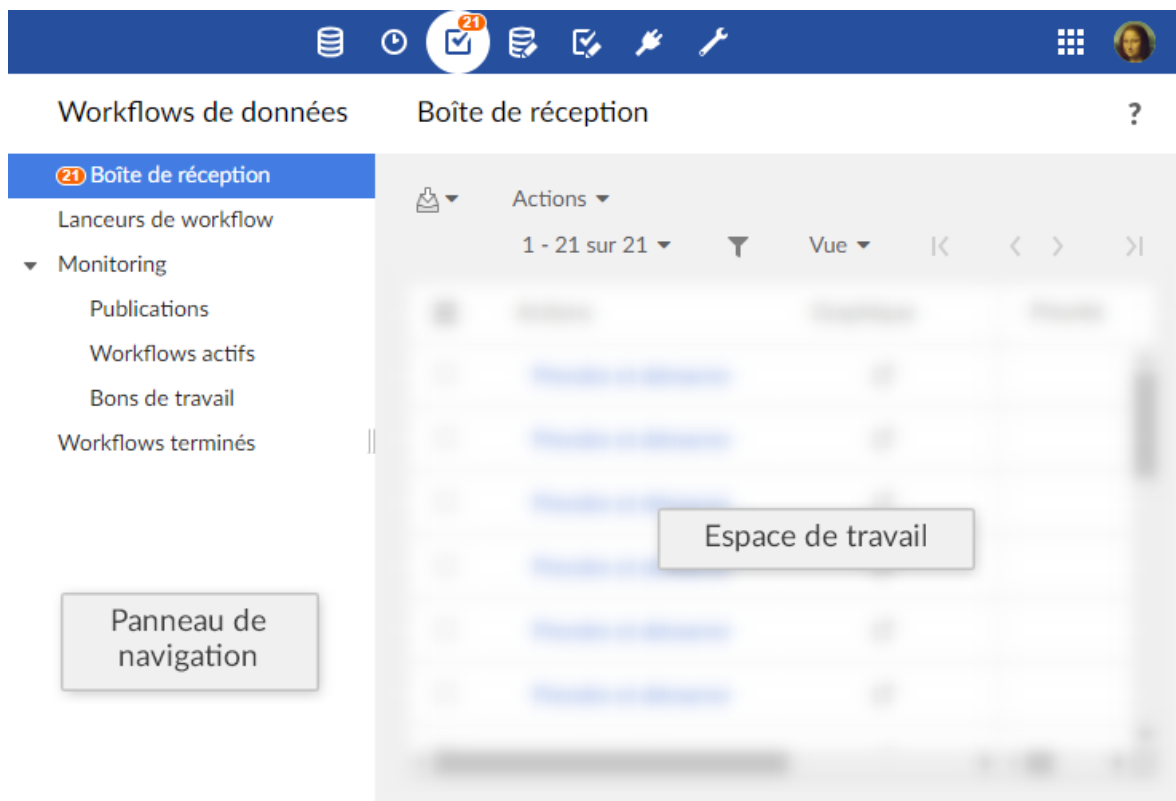
Utilisation de l'interface utilisateur de la section Workflow de données

Ce chapitre contient les sections suivantes :

1. [Navigation dans l'interface utilisateur](#)
2. [Règles de navigation](#)
3. [Vues personnalisées](#)
4. [Colonnes spécifiques](#)
5. [Filtrage d'items dans les vues](#)
6. [Vue graphique d'un workflow de données](#)

31.1 Navigation dans l'interface utilisateur

La fonctionnalité workflows de données se trouve dans la section **Workflows de données** dans l'interface utilisateur de TIBCO EBX®.



Note

Seuls les utilisateurs autorisés peuvent accéder à cet écran via la 'Perspective avancée' ou via une perspective spécifique. Seuls les utilisateurs autorisés peuvent accéder à ces interfaces spécifiques.

Le panneau de navigation est composé de plusieurs items. Ces items sont visibles en fonction des permissions globales associées. Les différents items sont :

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 travail aux utilisateurs ou rôles.
Workflows terminés	Workflows de données, dont l'exécution est terminée, 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.

Note

Chaque section est accessible par composant web, par exemple pour intégration avec un portail, ou programmatiquement par la classe Servicekey dans l'API Java.

Voir aussi

[Using TIBCO EBX® as a Web Component](#) [p 229]

ServiceKey^{API}

31.2 Règles de navigation

Corbeille de bons de travail

Par défaut, une fois qu'un bon de travail est exécuté, la corbeille de bons de travail est affichée.

Ce comportement peut être modifié en fonction de la stratégie d'avancement de l'étape suivante. Cette stratégie permet d'enchaîner plusieurs étapes à la suite sans repasser par la corbeille de tâches.

Voir [la stratégie d'avancement d'une étape de workflow](#) [p 172] dans la modélisation de workflow.

Lanceurs de workflow

Par défaut, une fois qu'un workflow est lancé, la table des lanceurs de workflow est affichée.

Ce comportement est modifiable dans la configuration du modèle : il est possible d'ouvrir directement la première étape sans afficher la table des lanceurs de workflow.

Voir [l'ouverture automatique de la première étape d'un workflow](#) [p 187] dans la modélisation de workflow.

31.3 Vues personnalisées

Il est possible de définir des vues sur les différentes tables du workflow et de bénéficier de tous les mécanismes associés (dont la publication).

Les permissions pour créer et gérer les vues des tables de workflows sont les mêmes que les permissions des vues de tables de données. Il peut s'avérer nécessaire de modifier les permissions dans la section 'Administration' pour bénéficier de cette fonctionnalité, en sélectionnant *Gestion des workflows* > *Workflows*.

Voir les [Vues](#) [p 131] pour plus d'informations.

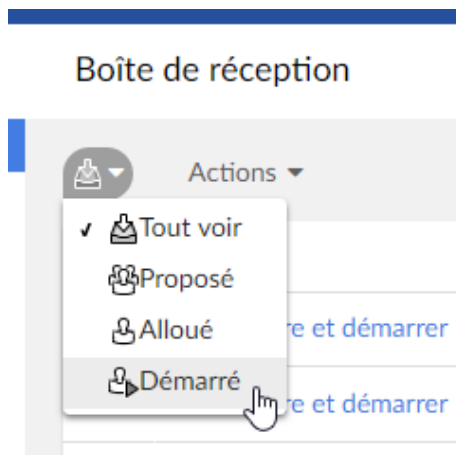
31.4 Colonnes spécifiques

Par défaut, les colonnes spécifiques sont masquées dans les vues qui peuvent en bénéficier (corbeille, monitoring des bons de travail, monitoring des workflows actifs et workflows terminés).

Il faut créer une vue personnalisée et l'appliquer pour afficher les colonnes spécifiques souhaitées. Pour chaque bon de travail ou workflow, la correspondance définie dans le modèle de workflow associé est alors appliquée. Si une expression est définie pour une colonne et contient des variables du contexte de données, ces variables sont évaluées lors de son affichage. Si une expression contient des expressions natives qui dépendent de la locale, l'expression est évaluée dans la locale par défaut.

31.5 Filtrage d'items dans les vues

Dans certaines vues, comme la '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.



31.6 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. Pour cela, cliquer 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.

Si des étapes ont été définies comme masquées dans la modélisation de workflow, elles sont automatiquement masquées dans la vue graphique de workflow pour les utilisateurs non-administrateurs (non-administrateurs built-in et non-administrateurs de workflow). Un bouton est disponible pour afficher les étapes masquées. Le choix des utilisateurs (montrer ou masquer les étapes) est sauvegardé par utilisateur, par publication, pendant la session utilisateur.

Pour les tâches utilisateur dans le nouveau mode (un seul bon de travail), les informations principales à propos du bon de travail unique sont directement affichées dans la vue graphique de workflow, si elles sont disponibles : l'avatar de l'utilisateur associé au bon de travail et la décision qui a été prise pour le bon de travail (accepté ou rejeté).

CHAPITRE 32

Bons de travail

Ce chapitre contient les sections suivantes :

1. [Présentation des bons de travail](#)
2. [Travail sur un bon de travail en tant que participant](#)
3. [Priorité de bons de travail](#)

32.1 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, lancés depuis les publications du modèle, génèrent un bon de travail individuel pour chacun des participants listés dans la tâche utilisateur.

Voir aussi [Overview](#) [p 661]

États d'un bon de travail

Lorsqu'un bon de travail est émis, pendant l'exécution d'un workflow de données, pour une tâche utilisateur définie dans le modèle, le bon de travail peut prendre plusieurs états : proposé, alloué, démarré, et terminé.

Création des bons de travail

Mode par défaut

Dans le mode par défaut, un seul bon de travail est généré quelle que soit la liste de profils définis.

Par défaut, si un seul utilisateur est défini dans la liste des profils, le bon de travail créé est à l'état *alloué*.

Par défaut, dans les autres cas, le bon de travail créé est à l'état *proposé*.

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 profils de la tâche utilisateur.

Mode de compatibilité

Par défaut, pour chaque utilisateur défini comme participant de la tâche utilisateur, le workflow de données crée un bon de travail à l'état *alloué*.

Par défaut, pour chaque rôle défini comme participant de la tâche utilisateur, le workflow de données crée un bon de travail à l'état *proposé*.

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.

Variations des états des bons de travail

Lorsque le bon de travail est à l'état *alloué*, l'utilisateur défini peut directement commencer à travailler sur le bon de travail alloué en effectuant l'action 'Prendre et démarrer'. Le bon de travail passe alors à l'état *démarré*.

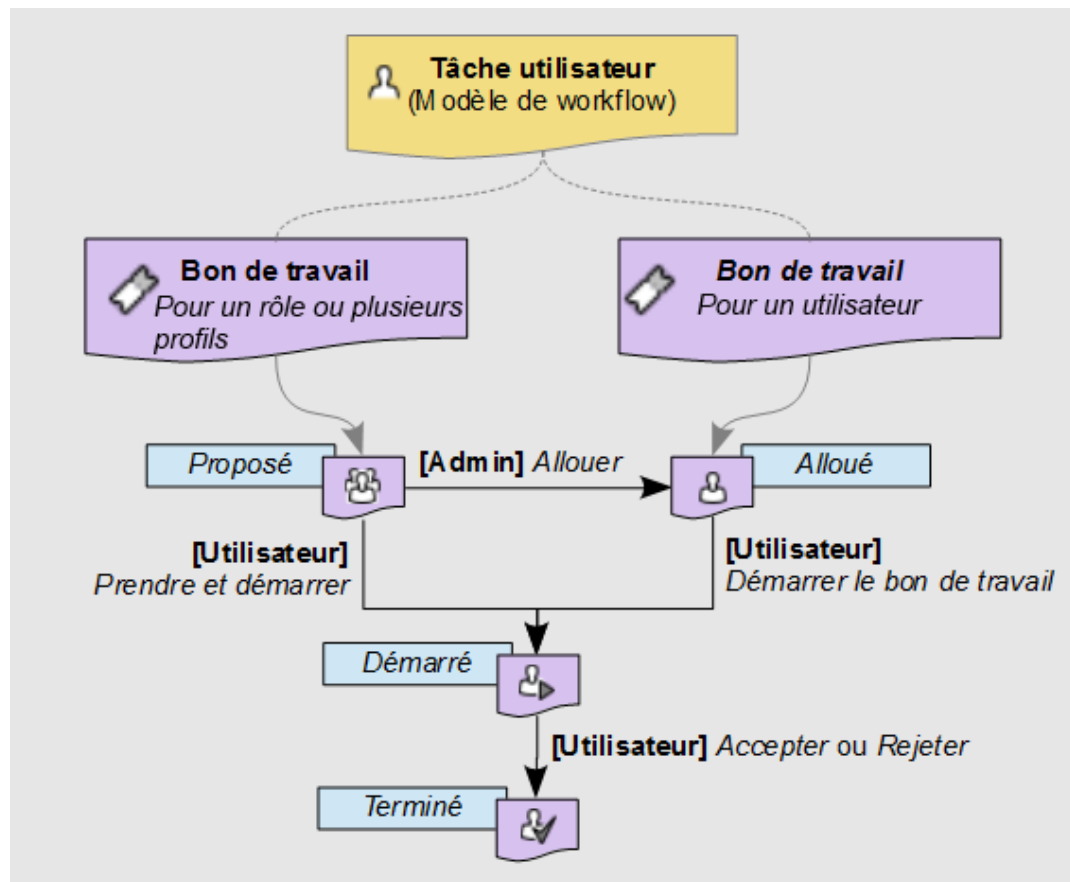
Lorsque le bon de travail est à l'état *proposé*, tout utilisateur ou tout membre des rôles auxquels il est proposé peut prendre le bon de travail en utilisant l'action 'Prendre et démarrer'. Le bon de travail passe ainsi à l'état *démarré*.

Avant qu'un utilisateur ait pris le bon de travail proposé, un gestionnaire des allocations du workflow de données peut intervenir pour affecter manuellement le bon de travail à un utilisateur spécifique, passant ainsi le bon de travail à 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*.

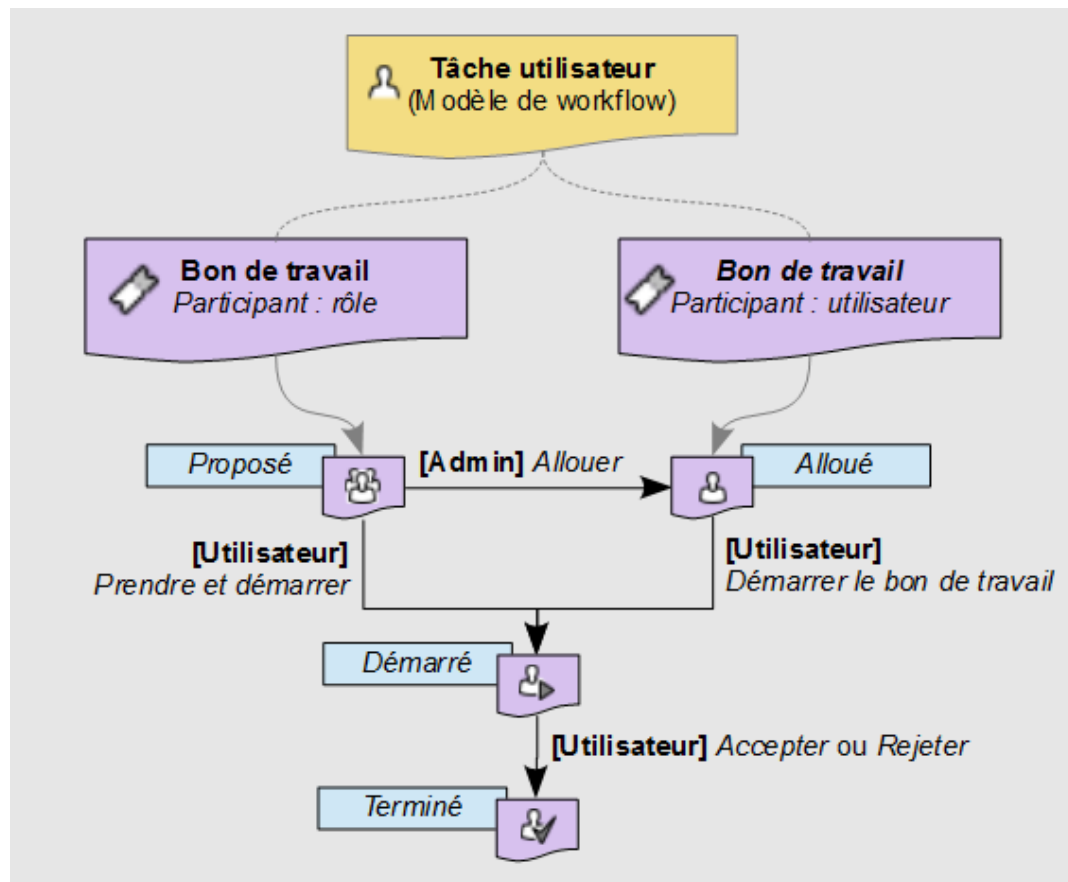
Une fois que l'utilisateur, qui a démarré le bon de travail, a réalisé l'action demandée, l'action terminale 'Accepter' ou 'Rejeter' place le bon de travail dans l'état *terminé*. Lorsqu'un utilisateur termine un bon de travail, le workflow de données passe automatiquement à l'étape suivante définie dans le modèle de workflow.

Diagramme des états de bon de travail

Mode par défaut



Mode de compatibilité



32.2 Travail sur un bon de travail en tant que participant

Tous les bons de travail disponibles (qui vous sont soit proposés, soit alloués), 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 administrateurs, les superviseurs du workflow et les autres participants au workflow de données. 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 continue à partir du même point.

32.3 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 TIBCO EBX®.

Voir aussi [tâche utilisateur \(glossaire\)](#) [p 32]

Concepts apparentés [Tâche utilisateur](#) [p 173]

CHAPITRE 33

Lancement et monitoring de workflows de données

Ce chapitre contient les sections suivantes :

1. [Lancement d'un workflow de données](#)
2. [Activités de monitoring](#)
3. [Gestion de l'allocation des bons de travail](#)

33.1 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 en utilisant la vue 'Lanceurs de workflow' dans le panneau de navigation. Pour créer un nouveau workflow de données, depuis une publication de modèle de workflow, cliquer 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.

33.2 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.

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

33.3 Gestion de l'allocation des bons de travail

Lorsque vous êtes autorisé à gérer l'allocation des bons de travail, vous pouvez allouer manuellement des bons de travail durant l'exécution des workflows associés au modèle de workflow. Dans ce cas, vous pouvez effectuer une ou plusieurs des actions ci-dessous sur les bons de travail.

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> .
Réallouer	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](#) [p 205]

[Permissions sur les workflows de données associés](#) [p 189]

Concepts apparentés [Modèles de workflow](#) [p 166]

CHAPITRE 34

Administration de workflows de données

Si vous disposez de permissions pour administrer des workflows de données, les vues 'Publications', 'Workflows actifs', et 'Bons de travail' associés seront accessibles sous le menu 'Monitoring' du panneau de navigation. Dans ces vues, sous les menus 'Actions' sur les lignes des tables, vous pourrez accéder aux 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.

Ce chapitre contient les sections suivantes :

1. [Présentation de l'exécution de workflow de données](#)
2. [Actions d'administration de workflow de données](#)

34.1 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 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.
- invocation de sous-workflows qui lance les sous-workflows associés et attend que les sous-workflows en cours soient terminés.
- tâche d'attente qui met en pause le workflow jusqu'à ce qu'un événement spécifique soit reçu.

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.
- **En attente de sous-workflows** : Le jeton est positionné sur une invocation de sous-workflows et attend la terminaison de tous les sous-workflows lancés.
- **En attente d'événement** : Le jeton est positionné sur une tâche d'attente et attend de recevoir un événement donné.
- **Terminé** : Le jeton a atteint la fin du workflow de données.
- **Erreur** : Une erreur est survenue.

Voir aussi [Workflow management](#) [p 461]

34.2 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.

Dépuration 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. Pour cela, suivez le processus décrit dans la section [Désactivation d'une publication de workflow](#) [p 214].

2. Dépublier la publication de workflow en sélectionnant *Actions > Dépublier* de la ligne de la publication cible.

Note

A la dépublication d'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 tout bon de travail associé. Toute perte de données résultant d'une fin prématurée est alors définitive.

Actions sur workflows de données

Dans les vues tabulaires des workflows de données, chaque enregistrement porte un menu *Actions* qui permet d'exécuter des services sur un workflow 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 et sous-workflows liés, et le jeton est repositionné au début de l'étape courante.

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

Terminer un workflow de données actif et le purger

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, tous les bons de travail et sous-workflows associés.

Note

Cette action n'est pas disponible pour les workflows dans l'état 'En cours d'exécution' et pour les sous-workflows lancés par d'autres workflows.

Note

Les historiques du workflow ne sont pas supprimés.

Forcer la terminaison d'un workflow de données actif

Pour forcer la terminaison d'un workflow de données en cours d'exécution, sélectionnez *Actions > Forcer la terminaison* dans la ligne du workflow cible dans la table 'Workflows actifs'. L'action

stoppe l'exécution du workflow de données et supprime les éventuels bons de travail et sous-workflows associés.

Note

Cette action est disponible pour les sous-workflows, et pour les workflows en erreur bloqués sur la dernière étape.

Note

Les historiques du workflow ne sont pas supprimés.

Forcer le réveil d'un workflow en attente

Pour réveiller un workflow qui est en attente d'événement, sélectionner **Actions > Forcer le réveil** à partir du workflow dans la table 'Workflows actifs'. Cela entraîne le réveil du workflow. Avant d'effectuer cette action, l'administrateur doit mettre à jour le contexte de données afin de s'assurer que le workflow peut exécuter les tâches suivantes.

Note

Cette action est disponible uniquement pour les workflows qui sont à l'état 'en attente d'événement'.

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, vous pouvez effectuer un nettoyage en sélectionnant *Actions > Purger* dans la ligne du workflow cible de la table 'Workflows terminés'.

Un workflow purgé n'est plus visible dans la vue 'Workflows terminés'. Cependant, son historique reste consultable dans la zone d'administration technique.

Note

Cette action n'est pas disponible pour les sous-workflows lancés par d'autres workflows.

Voir aussi [Workflow management](#) [p 461]

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 [Permissions sur les workflows de données associés](#) [p 189]

Services de données

CHAPITRE 35

Introduction aux services de données

Ce chapitre contient les sections suivantes :

1. [Présentation](#)
2. [Utilisation de l'interface utilisateur de la section Services de données](#)

35.1 Présentation

Fonction du service de données

Un [service de données](#) [p 33] est :

- un web service standard qui permet d'interagir avec TIBCO EBX®.
Les services de données SOAP peuvent être générés dynamiquement à partir d'un modèle de données dans la section 'Services de données'.
- un service REST qui permet d'interroger le contenu du référentiel EBX®.
Un service RESTful prédéfini ne nécessite pas d'interface de service, il est auto-descriptif par l'intermédiaire des méta-données retournées.

Ils peuvent être utilisés pour accéder à une partie des fonctionnalités disponibles par l'interface utilisateur.

Voir aussi

[WSDL/SOAP](#) [p 694]

[REST](#) [p 748]

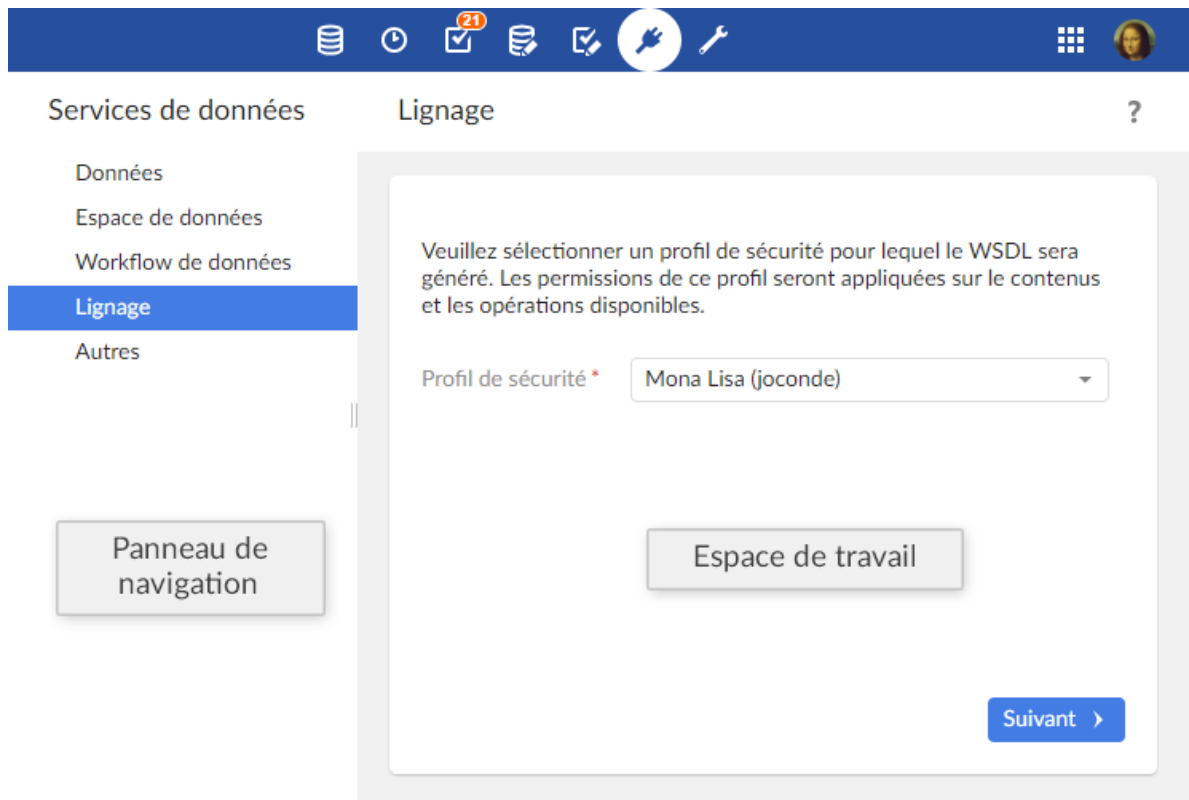
Lignage

Le [lignage](#) [p 34] établit des profils de droit d'accès utilisés par les services de données. Quand les services de données accèdent aux interfaces WSDL, ils utilisent les profils de droit d'accès définis par ce mécanisme.

Glossaire

Voir aussi [Services de données](#) [p 33]

35.2 Utilisation de l'interface utilisateur de la section Services de données



Note

Seuls les utilisateurs autorisés peuvent accéder à cette section via la 'Perspective avancée'.

Concepts apparentés

[Espaces de données](#) [p 100]

[Jeux de données](#) [p 122]

[Workflows de données](#) [p 196]

[Introduction to data services](#) [p 694]

CHAPITRE 36

Génération de WSDL pour services de données

Ce chapitre contient les sections suivantes :

1. [Générer un WSDL pour accéder aux données](#)
2. [Générer un WSDL pour accéder à un espace de données](#)
3. [Générer un WSDL pour contrôler un workflow de données](#)
4. [Générer un WSDL pour un lignage](#)
5. [Générer un WSDL pour l'administration](#)
6. [Générer un WSDL pour modifier l'annuaire par défaut](#)

36.1 Générer un WSDL pour accéder aux données

La génération de WSDL pour l'accès aux données est disponible en sélectionnant 'Données' dans la section 'Services de données'.

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

1. Sélectionner si le WSDL sera utilisé pour des opérations sur un jeu de données ou sur une table.
2. Identifier l'espace de données et le jeu de données ciblés par les opérations.
3. Sélectionner les tables sur lesquelles les opérations sont autorisées, ainsi que les opérations permises.
4. Télécharger le fichier WSDL généré en cliquant sur le bouton 'Télécharger le WSDL'.

Opérations disponibles sur un jeu de données

Les opérations suivantes sur les jeux de données sont disponibles en utilisant le WSDL généré :

- Sélectionner les données d'un jeu de données pour un espace de données ou une image.
- Récupérer les changements du jeu de données entre espaces de données ou images
- Actualiser une unité de réplication en base de données

Voir aussi

[WSDL download from HTTP protocol](#) [p 707]

[*Operations generated from a data model*](#) [p 713]

Opérations disponibles sur une table d'un jeu de données

Si sélectionnées, les opérations suivantes sur les tables sont disponibles en utilisant le WSDL généré :

- 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
- Obtenir les droits d'accès

Voir aussi

[*WSDL download from HTTP protocol*](#) [p 707]

[*Operations generated from a data model*](#) [p 713]

36.2 Générer un WSDL pour accéder à un espace de données

La génération de WSDL pour la manipulation d'un espace de données est accessible en sélectionnant '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 le WSDL**.

Opérations disponibles sur un espace de données

Les opérations suivantes sur les espaces de données sont disponibles en utilisant le WSDL généré :

- Créer un espace de données
- Créer une image
- Fermer un espace de données
- Fermer une image
- Fusionner un espace de données
- Valider un espace de données ou une image
- Valider un jeu de données
- Verrouiller un espace de données
- Déverrouiller un espace de données

Voir aussi

[*WSDL download from HTTP protocol*](#) [p 707]

[*Operations on datasets and dataspace*](#) [p 734]

36.3 Générer un WSDL pour contrôler un workflow de données

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

Opérations disponibles pour contrôler un workflow de données

Les opérations suivantes sur les espaces de données sont disponibles en utilisant le WSDL généré :

- Démarrer un workflow
- Réveiller un workflow
- Terminer un workflow

Voir aussi

[WSDL download from HTTP protocol](#) [p 707]

[Operations on data workflows](#) [p 740]

36.4 Générer un WSDL pour un lignage

La génération d'un WSDL pour un lignage est accessible en sélectionnant 'Lignage' dans la section 'Services de données', sous réserve que des profils aient été autorisés par un profil administrateur dans *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](#) [p 221].

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

1. Sélectionner un rôle ou un utilisateur, dont les permissions seront appliquées. Un rôle ou un utilisateur doit être autorisé à être utilisé pour le lignage par un administrateur.
2. Identifier l'espace de données et le jeu de données ciblés par les opérations.
3. Sélectionner les tables sur lesquelles les opérations sont autorisées, ainsi que les opérations permises.
4. Télécharger le fichier WSDL généré en cliquant sur le bouton **Télécharger le WSDL**.

Voir aussi [Lignage](#) [p 218]

36.5 Générer un WSDL pour l'administration

Cette action ne peut être effectuée que par un administrateur.

La génération d'un WSDL pour :

- modifier l'accès à l'interface utilisateur
- récupérer les informations système

est disponible en sélectionnant 'Administration' dans la section 'Services de données'.

Opérations disponibles pour l'administration

- Fermer l'interface utilisateur
- Ouvrir l'interface utilisateur
- Obtenir les informations système

Voir aussi

[*WSDL download from HTTP protocol*](#) [p 707]

[*User interface operations*](#) [p 744]

[*System information operation*](#) [p 744]

36.6 Générer un WSDL pour modifier l'annuaire par défaut

Cette action ne peut être effectuée que par un administrateur et seulement si l'annuaire par défaut est utilisé.

La génération d'un WSDL pour modifier l'annuaire d'accès est accessible en sélectionnant 'Annuaire' dans la section 'Services de données'.

Opérations disponibles pour modifier l'annuaire par défaut

Les WSDL générés pour accéder aux tables sont les mêmes que ceux utilisés pour la [*génération d'un WSDL d'accès aux données*](#) [p 221].

Voir aussi

[*WSDL download from HTTP protocol*](#) [p 707]

[*Directory services*](#) [p 743]

Manuel de référence (en anglais)

Intégration

CHAPITRE 37

Overview of integration and extension

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

Ce chapitre contient les sections suivantes :

1. [User interface customization and integration](#)
2. [Data services](#)
3. [XML and CSV import/export services](#)
4. [Programmatic services](#)

37.1 User interface customization and integration

The EBX® graphical interface can be customized through various EBX® APIs.

It can also be integrated into any application that is accessible through a [supported web browser](#) [p 332].

See [Interface customization](#) [p 658] for more information.

37.2 Data services

The data services module provides a means for external systems to interact with EBX® using one of following:

- Web Services Description Language (WSDL/SOAP) standard
- Representational state transfer (REST)

Voir aussi

[WSDL/SOAP data services](#) [p 694]

[REST data services](#) [p 748]

37.3 XML and CSV import/export services

EBX® 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

[*Import et export XML*](#) [p 143]

[*Import et export CSV*](#) [p 149]

37.4 Programmatic services

Programmatic services allow executing procedures in a well-defined context, for example in a scheduled task or in a batch.

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 EBX® **built-in optimization services** `AdaptationTreeOptimizerSpecAPI` are not sufficient.

Voir aussi `ProgrammaticServiceAPI`

CHAPITRE 38

Using TIBCO EBX® as a Web Component

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Integrating EBX® Web Components into applications](#)
3. [Repository element and scope selection](#)
4. [Combined selection](#)
5. [Request specifications](#)
6. [Example calls to an EBX® Web Component](#)

38.1 Overview

EBX® can be used as a user interface Web Component, called through the HTTP protocol. An EBX® 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 EBX®, 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 EBX® 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](#) [p 332]

38.2 Integrating EBX® Web Components into applications

A web application that calls an EBX® 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 EBX® Web Component [request specifications](#) [p 231].
2. A Java application, for example:
 - A Java web application running on the same application server instance as the EBX® repository it targets or on a different application server instance.

- An EBX® [User service](#) [p 658] or a [Custom widget](#) [p 659], in which case, the new session will automatically inherit from the parent EBX® session.

Note

In Java, the recommended method for building HTTP requests that call EBX® web components is to use the class `UIHttpManagerComponentAPI` in the API.

38.3 Repository element and scope selection

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

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

The repository elements that can be selected are as follows:

- Dataspace or snapshot
- Dataset
- Node
- Table or a published view
- Table record

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

Voir aussi [Scope](#) [p 234]

Voir aussi [firstCallDisplay](#) [p 234]

It is also possible to select a specific perspective as well as a perspective action.

By default, the selection of the element is done in the context of the perspective of the user if the scope is "full".

Voir aussi [Perspective](#) [p 19]

38.4 Combined selection

A URL of a Web component can specify a perspective and an action or an entity (dataspace, dataset, etc). Thus, for a Web component that has specified in its URL a perspective and an entity (but no action), if an action of the perspective matches this entity, then this action will be automatically selected.

Otherwise, if no action matches this entity, no action will be selected but the entity is opened regardless.

If an action is specified at the same time than an entity, this last is ignored and the action will be selected.

Specific case

If the target entity is a record and if an action is on the table that contains this record, then this action will be selected and the record will be opened inside the action.

In the same way, if a workflow work item is targeted by the web component, and if an action on « inbox » exists in the perspective, then this action will be selected and the work item will be opened inside it.

Known limitations

If the Web component specifies a predicate to filter a table, the perspective action must specify the exact same predicate to be selected.

In the same way, if the perspective action specifies a predicate to filter a table, the Web component must specify the exact same predicate to establish the match.

38.5 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 through the repository login page. See <code>Directory^{API}</code> 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 <code>AccessRule^{API}</code> 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> . For more information, see Exit policy [p 427].	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 dataspace.	No
version	Selects the specified snapshot.	No
instance	Selects the specified dataset. The value must be the reference of a dataset that exists in the selected dataspace or snapshot.	Only if xpath or viewPublication is specified.
viewPublication	Specifies the publication name of the tabular or hierarchical view to apply to the selected content. This publication name is the one declared during the publication of the view. It can be found in the 'Administration' area under <i>Views configuration > Views</i>. All settings of the view, that is, its filters, sort order, and displayed columns, are applied to the result. A dataspace and a dataset must be selected in order for this view to be applied. The target table selection is not necessary, as it can be automatically determined based on the definition of the view. This parameter can be combined with the predicate specified in the xpath parameter as a logical 'AND' operation.	No
xpath	Specifies a node selection in the dataset. Value may be a valid absolute path located in the selected dataset. The notation must conform to a simplified XPath, with abbreviated syntax. It can also be a predicate surrounded by "[" and "]" (to be encoded using %5B and %5D respectively) if a table can be automatically selected using other Web Component parameters (for example, viewPublication or workflowView). For XPath syntax, see XPath supported syntax [p 251] See <code>UIHttpManagerComponent.setPredicate^{API}</code> for more information.	No
service	Specifies the service to access. For more information on built-in User services, see Built-in services [p 237]. In the Java API, see <code>ServiceKey^{API}</code> for more information.	No
workflowView	Specifies the workflow section to be selected. See <code>WorkflowView^{API}</code> for more information.	No.
perspectiveName	Specifies the name of the perspective to be selected. If this parameter is specified, the scope parameter can have only two values: full and data.	Only if perspectiveActionId or perspectiveActionName is specified.
perspectiveActionId	Deprecated. Please consider using perspectiveActionName instead. Specifies the identifier of the perspective action to be selected.	No.

Parameter	Description	Required
perspectiveActionName	Specifies the unique name of the perspective action to be selected.	No.

Layout parameters

Parameter	Description	Required
scope	Specifies the scope to be used by the web component. Value can be full, data, dataspace, dataset or node. See <code>UIHttpManagerComponent.Scope^{API}</code> for more information.	No, default will be computed to be the smallest possible according to the target selection.
firstCallDisplay	Specifies which display must be used instead of the one determined by the combination of selection and scope parameter. Possible values are: • auto: The display is automatically set according to the selection. • view: Forces the display of the tabular view or of the hierarchical view. • record: If the predicate has at least one record, forces the display of the first record in the list. For example, <code>firstCallDisplay=view</code> <code>firstCallDisplay=view:hierarchyExpanded</code> <code>firstCallDisplay=record</code> <code>firstCallDisplay=record:{predicate}</code> See <code>UIHttpManagerComponent.setFirstCallDisplay^{API}</code> for more information. See <code>UIHttpManagerComponent.setFirstCallDisplayHierarchyExpanded^{API}</code> for more information. See <code>UIHttpManagerComponent.setFirstCallDisplayRecord^{API}</code> for more information.	No, default will be computed according to the target selection.
closeButton	Specifies how to display the session close button. Value can be logout or cross. See <code>UIHttpManagerComponent.CloseButtonSpec^{API}</code> for more information.	No. If scope is not full, no close button will be displayed by default.
dataSetFeatures	Specifies which features to display in a UI service at the dataset level or a form outside of a table. These options pertain only to features in the workspace. It is recommended to use this property with the smallest scope possible, namely dataset or node. Syntax: <code><prefix> ":" <feature> [", " <feature>]*</code> where • <code><prefix></code> is hide or show, • <code><feature></code> is services, title, save, or revert. For example, <code>hide:title</code> <code>show:save, revert</code> See <code>UIHttpManagerComponent.DataSetFeatures^{API}</code> for more information.	No.
viewFeatures	Specifies which features to display in a tabular or a hierarchy view (at the table level).	No.

Parameter	Description	Required
	These options pertain only to features in the workspace. It is recommended to use this property with the smallest scope possible, namely dataset or node. Syntax: <pre><prefix> ":" <feature> [", " <feature>]*</pre> where • <prefix> is hide or show, • <feature> is create, views, selection, filters, services, refresh, title, or breadcrumb. For example, <pre>hide:title,selection</pre> <pre>show:service,title,breadcrumb</pre> See <code>UIHttpManagerComponent.ViewFeatures^{API}</code> for more information.	
recordFeatures	Specifies which features must be displayed in a form at the record level. These options pertain only to features in the workspace. It is recommended to use this property with the smallest scope possible, namely dataset or node. Syntax: <pre><prefix> ":" <feature> [", " <feature>]*</pre> where • <prefix> is hide or show, • <feature> is services, title, breadcrumb, save, saveAndClose, close, or revert. For example, <pre>hide:title</pre> <pre>show:save,saveAndClose,revert</pre> See <code>UIHttpManagerComponent.RecordFeatures^{API}</code> for more information.	No.
pageSize	Specifies the number of records that will be displayed per page in a table view (either tabular or hierarchical).	No.
startWorkItem	Specifies a work item must be automatically taken and started. Value can be true or false. See <code>ServiceKey.WORKFLOW^{API}</code> for more information.	No. Default value is false, where the target work item state remains unchanged.

38.6 Example calls to an EBX® Web Component

Minimal URI:

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

Logs in as the user 'admin' and selects the 'Reference' dataspace:

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

Selects the 'Reference' dataspace 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"]
```

Note

For clarity purposes, the above URLs are not encoded and this can make them incompatible with some application servers.

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 dataset 'instanceId' in the dataspace '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"]
```

Note

For clarity purposes, the above URLs are not encoded and this can make them incompatible with some application servers.

CHAPITRE 39

Built-in user services

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

- [when defining workflow model tasks](#) [p 173]
- [when defining perspective action menu items](#) [p 20]
- [as extended user services when used with service extensions](#) [p 685]
- [when using EBX® as a Web Component](#) [p 229]

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

Ce chapitre contient les sections suivantes :

1. [Access data \(default service\)](#)
2. [Create a new record](#)
3. [Duplicate a record](#)
4. [Export data to an XML file](#)
5. [Export data to a CSV file](#)
6. [Import data from an XML file](#)
7. [Import data from a CSV file](#)
8. [Access a dataspace](#)
9. [Validate a dataspace, a snapshot or a dataset](#)
10. [Merge a dataspace](#)
11. [Access the dataspace merge view](#)
12. [Compare contents](#)
13. [Data workflows](#)

39.1 Access data (default service)

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

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

Input parameters

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

39.2 Create a new record

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

Service name parameter: `service=@creation`

Input parameters

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

Output parameters

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

39.3 Duplicate a record

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

Service name parameter: `service=@duplicate`

Input parameters

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

Output parameters

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

39.4 Export data to an XML file

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

Service name parameter: service=@exportToXML

Input parameters

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

39.5 Export data to a CSV file

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

Service name parameter: `service=@exportToCSV`

Input parameters

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

39.6 Import data from an XML file

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

Service name parameter: service=@importFromXML

Input parameters

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

39.7 Import data from a CSV file

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

Service name parameter: service=@importFromCSV

Input parameters

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

39.8 Access a dataspace

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

Service name parameter: `service=@selectDataSpace`

Input parameters

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

39.9 Validate a dataspace, a snapshot or a dataset

Workflows automatically consider the validation service as complete.

Service name parameter: service=@validation

Input parameters

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

Output parameters

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

39.10 Merge a dataspace

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

Service name parameter: `service=@merge`

Input parameters

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

Output parameters

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

39.11 Access the dataspace merge view

The merge.view service is automatically considered complete.

Service name parameter: service=@merge.view

Input parameters

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

39.12 Compare contents

Workflows automatically consider the compare service as complete.

Service name parameter: service=@compare

Input parameters

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

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

39.13 Data workflows

This service provides access to the data workflows user interfaces.

Service name parameter: `service=@workflow`

Note

This service is for perspectives only.

Input parameters

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

CHAPITRE 40

Supported XPath syntax

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Example expressions](#)
3. [Syntax specifications for XPath expressions](#)
4. [Java API](#)

40.1 Overview

The XPath notation used in TIBCO EBX® 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.

40.2 Example expressions

The general XPath expression is:

path[predicate]

Absolute path

/library/books/

Relative paths

./Author

../Title

Root and descendant paths

//books

Table paths with predicates

../../books/[author_id = 0101 and (publisher = 'harmattan')]

/library/books/[not(publisher = 'dumesnil')]

Complex predicates

starts-with(col3,'xxx') and ends-with(col3,'yyy') and osd:is-not-null(./col3))

```
contains(col3 , 'xxx') and ( not(col1=100) and date-greater-than(col2, '2007-12-30') )
```

Predicates with parameters

`author_id = $param1 and publisher = $param2` where the parameters `$param1` and `$param2` refer respectively to 0101 and 'harmattan'

`col1 < $param1 and col4 = $param2` where the parameters `$param1` and `$param2` refer respectively to 100 and 'true'

`contains(col3, $param1) and date-greater-than(col2, $param2)` where the parameters `$param1` and `$param2` refer respectively to 'xxx' and '2007-12-30'

Note

The use of this notation is restricted to the Java API since the parameter values can only be set by the method `Request.setXPathParameterAPI` of the Java API.

Search predicate

- Syntax: `osd:search(fields, queryString[, templateKey])`
- Examples:
 - `osd:search('col1', 'xxx')`
 - `osd:search('col1,col2', 'xxx')`
 - `osd:search('', 'xxx')`
 - `osd:search(col1, 'xxx', myTemplate@myModule)`

The `osd:search` function tries to match a term, or a list of terms, against the set of fields of the current table. This function is generic, handling every field datatype supported by EBX®. When no fields are specified, it searches against all fields for current table. For a more advanced usage, the query string supports specialized operators, see [Caractères spéciaux](#) [p 129] for more information.

For any concerned field, if a label exists, the search targets the label, rather than the value; however, this is not yet supported in some cases. See [Limitations](#) [p 316] for more information.

The predicate `osd:search` is localized.

Note

The locale can be set by the methods of the Java API `Request.setLocaleAPI` or `Request.setSessionAPI`.

Note

The identifier of a **search template** `SearchTemplateAPI` can be specified, to customize the behavior of the search.

Predicates for validation search

```
osd:has-validation-item()
```

```
osd:has-validation-item('error,info')
```

```
osd:contains-validation-message('xxx')
```

```
osd:contains-validation-message('xxx', 'info,warning')
```

- XPath functions for validation search cannot be used on XPath predicates defined on associations and foreign key filters.

- The predicates `osd:label`, `osd:contains-record-label` and `osd:contains-validation-message` are localized.

Note

The locale can be set by the methods of the Java API `Request.setLocaleAPI` or `Request.setSessionAPI`.

Attention

To ensure that the search is performed on an up-to-date validation report, it is necessary to perform an explicit validation of the table just before using these predicates.

40.3 Syntax specifications for XPath expressions

Overview

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

Predicate specification

Expression	Format	Notes/Example
<predicate>	Example: A and (B or not(C)) A,B,C: <atomic expression>	Composition of: logical operators parentheses, not() and atomic expressions.
<atomic expression>	<path><comparator><criterion> or method(<path>,<criterion>)	royalty = 24.5 starts-with(title, 'Johnat') booleanValue = true
<path>	<relative path> or <i>osd:label</i> (<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>, <i>osd:is-null</i> method or <i>osd:is-not- null</i> method	
<date, time & dateTime method>	date-less-than, date-equal or date- greater-than	
<string method>	matches, starts-with, ends-with, contains, <i>osd:is-empty</i> , <i>osd:is- not-empty</i> , <i>osd:is-empty-or-nil</i> , <i>osd:is-neither-empty-nor-nil</i> , <i>osd:is-equal-case-insensitive</i> , <i>osd:starts-with-case-insensitive</i> , <i>osd:ends-with-case-insensitive</i> , <i>osd:contains-case-insensitive</i> , <i>osd:contains-record-label</i> , or <i>osd:search</i>	
<criterion>	<boolean criterion>, <numeric criterion>, <string criterion>, <date criterion>, <time criterion>, or <dateTime criterion>	
<boolean criterion>	true, false	
<numeric criterion>	An integer or a decimal	-4.6
<string criterion>	Quoted character string	'azerty'

Expression	Format	Notes/Example
<date criterion>	Quoted and formatted as 'yyyy-MM-dd'	'2007-12-31'
<time criterion>	Quoted and formatted as 'HH:mm:ss' or 'HH:mm:ss.SSS'	'11:55:00'
<dateTime criterion>	Quoted and formatted as 'yyyy-MM-ddTHH:mm:ss' or 'yyyy-MM-ddTHH:mm:ss.SSS'	'2007-12-31T11:55:00'

XPath 1.0 formula

It is possible to use an XPath 1.0 formula in the criterion value part of an atomic predicate expression (right-hand side).

For example, instead of `[./a=3]`, you may use the expression `[./a=(floor(./d)+ 2.0)]`.

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

Predicate on label

The `osd:label()` function can be applied to the path portion of the atomic predicate, in order to resolve the predicate on the label instead of the value. In this case, only string operators and string criteria can be used, i.e. `ends-with(osd:label(./price), '99')`.

A predicate on label is localized, so the criterion must be expressed in the same locale as the predicate-filtered request. For example: `request.setLocale(Locale.FRENCH); request.setXPathFilter("osd:label(./delivery_date)='30/12/2014'");`

Note

It is forbidden to use the `osd:label` function if the right part of the predicate is a contextual value.

Note

If the `osd:label` function is used in a data model, for example on a selection or in the filter predicate of a table reference node, the default locale of the data model (as defined in its module declaration) must be used for the criterion format (even though this is generally not recommended).

Voir aussi `SchemaNode.displayOccurrenceAPI`

Contextual values

For predicates that are relative to a selected node, the criterion value (that is, the right-hand side of the predicate) can be replaced with a contextual path using the syntax `${<relative-path>}` where *<relative-path>* is the location of the element relative to the selected node.

Note

When calling a method, the criterion is the second parameter, and the first parameter cannot be a relative value.

Aggregated lists

For predicates on aggregated lists, the predicate returns true regardless of the comparator if one of the list elements verifies the predicate.

Note

Special attention must be paid to the comparator `!=`. For example, for an aggregated list, `./list != 'a'` is not the same as `not(./list = 'a')`. Where the list contains the elements (e1,e2,...), the first predicate is equivalent to `e1 != 'a' or e2 != 'a' ...`, while the second is equivalent to `e1 != 'a' and e2 != 'a' ...`.

'Null' values

Null values must be explicitly treated in a predicate using the operators `osd:is-null` and `osd:is-not-null`.

For example, `/root/products[./price<100]` or `/root/products[./price!=100]` will not return any products whose prices are not set (null). For the latter case to return unset values as well, the predicate must instead be: `/root/products[./price!=100 or osd:is-null(./price)]`.

How to manage single and double quotes in literal expressions

By default, a literal expression is delimited by single quotes (`'`). If the literal expression contains single quotes and no double quotes, the expression must be delimited by double quotes (`"`). If the literal expression contains both single and double quotes, the single quotes must be doubled.

The method `XPathExpressionHelper.encodeLiteralStringWithDelimitersAPI` in the Java API handles this.

Examples of using `encodeLiteralStringWithDelimiters`

Value of Literal Expression	Result of this method
Coeur	'Coeur'
Coeur d'Alene	"Coeur d'Alene"
He said: "They live in Coeur d'Alene".	'He said: "They live in Coeur d''Alene".'

Extraction of foreign keys

In EBX®, the foreign keys are grouped into a single field with the [osd:tableRef](#) [p 554] 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/tableA[ fkB = '123|2008-01-21' ]`, where the string "123|2008-01-21" is a representation of the entire primary key value.
See **Syntax of the internal String representation of primary keys** `PrimaryKey.syntaxAPI` for more information.
- `/root/tableA[ fkB/id = 123 and date-equal(fkB/date, '2008-01-21') ]`, where this predicate is a more efficient equivalent to the one in the previous example.
- `/root/tableA[ fkB/id >= 123 ]`, where any number operator could be used, as the targeted primary key field is of type `xs:int`.
- `/root/tableA[ date-greater-than( ./fkB/date, '2007-01-01' ) ]`, where any date operator could be used, as the targeted primary key field is of type `xs:date`;
- `/root/tableA[ fkB = "" ]` is not valid as the targeted primary key has two columns.
- `/root/tableA[ osd:is-null(fkB) ]` checks if a foreign key is null (not defined).

40.4 Java API

Using the XPath in the Java API:

In the Java API, the `xPathFilter` class allows to define XPath predicates and to execute requests on them.

The `xPathExpressionHelper` class provides utilitarian methods to handle XPath predicates and paths.

Localisation

CHAPITRE 41

Labeling and localization

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Value formatting policies](#)
3. [Syntax for locales](#)

41.1 Overview

TIBCO EBX® offers the ability to handle the labeling and the internationalization of data models.

Localizing user interactions

In EBX®, language preferences can be set for two scopes:

1. Session: Each user can select a default locale from the user pane.
2. Data model: If a data model has been localized into other languages than those natively supported by EBX®, the user can select one of those languages for that particular data model. See [Extending TIBCO EBX® internationalization](#) [p 263] for more information.

Textual information

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

- Dataspaces, snapshots and datasets can have their own label and description. The label is independent of the unique name, so that it remains localizable and modifiable;
- Any node in the data model can have a static label and description;
- Values can have a static label when they are enumerated;
- Validation messages can be customized, and permission restrictions can provide text explaining the reason;
- Each record is dynamically displayed according to its content, as well as the context in which it is being displayed (in a hierarchy, as a foreign key, etc.);

All this textual information can be localized into the locales that are declared by the module.

Voir aussi

[Labels and messages](#) [p 591]

[Tables declaration](#) [p 549]

[Foreign keys declaration](#) [p 554]

41.2 Value formatting policies

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

A formatting policy is used to define how to display the values of [simple types](#) [p 534].

For each locale declared by the module, its formatting policy is configured in a file located at /WEB-INF/ebx/{locale}/frontEndFormattingPolicy.xml. For instance, to define the formatting policy for Greek (el), the engine looks for the following path in the module:

```
/WEB-INF/ebx/el/frontEndFormattingPolicy.xml
```

If the corresponding file does not exist in the module, the formatting policy is looked up in the class-path of EBX®. If the locale-specific formatting policy is not found, the formatting policy of en_US is applied.

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

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

The elements date, dateTime and time are mandatory.

The group and decimal separators that appear in the formatted numbers can be modified by defining the attributes groupingSeparator and decimalSeparator for the elements decimal and int.

41.3 Syntax for locales

There are two ways to express a locale:

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

In any XML file (XSD, formatting policy file, etc.) read by EBX®, either syntax is allowed.

For a web path, that is, a path within the web application, only the Java syntax is allowed. Thus, formatting policy files must be located in directories whose locale names respect the Java syntax.

Voir aussi [Extending TIBCO EBX® internationalization](#) [p 263]

CHAPITRE 42

Extending TIBCO EBX® internationalization

Ce chapitre contient les sections suivantes :

1. [Overview of the native EBX® localization](#)
2. [Extending EBX® user interface localization](#)
3. [Localized resources resolution](#)
4. [Known limitations](#)

42.1 Overview of the native EBX® localization

By default, the EBX® built-in user interface is provided in English (en-US) and French (fr-FR).

Localization consists of a formatting policy and a set of message files (resource bundle):

- For English, localization is provided by a formatting policy and a set of message files with no locale defined,
- For French, localization is provided by a formatting policy and a set of message files with locale set to "fr".

EBX® provides an option to add locales in order to extend the localization of the user interface and to internationalize the documentation of data models and associated services.

42.2 Extending EBX® user interface localization

EBX® supports the localization of its user interface 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 followed:

- Declare the new locale in the EBX® main configuration file. For example:
ebx.locales.available=en-US, fr-FR, xx

- The first locale is always considered the default.
- The built-in locales, en-US and fr-FR, can be removed if required.

See [Configuring EBX® localization](#) [p 375].

- Deploy the following files in the EBX® class-path:
 - A formatting policy file, named `com.orchestranetworks.i18n.frontEndFormattingPolicy_xx.xml`,
 - A set of localized message files (*_xx.mxml) in a resource bundle.

Note

The files must be ending with ".mxml".

42.3 Localized resources resolution

Since version 5.7.0, localized resources are resolved on a locale-proximity base, with the following lookup mechanism:

- resourceName + "_" + language + "_" + country + "_" + variant + ".mxml"
- resourceName + "_" + language + "_" + country + ".mxml"
- resourceName + "_" + language + ".mxml"
- resourceName + ".mxml"

Note

The resolution is done at the localized message level. It is therefore possible to define one or more files for a locale that only includes messages for which specific localization is required.

42.4 Known limitations

Non extendable materials

Localization of the following cannot be extended:

- EBX® product documentation,
- EBX® HTML editor and viewer.

Persistence

CHAPITRE 43

Overview of persistence

This chapter is an introduction to history tables and replicated tables.

Note

The term *mapped mode* [p 267] refers to any tables that are stored as-is, and thus whose contents can be accessed directly in the database.

Ce chapitre contient les sections suivantes :

1. [Primary persistence of managed master data](#)
2. [Historization](#)
3. [Replication](#)
4. [Mapped mode](#)

43.1 Primary persistence of managed master data

Data that is modeled in and governed by the EBX® repository are primarily persisted in the relational database, using generic tables (common to all datasets and data models).

43.2 Historization

Master data tables can activate historization in order to track modifications to their data, regardless of whether they are replicated.

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

Voir aussi [History](#) [p 269]

43.3 Replication

Replication enables direct SQL access to tables of master data, by making a copy of data in the repository to replica tables in the relational database. Replication can be enabled on any table regardless of 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 EBX®.

Voir aussi [Replication](#) [p 277]

43.4 Mapped mode

Overview of mapped mode

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

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

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

Voir aussi

[Database mapping administration](#) [p 457]

[Data model evolutions](#) [p 283]

Structural constraints

When a mapped mode is set, some EBX® data model constraints will generate a "structural constraint" on the underlying RDBMS schema. This concerns the following constraining 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 EBX®. Hence, the above constraints become *blocking constraints*. A blocking constraint means that updates are rejected if they do not comply: when a transaction does not comply with a blocking constraint, it is cancelled and a `ConstraintViolationException`^{API} is thrown.

Voir aussi [Blocking and non-blocking constraints](#) [p 581]

Data model restrictions due to mapped mode

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

- [Limitations of supported databases](#) [p 336]
- Unlimited-length strings: All string fields, except foreign keys, of type `xs:string`, its derived types, and `xs:anyURI` must define a 'maxLength' or 'length' facet. Since a foreign key field is composed of the final primary key field(s) of its target table(s), this facet requirement applies to each of those final primary key fields instead of the foreign key field itself. Additionally, limitations of the underlying database concerning the maximum length of its character types apply, such as `VARCHAR` and `NVARCHAR2`.

- Large lists of columns might not be indexable. Example for Oracle: the database enforces a limit on the maximum cumulated size of the columns included in an index. For strings, this size also depends on the character set. If the database server fails to create the index, you should consider redesigning your indexes, typically by using a shorter length for the concerned columns, or by including fewer columns in the index. The reasoning is that an index leading to this situation would have headers so large that it could not be efficient anyway.
- Fields of type `type="osd:password"` are ignored.
- Terminal complex types are supported; however, they cannot be globally set to `null` at record-level.

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, 1600 for PostgreSQL). Note that a history table contains twice as many fields as declared in the schema (one functional field, plus one generated field for the operation code).

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

Voir aussi [Data model evolutions](#) [p 283]

CHAPITRE 44

History

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Configuring history](#)
3. [History views and permissions](#)
4. [SQL access to history](#)
5. [Impacts and limitations of historized mode](#)

44.1 Overview

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

It is an improvement over the deprecated [XML audit trail](#) [p 471].

Voir aussi

[Historique](#) [p 31]

[Replication](#) [p 277]

[Data model evolutions](#) [p 283]

44.2 Configuring history

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

Configuring history in the repository

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

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

- An internationalized label.
- A list of dataspace (branches) for which history is activated. It is possible to specify whether direct children and/or all descendants should also be concerned.

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

Profile Id	Description
ebx-referenceBranch	This profile is activated only on the reference dataspace.
ebx-allBranches	This profile is activated on all dataspaces.
ebx-instanceHeaders	This profile historizes dataset headers. However, this profile will only be setup in a future version, given that the internal data model only defines dataset nodes.

Configuring history in the data model

Activating table history

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

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

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

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

Historization must be activated for each table separately. See [model design](#) [p 530] documentation for more details.

Disabling history on a specific field or group

For a historized table, the default behavior is to historize all its supported elements (see [Impacts and limitations of historized mode](#) [p 274]).

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 descendants. 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 (dataset) 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 dataspace 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.

44.3 History views and permissions

Table history view

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

The next section explains how permissions are resolved.

For more information, see [vue historique de table](#) [p 31] section. To access the table history view from Java, the method `AdaptationTable.getHistoryAPI` must be invoked.

Permissions for table history

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

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

When defining a programmatic rule, it may be required to distinguish between the functional dataset context and the history view context, either because the expected permissions are not the same, or because some fields are not present in the history structure. This is the case for dataset fields, computed values and [fields for which history has been disabled](#) [p 270]. The methods `Adaptation.isHistoryAPI`

and `AdaptationTable.getHistoryAPI` can then be used in the programmatic rule in order to implement specific behavior for history.

Note

There is currently a limitation when a table has a scripted permission rule on record specified: for security reason access to the table history is totally disabled for everyone but the built-in administrator profile. Access for other users will be allowed in a future version.

Transaction history views

The transaction history view gives access to the executed transactions, independently of a table, a dataset 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 Dataspaces area by selecting a historized dataspace and using the **Actions** menu in the workspace.

For more information, see [vue historique des transactions](#) [p 31].

44.4 SQL access to history

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

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 TIBCO EBX®, as specified in the section [Rules for the database access and user privileges](#) [p 413].

Relational schema overview

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

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

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

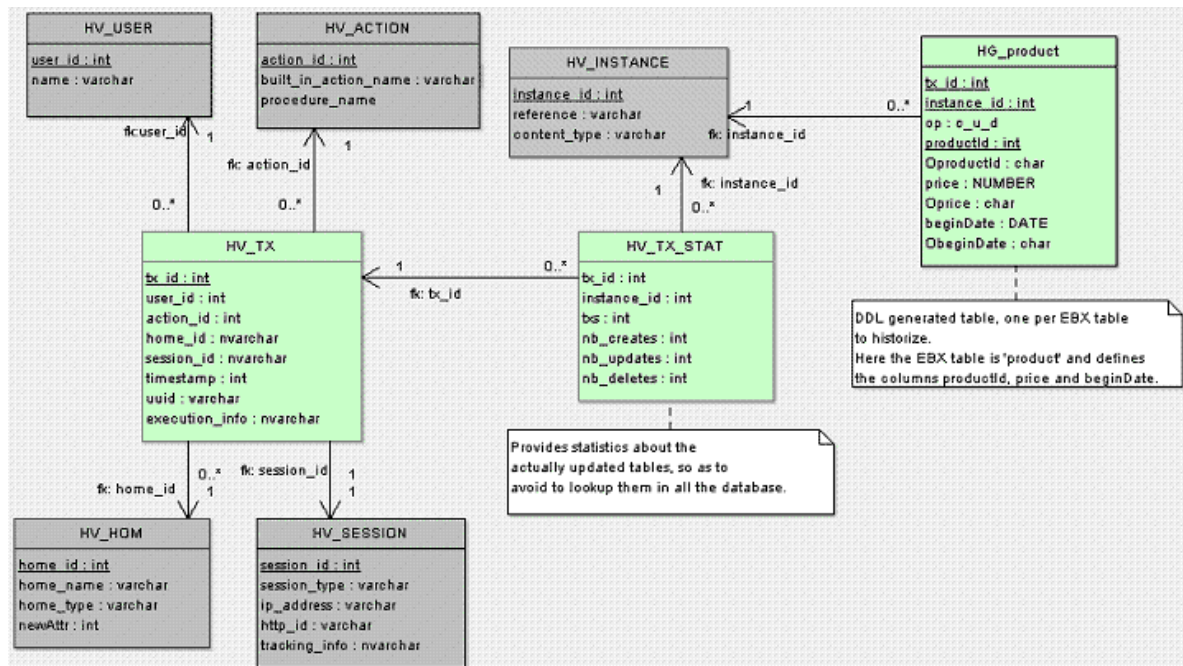
Example of a generated history table

In the following example, we are historizing a table called product. Let us assume this table declares three fields in EBX® 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.

Combination of operations

If several operations are combined in the same transaction, the operation field is resolved as follows:

- C + U -> C
- D + U -> D
- D + C -> U
- C + D -> {} (no entry in history)

Values for operation fields

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

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

If [inheritance](#) [p 288] 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.

44.5 Impacts and limitations of historized mode

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

Validation

Some EBX® data model constraints become blocking constraints when table history is activated. For more information, see the section [Structural constraints](#) [p 267].

Data model restrictions for historized tables

Some restrictions apply to data models containing historized tables:

- [Data model restrictions due to mapped mode](#) [p 267]
- Limitations exist for two types of aggregated lists: aggregated lists under another aggregated list, and aggregated lists under a terminal group. Data models that contain such aggregated lists can be used, however these lists will be ignored (not historized).
- Computed values are ignored.
- Linked fields 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](#) [p 283]

Other limitations of historized mode

- No data copy is performed when a table with existing data is activated for history.
- Global operations on datasets are not historized (create an instance and remove an instance), even if they declare a historized table.
- Default labels referencing a non-historized field are not supported for historized tables.

As a consequence, default labels referencing a computed field are not supported for historized tables.

The workaround is to implement the `UILabelRenderer` interface and adapt the label computation for history.

- D3: the history can be enabled in the delivery dataspace of a primary node, but in the delivery dataspace of the replica nodes, the historization features are always disabled.
- Recorded user in history: for some specific operations, the user who performs the last operation and the one recorded in the corresponding history record may be different.

This is due to the fact that these operations are actually a report of the data status at a previous state:

- Archive import: when importing an archive on a dataspace, the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is the user who performs the import.
- Programmatic merge: when performing a programmatic merge on a dataspace, the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is the user who performs the merge.
- D3: for distributed data delivery feature, when a broadcast is performed, the data from the primary node is reported on the replica node and the time and user of the last operation performed in the child dataspace are preserved, while the user recorded in history is 'ebx-systemUser' who performs the report on the replica node upon the broadcast.

CHAPITRE 45

Replication

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Configuring replication](#)
3. [Accessing a replica table using SQL](#)
4. [Requesting an 'onDemand' replication refresh](#)
5. [Impact and limitations of replication](#)

45.1 Overview

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

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

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

Voir aussi

[History](#) [p 269]

[Data model evolutions](#) [p 283]

[Repository administration](#) [p 412]

Note

replicated table: refers to a primary data table that has been replicated

replica table (or replica): refers to a database table that is the target of the replication

45.2 Configuring replication

Enabling replication

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

The nested elements are as follows:

Element	Description	Required
<code>name</code>	Nom de l'unité de réplication. Ce nom identifie l'unité de réplication dans le modèle de données. Ce nom doit être unique.	Yes
<code>dataSpace</code>	Indique l'espace de données concerné par la réplication. Cet espace de données ne peut ni être une version ni être relationnel.	Yes
<code>dataSet</code>	Indique le jeu de données concerné par la réplication.	Yes
<code>refresh</code>	Specifies the data synchronization policy. The possible policies are: <code>onCommit</code>: The replica table content in the database is always up to date with respect to its source table. Every transaction that updates the EBX® source table triggers the corresponding insert, update, and delete statements on the replica table. <code>onDemand</code>: The replication of specified tables is only done when an explicit refresh operation is performed. See Requesting an 'onDemand' replication refresh [p 280].	Yes
<code>table/path</code>	Indique le chemin de la table dans le modèle de données qui doit être répliquée dans la base de données.	Yes
<code>table/nameInDatabase</code>	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
<code>table/element/path</code>	Indique le chemin de la liste agrégée dans la table qui doit être répliquée dans la base de données.	Yes
<code>table/element/nameInDatabase</code>	Indique le nom de la table dans la base de données qui contiendra les données répliquées de la liste agrégée. 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>
    </osd:replication>
  </xs:appinfo>
</xs:annotation>
</xs:schema>
```

```

</table>
<table>
  <path>/root/domain1/tableB</path>
  <nameInDatabase>PRODUCT_REF_B</nameInDatabase>
  <element>
    <path>/retailers</path>
    <nameInDatabase>PRODUCT_REF_B_RETAILERS</nameInDatabase>
  </element>
</table>
</osd:replication>
</xs:appinfo>
</xs:annotation>
...
</xs:schema>

```

Notes:

- See [Data model restrictions for replicated tables](#) [p 280]
- If, at data model compilation, the specified dataset and/or dataspace does not exist in the current repository, a warning is reported, but the replica table is created in the database. Once the specified dataspace and dataset are created, the replication becomes active.
- At data model compilation, if a table replication is removed, or if some of the above properties has changed, the replica table is dropped from the database, and then recreated with the new definition if needed.

Disabling replication on a specific field or group

For a replicated table, the default behavior is to replicate all its supported elements (see [Data model restrictions for replicated tables](#) [p 280]).

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.

45.3 Accessing a replica table using SQL

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

Voir aussi [SQL access to history](#) [p 272]

Finding the replica table in the database

For every replicated EBX® table, a corresponding table is generated in the RDBMS. Using the EBX® 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 EBX® uses.

Voir aussi [Rules for the database access and user privileges](#) [p 413]

SQL reads

Direct SQL reads are possible in well-managed, preferably short-lived transactions. However, for such accesses, EBX® 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.

45.4 Requesting an 'onDemand' replication refresh

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

There are several ways to request a replication refresh:

- **User interface:** In the dataset 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](#) [p 740] for data services for more information.
- **Java API:** Call the `ReplicationUnit.performRefreshAPI` methods in the `ReplicationUnit` API to launch a refresh of the replication unit.

45.5 Impact and limitations of replication

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

See [Supported databases](#) [p 336] for the databases for which replication is supported.

Validation

Some EBX® data model constraints become blocking constraints when replication is enabled. For more information, see [Structural constraints](#) [p 267].

Data model restrictions for replicated tables

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

- [Data model restrictions due to mapped mode](#) [p 267]
- Dataset inheritance is not supported for the 'onCommit' refresh policy if the specified dataset is not a root dataset or has not yet been created. See [dataset inheritance](#) [p 289] for more information.

- Field inheritance is also only supported for the 'onDemand' refresh policy. This means that, at data model compilation, an error is reported if the refresh mode is 'onCommit' and the table to be replicated has an inherited field. See [inherited fields](#) [p 290] for more information.
- Computed values are ignored.
- Linked fields are ignored.
- Limitations exist for two types of aggregated lists: aggregated lists under another aggregated list, and aggregated lists under a terminal group. Data models that contain such aggregated lists can be used, however these lists will be ignored (not replicated).
- User-defined attributes are not supported. A compilation error is raised if they are included in a replication unit.

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](#) [p 283]

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 bulk of all replica tables in a replication unit to fit into the 'UNDO' tablespace.
- It is recommended to provide enough space in the buffer cache to allow those transactions to run with minimal disk access.
- It is recommended to provision 'REDO' log groups big enough to avoid those transactions to wait for the 'db_writer' process.

Distributed data delivery (D3)

Replication is available on both D3 primary and replica delivery dataspace. On the primary dataspace, the replication behavior is the same as in a standard semantic dataspace, but on replica dataspace, the replicated content is that of the last broadcast snapshot.

In a replica delivery dataspace, some restrictions occur:

- The refresh policy defined in the data model has no influence on the behavior described above: replication always happens on snapshot.
- The action item Refresh replicas is not available.
- It is not allowed to invoke the `ReplicationUnit.performRefreshAPI` method.

Voir aussi [D3 overview](#) [p 478]

Other limitations of replication

- [Limitations of supported databases](#) [p 336]

- 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 46

Data model evolutions

This chapter describes the modifications that are possible on data models, as well as potential limitations.

Attention

Whenever the data modeler performs an evolution on the primary key of a table, the resulting definition is considered as a new table. In such cases, if existing data must be preserved in some ways, a data migration plan must be set up and operated before the new data model is published or deployed. It can also be noted that data is not destroyed immediately after the data model evolution; if the data model is rolled back to its previous state, then the previous data is retrieved.

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 TIBCO EBX® from a module. Otherwise, the configuration modifications are not taken into account.

Voir aussi [Mapped mode](#) [p 267]

Ce chapitre contient les sections suivantes :

1. [Types of permitted evolutions](#)
2. [Limitations/restrictions](#)

46.1 Types of permitted evolutions

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

Model-level evolutions

The following modifications can be made to existing data models:

- Replication units can be added to the data model. If their refresh policy is 'onCommit', the corresponding replica tables will be created and refreshed on next schema compilation.
- Replication units can be removed from the data model. The corresponding replica tables will be dropped immediately.

- The data model can be deleted. If it declares replication units, the corresponding replica tables will be dropped immediately. If it contains historized tables, this change marks the associated mapped tables as disabled. See [Database mapping](#) [p 457] for the actual removal of associated database objects.

Table-level evolutions

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

- A new table can be added. Upon creation, the table can also declare one or more mapped modes.
- An existing table can be deleted. If it declares replication units, the corresponding replica tables will be dropped immediately. If it historized, this change marks the mapped table as disabled. See [Database mapping](#) [p 457] for the actual removal of associated database objects.
- History can be enabled or disabled on a table. History will not take into account the operations performed while it is disabled.
- A table can be renamed. Data should be manually migrated, by exporting then re-importing an XML or archive file, because this change is 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. The data of the deleted field will be removed from each record upon its next update. For a replica table, the corresponding column is automatically removed. In history mode, the field is marked as disabled.
- A field can be specifically disabled from the history or replication which applies to its containing table, by using the attribute `disable="true"`. For a replica table, the corresponding column is automatically removed. For a history table, the column remains but is marked as disabled. See [Disabling history on a specific field or group](#) [p 270] and [Disabling replication on a specific field or group](#) [p 279].
- The facets of a field can be modified, except for the facets listed under [Limitations/restrictions](#) [p 284].

The following changes are accepted, but they can lead to a loss of data. Data should be migrated manually, by exporting then re-importing an XML or archive file, since these changes are considered to be a combination of deletion and creation:

- A field can be renamed.
- The type of a field can be changed.

46.2 Limitations/restrictions

Limitations related to primary key evolutions

When a primary key definition is modified:

- The content of the table will be reset to an empty content, in all datasets and dataspace.

- If the new primary key has been used in the past, the content of the table will be reset to the previous data existing at the time this primary key was used, in all datasets and dataspace.
- The modification will be rejected if the table has - or has had - history activated in the existing dataspace. A possible workaround: first drop the history table associated with the dedicated table, then proceed to modifying the primary key. For the procedure to purge mapped table database resources, see [Database mapping](#) [p 457].

Note

If the modified primary key is referenced in the primary key of another table, all the limitations mentioned above apply to the target table.

Limitations related to foreign key evolutions

- When the declaration of a `osd:tableRef` facet is added or modified, or when the primary key of its target table is modified, the existing values will restart from empty (except if this modification is reverting to a previous definition; in this case, the previous content will be retrieved).
- In replication mode, the structure of a foreign key field is set to match that of the target primary key. A single field declaring an `osd:tableRef` constraint may then be split into a number of columns, whose number and types correspond to that of the target primary key. Hence, the following cases of evolutions will have an impact on the structure of the mapped table:
 - declaring a new `osd:tableRef` constraint on a table field;
 - removing an existing `osd:tableRef` constraint on a table field;
 - adding (resp. removing) a column to (resp. from) a primary key referenced by an existing `osd:tableRef` constraint;
 - modifying the type or path for any column of a primary key referenced by an existing `osd:tableRef` constraint.

These cases of evolution will translate to a combination of field deletions and/or creations. Consequently, the existing data should be migrated manually.

Limitations related to field-level evolutions

When changing the type of a field to an incompatible type or cardinality, the field will be considered as a new one, and start with an empty content. The previous content will be retrieved if the model is rolled back to a previous definition.

- The following types are fully inter-convertible (meaning these types have the same exact persistent representation, and can be substituted to each other in the following charts):
 - `xs:string`
 - `osd:color`
 - `osd:datasetName`
 - `osd:dataspaceKey`
 - `osd:email`
 - `osd:html`
 - `osd:local`
 - `osd:resource`

- xs:nmtoken
- xs:nmtokens
- osd:text
- xs:anyUri
- xs:name
- The following conversions are fully supported (that is, regardless of their cardinalities):
 - xs:decimal to xs:string
 - xs:datetime to xs:string
 - xs:date to xs:string
 - xs:integer to xs:string
 - xs:int to xs:decimal
 - xs:integer to xs:decimal
 - xs:decimal to xs:integer (losing the decimal part)
 - xs:int to xs:integer
 - xs:datetime to xs:date (losing the time part)
 - xs:date to xs:datetime (defaulting the time part to 0)
- The following conversions are possible only if the original type is single-valued:
 - xs:boolean to xs:string
 - xs:time to xs:string
 - xs:int to xs:string
 - xs:long to xs:string

The cardinality of a type can be changed; when the conversion is supported, it has the following behavior:

- 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. Other values are lost.

Attention

Groups and complex types do not support conversion to (and from) any other types. Moreover, when a group or complex type changes between single-occurrenced and multi-occurrenced, the conversion is supported only if the group or complex type is terminal.

Divers

CHAPITRE 47

Inheritance and value resolution

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Dataset inheritance](#)
3. [Inherited fields](#)
4. [Optimize & Refactor service](#)

47.1 Overview

The principle of inheritance is to mutualize resources that are shared by multiple contexts or entities. TIBCO EBX® offers mechanisms for defining, factorizing and resolving data values: *dataset inheritance* and *inherited fields*.

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

Note

Inheritance mechanisms described in this chapter should not be confused with "structural inheritance", which usually applies to models and is proposed in UML class diagrams for example.

Voir aussi [Inheritance \(glossary\)](#) [p 29]

Dataset inheritance

Dataset inheritance is particularly useful when data applies to global enterprise contexts, such as subsidiaries or business partners.

Given a hierarchy of datasets, it is possible to factorize common data into the root or intermediate datasets and define specialized data in specific contexts.

The dataset inheritance mechanisms are detailed below in [Dataset inheritance](#) [p 289].

Inherited fields

Contrary to dataset inheritance, which exploits global built-in relationships between datasets, inherited fields exploit finer-grained dependencies that are specific to the data structure. It allows factorizing and specializing data at the business entities-level.

For example, if the model specifies that a 'Product' is associated with a 'FamilyOfProducts', it is possible that some attributes of 'Product' inherit their values from the attributes defined in the associated 'FamilyOfProducts'.

Note

When using both inheritance mechanisms in the same dataset, field inheritance has priority over dataset inheritance.

Computed values (functions)

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

The function is able to take into account the current context, such as the values of the current record or computations based on another table, and to send requests to third-party systems.

Voir aussi [Computed values](#) [p 587]

47.2 Dataset inheritance

Dataset inheritance declaration

The dataset inheritance mechanism is declared as follows in a data model:

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

The element `osd:inheritance` defines the property `dataSetInheritance` to specify the use of inheritance on datasets based on this data model. The following values can be specified:

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

If not specified, the inheritance mechanism is disabled.

Value lookup mechanism

The dataset inheritance lookup mechanism for values proceeds as follows:

1. If the value is locally defined, it is returned.
It can be explicitly `null`.
2. Otherwise, looks up the first locally defined value according to the built-in child-to-parent relationship of the dataset in the hierarchy of datasets.
3. If no locally defined value is found, the default value is returned.
If no default value is defined, `null` is returned.

Note: Default values cannot be defined on:

- A single primary key node

- Auto-incremented nodes
- Nodes defining a computed value

Record lookup mechanism

Like values, table records can also be inherited as a unit by multiple contexts, but they can also be partially redefined (*overwritten*), defined for a specific context (*root mode*), or be *occulted*.

Formally, a table record has one of four distinct definition modes:

<i>root record</i>	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.
<i>overwriting record</i>	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.
<i>inherited record</i>	Not locally defined in the current table and has a parent record. All values are inherited. Functions are always resolved in the current record context and are not inherited.
<i>occulting record</i>	Specifies that, if a parent with the same primary key is defined, this parent will not be visible in table descendants.

Voir aussi [Héritage entre jeux de données](#) [p 161]

Defining inheritance behavior at the table level

It is also possible to specify management rules in the declaration of a table in the data model.

Voir aussi [Properties related to dataset inheritance](#) [p 553]

47.3 Inherited fields

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

Field inheritance declaration

Specific inheritance must be specified on terminal nodes in the underlying data model and is declared as follows:

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

```
<sourceNode>color</sourceNode>
</osd:inheritance>
</xs:appinfo>
</xs:annotation>
</xs:element>
```

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

If `sourceRecord` is not defined in the data model, the inherited fields are fetched from the current record.

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

The following conditions must be satisfied for specific inheritance:

- The element `sourceNode` is mandatory.
- The expression for the path to the source record must be a consistent path of foreign keys, from the current element to the source record. This expression must involve only one-to-one and zero-to-one relationships.
- The `sourceRecord` cannot contain any aggregated list elements.
- Each element of the `sourceRecord` must be a foreign key.
- If the inherited field is also a foreign key, the `sourceRecord` cannot refer to itself to get the path to the source record of the inherited value.
- Every element of the `sourceRecord` must exist.
- The source node must belong to the table containing the source record.
- The source node must be terminal.
- The source node must be writeable.
- The source node type must be compatible with the current node type.
- The source node cardinalities must be compatible with those of the current node.
- The source node cannot be the same as the inherited field if the fields to inherit from are fetched into the same record.

Value lookup mechanism

The lookup mechanism for inherited fields values proceeds as follows:

1. If the value is locally defined, it is returned.
It can be explicitly `null`
2. Otherwise, looks up the source record and value to inherit from, according to the properties that are defined in the data model.
3. The process is recursive; if the source node does not locally define a value, it is then looked up according to the inheritance behavior of the source node.

47.4 Optimize & Refactor service

EBX® provides a built-in user service for optimizing the dataset inheritance in the hierarchy of datasets. This service performs the following functions:

- **Handles duplicated values:** Detects and removes all parameter values that are duplicates of the inherited value.
- **Mutualizes common values:** Detects and mutualizes the common values among the descendants of a common ancestor.

Procedure details

Datasets are processed from the bottom up, which means that if the service is run on the dataset at level N , with $N+1$ being the level of its children and $N+2$ being the level of its children's children, the service will first process the datasets at level $N+2$ to determine if they can be optimized with respect to the datasets at level $N+1$. Next, it would proceed with an optimization of level $N+1$ against level N .

Note

- These optimization and refactoring functions do not handle default values that are declared in the data model.
- The highest level considered during the optimization procedure is always the dataset on which the service is run. This means that optimization and refactoring are not performed between the target dataset and its own ancestors.
- Table optimization is performed on records with the same primary key.
- Inherited fields are not optimized.
- *The optimization and refactoring functions do not modify the resolved view of a dataset, if it is activated.*

Service availability

The 'Optimize & Refactor' service is available on datasets that have child datasets and have the 'Activated' property set to 'No' in their dataset information.

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

Note

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

CHAPITRE 48

Permissions

Permissions dictate the access each user has to data and actions.

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Important considerations about permissions](#)
3. [Defining user-defined rules](#)
4. [Defining dynamic rules](#)
5. [Resolving permissions on data](#)
6. [Resolving permissions on services](#)
7. [Resolving permissions on actions](#)

48.1 Overview

Permissions are related to whether actions are authorized or not. They are also related to access rights, that is, whether an entity is hidden, read, or read-write. The main entities controlled by permissions are:

- Dataspace
- Dataset
- Table
- Group
- Field

Users, roles and profiles

The definition and resolution of permissions make extensive use of the notion of *profiles*, which is the generic term applied to users or roles.

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

These relationships are defined in the user and roles directory. See [Users and roles directory](#) [p 451].

Special definitions:

- A *built-in administrator* is a member of the built-in role 'ADMINISTRATOR'.

- An *owner of a dataset* is a member of the *owner* attribute specified in the information of a root dataset. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the dataset.
- An *owner of a dataspace* is a member of the *owner* attribute specified for a dataspace. In this case, the built-in role 'OWNER' is activated when permissions are resolved in the context of the dataspace.

Permission rules

A permission rule defines the authorization granted to a profile for a particular entity.

User-defined permission rules are created through the user interface. See the section [Defining user-defined rules](#) [p 298].

Dynamic permission rules can be either programmatic rules created by developers, or scripted rules created by administrators. See the section [Defining dynamic rules](#) [p 302].

Resolution of permissions

Permissions are always resolved in the context of an authenticated user session, thus permissions are mainly based on the user profiles.

In general, resolution of permissions is performed restrictively between a given level and its parent level. Thus, at any given level, a user cannot have a higher permission than the one resolved at a parent level.

Dynamic permissions are always considered to be restrictive.

Note

In the Java API, the class `SessionPermissionsAPI` provides access to the resolved permissions.

Voir aussi

[Resolving permissions on data](#) [p 304]

[Resolving permissions on services](#) [p 307]

[Resolving permissions on actions](#) [p 309]

Owner and administrator special permissions

On a dataset

A built-in administrator or owner of a dataset can perform the following actions:

- Manage its permissions
- Change its owner, if the dataset is a root dataset

- Change its general information (localized labels and descriptions)

Attention

While the definition of permissions can restrict a built-in administrator or dataset owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

On a dataspace

To be a *super owner* of a dataspace, a user must either:

- Own the dataspace and be allowed to manage its permissions, or
- Own a dataspace that is an ancestor of the current dataspace and be allowed to manage the permissions of that ancestor dataspace.

A built-in administrator or super owner of a dataspace can perform the following actions:

- Manage its permissions of dataspace.
- Change its owner
- Lock it or unlock it
- Change its general information (localized labels and descriptions)

Furthermore, in a workflow, when using a "Create a dataspace" or "Create a snapshot" built-in script task, resolved permissions are computed using the owner defined in the script task's configuration, rather than the current session. This is because, in these cases, the current session is associated with a system user.

Attention

While the definition of permissions can restrict a built-in administrator or dataspace owner's right to view data or perform certain actions, it remains possible for them to modify their own access, as they will always have access to permissions management.

Impact of merge on permissions

When a dataspace is merged, the permissions of the child dataset are merged with those of the parent dataspace if and only if the user specifies to do so during the merge process. The permissions of its parent dataspace are never impacted.

If some elements are hidden for the profile attempting to perform a merge, it will not be possible to proceed as the impacts of the merge on data will not be fully visible.

48.2 Important considerations about permissions

In this section are listed some very important information that must be kept in mind while working with permissions.

Actions and user services granting high privileges

The following actions and their related user services must only be allowed to trusted administrators:

- 'Create dataspace' (gives the 'owner' role, which grants the right to define the dataspace permissions)
- 'Create dataset' (gives the 'owner' role, which grants the right to define the dataset permissions)
- 'Import archive' (allows writing the archive content regardless of any permission)

Note

See the [Owner and administrator special permissions](#) [p 294] section for more information about the privileges granted to these profiles.

API access without permission checks

Developers and administrators must be aware that some parts of the API can run without any permission check. In general if the code run in a context with a `SessionAPI` provided, it means that permissions will be checked. Here are some specific cases where permissions are not checked:

- When a Java procedure disables all permission checks by using `ProcedureContext.setAllPrivilegesAPI`.
- When accessing EBX data by directly querying your database, in the case a table enables the [replication mode](#) [p 266], or the [historization](#) [p 266]. This is because EBX permissions are not "translated" in the underlying database. As a consequence, either the database access must be globally restricted or proper permissions must be defined in it.

Using permission for hiding information in the UI

Using the permissions only to hide in the UI some non sensitive information is highly unadvised, especially if this information is likely to be used for filtering / joining / sorting in some queries. In such cases, UI-only hiding methods should be used instead. For instance by setting the field as [hidden for default views](#) [p 599] in the datamodel property and/or by creating [views](#) [p 437] for the concerned users.

Limitations of the permission checks in Query API

The permission check performed when specifying a session in a `QueryAPI` or `RequestAPI` will throw a `QueryPermissionExceptionAPI` if any field used in the query is hidden for the current user. However there are some specificities to know that are described hereafter:

- Fields belonging to primary keys are not checked and are always considered as usable.
- `AccessRuleAPI` set on fields which are record-dependent are ignored by this permission check. In other words, method `AccessRule.getPermissionAPI` is only called with the dataset as the `aDataSetOrRecord` parameter, never with a record.

As a consequence to the last point, it is recommended be very vigilant when using this kind of rules because a malicious user could "guess" sensitive information by filtering or sorting on these nodes when using a component relying on these API. For instance, a developer could decide to prevent the query to run as soon as a field has an `AccessRuleAPI` defined on it, to remove such criteria before executing the query, or to completely hide the records for which some fields are confidential.

Scripted permission rules on records and table history

There is currently a limitation when a table has both the history activated and a scripted permission rule on record specified: for security reason access to the table history is totally disabled for everyone. Access to history will be allowed in a future version.

Using hidden fields in custom display labels

Resolution of custom display labels for tables ('defaultLabel' property) and relationships ('display' property) takes into account permission. As soon as an hidden field is detected in the label, the primary key will be displayed instead.

Note

This is not the case when using API like `TableRefDisplayAPI` or `Adaptation.getLabelOrNameAPI`. Since the provided contexts do not contain the current session, no permission check can be performed. As a consequence, developer should make sure that no confidential data is exposed when using these APIs.

Note

Also note that quick search will ignore nodes with hidden fields in custom display label in the context of history view and/or in a child dataset.

Because of this behavior it is highly discouraged to use labels for filtering in a query. When labels with hidden fields are used, it will be replaced by the pk value and the filter will become inconsistent.

Linked field permission check

When a [linked field](#) [p 568] access permission is computed, the result is the minimum between the permission applying to the node in the main table and the node in the target table. Practically it means that if a field is hidden in a table, all linked fields pointing on it in other tables will also be hidden.

Table action permission related limitations

When performing actions on a table (create, delete, overwrite or occult) in a procedure, the current user session access right on the table node is ignored during the permission resolution. Should this check be performed, the client code must explicitly call `SessionPermissions.getNodeAccessPermissionAPI` beforehand in the procedure.

Permission cache life cycle

To optimize the resolution of permissions for both data and user services, a dedicated cache is implemented at the session level. All permissions are cached including dynamic rules, it means that a rule result should not change for the duration of the cache which is explained below.

The session cache life cycle depends on the context, as described hereafter:

- In the UI, the cache is cleared for every non-ajax event (i.e on page display, pop-up opening, etc.).

- In programmatic procedures, the cache lasts until the end of the procedure, unless explicitly cleared (see below).

Attention

When modifying permissions in a procedure context (by importing an EBX® archive or merging a dataspace programmatically), the session cache **must** be cleared via a call to `Session.clearCacheAPI`. Otherwise, these modifications will not be reflected until the end of the procedure.

48.3 Defining user-defined rules

Each level has a similar schema, which allows defining permission rules for profiles.

Defining dataspace user-defined rules

For a given dataspace, the allowable permissions for each profile are as follows:

Dataspace access	Authorization
Write	• Can view the dataspace. • Can access datasets according to dataset permissions.
Read-only	• Can view the dataspace and its snapshots. • Can view child dataspace, if allowed by permissions. • Can view contents of the dataspace, though cannot modify them.
Hidden	• Can neither see the dataspace nor its snapshots. • If allowed to view child dataspace, can see the current dataspace but cannot select it. • Cannot access the dataspace contents, including datasets. • Cannot perform any actions on the dataspace.

Restriction policy	Indicates whether this dataspace profile-permission association should have priority over other permissions rules.
Create a child dataspace	Indicates whether the profile can create child dataspace from the current dataspace.
Create a child snapshot	Indicates whether the profile can create snapshots of the current dataspace.
Initiate merge	Indicates whether the profile can merge the current dataspace with its parent dataspace.
Export archive	Indicates whether the profile can export the current dataspace as an archive.
Import archive	Indicates whether the profile can import an archive into the current dataspace.
Close a dataspace	Indicates whether the profile can close the current dataspace.
Close a snapshot	Indicates whether the profile can close a snapshot of the current dataspace.
Rights on services	Indicates if a profile has the right to execute services on the dataspace. By default, all dataspace services are

allowed. A built-in administrator or super owner of the current dataspace or a given user who is allowed to modify permissions on the current dataspace can modify these permissions to restrict dataspace services for certain profiles.

Permissions of child dataspace when created

When a user creates a child dataspace, the permissions of this new dataspace are automatically assigned to the profile's owner, based on the permissions defined under 'Permissions of child dataspace when created' in the parent dataspace. If multiple permissions are defined for the owner through different roles, the owner's profile behaves like any other profile and [permissions are resolved](#) [p 294] as usual.

Defining dataset user-defined rules

For a given dataset, the allowable permissions for each profile are as follows:

Actions on datasets

Restriction policy	Indicates whether this dataset profile-permission association should have priority over other permissions rules.
Create a child dataset	Indicates whether the profile has the right to create a child dataset of the current dataset.
Duplicate dataset	Indicates whether the profile has the right to duplicate the current dataset.
Change the dataset parent	Indicates whether the profile has the right to change the parent dataset of a given child dataset.

Actions on tables

The action rights on default tables are defined at the dataset level. It is then possible to override these default rights for one or more tables. The allowable permissions for each profile are as follows:

Create a new record	Indicates whether the profile has the right to create records in the table.
Overwrite inherited record	Indicates whether the profile has the right to overwrite inherited records in the table.
Occult inherited record	Indicates whether the profile has the right to occult inherited records in the table.
Delete a record	Indicates whether the profile has the right to delete records in the table.

Access rights on node values

Permissions defined on specific terminal nodes override their default access rights.

Read-write	Can view and modify node values.
Read	Can view nodes, but cannot modify their values.
Hidden	Cannot view nodes.

Permissions on services

A built-in administrator or an owner of the current dataspace can modify the service default permission to either restrict or grant access to certain profiles.

Enabled	Grants service access to the current profile.
Disabled	Forbids service access to the current profile. It will not be displayed in menus, nor will it be launchable via web components.
Default	Sets the service permission to enabled or disabled, according to the default permission defined upon service declaration. See <code>ActivationContext.setDefaultPermission^{API}</code> for more information.

48.4 Defining dynamic rules

Dynamic rules give the possibility to define more precisely the conditions for accessing data or user services depending on the context.

There are different types of programmatic rules:

- the `AccessRuleAPI`, described in the section below [Defining access rules on data](#) [p 302].
- the scripted record permission rule, described in the section below [Defining scripted permission rules on data](#) [p 302].
- the `ServiceActivationRule` [p 303], described in the section below [Defining activation rules on service](#) [p 303].
- the `ServicePermissionRuleAPI`, described in the section below [Defining permission rules on service](#) [p 303].

Defining scripted permission rules on data

scripted permission rules are rules that dynamically define, depending on the context, the read/write rights on the records of a table.

To define such a rule, a [record permission script](#) [p 910] must be created in the DMA. A script editor is available on the table node definition, in the "Extensions" tab.

Defining access rules on data

`AccessRules` are rules that programmatically define, depending on the context, the read/write rights on a data model node or on the records of a table.

The definition of an `AccessRule` is performed as follows:

1. Creation of a rule in the form of a Java class implementing the `AccessRuleAPI` or `AccessRuleForCreateAPI` interface.
2. Assignment of this rule to concerned nodes in the schema extension: `SchemaExtensionsAPI`.

According to the rule target (model node(s) or records) and type (`AccessRule` or `AccessRuleForCreate`), several methods such as `SchemaExtensionsContext.setAccessRuleOnOccurrenceAPI` or `SchemaExtensionsContext.setAccessRuleForCreateOnNodeAPI` can be used.

The rule thus assigned is said to be "local" and is only executed when the target entity is requested. See [Resolving permissions on data](#) [p 304] for more information.

Attention

Only one `AccessRule` can be defined for each node, dataspace or record. Only one `AccessRuleForCreate` can be defined for each table child node. The definition of a new programmatic rule of one type will lead to the replacement of the existing one.

Defining activation rules on service

The `ServiceActivationRules` allow to specify if a service is activated or not for a given dataspace or dataset. A service that has been deactivated through this rule is never available in the entity for which it is deactivated, regardless of the current profile, for execution or display, even in permission screens.

The definition of a `ServiceActivationRule` is carried out as follows:

1. Creation of a rule in the form of a Java class implementing the `ServiceActivationRuleForDataspaceAPI` interface or `ServiceActivationRuleForDatasetAPI`, depending on the service type.
2. Assignment of this rule to the impacted services at their declaration level, depending on the service type, via the `ActivationContextOnDataspace.setActivationRuleAPI` or `ActivationContextWithDatasetSet.setActivationRuleAPI` methods.

The resulting assigned rule will be evaluated during the service activation evaluation. See [Resolving permissions on services](#) [p 307] for more information.

Defining permission rules on service

The `ServicePermissionRules` are advanced rules allowing to dynamically define the display and execution conditions of a service depending on the context (current session, selected entity, etc.). The service should be activated for the current context beforehand for this type of rule to be triggered.

The definition of a `ServicePermissionRule` is carried out as follows:

1. Creation of a rule in the form of a Java class implementing the `ServicePermissionRuleAPI` interface.
2. Assignment of this rule to the impacted services:
 - Either, for new services, at their declaration level via the `ActivationContext.setPermissionRuleAPI` method.

The rule thus assigned is said to be "global" and is only executed when the service is activated for the current context. See [Resolving permissions on services](#) [p 307] for more information.

- Or, for existing services, in the **schema extension** `SchemaExtensionsAPI` via the `SchemaExtensionsContext.setServicePermissionRuleOnNodeAPI` and `SchemaExtensionsContext.setServicePermissionRuleOnNodeAndAllDescendantsAPI` methods. It is thus possible to assign a rule to any service, including standard services provided by EBX®, on one or more data model nodes: a table node, an association node, etc.

The rule thus assigned is said to be "local" and is only executed in the extended schema context and when the node corresponds to the one specified. See [Resolving permissions on services](#) [p 307] for more information.

Attention

Only one `ServicePermissionRule` can be defined for each model node. Thus, the definition of a new programmatic rule will replace the existing one.

48.5 Resolving permissions on data

Resolving user-defined rules

Access rights defined using the user interface are resolved on four levels: dataspace, dataset, record (if applicable) and node.

If a profile is associated with restrictive access rights at a given level, the minimum of all restrictive rights defined at that level is resolved. If no restrictions are defined at that level, the maximum of all access rights defined at that level is resolved.

When a restrictive permission is defined for a profile, it takes precedence over the other permissions potentially granted by the user's other roles. Generally, for all user-defined permission rules that match the current user session:

- If some rules with restrictions are defined, the minimum permissions of these restricted rules are applied.
- If no rules having restrictions are defined, the maximum permissions of all matching rules are applied.

Examples:

Given two profiles *P1* and *P2* concerning the same user, the following table lists the possibilities when resolving that user's permission to a service.

P1 authorization	P2 authorization	Permission resolution
Enabled	Enabled	Enabled. Restrictions do not make any difference.
Disabled	Disabled	Disabled. Restrictions do not make any difference.
Enabled	Disabled	Enabled, unless P2's authorization is a restriction.
Disabled	Enabled	Enabled, unless P1's authorization is a restriction.

The same restriction policy is applied for data access rights resolution.

In another example, a dataspace can be hidden from all users by defining a restrictive association between the built-in profile "Profile.EVERYONE" and the access right "hidden".

At any given level, the most restrictive access rights between those resolved at this level and higher levels are applied. For instance, if a user's dataset access permissions resolve to read-write access, but the container dataspace only allows read access, the user will only have read-only access to this dataset.

Note

The dataset inheritance mechanism applies to both values and access rights. That is, access rights defined on a dataset will be applied to its child datasets. It is possible to override these rights in the child dataset.

Access rights resolution example

In this example, there are three users who belong to the following defined roles and profiles:

User	Profile
User 1	• user1 • role A • role B
User 2	• user2 • role A • role B • role C
User 3	• user3 • role A • role C

The access rights of the profiles on a given element are as follows:

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

After resolution based on the role and profile access rights above, the rights that are applied to each user are as follows:

User	Resolved access rights
User 1	Hidden
User 2	Read
User 3	Read/Write

Resolving dataspace and snapshot access rights

At dataspace level, access rights are resolved as follows:

- If a user has several rights defined through multiple profiles:
 - If the rights include restrictions, the minimum of the restrictive profile-rights associations is applied.
 - Otherwise, the maximum of the profile-rights associations is applied.
- If the user has no rights defined:
 - If the user is a built-in administrator or owner of the dataspace, read-write access is given for this dataspace.
 - Otherwise, the dataspace will be hidden.

Resolving dataset access rights

At the dataset level, the same principle applies as at the dataspace level. After resolving the access rights at the dataset level alone, the final access rights are determined by taking the minimum rights between the resolved dataspace rights and the resolved dataset rights. For example, if a dataspace is resolved to be read-only for a user and one of its datasets is resolved to be read-write, the user will only have read-only access to that dataset.

Resolving node access rights

At the node level, the same principle applies as at the dataspace and dataset levels. After resolving the access rights at the node level alone, the final access rights are determined by taking the minimum rights between the resolved dataset rights and the resolved node rights.

Specific access rights can be defined at the node level. If no specific access right is defined, the default access right is used for the resolution process.

Note

The resolution procedure is slightly different for table and table child nodes.

Special case for table and table child nodes

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

For each user-defined permission rule that matches one of the user's profiles, the access rights for *N* are either:

1. The locally defined access rights for *N*;
2. Inherited from the access rights defined on the table node;
3. Inherited from the default access rights for dataset values.

All matching user-defined permission rules are used to resolve the access rights for *N*. Resolution is done according to the [restriction policy](#) [p 304].

The final resolved access rights will be the minimum between the dataspace, dataset and the resolved access right for *N*.

Resolving dynamic rules

There are three levels of resolution for dynamic access right rules: dataset, record and node. Since only one programmatic access rule can be set for a given level, the last rule set is the one used by the resolution procedure. However, a scripted rule can be specified on top of a programmatic rule at the table level.

Rule resolution on dataset

For a dataset, the last rule set is considered as the resolved rule

Rule resolution on record

For a record, the resolved rule is the minimum between the resolved rule set on the dataset and the rule set on this record.

See `SchemaExtensionsContext.setAccessRuleOnOccurrenceAPI` for more details.

Rule resolution on node

For a node that is a child node of a record, the resolved rule is the minimum between the resolved rule on the record and the rule set on this node.

For a child node of a dataset, the resolved rule is the minimum between the resolved rule set on the dataset and the rule set on this node.

See `SchemaExtensionsContext.setAccessRuleOnNodeAPI` for more details.

Display policy for foreign key drop-down menus

If a record is hidden due to access rules, it will not appear in foreign key drop-down menus.

Attention

The resolved access rights on a dataset or dataset node is the minimum between the resolved access rights defined in the user interface and the resolved dynamic rules, if any.

48.6 Resolving permissions on services

User services give the possibility to execute specific and advanced features from the user interface. Depending on their definition, these services can be called from a menu, as an action in a workflow, as a perspective item, or can be executed directly from a URL as a [Web component](#) [p 232].

Voir aussi [Overview](#) [p 661]

The permissions of a service are resolved as the service is called from the user interface, namely:

- During the execution, just before the service is displayed.
If the permission resolved in the user context is not enabled, a restriction message is displayed in place of the service.
- During the display of menus if the service is defined as displayable in menus.
If the permission resolved in the context for the user is not enabled, the service will not be displayed in the menu.

Thus, upon every request the resolution of permissions for a service is carried out as follows, in the following order and as long as conditions are respected:

1. The service activation has to correspond to the current context. This activation considers:
 - the selected entity type (dataset, table, record, etc.);
 - static activation rules defined within the `UserServiceDeclaration.defineActivationAPI` method;
 - the potential dynamic activation rule ([ServiceActivationRule](#) [p 303]) also defined within the `UserServiceDeclaration.defineActivationAPI` method.
2. When the service is activated for the current context, permissions for the user session will be evaluated:
 - If permissions have been defined via the user interface for the current user (or for their roles), their resolution must return enabled.

For more information, please refer to the [Resolving user-defined rules](#) [p 308] section.

 - If a [global permission rule](#) [p 303] is defined for the service, it must return enabled for the context provided (see `ServicePermissionRuleContextAPI`).
 - If a [local permission rule](#) [p 303] is defined for the selected node, it must return enabled for the context provided (see `ServicePermissionRuleContextAPI`).

Resolving user-defined rules

Example

In this example, there are two users belonging to different roles and profiles:

User	Profiles
User 1	• user1 • role A • role B
User 2	• role C • role D

The permissions associated with the roles and profiles defined on the dataset level are as follows:

Profile	Built-in service create (@creation)	Built-in service duplicate (@duplicate)	Built-in service compare (@compare)	Custom service 1 (custom1)	Custom service 2 (custom2)	Restriction policy
user1	Enabled	Disabled	Enabled	Disabled	Enabled	No
Role A	Enabled	Enabled	Disabled	Enabled	Disabled	Yes
Role B	Enabled	Disabled	Enabled	Enabled	Disabled	Yes
Role C	Enabled	Enabled	Disabled	Disabled	Disabled	No
Role D	Enabled	Disabled	Disabled	Enabled	Disabled	No

The services available to each user after permission resolution are as follows:

Users	Available services
User 1	Built-in service create (@creation)
	Custom service 1 (custom1)
User 2	Built-in service create (@creation)
	Built-in service duplicate (@duplicate)
	Custom service 1 (custom1)

Voir aussi [Resolving user-defined rules](#) [p 304]

48.7 Resolving permissions on actions

Actions are low-level operations for EBX® object manipulation on which it is possible to define execution rights for a profile. Unlike permissions on user services, which only impact the user interface, these rights are also applicable when an operation is carried out programmatically (i.e. via a Procedure^{API}) or indirectly (for example during data import, actions on the table (create, override, occult and delete) are evaluated).

Here is the list of actions on which rights can be defined:

Action object	Available actions
Dataspace	Create a child dataspace
	Create a snapshot
	Launch a merge
	Export an archive
	Import an archive
	Close the dataspace
	Close the snapshot
	Create a dataset
Dataset	Duplicate the dataset
	Delete the dataset
	Activate/deactivate the dataset
	Create a view
Table	Create a new record
	Override records
	Occult records
	Delete records

For the resolution of permissions on actions, only the permissions defined via the user interface for the current user (or their roles) will be taken into account, the restriction policy being applied as for any other permission defined via the user interface.

For more information, please refer to the [Resolving user-defined rules](#) [p 311] section.

Resolving user-defined rules

Example

In this example, we have two users belonging to different roles and profiles:

User	Profiles
User 1	• user1 • role A • role B
User 2	• role C • role D

Rights associated with roles and profiles on the actions of a given table are as follows:

Profile	Create a record	Override a record	Occult a record	Delete a record	Restriction policy
user1	No	Yes	No	Yes	No
Role A	Yes	No	Yes	No	Yes
Role B	No	Yes	Yes	No	Yes
Role C	Yes	No	No	No	No
Role D	No	No	Yes	No	No

The actions available to each user after resolving the rights are as follows:

Users	Available actions
User 1	Occult a record
User 2	Create a record
	Occult a record

Voir aussi [Resolving user-defined rules](#) [p 304]

CHAPITRE 49

Criteria editor

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Conditional blocks](#)
3. [Atomic criteria](#)

49.1 Overview

The criteria editor is included in several different areas of the user interface. It allows defining table filters, as well as validation and computation rules on data. This editor is based on the XPath 1.0 W3C Recommendation.

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

Voir aussi [Supported XPath syntax](#) [p 251]

49.2 Conditional blocks

Conditional blocks are made up of atomic criteria and other conditional blocks. They express a condition based on the criteria. The following types of blocks exist:

- **No criteria match:** None of the criteria in the block match.
- **Not all criteria match:** At least one criterion in the block does not match.
- **All criteria match:** All criteria in the block match.
- **At least one criterion matches:** One or more of the criteria match.

49.3 Atomic criteria

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

Field	Specifies the field of the table to which the criterion applies.
Operator	Specifies the operator used. Available operators depend on the data type of the field.
Value	Specifies the value or expression. See Expression [p 314] below.
Code only	If checked, specifies searching the underlying values for the field instead of labels, which are searched by default.

Expression

The expression can either be a fixed value or a formula. When creating a filter, only fixed values are authorized. During creation of a validation or computation rule, a formula can be created using the wizard.

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

CHAPITRE 50

Search

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Search strategies for string fields](#)
3. [Search strategy for primary key fields](#)
4. [Excluding a field from search \('Void' indexing\)](#)
5. [Assigning a search strategy to a field](#)

50.1 Overview

A search strategy defines how a field is indexed and queried. Any field is associated with a default search strategy, primarily based on its data type.

Search strategies are specified in the Data Model Assistant:

- when editing a field, its search strategies can be set in the 'Extensions' tab;
- at the data model level, custom search strategies can be specified, under 'Extensions > Search' element in the left pane;

Voir aussi [Recherche rapide](#) [p 128]

Value-labeling

Value-labeling is a global feature in EBX® to display user-friendly labels instead of raw values. For example, in the user interface, a foreign key field displays the label of the linked record, or a field based on a static enumeration displays the localized label associated with the raw value, as specified by the data model.

If a field supports value-labeling, the Quick search and the sort in the user interface usually apply on the displayed label, to preserve an intuitive user interface.

There are some exceptions, where raw value is still used by the quick search and the sort operation:

- Boolean and Locale data types. This is a temporary limitation.
- Programmatic labels and programmatic enumeration constraints (a foreign key specifying a `TableRefDisplay` or whose display depends on a `UILabelRenderer` specified on the target table, or a field constrained by a `ConstraintEnumeration`). It is recommended to use alternative solutions (display patterns and foreign keys).

- Enumeration constraint defined using another node (`<osd:enumeration osd:path=...`). It is recommended to use an alternative solution (a foreign key).

Obviously, if a field is displayed through a `UIWidget` (or a `UIBean`), to preserve an intuitive user interface, it is expected for the custom component to display the label (or the value, if this field does not enable value-labeling).

Limitations

In general, the following fields are not included in the Quick search and they are not optimized for other operators:

- computed fields with non-local dependency;
- inherited fields.

In the specific cases of inherited dataset or history view, legacy search is used. This implies that the size of the table cannot be quickly estimated, and might not be presented in the UI. It also implies that Quick search:

- considers all searchable fields (including computed fields with non-local dependency);
- behaves like a 'contains' (Lucene syntax cannot be used);
- may perform poorly on tables with large volumes.

Regarding the Advanced Search pane, all fields will be available, except those of type `osd:locale` which are not defined as enumerations, and those of type `osd:resource`.

50.2 Search strategies for string fields

Basic built-in strategies for strings

'Texte'	La stratégie de recherche 'Texte' est adaptée pour les champs contenant plusieurs mots: descriptions, textes ou commentaires. Cette stratégie prend en charge la recherche plein texte et la recherche approchée. Le tri et certaines fonctions, tels que les opérateurs «égal» et «commence par», ne sont pas pertinents et ne sont pas pris en charge. L'indexation de cette stratégie est légère, consommant peu d'espace disque. See also Recherche rapide [p 128]
'Code'	La stratégie de recherche 'Code' est destinée aux codes et identifiants. Les valeurs sont considérées comme un seul symbole (ou mot) court, permettant tout type de filtre sensible à la casse et insensible à la casse. La recherche plein texte n'est pas pertinente, elle est remplacée par un filtre d'inclusion (sans transformation et ne tenant pas compte de la casse).
'Nom'	La stratégie de recherche 'Nom' est destinée aux noms et aux libellés qui ne contiennent que quelques mots. Cette stratégie permet la même recherche avancée que la stratégie 'Texte', tout en permettant le tri et en prenant également en charge les divers filtres de 'Code'. La stratégie 'Nom' offre le plus de capacités, mais son indexation consomme aussi plus d'espace disque. Il est conseillé de définir 'Texte', 'Code', voire 'Exclu de la recherche' si le champ le permet.

Default strategy for string fields

The 'Name' strategy is applied to string fields by default, except:

- If the field is part of the primary key, it is set by default to 'Code'.
- If the field is a foreign key, it is forced to 'Code' and cannot be changed.
- If the field has a built-in datatype extending `xs:string`, then it has a strategy relevant to its datatype; for instance `osd:text`, `osd:Name`, `osd:email`, `osd:html`, etc.

As the default strategy 'Name' can be irrelevant and consumes more disk space, the data model compilation reports warnings for fields with the 'Name' strategy set as default, so as to ensure that strategies are defined on purpose. We advise to choose the 'Text' strategy, when the length of the expected values is greater than 80, as a rough estimate. Long values (> 32766 bytes once encoded into UTF-8) will not be fully indexed with the 'Name' or 'Code' strategy. Quick search is not affected, but sorting will consider only the first 1000 characters, and some operators ('equals' and 'ends-with', ...) will not return the correct results.

Advanced custom strategies

Some strategies accept parameters, for example to define stop words, or a specific language. This is done by creating a record in the 'Search strategies' table of the 'Search' data model extension. The new parameterized strategy will be available for selection in the 'Extension' tab, for compatible fields.

50.3 Search strategy for primary key fields

Primary key fields must have a sortable search strategy. This excludes the 'Void' strategy for all data types, and the 'Text' strategy for strings.

50.4 Excluding a field from search ('Void' indexing)

The 'Excluded from search' (or void) strategy deactivates indexing, making filter, search, or sort impossible. It is available for all data types, and is intended for fields that are never queried. Values can still be accessed through their record. Disabling the indexing reduces the disk space consumed and speeds up some operations like data import.

50.5 Assigning a search strategy to a field

A search strategy can be associated with a field, by means of a **search template** SearchTemplate^{APT}. This is done in the 'Extension' tab of the field, in the Data Model Assistant. Assigning multiple search strategies to a field requires registering additional search templates into a module. Only the addons EBX® Information Search and EBX® Match and merge are concerned by additional search templates.

CHAPITRE 51

Performance guidelines

Ce chapitre contient les sections suivantes :

1. [Basic performance checklist](#)
2. [Checklist for dataspace usage](#)
3. [Memory management](#)
4. [Validation performance](#)
5. [Mass updates](#)
6. [Accessing tables](#)
7. [Applications server](#)

51.1 Basic performance checklist

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

Expensive programmatic extensions

For reference, the table below details the programmatic extensions that can be implemented.

Use case	Programmatic extensions that can be involved
Validation	• programmatic constraints <code>Constraint^{API}</code> • computed values <code>ValueFunction^{API}</code>
Table access	• record-level permission rules <code>SchemaExtensionsContext.setAccessRuleOnOccurrence^{API}</code> • programmatic filters <code>AdaptationFilter^{API}</code>
EBX® content display	• computed values <code>ValueFunction^{API}</code> • UI Components <code>UIBeanEditor^{API}</code> • node-level permission rules <code>SchemaExtensionsContext.setAccessRuleOnNode^{API}</code>
Data update	• triggers Package <code>com.orchestranetworks.schema.trigger^{API}</code>

For large volumes of data, using algorithms of high computational complexity has a serious impact on performance. For example, the complexity of a constraint's algorithm is $O(n^2)$. If the data size is 100, the resulting cost is proportional to 10 000 (this generally produces an immediate result). However, if the data size is 10 000, the resulting cost will be proportional to 10 000 000.

Another reason for slow performance is calling external resources. Local caching usually solves this type of problem.

If one of the use cases above displays poor performance, it is recommended to track the problem, either by code analysis or by using a Java profiling tool.

Directory integration

Authentication and permissions management involve the [user and roles directory](#) [p 451].

If a specific directory implementation is deployed and accesses an external directory, it can be useful to ensure that local caching is performed. In particular, one of the most frequently called methods is `Directory.isUserInRoleAPI`.

Aggregated lists

In a data model, when an element's cardinality constraint `maxOccurs` is greater than 1 and no `osd:table` is declared on this element, it is implemented as a Java `List`. This type of element is called an [aggregated list](#) [p 545], as opposed to a table.

It is important to consider that there is no specific optimization when accessing aggregated lists, in terms of iterations, user interface display, etc. Besides performance concerns, aggregated lists are limited with regard to many functionalities that are supported by tables. See [tables introduction](#) [p 549] for a list of these features.

Attention

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

51.2 Checklist for dataspace usage

[Dataspaces](#) [p 100] are an invaluable tool for managing complex data life cycles. While this feature brings great flexibility, it also implies a certain overhead, which should be taken into consideration for optimizing usage patterns.

This section reviews the most common performance issues that can appear in case of an intensive use of many dataspace containing large tables, and how to avoid them.

Note

Sometimes, the use of dataspace is not strictly needed. As an extreme example, consider the case where every transaction triggers the following actions:

1. A dataspace is created.
2. The transaction modifies some data.
3. The dataspace is merged, closed, then deleted.

In this case, no future references to the dataspace are needed, so using it to make isolated data modifications is unnecessary. Thus, using `ProcedureAPI` already provides sufficient isolation to avoid conflicts from concurrent operations. It would then be more efficient to directly do the modifications in the target dataspace, and get rid of the steps which concern branching and merging.

For a developer-friendly analogy, referring to a source-code management tool (CVS, SVN, etc.): when you need to perform a simple modification impacting only a few files, it is probably sufficient to do so directly on the main branch. In fact, it would be neither practical nor sustainable, with regard to file tagging/copying, if every file modification involved branching the whole project, modifying the files, then merging the dedicated branch.

Insufficient memory

When a table is accessed, the EBX® Java memory cache is used. It ensures a much more efficient access to data when this data is already loaded in the cache. However, if there is not enough space for working data, swaps between the Java heap space and the underlying database can heavily degrade overall performance.

Such an issue can be detected by monitoring the [monitoring log file](#) [p 322]. If it occurs, various actions can be considered:

- reducing the number of child dataspace that contain large tables;
- or (obviously) allocating more memory, or optimizing the memory used by applications for non-EBX® objects.

Voir aussi [Memory management](#) [p 322]

Reorganizing database tables

As with any database, inserting and deleting large volumes of data may lead to fragmented data, which can deteriorate performance over time. To resolve the issue, reorganizing the impacted database tables is necessary. See [Monitoring and cleanup of the relational database](#) [p 417].

A specificity of EBX® is that creating dataspace and snapshots adds new entries to tables `GRS_DTR` and `GRS_SHR`. When poor performance is experienced, it may be necessary to schedule a reorganization of these tables, for large repositories in which many dataspace are created and deleted.

Voir aussi [Monitoring and cleanup of the relational database](#) [p 417]

51.3 Memory management

Monitoring

Indications of EBX® load activity are provided by monitoring the underlying database, and also by the ['monitoring' logging category](#) [p 376].

If the numbers for *cleared* and *built* objects remain high for a long time, this is an indication that EBX® is swapping on the application server.

Tuning hardware resources

Disk space

The master data stored in the database is additionally indexed into persistent Lucene indexes, serving to accelerate all the queries issued by EBX®. This comes at the cost of additional storage space: a rule of thumb is to plan for 10 times the space occupied in the relational database.

Disk latency

In order to maintain good overall performance, it is particularly important for the disk [storing the Lucene indexes](#) [p 373] to have low latency.

Memory allocated to the application server

Since the query engine retrieves the necessary information from persistent storage, the memory allocated to the Java Virtual Machine (usually specified by the `-Xmx` parameter) can be kept low. We recommend to stay below 32 GB, which should fit all reasonable use cases, and allow benefiting from the [compressed Oups](#) feature.

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

Memory allocated to the operating system

On the OS running the application server, it is important to leave sufficient room to the OS cache, letting it optimize access to the persistent Lucene indexes. Indeed, once these have been loaded from the file system, the OS use its memory cache to speed up subsequent accesses to this same data, and avoid reloading it from the disk every time. This is only possible if sufficient RAM has been left for this purpose.

51.4 Validation performance

The internal validation framework will optimize the work required during successive requests to update the validation report of a dataset or a table. The incremental validation process behaves as follows:

- The first call to a dataset or table validation report performs a full validation of the dataset or the table.
- The next call to the validation report will compute the changes performed since the last validation. The validation report will be updated according to these changes.

- Validation reports are stored persistently in the TIBCO EBX® repository. This reduces the amount of memory dedicated to validation reports when datasets have a large amount of validation messages. Also, validation reports are not lost when the application server restarts.
- Validation reports can be reset using the API or manually in the user interface by an administrator user (this option is available from the validation report section in EBX®). As a consequence, resetting validation reports must be used with caution since associated datasets or tables will be fully revalidated during the next call to their validation reports.

See `Adaptation.resetValidationReportAPI` for more information.

Certain constraints are systematically re-validated, even if no updates have occurred since the last validation. These are the constraints with *unknown dependencies*. An element has unknown dependencies if:

- It specifies a **programmatic constraint** `ConstraintAPI` in the default *unknown dependencies* mode.
- It declares a **computed value** `ValueFunctionAPI`, or it declares a dynamic facet that depends on an element that is itself a **computed value** `ValueFunctionAPI`.
- It is an [inherited field](#) [p 290] or it declares a dynamic facet that depends on a node that is itself an [inherited field](#) [p 290].

Consequently, on large tables, it is recommended to:

- Avoid constraints with unknown dependencies (or at least to minimize the number of such constraints). 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.dependenciesAPI`.
- To use **constraints on tables** `ConstraintOnTableAPI` instead of **programmatic constraints** `ConstraintAPI` defined at field level. Indeed, if a table defines constraints at field level, then the validation process will iterate over all the records to check if the value of the associated field complies with the constraint. Using **constraints on tables** `ConstraintOnTableAPI` gives the opportunity to execute optimized queries on the whole table.
- Avoid the use of the facet [pattern](#) since its check is not optimized on large tables. That is, if a field defines this facet then the validation process will iterate over all the records to check if the value of the associated field complies with the specified pattern.

51.5 Mass updates

Mass updates can involve several hundred thousands of insertions, modifications and deletions. These updates are usually infrequent (usually initial data imports), or are performed non-interactively (nightly batches). Thus, performance for these updates is less critical than for frequent or interactive operations. However, similar to classic batch processing, it has certain specific issues.

Transaction threshold

It is generally not advised to use a single transaction when the number of atomic updates in the transaction is beyond the order of 10^4 . Large transactions require a lot of resources, in particular, memory, from EBX® and from the underlying database.

To reduce transaction size, it is possible to:

- Specify the property [ebx.manager.import.commit.threshold](#) [p 383]. However, this property is only used for interactive archive imports performed from the EBX® user interface.
- Explicitly specify a **commit threshold** `ProcedureContext.setCommitThresholdAPI` inside the batch procedure.
- Structurally limit the transaction scope by implementing `ProcedureAPI` for a part of the task and executing it as many times as necessary.

On the other hand, specifying a very small transaction size can also hinder performance, due to the persistent tasks that need to be done for each commit.

Note

If intermediate commits are a problem because transactional atomicity is no longer guaranteed, it is recommended to execute the mass update inside a dedicated dataspace. This dataspace will be created just before the mass update. If the update does not complete successfully, the dataspace must be closed, and the update reattempted after correcting the reason for the initial failure. If it succeeds, the dataspace can be safely merged into the original dataspace.

Triggers

If required, triggers can be deactivated using the method `ProcedureContext.setTriggerActivationAPI`.

51.6 Accessing tables

Functionalities

Tables are commonly accessed through EBX® UI, data services and also through the `RequestAPI` and `QueryAPI` APIs. This access involves a unique set of functions, including a *dynamic resolution* process. This process behaves as follows:

- **Inheritance:** Inheritance in the dataset tree takes into account records and values that are defined in the parent dataset, using a recursive process. Also, in a root dataset, a record can inherit some of its values from the data model default values, defined by the `xs:default` attribute.
- **Value computation:** A node declared as an `osd:function` is always computed on the fly when the value is accessed. See `ValueFunction.getValueAPI`.
- **Filtering:** An [XPath predicate](#) [p 251], a **programmatic filter** `AdaptationFilterAPI`, or a record-level **permission rule** `SchemaExtensionsContext.setAccessRuleOnOccurrenceAPI` requires a selection of records.
- **Sort:** A sort of the resulting records can be performed.

Query on tables

Architecture and design

In order to improve the speed of operations on tables, persistent Lucene indexes are managed by the EBX® engine.

Attention

Faster access to tables is ensured if indexes are ready and maintained in the OS memory cache. As mentioned [above](#) [p 322], it is important for the OS to have enough space allocated.

Performance considerations

The query optimizer favors the use of indexes when computing a request result. If a query cannot take advantage of the indexes, it will be resolved in Java memory, and experience poor performance on large volumes. The following guidelines apply:

Attention

- Only XPath predicates and SQL queries can benefit from index optimization.
- Some fields and some datasets cannot be indexed, as described in section [Limitations](#) [p 316].
- XPath predicates on a multivalued field cannot benefit from index optimization, except for the `osd:search` function.
- XPath predicates using the `osd:label` function cannot benefit from index optimization

If indexes have not yet been built, additional time is required to build and persist the indexes, on the first access to the table.

Accessing the table data blocks is required when the query cannot be computed against any index (whether for resolving a rule, filter or sort), as well as for building the index. If the table blocks are not present in memory, additional time is needed to fetch them from the database.

It is possible to get information through the [monitoring](#) [p 322] and request logging categories.

Accessing and modifying a table

The following access lead to poor performance, and must be avoided:

- Access a table after a few modifications, repeatedly. It implies the index state to be refreshed after each modification. The cost of refreshing makes this pattern ineffective. Instead, perform a single query and apply the modification when browsing the results.
- If there is an ongoing access to the same table, concurrently to the previous case, it prevents outdated index files to be deleted. As a consequence, the size of the index on disk increases, and the server may run out of disk space in extreme cases. When the concurrent access is closed, the index size is back to normal. This is usually a sign that a Request or a Query is not properly closed.

Voir aussi

`RequestResult.close`^{API}

`QueryResult.close`^{API}

Other operations on tables

The new records creations or record insertions depend on the primary key index. Thus, a creation becomes almost immediate if this index is already loaded.

Setting a fetch size

In order to improve performance, a fetch size should be set according to the expected size of the result of the request on a table. If no fetch size is set, the default value will be used.

- On a history table, the default value is assigned by the JDBC driver: 10 for Oracle and 0 for PostgreSQL.

Attention

On PostgreSQL, the default value of 0 instructs the JDBC driver to fetch the whole result set at once, which could lead to an `OutOfMemoryError` when retrieving large amounts of data. On the other hand, using `fetchSize` on PostgreSQL will invalidate server-side cursors at the end of the transaction. If, in the same thread, you first fetch a result set with a `fetchsize`, then execute a procedure that commits the transaction, then, accessing the next result will raise an exception.

Voir aussi

`Request.setFetchSize`^{API}

`RequestResult`^{API}

51.7 Applications server

Configuration

To speed up the persisted data access, it is required to perform a `ebx-1z4.jar` native installation.

See [Data compression library](#) [p 340] for more information.

Startup

To speed up the web applications server startup, the [JAR files scanner](#) [p 347] should be configured.

Guide d'administration (en anglais)

CHAPITRE 52

Administration overview

The Administration section in TIBCO EBX® is the main point of entry for all administration tasks. In this overview are listed all the topics that an administrator needs to master. Click on your topic of interest in order to access the corresponding chapter or paragraph in the documentation.

Ce chapitre contient les sections suivantes :

1. [Repository management](#)
2. [Disk space management](#)
3. [Data model](#)
4. [Perspectives](#)
5. [Administrative delegation](#)

52.1 Repository management

For storage optimization, it is recommended to maintain a repository (persistence RDBMS) to the necessary minimum. To this end, it is recommended to regularly perform a purge of snapshots and obsolete dataspace and to consider using a backup file system.

See also [Cleaning up dataspace, snapshots, and history](#) [p 417] and [Deleting dataspace, snapshots, and history](#) [p 418].

It is also possible to archive files of the file system type in order to reduce the storage costs, see [EBX® monitoring](#) [p 416].

Administration tasks can be scheduled by means of the task scheduler, using built-in tasks, see [Task scheduler](#) [p 465].

Object cache

EBX® maintains an object cache in memory. The object cache size should be managed on a case by case basis according to specific needs and requirements (pre-load option and pre-validate on the reference dataspace, points of reference, and monitoring), while continuously monitoring the repository health reports (./ebxLog/monitoring.log).

See [Memory management](#) [p 322].

Obsolete contents

Keeping obsolete contents in the repository can lead to a slow server startup and slow responsiveness of the interface. It is strongly recommended to delete obsolete content.

For example: datasets referring to deleted data models or undeployed add-on modules. See [Deploying and registering TIBCO EBX® add-ons](#) [p 393].

Workflow

Cleanup

The workflow history and associated execution data have to be cleaned up on a regular basis.

The workflow history stores information on completed workflows, their respective steps and contexts. This leads to an ever-growing database containing obsolete history and can thus lead to poor performance of the database if not purged periodically. See [Workflow history](#) [p 464] for more information.

Email configuration

It is required to configure workflow emails beforehand in order to be able to implement workflow email notifications. See [Configuration](#) [p 462] for more information.

52.2 Disk space management

Purge of logs

The log file size will vary according to the log level (and to the selected severity level) and disk space needs to be accordingly managed.

An automatic purge is provided with EBX®, allowing to define how many days should log files be stored. After the defined period, log files are deleted.

Any customized management of the purge of logs (backup, archiving, etc.) is the user's responsibility.

```
#####
## 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
```

Audit trail

EBX® is provided with a default audit trail manager. Any customized management (including purge, backups, etc.) is the user's responsibility.

If the audit trail is unwanted, it is possible to fully deactivate it. See [Activating the XML audit trail](#) [p 375] and [Audit trail](#) [p 471] for more information.

52.3 Data model

Publication management

The management of publications of [embedded data models](#) [p 93]. See [Data model administration](#) [p 455] for more information on the management of these publications and the administration tasks that can be performed (delete, import and export).

Refresh data models

It is possible to update the data models that are using XML Schema documents not managed by EBX®. See [Data model refresh tool](#) [p 525] for more information.

52.4 Perspectives

EBX® offers extensive UI customization options. Simplified interfaces ([Recommended perspectives](#)) [p 436] dedicated to each profile accessing the system can be parameterized by the administrator. According to the profile of the user logging in, the interface will offer more or less options and menus. This allows for a streamlined work environment.

See [Advanced perspective](#) [p 424] for more information.

52.5 Administrative delegation

EBX® is provided with the built-in administrator profile by default. An administrator can delegate administrative rights to a non-administrator user, either for specific actions or for all activities.

The administrative delegation is defined under 'Administration' in the [global permissions](#) [p 423] profile.

Access to the administration section can be granted to specific profiles via the global permissions in order to delegate access rights on corresponding administration datasets.

If all necessary administrative rights have been delegated to non-administrator users, it becomes possible to disable the built-in 'Administrator' role.

Voir aussi [Configuring the user and roles directory](#) [p 374]

Installation & configuration

CHAPITRE 53

Supported environments

Ce chapitre contient les sections suivantes :

1. [Browsing environment](#)
2. [Supported application servers](#)
3. [Supported databases](#)

53.1 Browsing environment

Supported web browsers

The TIBCO EBX® web interface supports the following browsers:

Microsoft Edge chromium	As Microsoft Edge chromium is updated frequently and it is not possible to deactivate automatic updates, TIBCO Software Inc. only tests and makes the best effort to support the latest version available.
Microsoft Edge	Minimum supported version is 44 Compatibility mode is not supported.
Microsoft Internet Explorer 11	Compatibility mode is not supported. Performance limitations: page loading with IE11 is two times slower. This issue is observed when forms have many input components, and particularly many multi-occurrence groups. Graphical layout: graphical rendering in IE11 can slightly differ from other browsers (for example, the alignment of some labels, icons and other components can be off by a few pixels).
Mozilla Firefox ESR 68 (see details)	As Mozilla Firefox is updated frequently, TIBCO Software Inc. only fully supports version ESR 68. See Mozilla Firefox ESR for more details.
Google Chrome	As Google Chrome is updated frequently and it is not possible to deactivate automatic updates, TIBCO Software Inc. only tests and makes the best effort to support the latest version available.

Screen resolution

The minimum screen resolution for EBX® is 1024x768.

Refreshing pages

Browser page refresh is not supported by EBX®. When a page refresh is performed, the last user action is re-executed, and therefore could cause issues. It is thus imperative to use the action buttons and links offered by EBX® instead of refreshing the page.

'Previous' and 'Next' buttons

The 'previous' and 'next' buttons of the browser are not supported by EBX®. When navigating through page history, an obsolete user action is re-executed, and therefore could cause issues. It is thus imperative to use the action buttons and links offered by EBX® rather than the browser buttons.

Browser configuration

The following features must be activated in the browser configuration, for the user interface to work properly:

- JavaScript
- Ajax
- Pop-ups

Attention

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

Limitations

Some browsers may have a limitation on the number of iframes that can be embedded. If this is the case, it limits to the number of items that can be pushed in the breadcrumb. Please check the browser documentation for more details.

53.2 Supported application servers

EBX® supports the following configurations:

- Java Runtime Environment: JRE 8 or 11, which necessarily includes the limitations specified by the Java Virtual Machine implementation vendor. For example, for JRE and JDK 8, 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 3.0 (inclusive) up to 5.0 (exclusive): for example Tomcat 7.0 (inclusive) up to 10.0 (exclusive), WebSphere Application Server 8.5.5 or higher, WebLogic Application Server 14c or higher, JBoss EAP 6.0 or higher. See [Java EE deployment overview](#) [p 347].
- The application server must support the JSON Processing 1.1 ([JSR 374](#)), or allow the uses of the implementation embedded in the `ebx.jar` library. For example, Tomcat does not provide any library to support this specification (only the embedded one can be used), WebSphere Application Server allows reversing the classloading system (making the embedded one a priority), WebLogic Application Server 14c or higher supports this specification, JBoss EAP allows including or excluding the available libraries.
- The application server must use UTF-8 encoding for HTTP query strings from EBX®. 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 instruct the server to use the encoding of the request body by setting the parameter `useBodyEncodingForURI` to 'true' in `server.xml`.

Attention

- Limitations apply regarding clustering and hot deployment/undeployment:

Clustering: EBX® does not include a cache synchronization mechanism, thus it cannot be deployed into a cluster of active instances. See [Technical architecture](#) [p 412] for more information.

Hot deployment/undeployment: EBX® does not support hot deployment/undeployment of web applications registered as EBX® modules, or of EBX® built-in web applications.

- WebSphere Application Server's Java SDKs under version 8.0.4.10 are incompatible with the embedded Apache Calcite 1.26.0 third-party library. It is highly recommended to use the latest Java SDK available and compatible with the application server.

53.3 Supported databases

The EBX® repository supports the relational database management systems listed below, with the suitable JDBC drivers. It is important to follow the database vendor recommendations and update policies regarding the database itself, as well as the JDBC driver.

Oracle Database 12c or higher (but excluding 18c).	The distinction of null values bears certain limitations. On simple <code>xs:string</code> elements, Oracle does not support the distinction between empty strings and null values. See Empty string management [p 586] for more information. The user with which EBX® connects to the database requires the following privileges: • CREATE SESSION, • CREATE TABLE, • ALTER SESSION, • CREATE SEQUENCE, • A non-null quota on its default tablespace.
PostgreSQL 10 or higher.	The user with which EBX® connects to the database needs the CONNECT privilege on the database hosting the EBX® repository. Other than this, the default privileges on the public schema of this database are suitable. Also, see this limitation [p 285] regarding the evolution of datamodels in mapped modes.
Amazon Aurora PostgreSQL 2.3 (compatible with PostgreSQL 10.7) or higher.	The comments in the above section for PostgreSQL apply.
Google Cloud SQL for PostgreSQL 10 or higher.	The comments in the above section for PostgreSQL apply.
SAP HANA Database 2.0 or Higher.	When using SAP HANA Database as the underlying database, certain schema evolutions are not supported. It is, for example, impossible to reduce the length of a column; this is a limitation of HANA, as mentioned in the SQL reference guide: "For row table, only increasing the size of VARCHAR and NVARCHAR type column is allowed."
Microsoft SQL Server 2012 SP4 or higher.	When used with Microsoft SQL Server, EBX® uses the default database collation to compare and sort strings stored in the database. This applies to strings used in the data model definition, as well as data stored in history tables. The default database collation can be specified when the

database is created. Otherwise, the collation of the database server is used. To avoid naming conflicts or unexpected behaviors, a case- and accent-sensitive collation must be used as the default database collation (the collation name is suffixed by "CS_AS" or the collation is binary).

The default setting to enforce transaction isolation on SQL Server follows a pessimistic model. Rows are locked to prevent any read/write concurrent accesses. This may cause liveliness issues for mapped tables (history or relational). To avoid such issues, it is recommended to activate [snapshot isolation](#) on your SQL Server database.

The user with which EBX® connects to the database requires the following privileges:

- CONNECT, SELECT and CREATE TABLE on the database hosting the EBX® repository,
- ALTER, CONTROL, UPDATE, INSERT, DELETE on its default schema.

Microsoft Azure SQL Database

EBX® has been qualified on Microsoft Azure SQL Database v12 (12.00.700), and is regularly tested to verify compatibility with the current version of the Azure database service.

When used with Microsoft Azure SQL, EBX® uses the default database collation to compare and sort strings stored in the database. This applies to strings used in the data model definition, as well as data stored in history tables. The default database collation can be specified when the database is created. Otherwise, the database engine server collation is used. To avoid naming conflicts or unexpected behaviors, a case- and accent-sensitive collation must be used as the default database collation (the collation name is suffixed by "CS_AS" or the collation is binary).

The user with which EBX® connects to the database requires the following privileges:

- CONNECT, SELECT and CREATE TABLE on the database hosting the EBX® repository,
- ALTER, CONTROL, UPDATE, INSERT, DELETE on its default schema.

H2 v1.4.196 or higher.

H2 is not supported for production environments.

The default H2 database settings do not allow consistent reads when records are modified.

For other relational databases, please contact the Support team at <https://support.tibco.com>.

Attention

In order to guarantee the integrity of the EBX® repository, it is strictly forbidden to perform direct modifications to the database (for example, using direct SQL writes).

Voir aussi

[*Repository administration*](#) [p 412]

[*Data source of the EBX® repository*](#) [p 345]

[*Configuring the EBX® repository*](#) [p 373]

CHAPITRE 54

Java EE deployment

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Software components](#)
3. [Embedded third-party libraries](#)
4. [Required third-party libraries](#)
5. [Web applications](#)
6. [Deployment details](#)
7. [Installation notes](#)

54.1 Introduction

This chapter details deployment specifications for TIBCO EBX® on a Java application server. For specific information regarding supported application servers and inherent limitations, see [Supported environments](#). [p 332]

54.2 Software components

EBX® uses the following components:

- Library `ebx.jar`
- [Embedded](#) [p 340] and [required](#) [p 340] third-party Java libraries
- [EBX® built-in web applications](#) [p 343] and optional [custom web applications](#) [p 343]
- [EBX® main configuration file](#) [p 371]
- [EBX® repository](#) [p 412]
- [Default user and roles directory](#) [p 451], integrated within the EBX® repository, or a third-party system (LDAP, RDBMS) for the user authentication

Voir aussi [Supported environments](#) [p 332]

54.3 Embedded third-party libraries

To increase EBX® independence and interoperability, it embeds its own third-party libraries. Even if some of them have been modified, preventing conflicts, others must remain unchanged since they are official Java APIs.

The ones that can produce conflicts are:

- Apache Geronimo JSON
- Javax Activation
- Javax Annotations
- Javax JSON Bind
- Javax SAAJ API
- Javax WS RS
- Javax XML Bind

For more information regarding the versions or the details of the Third-Party Library, please refer to the: `TIB_ebx_6.0.3_license.pdf`.

Since those libraries are already integrated, custom web applications should not include them anew, otherwise linkage errors can occur. Furthermore, they should not be deployed aside from the `ebx.jar` library for the same reasons.

54.4 Required third-party libraries

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

Data compression library

The library named `ebx-lz4.jar` must be deployed separately from `ebx.jar`. It contains several compression implementations: JNI dedicated architecture libraries and Java fallbacks. It is possible to ensure optimal compression and decompression performance for EBX® repository by following prerequisites. If prerequisites can not be validated, EBX® will function in Java fallbacks safe or unsafe, but its performance will be degraded. The default location for `ebx-lz4.jar` library is beside `ebx.jar`.

To verify the compression implementation actually used by the EBX® repository, please check the value of 'Compression' in 'Administration > System Information', section 'Repository information'. It should be 'JNI - validated' for optimal performance. Otherwise, it will be 'Java[Safe|Unsafe] - validated' for Java fallbacks.

Performance prerequisites

The JNI access is allowed to the following operating system architectures: i386, x86, amd64, x86_64, aarch64 or ppc64le. To verify this value, please check the value of 'Operating system architecture' in 'Administration > System Information', section 'System information'.

To enable JNI access for `ebx-1z4.jar`, the library should be loaded by the **system class loader** (also known as the application class loader). The deployment may be done by following the [specific instructions for your application server](#) [p 347].

Database drivers

The EBX® repository requires a database. Generally, the required driver is configured along with a data source, if one is used. Depending on the database defined in the main configuration file, one of the following drivers is required. Keep in mind that, whichever database you use, the version of the JDBC client driver must be equal to or higher than the version of the database server.

H2	Version 1.4.196 validated. Note that H2 is not supported in production environments. http://www.h2database.com/
Oracle JDBC	Oracle database 21c is validated on their latest patch set update. Determine the driver that should be used according to the database server version and the Java runtime environment version. Download the <code>ojdbc8.jar</code> certified library with JDK 8. Oracle database JDBC drivers download.
SQL Server JDBC	SQL Server 2012 SP4 and greater, with all corrective and maintenance patches applied, are validated. Remember to use an up-to-date JDBC driver, as some difficulties have been encountered with older versions. Include the <code>mssql-jdbc-8.4.1.jre8.jar</code> or <code>mssql-jdbc-8.4.1.jre11.jar</code> library, depending on the Java runtime environment version you use. Download Microsoft JDBC Driver 8.4.1 for SQL Server (zip).
PostgreSQL	PostgreSQL 10 and above validated Include the latest JDBC driver version 4.2 released for your database server and Java runtime environment. PostgreSQL JDBC drivers download.

Voir aussi

[Data source of the EBX® repository](#) [p 345]

[Configuring the EBX® repository](#) [p 373]

SMTP and emails

According to the web application server being used, the library `JavaMail` API for email management may already be provided, or must be added manually.

EBX® requires a library that is compatible with version 1.5.6 of this API. See [Activating and configuring SMTP and emails](#) [p 378] for more information on the configuration.

To facilitate manual installation, the `javax.mail-1.5.6.jar` has been provided and placed under the `ebx.software/lib/lib-mail` directory.

Voir aussi [JavaMail](#)

Secure Socket Layer (SSL)

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

- `jsse.jar`: <https://www.oracle.com/java/technologies/jsse-v103-for-cdc-v102.html>
- `ibmjse.jar`: <https://www.ibm.com/developerworks/java/jdk/security/>

Voir aussi [TIBCO EBX® main configuration file](#) [p 371]

Java Message Service (JMS)

When using JMS, version 1.1 or higher is required.

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

- For an application server based on Java EE (Java Platform Enterprise Edition), the required JMS provider library is available by default. See <https://www.oracle.com/java/technologies/java-ee-glance.html> 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 <https://www.oracle.com/java/technologies/java-se-glance.html> for more information.

Note

In EBX®, the supported JMS model is exclusively Point-to-Point (PTP). PTP systems allow working with queues of messages.

Voir aussi [TIBCO EBX® main configuration file](#) [p 371]

XML Catalog API

A library holding the XML Catalog API, introduced by the JAVA SE 9, is required if your web applications are running over a Java Runtime Environment 8 or below, except when a WebLogic 14c application server is used. To ease the installation steps, the following library has been bundled aside from `ebx.jar`, in the *EBX® CD*.

- `xml-apis-1.4.01.jar`, version 1.4.01, from August 20, 2011

See [Installation notes](#) [p 347] for more information.

54.5 Web applications

EBX® provides pre-packaged EARs that can be deployed directly if your company has no custom EBX® module web applications to add. If deploying custom web applications as EBX® modules,

it is recommended to rebuild an EAR containing the custom modules packaged at the same level as the built-in web applications.

For more information, see the note on [repackaging the EBX® EAR](#) [p 348] at the end of this chapter.

EBX® built-in web applications

EBX® includes the following built-in web applications.

Web application name	Description	Required
ebx	EBX® entry point, which handles the initialization on start up. See Deployment details [p 344] for more information.	Yes
ebx-root-1.0	EBX® root web application. Any application that uses EBX® requires the root web application to be deployed.	Yes
ebx-ui	EBX® user interface web application.	Yes
ebx-manager	EBX® user interface web application.	Yes
ebx-dma	EBX® data model assistant, which helps with the creation of data models through the user interface. Note: The data model assistant requires the ebx-manager user interface web application to be deployed.	Yes
ebx-dataservices	EBX® data services web application. Data services allow external interactions with the EBX® repository using the SOAP operations [p 713] and Web Services Description Language WSDL generation [p 705] standards or using the Built-in RESTful services [p 757]. Note: The EBX® web service generator requires the deployment of the ebx-manager user interface web application.	Yes

Custom web applications

It is possible to extend and customize the behavior of EBX® by deploying custom web applications which conform to the EBX® module requirements.

Voir aussi

[Packaging TIBCO EBX® modules](#) [p 513]

[Declaring modules as undeployed](#) [p 385]

54.6 Deployment details

Introduction

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

Attention

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

Voir aussi

[TIBCO EBX® main configuration file](#) [p 371]

[Supported application servers](#) [p 334]

User interface and web access

The web application 'ebx' (packaged as `ebx.war`) contains the servlet `FrontServlet`, which handles the initialization and serves as the sole user interface entry point for the EBX® 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 EBX® 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 performed using the attribute `crossContext` in the configuration file `server.xml`, as follows:

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

- When several EBX® 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 some 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 EBX® repository

Note

If the EBX® main configuration specifies the property `ebx.persistence.url`, then the environment entry below will be ignored by EBX® runtime. This option is only provided for convenience; it is always recommended to use a fully-configurable datasource. In particular, the size of the connection pool must be set according to the number of concurrent users. See [Configuring the EBX® repository](#) [p 373] for more information on this property.

The JDBC datasource for EBX® is specified in the deployment descriptor `WEB-INF/web.xml` of the 'ebx' web application as follows:

Reserved resource name	Default JNDI name	Description
<code>jdbc/EBX_REPOSITORY</code>	Weblogic: <code>EBX_REPOSITORY</code> JBoss: <code>java:/EBX_REPOSITORY</code>	JDBC data source for EBX® Repository. Java type: <code>javax.sql.DataSource</code>

Voir aussi

[Configuring the EBX® repository](#) [p 373]

[Rules for the database access and user privileges](#) [p 413]

Mail sessions

Note

If the EBX® 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 EBX® runtime. See [SMTP](#) [p 378] in the EBX® main configuration properties for more information on these properties.

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

Reserved resource name	Default JNDI name	Description
<code>mail/EBX_MAIL_SESSION</code>	Weblogic: <code>EBX_MAIL_SESSION</code> JBoss: <code>java:/EBX_MAIL_SESSION</code>	Java Mail session used to send emails from EBX®. Java type: <code>javax.mail.Session</code>

JMS connection factory

Note

If the EBX® main configuration does not activate JMS through the property `ebx.jms.activate`, the environment entry below will be ignored by the EBX® runtime. See [JMS](#) [p 379] in the EBX® main configuration properties for more information on this property.

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

Reserved resource name	Default JNDI name	Description	Required
jms/EBX_JMSConnectionFactory	Weblogic: EBX_JMSConnectionFactory JBoss: java:/ EBX_JMSConnectionFactory	JMS connection factory used by EBX® to create connections with the JMS provider configured in the operational environment of the application server. Java type: javax.jms.ConnectionFactory	Yes

Note

For deployment on WildFly, JBoss and WebLogic application servers with JNDI capabilities, you must update `EBX.ear` or `EBXForWebLogic.ear` for additional mappings of all required resource names to JNDI names.

JMS for data services

To configure data services to use JMS instead of the default HTTP, you must configure the [JMS connection factory](#) [p 346] and the following queues, declared in the `WEB-INF/web.xml` deployment descriptor of the 'ebx' web application. This is the only method for configuring JMS for data services.

When a SOAP request is received, the SOAP response is optionally returned if the header field `JMSReplyTo` is defined. If so, the fields `JMSCorrelationID` and `JMSType` are retained.

See [JMS](#) [p 379] for more information on the associated EBX® main configuration properties.

Note

If the EBX® main configuration does not activate JMS through the property `ebx.jms.activate`, then the environment entries below will be ignored by EBX® runtime. See [JMS](#) [p 379] in the EBX® main configuration properties for more information on this property.

Reserved resource name	Default JNDI name	Description	Required
<code>jms/EBX_QueueIn</code>	Weblogic: <code>EBX_QueueIn</code> JBoss: <code>java:/jms/EBX_QueueIn</code>	JMS queue for incoming SOAP requests sent to EBX® by other applications. Java type: <code>javax.jms.Queue</code>	No
<code>jms/EBX_QueueFailure</code>	Weblogic: <code>EBX_QueueFailure</code> JBoss: <code>java:/jms/EBX_QueueFailure</code>	JMS queue for failures. It contains incoming SOAP requests 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 [p 379] in the EBX® main configuration properties for more information on these properties.	No

JAR files scanner

To speed up the web applications server startup, the JAR files scanner configuration should be modified to exclude, at least, the `ebx.jar` and `ebx-addons.jar` libraries.

For example, on Tomcat, this should be performed in the `tomcat.util.scan.DefaultJarScanner.jarsToSkip` property from the `catalina.properties` file.

54.7 Installation notes

EBX® can be deployed on any Java EE application server that supports Servlet 3.0 up to 5.0 except. The following documentation on Java EE deployment and installation notes are available:

- [Installation note for JBoss EAP 7.1.x](#) [p 349]
- [Installation note for Tomcat 9.x](#) [p 355]
- [Installation note for WebSphere AS 9](#) [p 359]

- [Installation note for WebLogic 14c](#) [p 365]

Attention

- The EBX® installation notes on Java EE application servers do not replace the native documentation for each application server.
- These 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 EBX® 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 EBX® 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 EBX® modules, the EBX® web applications, as well as all the required shared libraries.

- In order to avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar` or other libraries in the class-loading system.
- In case of deployment on Oracle WebLogic server, please refer to the [Module structure](#) [p 513] section.

CHAPITRE 55

Installation note for JBoss EAP 7.1.x

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Requirements](#)
3. [JBoss Application Server installation](#)
4. [EBX® home directory configuration](#)
5. [JBoss Application Server and Java Virtual Machine configuration](#)
6. [JNDI entries configuration](#)
7. [Data source and JDBC provider configuration](#)
8. [EBX.ear application update](#)
9. [EBX.ear application deployment](#)
10. [EBX® application start](#)

55.1 Overview

Attention

- This chapter describes a *quick installation example* of TIBCO EBX® on JBoss Application Server.
- It does not replace the [documentation](#) of this 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.
- The complete description of the components needed by EBX® is given in chapter [Java EE deployment](#) [p 339].
- To avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar`, `ebx-lz4.jar` or other libraries in the class-loading system.

55.2 Requirements

- Java SE 8 or 11
- JBoss Application Server EAP 7.1
- Database and JDBC driver
- EBX® CD
- No CDI features in EBX®'s additional modules (since CDI will be automatically disabled)

Voir aussi [Supported environments](#) [p 332]

55.3 JBoss Application Server installation

This quick installation example is performed for a Linux operating system.

1. Download JBoss EAP 7.1 Installer jar version 7.1.x from:
<https://developers.redhat.com/products/eap/download/>
2. Run the Installer using `java -jar` command line.
For further installation details, refer to the [documentation](#).
3. Perform a standard installation:
 1. Select the language and click 'OK',
 2. Accept the License and click 'Next',
 3. Choose the installation path and click 'Next',
 4. Keep the 'Component Selection' as it is and click 'Next',
 5. Enter 'Admin username', 'Admin password' and click 'Next',
 6. On 'Installation Overview' click 'Next',
 7. On 'Component Installation' click 'Next',
 8. On 'Configure Runtime Environment' leave the selection as it is and click 'Next',
 9. When 'Processing finished' appear, click 'Next',
 10. Uncheck 'Create shortcuts in the start menu' and click 'Next',
 11. Generate 'installation script and properties file' in the JBoss EAP 7.1 installation root directory,
 12. Click 'done'.

55.4 EBX® home directory configuration

1. Create the `EBX_HOME` directory, for example `/opt/ebx/home`.
2. Copy from the `EBX® CD`, the `ebx.software/files/ebx.properties` file to `EBX_HOME`. In our example, we will have the following file:
`/opt/ebx/home/ebx.properties`.

3. If needed, edit the `ebx.properties` file to override the default database. By default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and the `h2.standalone` one must be commented.

55.5 JBoss Application Server and Java Virtual Machine configuration

1. Open the `standalone.conf` configuration file, placed in `<JBOSS_HOME>/bin` (or `jboss-eap.conf` file placed in `<JBOSS_HOME>/bin/init.d` for running the server as a service).
2. Add 'ebx.properties' and 'ebx.home' properties to the 'JAVA_OPTS' environment variable respectively set with `ebx.properties` file's path and `EBX_HOME` directory's path.
3. Set the 'JBOSS_MODULES_SYSTEM_PKGS' environment variable like the following:

```
JBOSS_MODULES_SYSTEM_PKGS="org.jboss.byteman,net.jpountz"
```

4. Copy from the *EBX® CD*, the [ebx.software/lib/ebx-lz4.jar](#) [p 340] Data compression library to a dedicated directory (for example `<JBOSS_HOME>/compress`).
5. Open the `standalone.sh` script file, placed in `<JBOSS_HOME>/bin`.
6. Create a 'CLASSPATH' environment variable like the following:

```
CLASSPATH="<path_to_the_data_compression_library>:${JBOSS_HOME}/jboss-modules.jar:${CLASSPATH}"

# For our example
# CLASSPATH="${JBOSS_HOME}/compress/ebx-lz4.jar:${JBOSS_HOME}/jboss-modules.jar:${CLASSPATH}"
```

7. Replace the launch command options for foreground and background executions like the following:

```
if [ "x$LAUNCH_JBOSS_IN_BACKGROUND" = "x" ]; then
# Execute the JVM in the foreground
eval \"\$JAVA\" -D\"[Standalone]\" $JAVA_OPTS \
-cp \"$CLASSPATH\" \
\"-Dorg.jboss.boot.log.file=\"$JBOSS_LOG_DIR/server.log\" \
\"-Dlogging.configuration=file:\"$JBOSS_CONFIG_DIR/logging.properties\" \
org.jboss.modules.Main \
$MODULE_OPTS \
-mp \"\"$JBOSS_MODULEPATH\"\" \
org.jboss.as.standalone \
-Djboss.home.dir=\"\"$JBOSS_HOME\"\" \
-Djboss.server.base.dir=\"\"$JBOSS_BASE_DIR\"\" \
\"$SERVER_OPTS\"
JBOSS_STATUS=$?
else
# Execute the JVM in the background
eval \"\$JAVA\" -D\"[Standalone]\" $JAVA_OPTS \
-cp \"$CLASSPATH\" \
\"-Dorg.jboss.boot.log.file=\"$JBOSS_LOG_DIR/server.log\" \
\"-Dlogging.configuration=file:\"$JBOSS_CONFIG_DIR/logging.properties\" \
org.jboss.modules.Main \
$MODULE_OPTS \
-mp \"\"$JBOSS_MODULEPATH\"\" \
org.jboss.as.standalone \
-Djboss.home.dir=\"\"$JBOSS_HOME\"\" \
-Djboss.server.base.dir=\"\"$JBOSS_BASE_DIR\"\" \
\"$SERVER_OPTS\" "&
...
fi
```

55.6 JNDI entries configuration

1. Open the `standalone-full.xml` file placed in `<JBOSS_HOME>/standalone/configuration`.

2. Add, at least, the following lines to the server tag in messaging-activemq subsystem:

```
<connection-factory
  name="jms/EBX_JMSConnectionFactory"
  entries="java:/EBX_JMSConnectionFactory"
  connectors="To Be Defined"/>
<jms-queue
  name="jms/EBX_D3ReplyQueue"
  entries="java:/jms/EBX_D3ReplyQueue"
  durable="true"/>
<jms-queue
  name="jms/EBX_QueueIn"
  entries="java:/jms/EBX_QueueIn"
  durable="true"/>
<jms-queue
  name="jms/EBX_QueueFailure"
  entries="java:/jms/EBX_QueueFailure"
  durable="true"/>
<jms-queue
  name="jms/EBX_D3MasterQueue"
  entries="java:/jms/EBX_D3MasterQueue"
  durable="true"/>
<jms-queue
  name="jms/EBX_D3ArchiveQueue"
  entries="java:/jms/EBX_D3ArchiveQueue"
  durable="true"/>
<jms-queue
  name="jms/EBX_D3CommunicationQueue"
  entries="java:/jms/EBX_D3CommunicationQueue"
  durable="true"/>
```

Caution: the connectors attribute value, from the connection-factory element, has to be defined. Since the kind of connectors is strongly reliant on the environment infrastructure, a default configuration can not be provided.

See [configuring messaging](#) for more information.

3. Add, at least, the following line to mail subsystem:

```
<mail-session name="mail" debug="false" jndi-name="java:/EBX_MAIL_SESSION"/>
```

55.7 Data source and JDBC provider configuration

1. After the launch of the JBoss Server, run the management CLI without the use of '--connect' or '-c' argument.
2. Use the 'module add' management CLI command to add the new core module. Sample for PostgreSQL configuration:

```
module add \
  --name=org.postgresql \
  --resources=<PATH_TO_JDBC_JAR> \
  --dependencies=javaee.api, sun.jdk, ibm.jdk, javax.api, javax.transaction.api
```

3. Use the 'connect' management CLI command to connect to the running instance.
4. Register the JDBC driver. When running in a managed domain, ensure to precede the command with '/profile=<PROFILE_NAME>'. Sample for PostgreSQL configuration:

```
/subsystem=\
datasources/jdbc-driver=\
  postgresql:add(\
    driver-name=postgresql,\
    driver-module-name=org.postgresql,\
    driver-xa-datasource-class-name=org.postgresql.xa.PGXADatasource\
  )
```

5. Define the datasource using the 'data-source add' command, specifying the appropriate argument values. Sample for PostgreSQL configuration:

```
data-source add \
  --name=jdbc/EBX_REPOSITORY \
```



```
--jndi-name=java:/EBX_REPOSITORY \
--driver-name=postgresql \
--connection-url=jdbc:postgresql://<SERVER_NAME>:<PORT>/<DATABASE_NAME> \
--user-name=<PERSISTENCE_USER> \
--password=<PERSISTENCE_PASSWORD>
```

55.8 EBX.ear application update

1. Copy from the *EBX® CD*, the `ebx.software/webapps/ear-packaging/EBX.ear` file to your working directory.
2. Uncompress the ear archive to add the application's specific required third-party libraries and additional web modules.

Mail: see [SMTP and emails](#) [p 341] for more information.

SSL: see [Secure Socket Layer \(SSL\)](#) [p 342] for more information.

JMS: see [Java Message Service \(JMS\)](#) [p 342] for more information.

XML Catalog API: see [XML Catalog API](#) [p 342] for more information.

3. Update the `/META-INF/application.xml` and `/META-INF/jboss-deployment-structure.xml` files according to the added additional web modules.
4. Compress anew the ear archive.

55.9 EBX.ear application deployment

1. Copy `EBX.ear` into the `JBoss_HOME/standalone/deployments` directory.

55.10 EBX® application start

1. After the launch of the JBoss Application Server, with the `<JBoss_HOME>/bin/standalone.sh -c standalone-full.xml` command line or through the service command, run the EBX® web application by entering the following URL in the browser: <http://localhost:8080/ebx/>.
2. At first launch, [EBX® Wizard](#) [p 391] helps to configure the default properties of the initial repository.

CHAPITRE 56

Installation note for Tomcat 9.x

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Requirements](#)
3. [Tomcat Application Server installation](#)
4. [EBX® home directory configuration](#)
5. [Tomcat Application Server and Java Virtual Machine configuration](#)
6. [EBX® and third-party libraries deployment](#)
7. [EBX® web applications deployment](#)
8. [EBX® application start](#)

56.1 Overview

Attention

- This chapter describes a *quick installation example* of TIBCO EBX® on Tomcat Application Server.
- It does not replace the [documentation](#) of this 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.
- Tomcat 10.x is not supported.
- The complete description of the components needed by EBX® is given in chapter [Java EE deployment](#) [p 339].
- To avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar`, `ebx-1z4.jar` or other libraries in the class-loading system.
- The description below uses the variable name `$CATALINA_HOME` to refer to the Tomcat installation directory, and from which most relative paths are resolved. However, if the `$CATALINA_BASE` directory has been set for a multiple instances configuration, it should be used for each of these references.

56.2 Requirements

- Java SE 8 or 11
- Apache Tomcat 9.x
- Database and JDBC driver
- EBX® CD

Voir aussi [Supported environments](#) [p 332]

56.3 Tomcat Application Server installation

1. Download Tomcat 9.x core binary distributions from:
<https://tomcat.apache.org/download-90.cgi>
2. Run the installer or extract the archive and perform a standard installation with default options

56.4 EBX® home directory configuration

1. Create *EBX_HOME* directory, for example `C:\EBX\home`, or `/home/ebx`
2. Copy from *EBX® CD* the `ebx.software/files/ebx.properties` file to *EBX_HOME*. In our example, we will have the following file:
`C:\EBX\home\ebx.properties`, or `/home/ebx/ebx.properties`
3. If needed, edit the `ebx.properties` file to override the default database. By default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and the `h2.standalone` one must be commented.

56.5 Tomcat Application Server and Java Virtual Machine configuration

1. Modify `$CATALINA_HOME/conf/server.xml` (or `$CATALINA_BASE/conf/server.xml`) file by adding the following line to the `<Host>` element:

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

In our example, we will have:

```
<Host name=...>
```

```
... ..
```

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

```
... ..
```

```
</Host>
```

2. Modify the `$CATALINA_HOME/conf/catalina.properties` (or `$CATALINA_BASE/conf/catalina.properties`) file by adding the following lines to the `tomcat.util.scan.DefaultJarScanner.jarsToSkip` property:

```
ebx.jar,\
```

ebx-addons.jar, \

ebx-lz4.jar, \

3. Configure the Java Virtual Machine properties

- For Windows' Command Prompt launch

Set the environment variables by creating a `setenv.bat` file either into `$CATALINA_HOME\bin` or `$CATALINA_BASE\bin`. This file will hold, at least, the following lines:

```
set EBX_HOME=<path_to_the_directory_ebx_home>
set EBX_OPTS="-Debx.home=%EBX_HOME% -Debx.properties=%EBX_HOME%\ebx.properties"
set JAVA_OPTS="%EBX_OPTS% %JAVA_OPTS%"
set CLASSPATH=<$CATALINA_HOME_or_$CATALINA_BASE>\compress\ebx-lz4.jar;%CLASSPATH%
```

Where `<$CATALINA_HOME_or_$CATALINA_BASE>` must be replaced by `%CATALINA_HOME%` or `%CATALINA_BASE%` if they have been configured. Otherwise this piece of text must be replaced by the Tomcat installation directory's path.

- For Windows users that have installed Tomcat as a service

Set Java options through the Tomcat service manager GUI (Java tab).

Be sure to set options on separate lines in the *Java Options* field of the GUI:

```
-Debx.home=<path_to_the_directory_ebx_home>
-Debx.properties=<path_to_the_directory_ebx_home>\ebx.properties
```

Update the service using the `//US//` parameter to set the proper classpath value.

```
C:\> tomcat9 //US//Tomcat9 --Classpath=<$CATALINA_HOME_or_$CATALINA_BASE>\compress\ebx-lz4.jar;%CLASSPATH%
```

Where `<$CATALINA_HOME_or_$CATALINA_BASE>` must be replaced by `%CATALINA_HOME%` or `%CATALINA_BASE%` if they have been configured. Otherwise this piece of text must be replaced by the Tomcat installation directory's path.

- For Unix shell launch

Set the environment variables by creating a `setenv.sh` file either into `$CATALINA_HOME/bin` or `$CATALINA_BASE/bin`. This file will hold, at least, the following lines:

```
EBX_HOME=<path_to_the_directory_ebx_home>
EBX_OPTS="-Debx.home=${EBX_HOME} -Debx.properties=${EBX_HOME}/ebx.properties"
export JAVA_OPTS="${EBX_OPTS} ${JAVA_OPTS}"
export CLASSPATH=<$CATALINA_HOME_or_$CATALINA_BASE>/compress/ebx-lz4.jar:${CLASSPATH}"
```

Where `<$CATALINA_HOME_or_$CATALINA_BASE>` must be replaced by `${CATALINA_HOME}` or `${CATALINA_BASE}` if they have been configured. Otherwise this piece of text must be replaced by the Tomcat installation directory's path.

Caution: Accounts used to launch EBX® must have create/update/delete rights on *EBX_HOME* directory.

Note

`<path_to_the_directory_ebx_home>` is the directory where we copied `ebx.properties`. In our example, it is `C:\EBX\home`, or `/home/ebx`.

Note

For a [Data compression library](#) [p 340] native installation, ensure to only reference it in the *CLASSPATH* environment variable.

56.6 EBX® and third-party libraries deployment

1. Copy third-party libraries from the *EBX® CD* to `$CATALINA_HOME/lib/` (or `$CATALINA_BASE/lib/`) directory, except for the [Data compression library](#) [p 340]. In our example, we will have:
`$CATALINA_HOME/lib/javax.mail-1.5.6.jar` coming from `ebx.software/lib/lib-mail` directory.
`$CATALINA_HOME/lib/h2-1.4.196.jar` (default persistence factory) coming from `ebx.software/lib/lib-h2` directory.
`$CATALINA_HOME/lib/xml-apis-1.4.01.jar` coming from `ebx.software/lib/lib-xml-apis` directory.
 The exact description of these components is given in chapter [Software components](#) [p 339]. Obviously, if those components are already deployed on the class-loading system, they do not have to be duplicated.
2. Create a directory dedicated to the [Data compression library](#) [p 340] (for example `$CATALINA_HOME/compress` or `$CATALINA_BASE/compress`) and copy it there.

Note

Ensure that the library is copied in the directory pointed out by the previously updated `CLASSPATH` environment variable.

3. Copy from *EBX® CD* the `ebx.software/lib/ebx.jar` file to `$CATALINA_HOME/lib/` (or `$CATALINA_BASE/lib/`) directory. In our example, we will have:
`$CATALINA_HOME/lib/ebx.jar`

56.7 EBX® web applications deployment

1. Copy from the *EBX® CD* the war files in `ebx.software/webapps/wars-packaging` to the `$CATALINA_HOME/webapps/` (or `$CATALINA_BASE/webapps/`) directory. In our example, we will have:
`$CATALINA_HOME/webapps/ebx.war`: Initialization servlet for EBX® applications
`$CATALINA_HOME/webapps/ebx-root-1.0.war`: Provides a common default module for data models
`$CATALINA_HOME/webapps/ebx-manager.war`: Master Data Management web application
`$CATALINA_HOME/webapps/ebx-dataservices.war`: Data Services web application
`$CATALINA_HOME/webapps/ebx-dma.war`: Data Model Assistant web application
`$CATALINA_HOME/webapps/ebx-ui.war`: User Interface web application

56.8 EBX® application start

1. After Tomcat launch, run EBX® web application by entering the following URL in the browser:
<http://localhost:8080/ebx/>
2. At first launch, [EBX® Wizard](#) [p 391] helps to configure the default properties of the initial repository.

CHAPITRE 57

Installation note for WebSphere AS 9

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Requirements](#)
3. [WebSphere Application Server installation](#)
4. [WebSphere Application Server and EBX® home directory configuration](#)
5. [Data source and JDBC provider configuration](#)
6. [Java Virtual Machine configuration](#)
7. [EBX® application deployment](#)
8. [EBX® application start](#)

57.1 Overview

Attention

- This chapter describes a *quick installation example* of TIBCO EBX® on WebSphere Application Server.
- It does not replace the [documentation](#) of this 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.
- The complete description of the components needed by EBX® is given in chapter [Java EE deployment](#) [p 339].
- To avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar`, `ebx-lz4.jar` or other libraries in the class-loading system.

57.2 Requirements

- WebSphere Application Server 9
- Database and JDBC driver

- EBX® CD
- No CDI features in EBX®'s additional modules (since CDI will be automatically disabled)

Voir aussi [Supported environments](#) [p 332]

57.3 WebSphere Application Server installation

This quick installation example is performed for a Linux operating system.

1. Download WebSphere AS 9 Installation Manager latest version from:
<https://www.ibm.com/support/pages/node/609575>
2. Run the Installation Manager and add the following repositories:
 - WebSphere Application Server V9.0:
<http://www.ibm.com/software/repositorymanager/V9WASBase>
 - WebSphere Application Server Network Deployment V9.0:
<http://www.ibm.com/software/repositorymanager/V9WASND>
3. Install the webSphere Application Server Network Deployment
For further installation details, refer to the [documentation](#).
4. Run the webSphere Customization Toolbox and perform a standard installation with default options:
 1. Create profile: click 'Create' then select 'Application Server', and click 'Next'
 2. Profile Creation Options: select 'Advanced profile creation' and click 'Next'
 3. Optional Application Deployment: select those options:
 - Deploy the 'Administrative Console'
 - Deploy the 'Installation Verification Tool' application
 Click 'Next'
 4. Profile Name and Location: enter a profile name (example: 'EbxAppSrvProfile') and a directory. In our example, we will get:
`/opt/IBM/WebSphere/AppServer/profiles/EbxAppSrvProfile`
Further, it will correspond to <PROFILE_HOME>.
Click 'Next'
 5. Node and Host Names: enter the node name (example: 'Node1'), the server name (example: 'EbxServer'), the host name (example: 'localhost'), and click 'Next'
 6. Administrative Security: check 'Enable administrative security' option, enter the user name, the password, and click 'Next'
 7. Security Certificate (part 1): select 'Create a new default personal certificate' and 'Create a new root signing certificate', and click 'Next'
 8. Security Certificate (part 2): keep as default and click 'Next'
 9. Port Value Assignment: keep as default and click 'Next'
 10. Linux Service Definition: check 'Run the application server process as a Linux service' option, enter the user name (example: 'ebx'), and click 'Next'

11. Web Server Definition: keep as default and click 'Next'
12. Profile Creation Summary: keep as default and click 'Create'
13. Profile Creation Complete: uncheck 'Launch the First steps console' option, and click 'Finish'

57.4 WebSphere Application Server and EBX® home directory configuration

1. Create the *EBX_HOME* directory, for example `/opt/ebx/home`
2. Copy from the *EBX® CD*, the `ebx.software/files/ebx.properties` file to *EBX_HOME*. In our example, we will have the following file:
`/opt/ebx/home/ebx.properties.`
3. If needed, edit the `ebx.properties` file to override the default database. By default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and the `h2.standalone` one must be commented.
4. Create the *EBX_LIB* directory, for example `/opt/ebx/home/lib`
5. Copy third-party libraries, from the *EBX® CD* or from other sources, to the *EBX_LIB* directory. In our example, for a PostgreSQL database, we will get:
`postgresql-X.X.X-driver.jar` (coming from another source than the *EBX® CD*).
`xml-apis-1.4.01.jar` (coming from the `ebx.software/lib/lib-xml-apis/` directory of the *EBX® CD*).
`ebx-lz4.jar` (coming from the `ebx.software/lib/` directory of the *EBX® CD*).
The complete description of these components is given in the chapter [Java EE deployment](#) [p 339]. If those components are already deployed on the class-loading system, they do not have to be duplicated (ex: `javax.mail-1.5.6.jar` is already present on the WebSphere Application Server).

57.5 Data source and JDBC provider configuration

1. Start the server with the following command line:
`sudo <PROFILE_HOME>/bin/startServer.sh <serverName>`
where:
`<PROFILE_HOME>` corresponds to the previously created profile home directory. In our example, we will get: `/opt/IBM/WebSphere/AppServer/profiles/EbxAppSrvProfile`.
`<serverName>` corresponds to the server to start. In our example, we will get: `EbxServer`.
2. Connect into the WebSphere Integrated Solutions Console, using the user name and password typed during the profile creation (Administrative Security step), by entering the following URL in the browser:
<https://localhost:9043/ibm/console>
3. On the left menu, go to 'Resources > JDBC > Data Sources', choose the JDBC 'Scope' (for example use 'Cell'), and click 'New'
4. Enter basic data source information:
 - Data source name: `EBX_REPOSITORY`

- JNDI name: jdbc/EBX_REPOSITORY

Click 'Next'

5. Select 'Create new JDBC provider', and click 'Next'

6. Create a new JDBC provider: (example with a PostgreSQL database)

- Database type: User-defined
- Implementation class name: org.postgresql.ds.PGConnectionPoolDataSource
- Name: PostgreSQL

Click 'Next'

7. Enter database class path information: (example with a PostgreSQL database)

- Class path: <EBX_LIB>/postgresql-X.X.X-driver.jar

In our example, <EBX_LIB> corresponds to /opt/ebx/home/lib.

Click 'Next'

8. Keep database specific properties for the data source as default and click 'Next'

9. Keep setup security aliases as default and click 'Next'

10. Click 'Finish'

11. Save the master configuration

12. Click on 'Data Sources > EBX_REPOSITORY'

13. On the right in the 'Configure additional properties' section, click on 'Additional Properties' and define the database account access:

- Define the user value to the according user
- Define the password value to the according password

14. Save the master configuration

15. Test the connection

57.6 Java Virtual Machine configuration

1. Click on 'Application Servers'

2. Click on the server name (for example: 'EbxServer')

3. Click on 'Process definition' under 'Server infrastructure > Java Process Management'

4. Click on 'Java Virtual Machine' under 'Additional Properties'

5. Add 'ebx.properties' and 'ebx.home' properties, in the 'Generic JVM arguments' section, respectively set to ebx.properties file's path and EBX_HOME directory's path.

6. Add, in the 'Classpath' section, the paths to the third-party libraries placed in the EBX_LIB directory except for the JDBC driver. In our example, we will get:

```
/opt/ebx/home/lib/xml-apis-1.4.01.jar
```

```
/opt/ebx/home/lib/ebx-lz4.jar
```

Note

Every library's path declaration must be on a separate line.

7. Click 'Ok'
8. Save the master configuration

57.7 EBX® application deployment

1. Copy from the *EBX® CD*, the `ebx.software/webapps/ear-packaging/EBX.ear` to the `<EBX_HOME>/ear` directory. In our example, we will get:
`/opt/ebx/home/ear/EBX.ear`
2. Connect into the WebSphere Integrated Solutions Console, using the user name and password typed during the profile creation (Administrative Security step), by entering the following URL in the browser:
<https://localhost:9043/ibm/console>
3. Click on 'WebSphere enterprise applications' under 'Applications > Application Types'
4. Install the EBX.ear
 1. Enterprise Applications: click on 'Install'
 2. Preparing for the application installation: Browse to the EBX.ear file. In our example, it is located under the `/opt/ebx/home/ear` directory.
Click 'Next'
 3. How do you want to install the application?: Select 'Fast Path...', then click 'Next'
 4. Select installation options: keep as default, then click 'Next'
 5. Map modules to servers: select all modules, then click 'Next'
 6. Map resource references to resources: copy the 'Resource Reference' value and paste it in the 'Target Resource JNDI Name' field, for every modules, then click 'Next'
 7. Warnings will appear related to `JNDI:mail/EBX_MAIL_SESSION` and `JNDI:jms/EBX_JMSConectorFactory`. This behavior is normal since these resources had not been configured.
Click 'Continue'
 8. Map resource environment references to resources: Copy the 'Resource Reference' value and paste it to the 'Target Resource JNDI Name' value, for every modules, then click 'Next'
 9. Warnings will appear related to unavailable resources. This behavior is normal since these resources had not been configured.
Click 'Continue'
10. Map virtual hosts for Web modules: select all modules and click 'Next'
11. Summary: keep as default, click 'Finish'
12. If installation succeeds, 'Application EBX® installed successfully' is logged.
Click 'Save'

5. On the left menu, go to 'Applications > Enterprise Applications'
6. Change EBX® application's class loader policy
 1. Click on EBX® resource's name
 2. On the 'configuration' pane, under 'Detail Properties', click on 'Class loading and update detection'
 3. Under 'General Properties', change 'Class loader order' to 'Classes loaded with local class loader first (parent last)' and click 'OK'
 4. Save the master configuration
7. On the left menu, go to 'Applications > Enterprise Applications', select EBX®, then click 'Start'
The EBX® 'Application status' will changed to a green arrow.

57.8 EBX® application start

1. After the launch of the WebSphere application Server, run the EBX® web application by entering the following URL in the browser:
<http://localhost:9080/ebx/>
or
<https://localhost:9443/ebx/>
2. At first launch, [EBX® Wizard](#) [p 391] helps to configure the default properties of the initial repository.

CHAPITRE 58

Installation note for WebLogic 14c

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Requirements](#)
3. [WebLogic Application Server installation](#)
4. [EBX® home directory configuration](#)
5. [WebLogic Application Server and Java Virtual Machine configuration](#)
6. [EBX® and third-party libraries deployment](#)
7. [Data source and JDBC provider configuration](#)
8. [EBX® application deployment](#)
9. [EBX® application start](#)

58.1 Overview

Attention

- This chapter describes a *quick installation example* of TIBCO EBX® on WebLogic Application Server.
- It does not replace the [documentation](#) of this 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.
- The complete description of the components needed by EBX® is given in chapter [Java EE deployment](#) [p 339].
- To avoid unpredictable behavior, the guideline to follow is to avoid any duplicates of `ebx.jar`, `ebx-1z4.jar` or other libraries in the class-loading system.

58.2 Requirements

- Certified Oracle Java SE 8 or 11

- WebLogic Server 14c
- Database and JDBC driver
- EBX® CD

Voir aussi [Supported environments](#) [p 332]

58.3 WebLogic Application Server installation

1. Download WebLogic 14c latest version from:
<https://www.oracle.com/middleware/technologies/fusionmiddleware-downloads.html>
2. Run the Oracle Fusion Middleware Weblogic installation wizard using a certified Oracle JDK and the `java -jar` command line
3. Perform a standard installation with default options and choose the appropriate installation directory
4. Leave the 'Automatically launch the Configuration Wizard' option activated to perform the next steps:
 1. Create Domain: choose 'Create a new domain' and specify the domain home directory, then click 'Next'
 2. Templates: keep as default and click 'Next'
 3. Administrator Account: enter a domain administrator username and password and click 'Next'
 4. Domain Mode and JDK: choose the production mode and your JDK installation home and click 'Next'
 5. Advanced configuration: check 'Administration server' and 'Topology'. That way, we create two independent domain nodes: an administration one and an application one.
Click 'Next'
 6. Administration Server: enter your administration node name (for example 'AdminServer') and listen port (by default 7001), then click 'Next'
 7. Managed Servers: add the application node name (for example 'EbxServer') and listen port (for example 7003), then click 'Next'
 8. Clusters: keep as default and click 'Next'
 9. Server Templates: keep as default and click 'Next'
 10. Machines: keep as default and click 'Next'
 11. Configuration Summary: click 'Create'
 12. Configuration Process: click 'Next'
 13. End Of Configuration: click 'Finish'

58.4 EBX® home directory configuration

1. Create `EBX_HOME` directory, for example `C:\EBX\home`, or `/home/ebx`
2. Copy from *EBX® CD* the `ebx.software/files/ebx.properties` file to `EBX_HOME`. In our example, we will have the following file:

C:\EBX\home\ebx.properties, or /home/ebx/ebx.properties

3. If needed, edit the `ebx.properties` file to override the default database. By default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and the `h2.standalone` one must be commented.

58.5 WebLogic Application Server and Java Virtual Machine configuration

1. Configure the launch properties for the *Managed Server* (for example 'EbxServer')

Edit the `<DOMAIN_HOME>/bin/startManagedWebLogic.sh` script file by adding the following lines:

```
EBX_HOME="<path_to_the_directory_ebx_home>"
EBX_OPTIONS="-Debx.home=${EBX_HOME} -Debx.properties=${EBX_HOME}/ebx.properties"
export JAVA_OPTIONS="${EBX_OPTIONS} ${JAVA_OPTIONS}"
```

2. Edit the `<DOMAIN_HOME>/bin/setDomainEnv.sh` script file by adding the following line:

```
PRE_CLASSPATH="<path_to_the_data_compression_library>"

# For our example
# PRE_CLASSPATH="${DOMAIN_HOME}/compress/ebx-lz4.jar"
```

58.6 EBX® and third-party libraries deployment

1. Copy third-party libraries from the *EBX® CD* to the `<DOMAIN_HOME>/lib` directory except for the [Data compression library](#) [p 340]. In our example, for an H2 standalone data base, we will have:

`<DOMAIN_HOME>/lib/h2-1.4.196.jar` (default persistence factory) coming from `ebx.software/lib/lib-h2` directory.

The complete description of the components needed by EBX® is given in chapter [Java EE deployment](#) [p 339]. Obviously, if those components are already deployed on the class-loading system, they do not have to be duplicated (ex: `javax.mail-1.5.6.jar` and `xml-apis-1.4.01.jar` are already present in the WebLogic Server).

2. Create a directory dedicated to the [Data compression library](#) [p 340] (for example `<DOMAIN_HOME>/compress`) and copy it there.

Note

Ensure that the library is copied in the directory pointed out by the previously updated `PRE_CLASSPATH` environment variable.

58.7 Data source and JDBC provider configuration

1. Start the 'Administration server' (for example 'AdminServer'), using:

```
<DOMAIN_HOME>/bin/startWebLogic.sh
```

2. Launch the 'WebLogic Server Administration Console' by entering the following URL in the browser:

<http://localhost:7001/console>.

Log in with the domain administrator username and password

3. Click on 'Services > Data sources' in the 'Domain Structure' panel, then click on 'New > Generic Data Source':
 1. Set: Type Name: EBX_REPOSITORY, JNDI Name: EBX_REPOSITORY, Database Type: *Your database type*
Click 'Next'
 2. Choose your database driver type, and click 'Next'
 3. Uncheck 'Supports Global Transactions', and click 'Next'
 4. Setup your database 'Connection Properties' and click 'Next'
 5. Click 'Test Configuration' and then 'Finish'
 6. Switch on the 'Targets' tab and select all Servers, then click 'Save'
 7. Restart the Administration server (for example 'AdminServer'), using:


```
<DOMAIN_HOME>/bin/stopWebLogic.sh
```

```
<DOMAIN_HOME>/bin/startWebLogic.sh
```

58.8 EBX® application deployment

1. Copy from the *EBX® CD* the `ebx.software/webapps/ear-packaging/EBXForWebLogic.ear` to the `EBX_HOME` directory. In our example, we will have:


```
C:\EBX\home\EBXForWebLogic.ear, or /home/ebx/EBXForWebLogic.ear
```
2. Launch the 'WebLogic Server Administration Console' by entering the following URL in the browser:


```
http://localhost:7001/console
```
3. Click on 'Lock and Edit' in the 'Change Center' panel
4. Click on 'Deployments' in the 'Domain Structure' panel, and click 'Install':
 1. Install Application Assistant: Enter in 'Path' the application full path to `EBXForWebLogic.ear` file, located in `C:\EBX\home\`, or `/home/ebx/` directory and click 'Next'
 2. Choose the installation type and scope: Click on 'Install this deployment as an application', 'Global' default scope and click 'Next'
 3. Select the deployment targets: Select a node (for example 'EbxServer') from the 'Servers' list and click 'Next'
 4. Optional Settings: keep as default and click 'Finish'
5. Click on 'Activate Changes', on the top left corner. The deployment status will change to 'prepared'
6. Switch to 'Control' tab, select the 'EBXForWebLogic' enterprise application, then click 'Start' > 'Servicing all requests'
7. Start the application node (for example 'EbxServer'), using:


```
<DOMAIN_HOME>/bin/startManagedWebLogic.sh EbxServer http://localhost:7001
```


58.9 EBX® application start

1. After WebLogic Application Server launch, run the EBX® web application by entering the following URL in the browser:
<http://localhost:7003/ebx/>
2. At first launch, [EBX® Wizard](#) [p 391] helps to configure the default properties of the initial repository.

CHAPITRE 59

TIBCO EBX® main configuration file

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Setting automatic installation on first launch](#)
3. [Setting the EBX® root directory](#)
4. [Configuring the EBX® repository](#)
5. [Configuring the user and roles directory](#)
6. [Configuring EBX® localization](#)
7. [Setting temporary files directories](#)
8. [Activating the XML audit trail](#)
9. [Configuring the EBX® logs](#)
10. [Activating and configuring SMTP and emails](#)
11. [Configuring data services](#)
12. [Activating and configuring JMS](#)
13. [Configuring distributed data delivery \(D3\)](#)
14. [Configuring REST toolkit services](#)
15. [Configuring Web access from end-user browsers](#)
16. [Configuring failover](#)
17. [Tuning the EBX® repository](#)
18. [Miscellaneous](#)

59.1 Overview

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

The main configuration file complements the [Java EE deployment descriptor](#) [p 344]. Administrators can also perform further configuration through the user interface, which is then stored in the EBX® repository.

Voir aussi[Deployment details](#) [p 344][UI administration](#) [p 423]**Location of the file**

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

1. By defining the Java system property 'ebx.properties'. For example, this property can be set by adding the option `-Debx.properties=<filePath>` to the java command-line command. See [Java documentation](#).
2. By defining the servlet initialization parameter 'ebx.properties'.
This standard Java EE setting must be specified in the `web.xml` file of the web application 'ebx'. EBX® accesses this parameter by calling the method `ServletConfig.getInitParameter("ebx.properties")` in the servlet `FrontServlet`.
See [getInitParameter](#) in the Oracle `ServletConfig` documentation.
3. By default, if nothing is specified, the main configuration file is located at `WEB-INF/ebx.properties` of the web application 'ebx'.

Note

In addition to specifying properties in the main configuration file, it is also possible to set the values of properties directly in the system properties. For example, using the `-D` argument of the java command-line command.

Custom properties and variable substitution

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

For example, the default configuration file provided with EBX® uses the custom property `ebx.home` to set a default common directory, which is then included in other properties.

59.2 Setting automatic installation on first launch

Repository can be automatically installed on first startup.

```
#####
## Installation on first launch.
## All values are ignored if the repository is already installed.
#####
## Enables repository installation on first startup (default is false).
ebx.install.enabled=true

## Following properties configure the repository. Values are optional and defaults are automatically generated.
ebx.install.repository.id=00275930BB88
ebx.install.repository.label=A Test

## Following properties specify the EBX administrator. These are ignored if a custom directory is defined.
ebx.install.admin.login=admin
ebx.install.admin.firstName=admin
ebx.install.admin.lastName=admin
ebx.install.admin.email=adamin@example.com

## Following property specifies the non-encrypted password used for the EBX administrator.
## It is ignored if a custom directory is defined. It cannot be set if property
ebx.install.admin.password.encrypted is set.
#ebx.install.admin.password=admin
```

```
## Following property specifies the encrypted password used for the EBX administrator.
## It is ignored if a custom directory is defined. It cannot be set if property ebx.install.admin.password is
## set.
## Password can be encrypted by using command:
## java -cp ebx.jar com.orchestranetworks.service.directory.EncryptPassword password_to_encrypt
ebx.install.admin.password.encrypted=8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918
```

59.3 Setting the EBX® root directory

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

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

Voir aussi [Monitoring and clean up of the file system](#) [p 419]

59.4 Configuring the EBX® repository

Before configuring the persistence properties of the EBX® repository, carefully read the section [Technical architecture](#) [p 412] in the chapter 'Repository administration'.

The required library (driver) for each supported database is described in the chapter [Database drivers](#) [p 341].

Voir aussi

[Repository administration](#) [p 412]

[Rules for the database access and user privileges](#) [p 413]

[Supported databases](#) [p 336]

[Data source of the EBX® repository](#) [p 345]

[Database drivers](#) [p 341]

```
#####
## 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 EBX® persistence system is H2 'standalone'.
#####
ebx.persistence.factory=h2.standalone
ebx.persistence.user=sa
ebx.persistence.password=

#####
## Case EBX® 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=xxxxxxx
#ebx.persistence.password=yyyyyyy
```

```
#####
## Case EBX® persistence system is Oracle database.
#####
#ebx.persistence.factory=oracle

## 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:oracle:thin:@127.0.0.1:1521:ebxDatabase
#ebx.persistence.driver=oracle.jdbc.OracleDriver
#ebx.persistence.user=xxxxxxx
#ebx.persistence.password=yyyyyyy

## Activate to use VARCHAR2 instead of NVARCHAR2 on Oracle; never modify on an existing repository.
#ebx.persistence.oracle.useVARCHAR2=false

#####
## Case EBX® persistence system is SAP Hana
#####
#ebx.persistence.factory=hana

## 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:sap://127.0.0.1:39041
#ebx.persistence.driver=com.sap.db.jdbc.Driver
#ebx.persistence.user=xxxxxxx
#ebx.persistence.password=yyyyyyy

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

## 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:sqlserver://127.0.0.1:1036;datasenname=ebxDatabase
#ebx.persistence.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#ebx.persistence.user=xxxxxxx
#ebx.persistence.password=yyyyyyy

#####
## Case EBX® persistence system is Microsoft Azure SQL database.
#####
#ebx.persistence.factory=azure.sql

## 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:sqlserver://myhost.database.windows.net:1433;database=ebxDatabase;encrypt=true;\
#trustServerCertificate=false;hostNameInCertificate=".database.windows.net";
#ebx.persistence.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#ebx.persistence.user=xxxxxxx
#ebx.persistence.password=yyyyyyy

#####
## Case EBX® 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=xxxxxxx
#ebx.persistence.password=yyyyyyy
```

59.5 Configuring the user and roles directory

This parameter specifies the Java directory factory class name. It must only be defined if not using the default EBX® directory.

Voir aussi

[Users and roles directory](#) [p 451]

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
```

59.6 Configuring EBX® localization

This parameter is used to configure the locales used at runtime. This list must contain all the locales that are exposed to the end-user. EBX® will not be able to display labels and messages in a language that is not declared in this list.

The default locale must be the first one in the list.

```
#####
## Available locales, separated by a comma.
## The first element in the list is considered as the default locale.
## If not set, available locales are 'en-US, fr-FR'.
##
#####
#ebx.locales.available=en-US, fr-FR
```

Voir aussi [Extending TIBCO EBX® internationalization](#) [p 263]

59.7 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
```

59.8 Activating the XML audit trail

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

```
#####
# The XML history has been replaced by an SQL history.
# This old XML history can be activated using the following variable.
# Default is false.
```

```
#####
ebx.history.xmlaudittrail.activated = false
```

Voir aussi [Audit trail](#) [p 471]

59.9 Configuring the EBX® logs

The most important logging categories are:

ebx.log4j.category.log.kernel	Logs for EBX® main features, processes, exceptions and compilation results of modules and data models.
ebx.log4j.category.log.workflow	Logs for main features, warnings and exceptions about workflow.
ebx.log4j.category.log.persistence	Logs related to communication with the underlying database.
ebx.log4j.category.log.setup	Logs for the compilation results of all EBX® objects, except for modules and data models.
ebx.log4j.category.log.validation	Logs for datasets validation results.
ebx.log4j.category.log.mail	Logs for the activity related to the emails sent by the server (see Activating and configuring SMTP and emails [p 378]). Note: This category must not use the Custom SMTP appender [p 378] in order to prevent infinite loops.
ebx.log4j.category.log.d3	Logs for D3 events on EBX®.
ebx.log4j.category.log.dataservices	Logs for data service events in EBX®.
ebx.log4j.category.log.monitoring	Raw logs for monitoring [p 322].
ebx.log4j.category.log.request	Logs all Request^{API} and Query^{API} issued in the EBX® repository having a duration exceeding ebx.logs.request.durationThreshold milliseconds. All queries are logged regardless of their duration, if log level is set to DEBUG.
ebx.log4j.category.log.restServices	Logs for REST services events in EBX®, including those from the REST Toolkit [p 897].

Some of these categories can also be written to through custom code using the `LoggingCategory`^{API} interface.

```
#####
## Log4J properties:
##
## We have some specific syntax extensions:
## - Appender ebxFilename:<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, ebxFilename:kernel, kernelMail
ebx.log4j.category.log.workflow= INFO, ebxFilename:workflow
ebx.log4j.category.log.persistence= INFO, ebxFilename:persistence
ebx.log4j.category.log.setup= INFO, Console, ebxFilename:kernel
ebx.log4j.category.log.mail= INFO, Console, ebxFilename:mail
ebx.log4j.category.log.frontEnd= INFO, Console, ebxFilename: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, ebxFilename: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 = EBX® 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, ebxFilename:monitoring
ebx.log4j.category.log.dataServices= INFO, ebxFilename:dataServices
ebx.log4j.category.log.d3= INFO, ebxFilename:d3
ebx.log4j.category.log.request= INFO, ebxFilename:request
ebx.log4j.category.log.restServices= INFO, ebxFilename:dataServices
```

Custom 'ebxFile' appender

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

The property `ebx.log4j.appender.ebxFilename.backup.Threshold` allows defining 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.ebxFilename.backup.Threshold=-1
```

Custom SMTP appender

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

Voir aussi [Activating and configuring SMTP and emails](#) [p 378]

Custom module log threshold

By default, the log level threshold of the logging category associated with a custom module is set to INFO.

This threshold can be customized by setting the property `ebx.log4j.category.log.wbp.xxxxxx` for the custom module `xxxxxx`.

Example: `ebx.log4j.category.log.wbp.mycompany-module=DEBUG`.

Voir aussi `ModuleContextOnRepositoryStartup.getLoggingCategory`^{API}

Add-on module log threshold

By default, the log level threshold of any add-on module is set to INFO.

The log level threshold can be customized by setting the property `ebx.log4j.category.log.addon.xxxxxx` for the add-on module `ebx-addon-xxxxxx`.

Example: `ebx.log4j.category.log.addon.daqa=DEBUG`

59.10 Activating and configuring SMTP and emails

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

Voir aussi [Mail sessions](#) [p 345]

```
#####
## SMTP and emails
#####

## Activate emails (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
```

59.11 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

# Specifies the default maximum pagination size value for the select
# operations. This configuration is used by SOAP and REST connectors.
# Default value is 10000, maximum recommended value is 100000
#ebx.dataservices.pagination.maxSize.default= 10000

# Upon WSDL generation, specifies if the target namespace value
# corresponds to the content before 5.5.0 'ebx-services'
# or 'urn:ebx:ebx-services' in conformity with the URI syntax.
# If the parameter is set to true, there is no check of the target
# namespace as URI at the WSDL generation.
# If unspecified, default is false.
#ebx.dataservices.wsdlTargetNamespace.disabledCheck=false

#####
## REST configuration
#####

# If activated, the HTTP request header 'Accept' is used to specify
# the accepted content type. If none is supported, an error is
# returned to the client with the HTTP code 406 'Not acceptable'.
# If deactivated, the header is ignored therefore the best content
# type is used.
# Default is false.
#ebx.dataservices.rest.request.checkAccept=false

# If activated, when a REST data service authentication negotiate fails,
# EBX response includes fallback to 'Basic' authentication method by setting
# the HTTP header 'WWW-Authenticate' to 'Basic'.
# Note: This property only activate/deactivate
# the authentication fallback.
# Default is false.
#ebx.dataservices.rest.auth.tryBasicAuthentication=false

# Authorization token timeout is seconds.
# Default value is 1800 seconds (30 minutes)
# This value is ignored if 'Token Authentication Scheme' is not activated.
#ebx.dataservices.rest.auth.token.timeout=1800
```

59.12 Activating and configuring JMS

Voir aussi [JMS for data services](#) [p 346]

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

## Activates JMS (true or false, default is false).
## If activated, the deployer must ensure that the entry 'jms/EBX_JMSConnectionFactory'
## are bound in the operational environment of the application server.
## The entry 'jms/EBX_QueueIn' should also be bound to enable handling Data Services
## request using JMS.
#ebx.jms.activate=false

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

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

59.13 Configuring distributed data delivery (D3)

See [Configuring D3 nodes](#) [p 497] for the main configuration file properties pertaining to D3.

Voir aussi

[JMS for distributed data delivery \(D3\)](#) [p 487]

[Introduction to D3](#) [p 478]

59.14 Configuring REST toolkit services

```
#####
## REST configuration
#####

# Defines the maximum number of bytes that will be extracted
# from the REST request body to build some DEBUG log messages.
# Default value is 8192 bytes.
# This value is ignored if DEBUG level is not activated on the restServices logger.
#ebx.restservices.log.body.content.extract.size=8192
```

59.15 Configuring Web access from end-user browsers

HTTP Authorization header policy

EBX® natively offers three policies to send and receive credentials using HTTP headers:

standard	It corresponds to the authentication scheme, using the HTTP Authorization header, described in the RFC 2617 .
ebx	To prevent HTTP Authorization header override issues, this policy acts the same as the standard but the credentials are stored in an EBX® specific HTTP header.
both	It is the combination of the two previously described policies.

```
#####
## EBX® authorization header policy for HTTP requests
##
## Possible values are: standard, ebx, both.
## standard:
##   the standard HTTP Authorization header holds the credentials
## ebx:
##   an EBX® specific HTTP header holds the credentials
## both:
##   both (standard and specific) HTTP headers hold the credentials
##
## Default value is: both.
#####
#ebx.http.authorization.header.policy=both
```

URLs computing

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

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

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

Voir aussi [URL policy \(deprecated\)](#) [p 426]

```
#####
## EBX® 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
## {ui.path}:
## If not defined, defaults to ebx-ui/
## {http.useHttpsSettings}:
## If true, force the use of SSL security even if the incoming requests do not
##
## Resulting address will be:
## EBX®: protocol://{host}:{port}/{path}
## UI: protocol://{host}:{port}/{ui.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.http.ui.path=ebx-ui/
#ebx.servlet.http.useHttpsSettings=false

#ebx.servlet.https.host=
#ebx.servlet.https.port=
ebx.servlet.https.path=ebx/
#ebx.servlet.https.ui.path=ebx-ui/

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

ebx.externalResources.useLocalUrl=true

#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
#ebx.externalResources.http.path=
#ebx.externalResources.http.useHttpsSettings=false

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

Proxy mode

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

The web server sends requests to the application server according to a path in the URL. The `servletAlias` and `uiServletAlias` paths are specified in the main configuration file.

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

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

To force the use of SSL security even if the incoming requests do not, `ebx.servlet.http.useHttpsSettings` and / or `ebx.externalResources.http.useHttpsSettings` properties must be set to `true`. Their default values are `false`.

The main configuration file may be 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.http.ui.path=uiServletAlias
#ebx.servlet.http.useHttpsSettings=false

#ebx.servlet.https.host=
#ebx.servlet.https.port=
ebx.servlet.https.path=servletAlias
ebx.servlet.https.ui.path=uiServletAlias

#####

ebx.externalResources.useLocalUrl=true

#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
ebx.externalResources.http.path=resourcesAlias
#ebx.externalResources.http.useHttpsSettings=false

#ebx.externalResources.https.host=
#ebx.externalResources.https.port=
ebx.externalResources.https.path=resourcesAlias
```

Attention

When proxy mode is used, the URL to the `ebx-dataservices` module must be configured through the lineage administration panel. Note that the provided URL must end its path with `/ebx-dataservices`.

Reverse-proxy mode

If URLs generated by EBX®, for requests and external resources, must contain a different protocol than the one from the incoming request, a specific server name, a specific port number or a specific path prefix, properties may be configured as follows:

```
#####
#ebx.servlet.useLocalUrl=false
```

```

ebx.servlet.http.host=reverseDomain
#ebx.servlet.http.port=
ebx.servlet.http.path=ebx/
#ebx.servlet.http.ui.path=ebx-ui/
#ebx.servlet.http.useHttpsSettings=false

ebx.servlet.https.host=reverseDomain
#ebx.servlet.https.port=
ebx.servlet.https.path=ebx/
#ebx.servlet.https.ui.path=ebx-ui/

#####
## Web parameters (for external resources)
## if nothing is set, values are taken from servlet.
#####
#ebx.externalResources.useLocalUrl=false

#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
#ebx.externalResources.http.path=
ebx.externalResources.http.useHttpsSettings=true

ebx.externalResources.https.host=reverseDomain
#ebx.externalResources.https.port=
ebx.externalResources.https.path=

```

Attention

When reverse-proxy mode is used, the URL to the ebx-dataservices module must be configured through the lineage administration panel. Note that the provided URL must end its path with /ebx-dataservices.

59.16 Configuring failover

These parameters are used to configure the failover mode and activation key, as well as heartbeat logging in DEBUG mode.

Voir aussi [Failover with hot-standby](#) [p 413]

```

#####
## Mode used to qualify the way in which a server accesses the repository.
## Possible values are: unique, failovermain, failoverstandby.
## Default value is: unique.
#####
#ebx.repository.ownership.mode=unique

## Activation key used in case of failover. The backup server must include this
## key in the HTTP request used to transfer exclusive ownership of the repository.
## The activation key must be an alphanumeric ASCII string longer than 8 characters.
#ebx.repository.ownership.activationkey=

## Specifies whether to hide heartbeat logging in DEBUG mode.
## Default value is true.
#ebx.repository.ownership.hideHeartBeatLogForDebug=true

```

59.17 Tuning the EBX® repository

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

The properties are configured as follows:

```

#####
## Technical parameters for memory and performance tuning
#####
# Import commit threshold allows to specify the commit threshold
# exclusively for the archive import launched directly from Manager.
#
# For more details about the commit threshold,
# see the JavaDoc ProcedureContext.setCommitThreshold().
# Default value is 0.

```

```
#
ebx.manager.import.commit.threshold=100
```

Voir aussi [Validation report page](#) [p 389]

59.18 Miscellaneous

Activating data workflows

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

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

Disabling user task legacy mode

This parameter specifies whether the creation service of a user task in legacy mode should be offered in the workflow modeling. The default value is true.

See `UserTask.UserTaskMode.LEGACY_MODEAPI` for more information.

```
## Disables legacy work item mode(default is true)
## Specify if the creation service of user task in legacy mode must be offered
## in workflow modeling.
#ebx.manager.workflow.legacy.userTaskMode=false
```

Disabling hierarchy plan view

This parameter specifies whether the hierarchy plan view is hidden. The default value is true.

```
## Activate or deactivate Workflow hierarchy plan view
ebx.manager.workflow.hierarchyPlanView.hidden=false
```

Log procedure starts

This parameter specifies whether starts of the procedure execution are logged.

```
#####
## Specifies whether transaction starts are logged. Default is false.
#####
ebx.logs.logTransactionStart = true
```

Log validation starts

This parameter specifies whether starts of datasets validation are logged.

```
#####
## Specifies whether validation starts are logged. Default is false.
#####
ebx.logs.logValidationStart = true
```

Request duration threshold for logs

This parameter specifies in milliseconds the threshold of duration of Request^{API} and Query^{API} to be logged. Logs are generated if logging category `ebx.log4j.category.log.request` level is not higher than INFO. If the level is DEBUG, all Request^{API} and Query^{API} are logged.

```
#####
## Specifies in milliseconds the threshold of duration of Requests and Queries
```



```
## to be logged
## Default value is 1000 ms.
## If unset, the default value is used.
#####
#ebx.logs.request.durationThreshold=1000
```

Request duration threshold for logs

This parameter specifies in milliseconds the delay between 2 logs for Request^{API} and Query^{API} that goes beyond the threshold of duration. If this value is greater than 0, and the query duration goes beyond the threshold of duration, it will be logged again repeatedly with at least this delay between each log. As log messages include duration, this is useful to track long queries duration.

```
#####
## Specifies in milliseconds the delay between 2 logs for Requests and Queries that goes
## beyond the threshold of duration. If this value is greater than 0, and the query duration
## goes beyond the threshold of duration, it will be logged again repeatedly with at least
## this delay between each log.
## Default value is 30000 ms.
## If unset, the default value is used.
#####
#ebx.logs.request.logAgainEvery=30000
```

Deployment site identification

This parameter allows specifying the email address to which technical log emails are sent.

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

Dynamically reloading the main configuration

Some parameters can be dynamically reloaded, without restarting EBX®. The parameter `thisfile.checks.intervalInSeconds` indicates how frequently the main configuration file is checked.

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

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

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

Declaring modules as undeclared

On application server startup, the initialization of deployed web applications / EBX® modules and the initialization of the EBX® repository are performed asynchronously. In order to properly initialize the EBX® repository, it is necessary to compile all the data models used by at least a dataset, hence EBX® will wait endlessly for referenced modules to be registered.

If a module is referenced by a data model but is not deployed (or no longer deployed), it is necessary to declare this module as undeployed to unlock the wait and continue the startup process.

Note

The `kernel` logging category indicates which modules are awaited.

Note

A module declared as undeployed cannot be registered into EBX® until it is removed from the property `ebx.module.undeployedModules`.

Note

Any data model based on an unregistered module will have an "undeployed module" compilation error.

Voir aussi

[Module registration](#) [p 514]

[Dynamically reloading the main configuration](#) [p 385]

```
#####
## Comma-separated list of EBX® modules declared
## as undeployed.
## If a module is expected by the EBX® repository but is
## not deployed, it must be declared in this property.
## Caution:
## if the "thisfile.checks.intervalInSeconds" property is deactivated,
## a restart is mandatory, otherwise it will be hot-reloaded.
#####
ebx.module.undeployedModules=
```

Module public path prefix

EBX® modules' public paths are declared in the 'module.xml' file of each module. A context prefix can be declared for all modules, without having to modify the 'module.xml' content, by specifying the property that follows.

This prefix will apply to any EBX® module, including core, add-on and specific modules.

When proxy and / or reverse-proxy mode are used, the `ebx.servlet.http[s].path` and `ebx.servlet.http[s].ui.path` properties must take into account this module public path prefix setting. Conversely, the `ebx.externalResources.http[s].path` property must end its path just before a potential prefix.

```
#####
ebx.servlet.useLocalUrl=true

#ebx.servlet.http.host=
#ebx.servlet.http.port=
ebx.servlet.http.path=reverse-proxy/prefix/ebx/
ebx.servlet.http.ui.path=reverse-proxy/prefix/ebx-ui/
#ebx.servlet.http.useHttpsSettings=false

#ebx.servlet.https.host=
#ebx.servlet.https.port=
ebx.servlet.https.path=reverse-proxy/prefix/ebx/
ebx.servlet.https.ui.path=reverse-proxy/prefix/ebx-ui/

#####
## Web parameters (for external resources)
## if nothing is set, values are taken from servlet.
#####
ebx.externalResources.useLocalUrl=true

#ebx.externalResources.http.host=
#ebx.externalResources.http.port=
```

```

ebx.externalResources.http.path=reverse-proxy/
#ebx.externalResources.http.useHttpsSettings=false

#ebx.externalResources.https.host=
#ebx.externalResources.https.port=
ebx.externalResources.https.path=reverse-proxy/

#####
## EBX® Module context path prefix
##
## If defined, applies to all EBX® modules public paths declared in
## any module.xml file (core, add-on and specific).
#####
ebx.module.publicPath.prefix=prefix/

```

See [URLs computing](#) [p 381] for more information.

EBX® run mode

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

When running in *development* mode, the [development tools](#) [p 525] are activated in EBX®, some features thus become fully accessible and more technical information is displayed.

Note

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

The additional features accessible when running in *development* mode include the following (non-exhaustive list):

Documentation pane	In the case of a computed value, the Java class name is displayed. A button is displayed giving access to the path to a node.
Compilation information	Module and schema compilation information is displayed in the dataset validation report.
Java bindings	The generation of Java bindings is available if the schema of the dataset mentions at least one binding.
Générateur de liens pour composant Web	The Générateur de liens pour composant Web is available on datasets and dataspace.
Data model assistant	Data model configuration and additional options, such as Services, Business Objects and Rules, Java Bindings, Toolbars and some advanced properties.
Workflow modeling	Declare specific script tasks.
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.
Product documentation	The product documentation is always the most complete one (i.e "advanced"), including administration and development chapters.

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

Note

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

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 can be of type [pattern] or "m:[pattern]:".
## the list can be void
#####
```

```
ebx.resource.exclude=CVS/*
```

Validation report page

The validation report page can display a finite number of items for each severity. This number can be tuned with this property.

```
#####
## Defines the maximum item displayed for each severity in the validation report page.
## Default value is 100.
#####
ebx.validation.report.maxItemDisplayed=200
```

Voir aussi [Tuning the EBX® repository](#) [p 383]

Validation report logs

This property allows to specify the number of validation messages to display in the logs when validating a dataset or a table.

```
#####
## Defines the maximum number of messages displayed in the logs.
## Default value is 100.
## When set to 0 or a negative value, the limit is not considered.
#####
ebx.validation.report.maxItemDisplayedInLogs=500
```

Voir aussi [Tuning the EBX® repository](#) [p 383]

CHAPITRE 60

Initialization and first-launch assistant

Deliverables can be found on [TIBCO eDelivery](#) (an account is mandatory in order to access eDelivery, please contact [the support team](#) to request one).

The TIBCO EBX® Configuration Assistant helps with the initial configuration of the EBX® repository. If EBX® does not have a repository installed upon startup and if the [automatic installation](#) [p 372] is not enabled, the configuration assistant is launched automatically.

Before starting the configuration of the repository, make sure that EBX® is correctly deployed on the application server. See [Java EE deployment](#) [p 339].

Note

The EBX® main configuration file must also be properly configured. See [TIBCO EBX® main configuration file](#) [p 371].

Ce chapitre contient les sections suivantes :

1. [Configuration steps](#)

60.1 Configuration steps

The EBX® 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 EBX® repository.

Validating the license agreement

In order to proceed with the configuration, you must read and accept the product license agreement.

Configuring the repository

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

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

Voir aussi [TIBCO EBX® main configuration file](#) [p 371]

Defining users in the default directory

If a custom user and roles directory is not defined in the EBX® main configuration file, the configuration assistant allows to define default users for the default user and roles directory.

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

Voir aussi [Users and roles directory](#) [p 451]

Validating the information entered

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

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

Installing the EBX® repository

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

CHAPITRE 61

Deploying and registering TIBCO EBX® add-ons

Note

Refer to the documentation of each add-on for additional installation and configuration information in conjunction with this documentation.

Ce chapitre contient les sections suivantes :

1. [Deploying an add-on module](#)
2. [Registering an add-on module](#)
3. [Activating an add-on module](#)
4. [Deleting an add-on module](#)

61.1 Deploying an add-on module

Note

Each add-on bundle version is intended to run with a specific EBX® version and all its fix releases. Make sure that the EBX® and add-on bundle versions are compatible, otherwise the add-on registration will abort.

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.

The EBX® add-on common JAR file, named `lib/ebx-addons.jar`, must be copied in the library directory shared by all web applications.

Note

The add-on log level can be managed in the [main configuration file](#) [p 378].

61.2 Registering an add-on module

Registering an add-on makes its configuration available in the admin section. Add-on features are only available to end-users when the add-on is also [activated](#) [p 394].

To register a new EBX® add-on in the repository:

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

Note

Unregistering an add-on will not delete any existing configuration, but will make it available in the UI until the add-on is registered again.

61.3 Activating an add-on module

Activating an add-on makes its features available to the end-users. Only registered add-ons can be activated.

To activate an EBX® add-on in the repository:

1. Navigate to the 'Administration' area.
2. Click the down-arrow in the navigation pane and select **Technical configuration > Add-ons registration**.
3. Select the registered add-on you are activating and enable the 'Activation' field.
4. Click on **Save**.

61.4 Deleting an add-on module

To delete an add-on module from the EBX® repository:

1. Navigate to the 'Administration' area.
2. Click the down-arrow in the navigation pane and select **Technical configuration > Add-ons registration**.
3. On the **Registered add-ons** page, tick the box corresponding to the add-on to be deleted.
4. In the 'Actions' menu, select 'Delete'.
5. Close and purge the Administration datasets related to the previously used add-on, as well as the including dataspace.

When an add-on is no longer deployed, a dataspace corresponding to the Administration dataset will then appear in the list of Reference children under the dataspace. When an add-on module is no longer deployed, it is thus necessary to close/delete and purge manually all data/dataspace related to the add-on.

EBX® Container Edition

CHAPITRE 62

Building the image

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Workstation requirement](#)
3. [Building the image](#)

62.1 Overview

TIBCO EBX® Container Edition image is Linux based (**amd64** architecture) and includes the following:

- Application Server Apache Tomcat® 9.0
- Java JDK 11

62.2 Workstation requirement

The image can be built on a workstation meeting following requirements:

- **Operating system:** Windows 10, macOS or Linux,
- **Docker Desktop** installed and running,
- Access to **Internet**.

62.3 Building the image

Download the installer

To build an EBX® image, one needs to download file **TIB_ebx_{ebx.version.public}_addon_5.X.Y_container_edition.amd64.zip** from TIBCO eDelivery.

Running the installer

To start installer on **Linux** or **macOS**:

- Unzip the **TIB_ebx_{ebx.version.public}_addon_5.X.Y_container_edition.amd64.zip**,
- Open a terminal in folder where file **ebx-ce-installer.sh** file is located,

- Execute command **./ebx-ce-installer.sh**.

To start installer on **Windows 10**:

- Unzip the **TIB_ebx_{ebx.version.public}_addon_5.X.Y_container_edition.amd64.zip**,
- Open a Windows PowerShell in folder where file **ebx-ce-installer.bat** file is located,
- Execute command **ebx-ce-installer.bat**.

Follow instructions and select **addons** to be added to the image.

The installer will then build the image and print a summary similar to:

```
*****
The TIBCO EBX Container Edition image was successfully created with following names:
ebx:latest
ebx:{ebx.version.public}-mame-tese-dama-dmdv-5.X.Y
This image includes the following addon(s):
Match and Merge (MAME)
Information Search (TESE)
Digital Asset Manager (DAMA)
Data Model and Data Visualization (DMDV)
To run this image with the default configuration and an embedded database, use command:
docker run -p 8080:8080 -d ebx:latest
You can test EBX by visiting http://localhost:8080 in a browser.
To run this image with other configurations, for example with an external database, see
documentation.
*****
```

Testing the image

The image can be run locally using command:

```
docker run -p 8080:8080 -d ebx:latest
```

In **production**, it is recommended to **not** use tag **latest**. The installer will always generate another tag depending on selected addons.

If previous sample, the tag is **{ebx.version.public}-mame-tese-dama-dmdv-5.X.Y** because the following addons were selected:

- Match and Merge (MAME),
- Information Search (TESE),
- Digital Asset Manager (DAMA),
- Data Model and Data Visualization (DMDV).

The image can then be run using command:

```
docker run -p 8080:8080 -d ebx:{ebx.version.public}-mame-tese-dama-dmdv-5.X.Y
```

If no addons are selected, the tag is **{ebx.version.public}**

The image can be run using command:

```
docker run -p 8080:8080 -d ebx:{ebx.version.public}
```

Sharing the image

The steps to share an image depends on customer's company infrastructure.

In following example, the image is pushed to a Docker private registry named **myregistry**:

```
docker tag ebx:{ebx.version.public} myregistry:5000/ebx:{ebx.version.public}
docker push myregistry:5000/ebx:{ebx.version.public}
```


CHAPITRE 63

Running the image

Ce chapitre contient les sections suivantes :

1. [Starting EBX®](#)
2. [Container access](#)
3. [Environment variables](#)
4. [Configuration files](#)
5. [Volumes](#)
6. [Linux user and group](#)
7. [Logs access](#)

63.1 Starting EBX®

First-launch assistant

To start EBX® with default configuration that includes an embedded H2 database, execute command:

```
docker run -p 8080:8080 -d ebx:{ebx.version.public}
```

Using a browser, you can connect to EBX® with URL <http://localhost:8080>. This will display the **first-launch assistant** that will help you configure EBX®.

For more information on the first launch assistant, see chapter [Initialization and first-launch assistant](#) [p 391].

Automatic initialization

To start EBX® with automatic initialisation on first startup and an embedded H2 database, execute command:

```
docker run -d -p 8080:8080 \  
-e "EBX_FLA_DISABLED=true" \  
-e "EBX_INSTALL_ADMIN_PASSWORD=<password>" \  
ebx:{ebx.version.public}
```

The EBX® repository will be automatically created on first startup.

Using a browser, you can connect to EBX® with URL <http://localhost:8080>. This will display the EBX® login screen. The username for administrator is **admin** and the password is the one specified in previous command.

Note

It's possible to specify another username for the administrator. For more information see [Automatic repository installation on first launch](#) [p 402].

Supported browsers

For details a supported browsers see: [Supported Web Browsers](#) [p 333].

63.2 Container access

The following command will start a bash shell inside the EBX® container:

```
docker exec -it <container-id> bash
```

63.3 Environment variables

This chapter describes the environment variables supported by **EBX® Container Edition**. All are optional.

Disabling First-launch assistant

For security reasons, one might want to disable the first-launch assistant in all circumstances. This is achieved by setting environment variable to **EBX_FLA_DISABLED** to **true**.

Automatic repository installation on first launch

If the repository is not yet initialized and first-launch assistant is disabled, EBX® will automatically trigger its installation if following mandatory variables are provided:

Name	Default	Description
EBX_INSTALL_ADMIN_LOGIN	admin	Sets the EBX® administrator login name. This parameter is ignored if repository variable EBX_FLA_DISABLED value is not true or if repository is already initialized.
EBX_INSTALL_ADMIN_PASSWORD		Sets the EBX® administrator login name. This parameter is mandatory if EBX_FLA_DISABLED value is true and is ignored if repository variable EBX_FLA_DISABLED value is not true or if repository is already initialized.

Note

If mandatory variables are not provided, EBX® will display an error message.

Example

To automatically install repository launch EBX® using following command:

```
docker run -d -p 8080:8080 \
-e "EBX_FLA_DISABLED=true" \
-e "EBX_INSTALL_ADMIN_LOGIN=<login-name>" \
-e "EBX_INSTALL_ADMIN_PASSWORD=<password>" \
ebx:{ebx.version.public}
```

URL configuration

Some EBX® features require generating URLs. Specific configuration may be required to achieve this, for example if EBX® is running behind a reverse proxy or on a Kubernetes cluster.

Name	Default	Description
EBX_IS_SECURED	If incoming request is HTTPS, "true" is assumed or else "false" is assumed.	If "true", the protocol "HTTPS" is assumed. If "false", the protocol "HTTP" is assumed.
EBX_HOSTNAME	The host name specified by the incoming HTTP(S) request.	The EBX® server host name.
EBX_PORT	The port number specified by the incoming HTTP(S) request.	The EBX® server port number.
EBX_ROOT_PATH	By default, the context path is empty.	If set, all EBX® urls will be prefixed by this value. The value must have a leading / and must not have a trailing / except if value is /. For example a valid value is /mdm/sales. Setting this variable is useful when running more than one instance of EBX with the same host name.
EBX_URL_DEFAULT		This environment variable is used when a background task needs to calculate a URL to EBX. It should be set to a full URL without the path component (EBX_ROOT_PATH applies for the path component). For example a valid value is: https://host_name. If EBX_URL_DEFAULT is not specified and EBX_HOSTNAME is specified, a default is calculated with following assumptions: - If EBX_IS_SECURED is not specified, 'false' is assumed, - If EBX_PORT is not specified, 443 is assumed if EBX_IS_SECURED is true or else 80 is assumed.

Database connectivity

For information on supported databases see chapter [Supported databases](#) [p 336].

By default, an embedded H2 database is used. Data for this H2 database is persisted at location **/ebx/data/h2**.

An external database may be configured using following variables:

Name	TIBCO EBX® main configuration file equivalent	Description
EBX_DB_FACTORY	ebx.persistence.factory	Specifies the type of database server.
EBX_DB_URL	ebx.persistence.url	The JDBC URL. It has format: jdbc:postgresql://<database_host>:<database_port_number>/<database_name> .
EBX_DB_USER	ebx.persistence.user	The database user id.
EBX_DB_PASSWORD	ebx.persistence.password	The database user password.

For more information on these variables see their TIBCO EBX® main configuration file equivalent in chapter [Configuring the EBX® repository](#) [p 373].

Note

The container includes JDBC drivers only for H2, PostgreSQL and Microsoft SQL Server. Using other databases that are supported by EBX® require adding the driver.

For instructions on how to add a driver, see [Adding a new JDBC driver](#) [p 410].

Example

To start EBX® that connects to a Postgres database, execute following command:

```
docker run -d -p 8080:8080 \
-e "EBX_DB_FACTORY=postgresql" \
-e "EBX_DB_URL=jdbc:postgresql://<server_name>:5432/<database_name>" \
-e "EBX_DB_USER=<user_name>" \
-e "EBX_DB_PASSWORD=<user_name>" \
ebx:{ebx.version.public}
```

Email connectivity

The EBX Mail service can be configured through the following environment variables :

Name	TIBCO EBX® main configuration file equivalent	Default	Description
EBX_SMTP_HOST	ebx.mail.smtp.host		SMTP server host name.
EBX_SMTP_PORT	ebx.mail.smtp.port		SMTP server port number.
EBX_SMTP_LOGIN	ebx.mail.smtp.login		SMTP server login id.
EBX_SMTP_PASSWORD	ebx.mail.smtp.password		SMTP server login password.
EBX_SMTP_SSL_ENABLED	ebx.mail.smtp.ssl.activate	true	Enables SSL. Value can be 'true' or 'false'.
EBX_WORKFLOW_MAIL_SENDER	ebx.manager.workflow.mail.sender		The workflow sender email. If not set, Workflows cannot send notifications.

More information on the used properties can be found in chapter [Activating and configuring SMTP and emails](#) [p 378].

Memory configuration

Environment variables **JAVA_MEMORY_PERCENT** may be used to configure the percentage of the container memory that is assigned to the JVM the runs EBX®. It must be an integer value between 0 and 100.

If not set, a default value is used at startup.

Note

This variable is for advanced usage. Setting it too low or too high may cause runtime issues.

Authentication for REST services

Basic authentication for REST services is not enabled by default.

To enable this feature set environment variable **EBX_REST_AUTHENTICATION_BASIC** to **true**.

63.4 Configuration files

Two Java property files are currently used to configure EBX®.

On startup EBX® reads property files in the following order:

- **/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties**

- `/my_custom/conf/ebx-container.properties`

File ebx-default.properties

The file `/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties` sets default EBX® configuration properties for the container.

It should **never** be modified at runtime as this may prevent easily updating EBX® to a next version, instead use `/opt/ebx/conf/ebx-container.properties`.

File ebx-container.properties

The file `/opt/ebx/conf/ebx-container.properties` is by default empty. Any property value specified here will override the value set by `ebx-default.properties`.

This file is useful to change a property at runtime. To change a property at run time, create a new file, for example `/my_custom/conf/ebx-container.properties`, containing the new property values and mount de parent folder from the host to the container:

```
docker run -v /my-custom/conf:/opt/ebx/conf -p 8080:8080 -d ebx:{ebx.version.public}
```

For the list of properties supported by EBX® see chapter [TIBCO EBX® main configuration file](#) [p 371].

63.5 Volumes

This image defines the following volumes:

Location	Description
<code>/ebx/data</code>	The EBX® root directory is located in this volume. It contains EBX® indexes and, when H2 embedded database is used, persisted data.
<code>/ebx/logs</code>	This volume is used for log files.

Note

The volume `/ebx/data` should be mapped to a persistent volume even when an external database is used. If not, EBX will have to rebuild its indexes on startup which may considerably increase boot time.

63.6 Linux user and group

The Linux process running EBX is owned by user **ebx** (uid 1500).

User ebx's primary group is **ebx** (guid 1500).

63.7 Logs access

Logs are sent to the stdout and stderr output streams and can be viewed using the following command:

```
docker logs <container-id>
```

Logs for both EBX® and Tomcat will be displayed.

Log files are also available under folder `/ebx/logs`:

- EBX® logs files are in folder **/ebx/logs/ebx**
- Tomcat logs files are in folder **/ebx/logs/tomcat**.

CHAPITRE 64

Customizing the image

The EBX® Container Edition image can be used as a parent image to create a customized image.

Ce chapitre contient les sections suivantes :

1. [Setting default configuration](#)
2. [Adding a custom module](#)
3. [Adding a new locale](#)
4. [Adding a new JDBC driver](#)

64.1 Setting default configuration

Setting default EBX® configuration should not be based on `/opt/ebx/conf/ebx-container.properties` as this file may be overridden at runtime.

Instead, proceed as following in the Docker file:

- Rename file `/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties`, for example to **ebx-default-original.properties**.
- Create a new file `/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties` container new property values. This file must define property **ebx.file.previous** set to the original property file new name, for example **ebx-default-original.properties**.

For the list of properties supported by EBX® see chapter [TIBCO EBX® main configuration file](#) [p 371].

Here is a sample Docker file that set the locale to "en-US"

```
FROM ebx:{ebx.version.public}

RUN mv /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties \
    /opt/ebx/webapps/ebx/WEB-INF/ebx-default-original.properties

RUN echo "ebx.file.previous=ebx-default-original.properties" >> \
    /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties
RUN echo "ebx.locales.available=en-US" >> /opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties
```

64.2 Adding a custom module

One can extend EBX® by developing custom modules. An EBX® module is a standard Java EE web application, packaging various resources such as XML Schema documents, Java classes and static resources.

For more information on EBX® module see chapter [Packaging TIBCO EBX® modules](#) [p 513].

With EBX® Container Edition, it is recommended to deploy modules as "unpacked" (exploded) WARs. This allows a faster startup and avoids unnecessarily increasing container size because Tomcat will not need to unpack WAR files.

The recommended way to add a module to an image is to:

- Copy the WAR to folder **/opt/ebx/webapps**. As stated previously, exploded format is recommended.
- Create an associated **Tomcat context XML file** named **module_war_name.xml** and copy it to folder **/opt/ebx/contexts**.
- Optionally, copy shared JARs to folder **/opt/ebx/lib**.

The **Tomcat context XML file** should have the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context docBase="${ebx.container.base}/webapps/module_war_name"/>
```

Using variable **\${ebx.container.base}** is required for correct support of environment variable **EBX_ROOT_PATH**.

For more information on Tomcat contexts see https://tomcat.apache.org/tomcat-9.0-doc/config/context.html#Defining_a_context.

Here is a sample Docker file:

```
FROM ebx:{ebx.version.public}
COPY "/module_name.xml" "/opt/ebx/conf"
COPY "/module_name" "/opt/ebx/webapps"
```

64.3 Adding a new locale

To add a new local you must have the jar file containing the language pack and add the locale to the ebx configuration.

Here is a sample Docker file

```
FROM ebx:{ebx.version.public}

COPY "<path_to_lib>" "/opt/ebx/lib"

RUN mv "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties" \
      "/opt/ebx/webapps/ebx/WEB-INF/ebx-default-original.properties"

RUN echo "ebx.file.previous=ebx-default-original.properties" >> \
      "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties"
RUN echo "ebx.locales.available=es" >> "/opt/ebx/webapps/ebx/WEB-INF/ebx-default.properties"
```

In this example, the lib is copied and the locale is set to "es" using the method described in the "[setting default configuration](#)" [p 409] section

64.4 Adding a new JDBC driver

Adding a new JDBC driver is similar to adding a new library. You simply have to copy the jar file in the **/opt/ebx/lib** folder with the correct permission. Here is an example with the Oracle JDBC driver :

```
FROM ebx:{ebx.version.public}

ADD \
  https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc11/21.3.0.0/ojdbc11-21.3.0.0.jar \
  "/opt/ebx/lib/"

RUN "chmod +r /opt/ebx/lib/ojdbc11-21.3.0.0.jar"
```

See [Database drivers](#) [p 341] for more information.

Technical administration

CHAPITRE 65

Repository administration

Ce chapitre contient les sections suivantes :

1. [Technical architecture](#)
2. [Auto-increments](#)
3. [Repository management](#)
4. [Monitoring management](#)
5. [Dataspaces](#)

65.1 Technical architecture

Overview

The main principles of the TIBCO EBX® technical architecture are the following:

- A Java process (JVM) that runs EBX® is limited to a single EBX® repository. This repository is physically persisted in a [supported relational database instance](#) [p 336], accessed through a [configured data source](#) [p 373].
- A repository cannot be shared by multiple JVMs at any given time. However, a failover architecture may be used. These aspects are detailed in the sections [Single JVM per repository](#) [p 413] and [Failover with hot-standby](#) [p 413]. Furthermore, to achieve horizontal scalability, an alternative is to deploy a [distributed data delivery \(D3\)](#) [p 478] environment.
- A single relational database instance can support multiple EBX® repositories (used by distinct JVMs). It is then required that they specify distinct table prefixes using the property `ebx.persistence.table.prefix`.

Voir aussi

[Configuring the EBX® repository](#) [p 373]

[Supported databases](#) [p 336]

[Data source of the EBX® repository](#) [p 345]

Rules for the database access 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.**

It is required for the database user specified by the [configured data source](#) [p 373] to have the 'create/alter' privileges on tables, indexes and sequences. This allows for [automatic repository installation and upgrades](#) [p 415].

Voir aussi

[SQL access to history](#) [p 272]

[Accessing a replica table using SQL](#) [p 279]

[Data source of the EBX® repository](#) [p 345]

Single JVM per repository

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

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

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

Attention

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

Failover with hot-standby

The exclusion mechanism described above is compatible with failover architectures, where only one server is active at any given time in an active/passive cluster. To ensure that this is the case, the main server must declare the property `ebx.repository.ownership.mode=failovermain`. The main server claims ownership of the repository database, as in the case of a single server.

A backup server can still start up, but it will not have access to the repository. It must declare the property `ebx.repository.ownership.mode=failoverstandby` to act as the backup server. Moreover, is required for both servers to define the same value for `ebx.repository.directory`, and to share the directory defined by this value. (This is, in particular, so that the Lucene indexes can be shared, i.e. not rebuilt on demand when the failover server starts.) Once started, the backup server is registered in the connection log. Its status can be retrieved using the Java API or through an HTTP request, as described in the section [Repository status information and logs](#) [p 414] 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 EBX® main configuration file. See [Configuring failover](#) [p 383].

The format of the request URL must be:

```
http[s]://<host>[:<port>]/ebx?activationKeyFromStandbyMode={value}
```

- Using the Java API, call the method `RepositoryStatus.wakeFromStandbyAPI`.

If the main server is still up and accessing the database, the following applies: the backup server marks the ownership table in the database, requesting a clean shutdown for the main server (yet allowing any running transactions to finish). Only after the main server has returned ownership can the backup server start using the repository.

Repository status information and logs

A log of all attempted Java process connections to the repository is available in the Administration area under '[History and logs](#) [p 269]' > 'Repository connection log'.

The status of the repository may be retrieved using the methods in the `RepositoryStatusAPI` API.

It is also possible to get the repository status information using an HTTP request that includes the parameter `repositoryInformationRequest` with one of following values:

state	The state of the repository in terms of ownership registration. • D: Java process is stopped. • O: Java process has exclusive ownership of the database. • S: Java process is started in failover standby mode, but is not yet allowed to interact with the repository. • N: Java process has tried to take ownership of the database but failed because another process is holding it.
heart_beat_count	The number of times that the repository has made contact since associating with the database.
info	Detailed information for the end-user regarding the repository's registration status. The format of this information may be subject to modifications in the future without explicit warning.

65.2 Auto-increments

Several technical tables can be accessed in the 'Administration' area of the EBX® user interface. These tables are for internal use only and their content should not be edited manually, unless removing obsolete or erroneous data. Among these technical tables are:

Auto-increments	Lists all auto-increment fields in the repository.

65.3 Repository management

Installation and upgrades

Automatic installation and upgrades

By complying with the [Rules for the database access and user privileges](#) [p 413], the repository installation or upgrade is done automatically.

Inter-database migration

EBX® provides a way to export the full content of a repository to another database. The export includes all dataspace, configuration datasets, and mapped tables. To operate this migration, the following guidelines must be respected:

- The source repository **must be shut down**: no EBX® server process must be accessing it; **not strictly complying with this requirement can lead to a corrupted target repository**;
- A new EBX® server process must be launched on the target repository, which must be empty. In addition to the classic Java system property `-Debx.properties`, this process must also specify `ebx.migration.source.properties`: the location of an EBX® properties file specifying the source repository. (It is allowed to provide distinct table prefixes between target and source.)
- The migration process will then take place automatically. Please note, however, that this process is not transactional: should it fail halfway, it will be necessary to delete the created objects in the target database, before starting over.
- After the migration is complete, an exception will be thrown, to force restarting the EBX® server process accessing the target repository.

Limitations:

- The names of the database objects representing the mapped tables (history, replication) may have to be altered when migrated to the target database, to comply with the limitations of its database engine (maximum length, reserved words, ...). Such alterations will be logged during the migration process.
- As a consequence, the names specified for replicated tables in the data model will not be consistent with the adapted name in the database. The first recompilation of this data model will force to correct this inconsistency.
- Due to different representations of numeric types, values for `xs:decimal` types might get rounded if the target database engine offers a lesser precision than the source. For example, a value of `10000000.1234567890123456789` in Oracle will get rounded to `10000000.123456789012345679` in SQL Server.

Repository backup

A global backup of the EBX® 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 called `archives` within the `ebx.repository.directory` (see [configuration](#) [p 371]). This directory is automatically created during the first export from EBX®.

Attention

As specified in the [security best practices](#) [p 510], access to this directory must be carefully protected. Also, if manually creating this directory, make sure that the EBX® process has read-write access to it. Furthermore, the administrator is responsible for cleaning this directory, as EBX® does not maintain it.

Note

The transfer of files between two EBX® environments must be performed using tools such as FTP or simple file copies by network sharing.

Repository attributes

A repository has the following attributes:

repositoryId	Uniquely identifies a repository within the scope of the company. It is 48 bits (6 bytes) and is usually represented as 12 hexadecimal digits. This information is used for generating UUIDs (Universally Unique Identifiers) for entities created in the repository, as well as transactions logged in history tables or in the XML audit trail. This identifier acts as the 'UUID node' part, as specified by <i>RFC 4122</i> .
repository label	Provides a user-friendly label that identifies the purpose and context of the repository. For example: "Production environment".
store format	Identifies the underlying persistence system, including the current version of its structure.

65.4 Monitoring management

Monitoring and cleanup of the relational database

Some entities accumulate during the execution of EBX®.

Attention

It is the *administrator's responsibility* to monitor and clean up these entities.

Database monitoring

The persistence data source of the repository must be monitored through RDBMS monitoring.

Database statistics

The performance of requests executed by EBX® requires that the database has computed up-to-date statistics on its tables. Since database engines regularly schedule statistics updates, this is usually not an issue. Yet, it could be necessary to explicitly update the statistics in cases where tables are heavily modified over a short period of time (e.g. by an import creating many records).

History tables: impact on UI

For history tables, some UI components use statistics to adapt their behavior in order to prevent users from executing costly requests unwillingly.

For example, the combo box will not automatically search on user input if the table contains a large volume of records. This behavior may also occur if the database's statistics are not up to date, because a table may be considered as containing a large volume of records even if it is not actually the case.

Cleaning up dataspaces, snapshots, and history

A full cleanup of dataspaces, snapshots, and history from the repository involves several stages:

1. Closing unused dataspaces and snapshots to keep the cache to a minimal size.
2. Deleting dataspaces, snapshots, and history.
3. Purging the remaining entities associated with the deleted dataspaces, snapshots, and history from the repository.

Closing unused dataspaces and snapshots

In order to keep the cache and the repository to a reasonable size, it is recommended to close any dataspaces and snapshots that are no longer required. This can be done in the following ways:

- Through the user interface, in the 'Dataspaces' area.
- From the 'Dataspaces / Snapshots' table under 'Dataspaces' in the 'Administration' area, using the **Actions** menu in the workspace. The action can be used on a filtered view of the table.
- Through the Java API, using the method `Repository.closeHomeAPI`.
- Using the data service "close dataspace" and "close snapshot" operations. See [Closing a dataspace or snapshot](#) [p 739] for more information.

Once the dataspace and snapshots have been closed, the data can be safely removed from the repository.

Note

Closed dataspace and snapshots can be reopened in the 'Administration' area, under 'Dataspace'.

Deleting dataspace, snapshots, and history

Dataspace, associated history and snapshots can be permanently deleted from the repository. However, the deletion of a dataspace does not necessarily imply the deletion of its history. The two operations are independent and can be performed at different times.

Note

The deletion of a dataspace, a snapshot, or of the history associated with them is recursive. The deletion operation will be performed on every descendant of the selected dataspace.

After the deletion of a dataspace or snapshot, some entities will remain until a repository-wide purge of obsolete data is performed. In particular, the complete history of a dataspace remains visible until a repository-wide purge is performed. Both steps, the deletion and the repository-wide purge, must be completed in order to totally remove the data and history. The process has been divided into two steps for performance issues. 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.

The process of deleting the history of a dataspace takes into account all history transactions recorded up until the deletion is submitted or until a date specified by the user. Any subsequent historized operations will not be included when the purge operation is executed. To delete new transactions, the history of the dataspace must be deleted again.

Note

It is not possible to set a deletion date in the future. The specified date will thus be ignored and the current date will be used instead.

The deletion of dataspace, snapshots, and history can be performed in a number of different ways:

- From the 'Dataspace/Snapshots' table under 'Dataspace' in the 'Administration' area, using the **Actions** menu button in the workspace. The action can be used on a filtered view of the table.
- Using the Java API, and more specifically the methods `Repository.deleteHomeAPI` and `RepositoryPurge.markHomeForHistoryPurgeAPI`.
- At the end of the data service "close dataspace" operation, using the parameters `deleteDataOnClose` and `deleteHistoryOnClose`, or at the end of a "merge dataspace" operation, using the parameters `deleteDataOnMerge` and `deleteHistoryOnMerge`.

Purging remaining entities after a dataspace, snapshot, or history deletion

Once items have been deleted, a purge can be executed to clean up remaining data from *all* deletions performed until that point. A purge can be initiated in the following ways:

- Through the user interface, by selecting in the 'Administration' area **Actions > Execute purge** in the navigation pane.
- Using the Java API, specifically the method `RepositoryPurge.purgeAllAPI`.

- Using the task scheduler. See [Task scheduler](#) [p 465] for more information.

The purge process is logged in the directory `${ebx.repository.directory}/db.purge/`.

Cleaning up other repository entities

It is the *administrator's responsibility* to monitor and regularly cleanup the following entities.

Purge

A purge can be executed to clean up the remaining data from *all* deletions, that is, deleted dataspace, snapshots and history performed up until that point. A purge can be initiated by selecting in the 'Administration' area **Actions > Execute purge** in the navigation pane.

Task scheduler execution reports

Task scheduler execution reports are persisted in the 'executions report' table, in the 'Task scheduler' section of the 'Administration' area. Scheduled tasks constantly add to this table as they are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

User interactions

User interactions are used by the EBX® component as a reliable means for an application to initiate and get the result of a service execution. They are persisted in the *ebx-interactions* administration section. It is recommended to regularly monitor the user interactions table, as well as to clean it, if needed.

Workflow history

The workflow events are persisted in the workflow history table, in the 'Workflow' section of the 'Administration' area. Data workflows constantly add to this table as they are executed. Even when an execution terminates normally, the records are not automatically deleted. It is thus recommended to delete old records regularly.

The steps to clean history are the following

- Make sure the process executions are removed (it can be done by selecting in the 'Administration' area of Workflows **Actions > Terminate and clean this workflow** or **Actions > Clean from a date** in the navigation pane).
- Clean main processes in history (it can be done by selecting in the 'Administration' area of Workflows history **Actions > Clear from a date** or **Actions > Clean from selected workflows** in the navigation pane).
- Purge remaining entities in workflow history using 'standard EBX® purge'

Voir aussi [the standard EBX® purge](#) [p 418]

Monitoring and clean up of the file system

Attention

In order to guarantee the correct operation of EBX®, the disk usage and disk availability of the following directories must be supervised by the administrator, as EBX® does not perform any clean up, except for Lucene indexes:

- **Lucene indexes:** `${ebx.repository.directory}/indexes-(...)/`

Lucene indexes: Indexes can require a lot of disk space; they are critical to the correct functioning of EBX®. In nominal usage, they must not be deleted or modified directly. However, there are cases where deleting these indexes might be needed:

- If the repository is recreated from scratch, whereas the directory `${ebx.repository.directory}/` is preserved; to ensure consistency of data, it is then required to delete the root directory of the indexes.
- More generally, if the indexes have become inconsistent with the repository data (this could happen in rare cases of bugs).

After deletion, the content of the indexes will be lazily recomputed per table, derived from the content of the repository. The deletion must happen at the root folder of the indexes: if a single directory is deleted at a lower level, the global structure of the index will become inconsistent. This operation, however, has a cost, and should generally be avoided.

- **XML audit trail:** `${ebx.repository.directory}/History/`
- **Archives:** `${ebx.repository.directory}/archives/`
- **Logs:** `ebx.logs.directory` [p 376]
- **Temporary directory:** `ebx.temp.directory` [p 375]

Attention

For **XML audit trail**, if large transactions are executed with full update details activated (contrary to the default setting), the required disk space can increase.

Attention

For pagination in the data services `getChanges` operation, a persistent store is used in the **Temporary directory**. Large changes may require a large amount of disk space.

Voir aussi

[XML audit Trail](#) [p 472]

[Tuning the EBX® repository](#) [p 383]

65.5 Dataspaces

Some dataspace administrative tasks can be performed from the 'Administration' area of EBX® by selecting 'Dataspaces'.

Dataspaces/snapshots

This table lists all the existing dataspaces and snapshots in the repository, whether open or closed. You can view and modify the information of dataspaces included in this table.

Voir aussi [Dataspace information](#) [p 106]

From this section, it is also possible to close open dataspaces, reopen previously closed dataspaces, as well as delete and purge open or closed dataspaces, associated history, and snapshots.

Voir aussi [Cleaning up dataspace, snapshots, and history](#) [p 417]

Dataspace permissions

This table lists all the existing permission rules defined on all the dataspace in the repository. You can view the permission rules and modify their information.

Voir aussi [Permissions sur un espace de données](#) [p 108]

Repository history

The table 'Deleted dataspace/snapshots' lists all the dataspace that have already been purged from the repository.

From this section, it is also possible to delete the history of purged dataspace.

CHAPITRE 66

UI administration

TIBCO EBX® comes with a full user interface called [Advanced perspective](#) [p 424] that includes all available features. The interface is fully [customizable](#) [p 428] (custom logo, colors, field size, default values, etc.) and available to built-in administrators.

Access to the advanced perspective can be restricted in order to simplify the end-user experience, through [global permissions](#) [p 423], giving the possibility to grant or restrict access to functional categories. Administrators can create simplified perspectives called [recommended perspectives](#) [p 436] for end-users, containing only the features and menus they need for their daily tasks.

Ce chapitre contient les sections suivantes :

1. [Global permissions](#)
2. [Advanced perspective](#)
3. [Recommended perspectives](#)
4. [Custom views](#)
5. [User session management](#)

66.1 Global permissions

Global permission rules can be created in EBX®.

The 'Display area' property allows restricting access to areas of the user interface. To define the access rules, select 'Global permissions' in the 'Administration' area.

Profil	Indicates on which profile the rule will be applied.
Restriction d'accès	Indique si les permissions définies ici restreignent celles définies pour d'autres profils. See the Restriction policy concept [p 304] for more information.
Espaces de données	Définit les permissions pour la section Espaces de données.
Modèles de données	Définit les permissions pour la section Modèles de données.
Modèles de workflow	Définit les permissions pour la section Modèles de workflow.
Workflows de données	Définit les permissions pour la section Workflows de données.
Services de données	Définit les permissions pour la section Services de données. Independently, it is also possible to: • Définit les permissions d'accès aux services REST prédéfinis sur http(s). • Définit les permissions d'accès aux services REST de OpenAPI. • Définit les permissions d'accès au connecteur SOAP sur http(s) et JMS. • Définit les permissions d'accès au connecteur WSDL sur http(s).
Administration	Définit les permissions pour la section Administration.

Note

Permissions can be defined by administrators and by the dataspace or dataset owner.

66.2 Advanced perspective

The advanced perspective and its parameterization are unique. It is the parent perspective from which any [new perspective](#) [p 436] will inherit.

Children perspectives can be created from that main perspective in order to offer a customized, simplified menu to the end-users. Thanks to dataset inheritance, these simplified perspectives will receive their parameters from the advanced perspective (the root dataset). These parameters can then be overridden on the newly created simplified perspectives. Simplified perspectives can be created underneath an existing simplified perspective, thus inheriting from the parent's parameters.

Voir aussi [Inheritance](#) [p 29]

The advanced perspective is available by default to all end-users but access can be restricted.

Note: Administrators can always access the advanced perspective even when it is deactivated.

It is possible to configure which perspective is applied by default when users log in. This 'default perspective' is based on two criteria: 'recommended perspectives', defined by administrators and 'favorite perspectives', defined by users.

Voir aussi

[Recommended perspectives](#) [p 436]

[Favorite perspectives](#) [p 21]

Perspective creation

To create a perspective, open the 'Select an administration feature' drop-down menu and click on the + sign to create a child dataset.

Voir aussi [Creating an inheriting child dataset](#) [p 126]

User interface

Options are available in the Administration area for configuring the web interface, in the 'User interface' section.

Attention

Be careful when configuring the [URL policy \(deprecated\)](#) [p 426]. If the web interface configuration is invalid, it can lead to the unusability of EBX®. If this occurs, use the "rescue mode" by setting `frontEnd.rescueMode.enable=true` in [EBX® main configuration file](#) [p 371], and accessing the following URL in your browser as a built-in administrator user: `http://.../ebx/?onwbpID=iebx-manager-rescue`.

Session configuration

These parameters configure the user session options:

User session default locale	Default session locale
Session time-out (in seconds)	Maximum duration of user inactivity before the session is considered inactive and is terminated. A negative value indicates that the session should never timeout.

Interface configuration

Entry policy

Describes the URL to access the application.

Login URL	If the user is not authenticated, the session is forwarded to this URL.

The entry policy defines an EBX® login page, replacing the default one.

If defined,

- it replaces an authentication URL that may have been defined using a specific user Directory^{API},
- it is used to build the permalinks in the user interface,
- if the URL is full, that is, starting with `http://` or `https://`, it replaces the URL of the workflow email configuration.

URL policy (deprecated)

Describes the URL and proxy policy. Both dynamic (servlet) and static (resources) URLs can be configured.

This configuration manner is deprecated and must be replaced by [URLs computing](#) [p 381]. After configuring the EBX® main configuration file, these configurations must be unset.

HTTP servlet policy	Header content of the servlet HTTP request: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
HTTPS servlet policy	Header content of the servlet HTTPS request: • if a field is not set, the default value is chosen (in an environment configuration), • if a default value is not set, the value in the initial request is used.
HTTP external resources policy	Header content of the external resources URL in HTTP: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
HTTPS external resources policy	Header content of the external resources URL in HTTPS: • if a field is not set, the default value in the environment configuration is used, • if a default value is not set, the value in the initial request is used.
Exit policy Describes how the application is exited.	
Normal redirection	Specifies the redirection URL used when exiting the session normally.
Error redirection	Specifies the redirection URL used when exiting the session due to an error. This feature is now deprecated and may be ignored by EBX®.
Redirection restrictions	Specifies the list of authorized domains and whether HTTPS is mandatory for each domain.

Graphical interface configuration

Activation & Allowed profiles

The 'Activated' radio button allows to activate or deactivate the perspective. When deactivated, the perspective will only be made available to the administrator.

The 'Allowed profiles' feature is used to give access to the perspective to a given profile. Several profiles can be added to the list of authorized profiles by clicking on the + icon below the numbered list.

The available perspective properties are:

Activé	Indique que la perspective est visible des utilisateurs autorisés.
Profils Autorisés	La liste des profils utilisateur autorisés pour la perspective.
Appareils autorisés	La liste des appareils autorisés pour la perspective. If not specified, only "EBX® Web Application" can display this perspective.
Sélection par défaut	L'élément de menu qui est sélectionné par défaut. This property is not available for the advanced perspective.

Application locking

EBX® 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

EBX® access security policy. These parameters only apply to 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 EBX®. 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 EBX® 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

EBX® 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 lors de l'initialisation de la table. 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 messages automatiquement	Defines the message severity threshold for displaying the messages pop-up.
Mode de compatibilité IE	Defines whether or not to compensate for Internet Explorer 8+ displaying EBX® in compatibility mode. 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'. See Specifying Document Compatibility Modes  for more information.
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 : hauteur des champs texte	La hauteur des champs de saisie de type texte dans les formulaires.
Formulaires : édition de liste	Nombre de lignes cachées à générer dans l'interface utilisateur, disponibles pour créer de nouvelles occurrences dans la liste.
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.
Sélection en liste : taille de page	Nombre maximum de lignes transmises à chaque requête du composant de sélection dans une liste (utilisé pour sélectionner les clé étrangères, les énumérations, etc.).
Formulaire d'enregistrement : mode de présentation des noeuds	Ajuste la manière d'afficher chaque noeud non terminal du formulaire d'enregistrement. En cas de modification de cette propriété, les utilisateurs doivent se déconnecter et se reconnecter afin que la nouvelle valeur soit prise en compte.
Formulaire d'enregistrement : affichage des noeuds de sélection et d'association en création	Si activé, les noeuds de sélection et d'association seront affichés dans les formulaires de création.
Densité d'affichage	Définit la densité d'affichage par défaut pour tous les utilisateurs. Si aucune densité n'a été sélectionnée par l'utilisateur, c'est cette valeur qui sera appliquée. Dans le cas contraire, le choix de l'utilisateur prévaut.
Affichage de l'avatar dans l'en-tête	Cette propriété définit le mode d'affichage de l'avatar dans l'en-tête. Il est notamment possible d'activer ou désactiver l'utilisation des avatars dans l'en-tête en modifiant cette propriété. Si aucune valeur n'est définie, la valeur par défaut est 'Avatar seulement'.
Affichage de l'avatar dans l'historique	Cette propriété définit le mode d'affichage de l'avatar dans l'historique. Il est notamment possible d'activer ou désactiver l'utilisation des avatars dans l'historique en modifiant cette propriété. Si aucune valeur n'est définie, la valeur par défaut est 'Avatar seulement'.

Affichage de l'avatar dans le workflow

Cette propriété définit le mode d'affichage de l'avatar dans le workflow. Il est notamment possible d'activer ou désactiver l'utilisation des avatars dans le workflow en modifiant cette propriété. Si aucune valeur n'est définie, la valeur par défaut est 'Avatar seulement'.

Default option values

Defines default values for options in the user interface.

Import/Export

CSV : jeu de caractères

Définit le jeu de caractères utilisé par défaut pour les imports et les exports CSV.

CSV : séparateur de champ

Définit le caractère séparateur utilisé par défaut pour les imports et les exports CSV.

CSV : séparateur de liste

Définit le caractère séparateur de liste utilisé par défaut pour les imports et les exports CSV.

Mode d'import

Spécifie le mode utilisé par défaut pour les imports.

Valeurs XML manquantes à nul

Si 'Oui', quand 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 EBX® 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. Si l'URL est relative, préfixez-la avec "../" afin de remonter à l'URL racine de l'application.
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. Si l'URL est relative, préfixez-la avec "../" afin de remonter à l'URL racine de l'application.
URL du logo (GIF/JPG)	L'image GIF/JPG est utilisée quand les images PNG et SVG ne sont pas renseignées. 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. Si l'URL est relative, préfixez-la avec "../" afin de remonter à l'URL racine de l'application.
Principale	Couleur principale de l'interface utilisateur, utilisée pour les sélections et les surbrillances.
Bandeau	Couleur de fond du bandeau de l'interface utilisateur. Par défaut, définie à la même valeur que la couleur principale.
Pastille du workflow	Couleurs de fond et de texte/bordure de la pastille du workflow (compteur de workflows utilisateur).
Boutons primaires	Couleur des boutons sélectionnés par défaut. Par défaut, définie à la même valeur que la couleur principale.

Texte des boutons style lien	Couleur du texte de quelques boutons qui ont un style lien (le texte n'est ni foncé ni clair, il est coloré). Par défaut, définie à la même valeur que la couleur principale.
Bordure de l'onglet sélectionné	Couleur de bordure de l'onglet sélectionné. Par défaut, définie à la même valeur que la couleur principale.
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.

Child perspective menu

An unlimited number of child perspectives can be created. Child perspectives inherit from the parameters of the 'Advanced perspective'. Some of these parameters can be overridden as detailed hereafter.

Activation & Allowed profiles

See [Activation and Allowed profiles for the Advanced perspective](#) [p 428] for more information.

Note

Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

Perspective Menu

This view displays the perspective menu. It is a hierarchical table view.

From this view, a user can create, delete or reorder menu item records.

Voir aussi [Hierarchical table view](#) [p 29]

Section Menu Item	This is a top level menu item. It contains other menu items.
Menu group	This is a container for other menu items.
Action Menu Item	This menu item displays a user service in the workspace area.

Menu item properties

When creating a record in the 'Perspective' Menu, the available perspective properties are:

Type	Le type de l'élément de menu Voir aussi Menu item types [p 434]
Parent	Le parent de l'élément de menu. This property is not available for section menu items.
Libellé	Le libellé de l'élément de menu. The label is optional for action menu items. If not specified, the label will be dynamically generated by EBX® when the menu item is displayed.
Appareils autorisés	La liste des appareils autorisés pour cet élément. If not specified, all devices can display this menu item. Currently only two devices are supported : "EBX® Web Application" and "EBX® GO".
 Icône	L'icône pour l'élément de menu. Icon can be either "standard" (provided by EBX®) or an image, specified by a URL, that can be hosted on any web server. Icons size should be 16x16 pixels. This property is not available for section menu items.
Séparateur haut	Indique que la section élément de menu a un séparateur haut. This property is only available for section menu items.
Action	The user service to execute when the user clicks on the menu item. Voir aussi User interface services [p 661] If an end-user is allowed to view the perspective but not to execute the user service, an "access denied" message will be displayed when the user clicks on the menu item. This property is only available for action menu items.
Selection après fermeture	L'élément de menu qui sera sélectionné quand le service se terminera. Built-in services use this property when the user clicks on the 'Close' button.

This property is only available for action menu items.

Ergonomics and layout

See [Ergonomics and layout for the Advanced perspective](#) [p 430] for more information.

Note

Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

Colors and themes

See [Colors and themes for the Advanced perspective](#) [p 433] for more information.

Note

Any specific parameter set for this perspective will override the default parameters that have been set in the 'Advanced perspective' configuration.

66.3 Recommended perspectives

It is possible for a perspective administrator to configure recommended perspectives dedicated to a specific audience. These recommended perspectives are a way to choose which perspective is applied by default when a user logs in, based on their role.

However, users always have the possibility to switch between the various perspectives that are available to them and to set one as their favorite. See [Perspectives favorites](#) [p 21] for more information.

To configure recommended perspectives, go to *User interface > Recommended perspectives > Manage recommended perspectives*.

Managing recommended perspectives

The main screen shows an ordered list of records associating a profile with a perspective. Note that the order here is important since a user can match more than one record (see [Resolution](#) [p 436] for more information).

- To add an entry, use the 'Create' action.
- To edit an entry, first select it in the list by clicking on it, then click on the 'Edit' action, or simply double-click on it.
- To remove an entry, first select it in the list, then click on the 'Delete' action.
- To move an entry, first select it in the list, then use the actions in the toolbar to the right of the list.

Resolution

When a user logs in, the following algorithm determines which perspective is selected by default:

```
// 1) favorite perspective
IF the user has a favorite perspective
AND this perspective is active
AND the user is authorized for this perspective
  SELECT this perspective
  DONE

// 2) recommended perspective
FOR EACH association in the recommended perspectives list, in the declared order
  IF the user is in the declared profile
```

```

    AND the associated perspective is active
    AND the user is authorized for the associated perspective
    SELECT this perspective
    DONE

// 3) advanced perspective
IF the advanced perspective is active
AND the user is authorized for this perspective
    SELECT this perspective
    DONE

// 4) any perspective
SELECT any active perspective for which the user is authorized
DONE

```

66.4 Custom views

Users can create and manage custom views directly from the 'View' menu on tables. This administration section is the central point to manage these custom views.

Vues

This table contains all custom views defined on any table. Only a subset of fields is editable:

Documentation	Libellés et descriptions localisés.
Propriétaire	Profil propriétaire (auteur) de cette spécification de vue.
Groupe de la vue	Groupe d'appartenance de cette vue.
Partager avec	Autres profils pouvant utiliser cette vue depuis le menu 'Vue'.

Permissions des vues

This table allows to manage permissions relative to custom views, by data model and profile. The following permissions can be configured (the default value is applied when no permission is set for a given user):

Permission	Description	Default value
Recommander des vues	Autorise l'utilisateur à gérer les vues recommandées.	If the user is the dataset owner, the default value is 'Yes', otherwise it is 'No'.
Gérer des vues	Définit les vues que l'utilisateur peut modifier et supprimer.	If the user is a built-in administrator, the default value is 'Owned + shared', otherwise it is 'Owned'.
Partager des vues	Définit les vues pour lesquelles l'utilisateur peut modifier le champ 'Partager avec'.	If the user is a built-in administrator, the default value is 'Owned + shared', else if the user is the dataset owner, it is 'Owned', otherwise it is 'None'.
Publier des vues	Autorise l'utilisateur à publier des vues afin de les rendre accessibles à tous les autres utilisateurs grâce aux composants web, tâches utilisateur du workflow, ou services de données.	If the user is a built-in administrator, the default value is 'Yes', otherwise it is 'No'.

66.5 User session management

This tool lists all user sessions and allows terminating active sessions when necessary.

For example: it is possible to invalidate and terminate all currently open and active sessions for maintenance purposes. The access to the user interface can be temporarily closed, with an unavailability message being displayed, through [Application locking](#) [p 428]. After active sessions are terminated, users will not be able to reconnect and will see the unavailability message. The maintenance operation can then be performed.

CHAPITRE 67

UI – Workflow launcher

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Workflow launcher in data section](#)
3. [Creating and setting a launcher](#)
4. [Activating workflow launcher](#)
5. [Launching a workflow](#)
6. [Adding workflow launcher on a custom toolbar](#)
7. [Access a launcher after workflow model modification](#)

67.1 Introduction

A **Workflow Launcher** is a **user service** for **launching workflows in TIBCO EBX®** directly from the data section without passing by the data workflow's inbox. This feature does not create workflow publications but launches existing ones. It offers several advantages, including the ability to launch workflow publication directly from the **data section** (table view, hierarchy view or record form view). In this way, the user experience is improved by avoiding to the user shifting his attention back and forth between the data section and the data workflow section. Hence, the user can launch a workflow while still focusing on his main task.

The second advantage is that it allows to launch the same workflow publication from any data selection. Thanks to the **dynamic mapping** of the [workflow data context](#) [p 32] with the current data selection. The dynamic mapping offers the possibility to initialize the data context inputs at launch time. Hence, in order to launch the same workflow from n different data selections, it is not longer necessary to duplicate n times the same workflow model with different data selections or to provide an initial user service to configure the data to select. The last solution is a programmatic solution which would solve the previously cited problems, however it is not the ideal solution because it does not fulfill the commitment zero line of code.

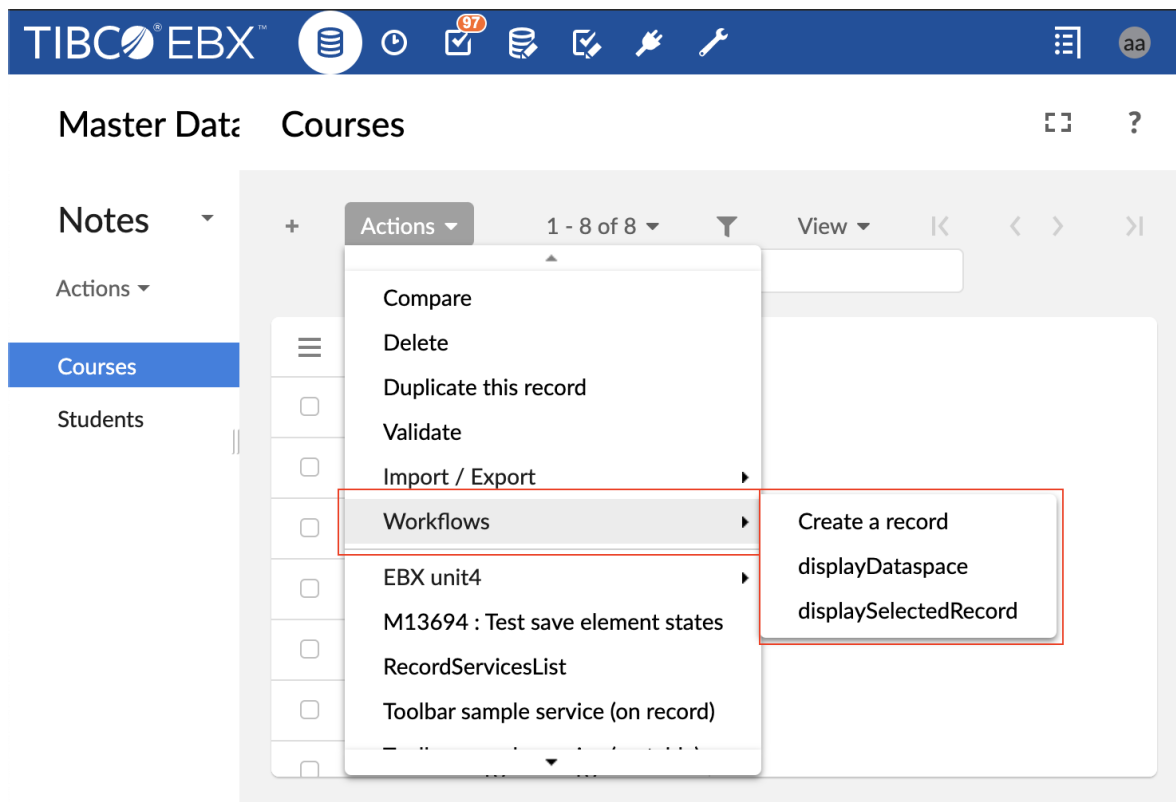
Understanding this feature requires familiarization with the following concepts:

Concept	Description/Link
Workflow model	See documentation [p 31]
Data context	See documentation [p 32]
Workflow publication	See documentation [p 33]
Publication name	A unique identifier of a workflow publication
Data Workflow	See documentation [p 32]
Data section	Is the data user interface [p 123] which displays the datasets and tables in EBX. it is accessible from the main header.
Workflow launchers dataset	It contains two tables Launchers and Activations . With this dataset the user can configure the launchers of workflow publications and activate them for a particular user(s) and table(s). It is available in the Administration area in a section called workflow management .
Launcher	An entity which is used by the service workflow launcher to identify the workflow publication to launch and how to initialize its data context .
Launchers	A table in the workflow launchers dataset, each record is a Launcher type. In order to be able to launch a workflow publication from a data section, a launcher which points to a workflow publication should be added in this table.
Activation	An entity which hides or shows the launcher of a workflow publication for a given user profile(s) and for a particular table(s) from the data section.
Activations	A table in the workflow launchers dataset, each record is an Activation type. In order to make a workflow available on the toolbar of a particular table and for a particular user, a record with the corresponding launcher should be created in this table.
Dynamic mapping of data context	Is the process of initialization of data context input variables when launching a workflow publication. Before, the values of the data context were static defined at modeling phase or set dynamically with Java coding, but now it is possible to set dynamically the values of the data contexts, i.e. at launch time, with zero line of code.
Reserved variables for data context	A set of reserved variables. These variables define which data selection should be mapped with a data context variable. The possible values are: \$(dataspace) , \$(dataset) , \$(table) and \$(record) . For example, if the value \$(table) is used for a data context variable, this means that this variable will be mapped with the current adaptation table reference, at launch time.

67.2 Workflow launcher in data section

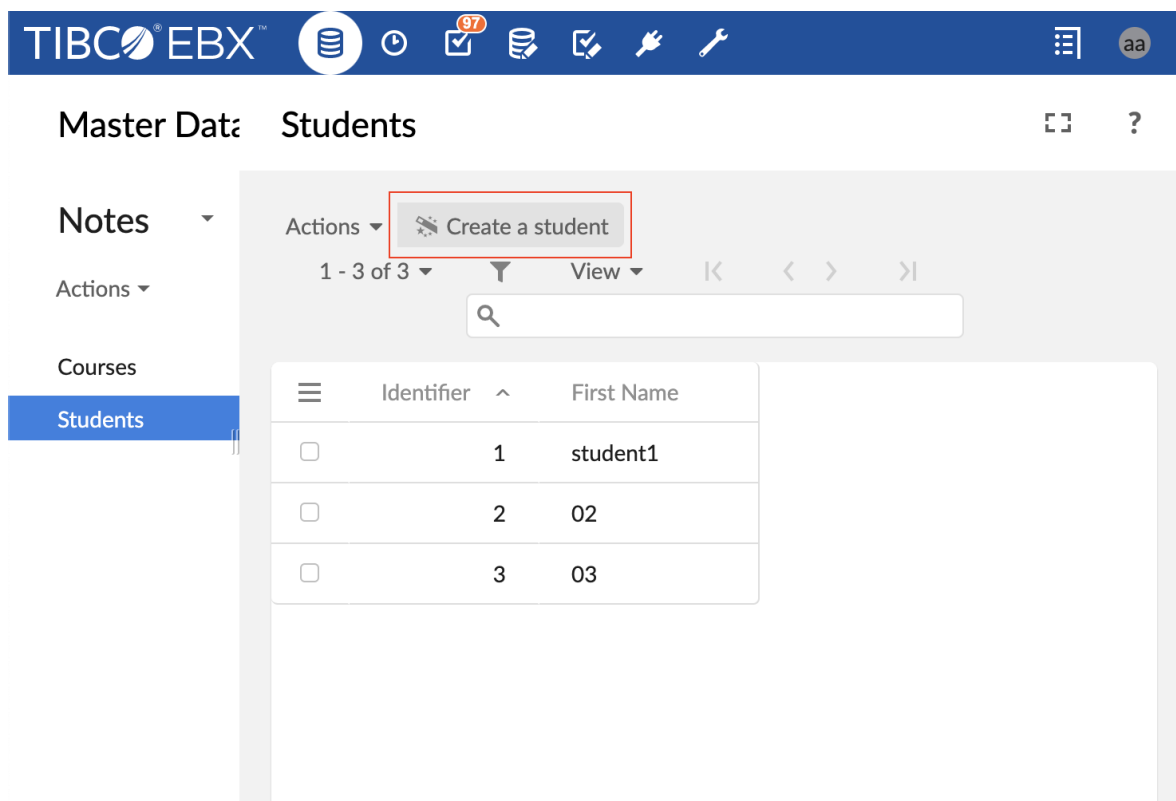
As previously stated, workflow launchers are now available directly on the toolbar of tables, records and hierarchies. There are a number of ways to display a workflow launcher on the toolbar of a table

in data section. The display depends on the type of the toolbar (default toolbar or custom toolbar in the DMA) and whether or not [smart filtering](#) [p 82] is applied. In the case of default toolbar, the action menu displays all the workflow launchers in a separate submenu called **Workflows** (see the screenshot below).



In the case of a custom toolbar, it is possible to define an action button (see the screenshot below) or action menu item for a particular workflow launcher. And finally, if the smart filtering policy is

activated, then all the workflow launchers that are displayed using an action button or action menu item will not appear in the Action menu.



In order to display a workflow launcher on a toolbar, first, a **launcher** must be **created** and **configured** (see [creating and setting workflow launcher](#) [p 442] section), then, an **activation** should be **created** for this launcher (see [activating workflow launcher](#) [p 444] section). Note that, if the current user has no [rights to launch a workflow publication](#) [p 191] then the workflow launcher will not be available on the toolbar. Also, if the workflow publication is deleted or if there is any errors or warnings in the configuration or activation of the launcher, then the corresponding workflow launcher will not be available on the toolbar.

The title and tooltip of the button that will be displayed on the toolbar, are computed in the following order of priority: the custom documentation of the launcher activation is used, otherwise the documentation of the launcher is used. If description is left empty, then the following description is used "This user service will launch a data workflow."

Note

Particular attention should be paid for the workflow launchers which are **available on the record form**. Only the **launchers which requires record selection** are displayed on the toolbar of a record form. A workflow launcher which requires record selection is the one for which one of the data context variable is mapped with the reserved variable **#{record}** in the configuration of its launcher.

67.3 Creating and setting a launcher

A launcher is the entity which is used by the service Workflow Launcher to identify the workflow publication to launch and how to initialize its Data context. In order to create a launcher for a particular workflow publication, first navigate to the administration area, then the workflow management

section, select the workflow launchers dataset, select the launchers table, and then add a new record. The second step, is to setup the following fields of the record:

Field name	Description
Nom du lanceur	A unique identifier. It is used to select a launcher in the activation phase [p 444].
Nom de publication du workflow	Le workflow qui sera lancé lors de l'exécution du lanceur. Un workflow doit d'abord être publié pour être disponible dans la liste.
Variables du contexte de données	This field contains the list of input variables which were defined in the data context configuration [p 187] phase of a workflow model. Each line is composed of a label, a value, and a toggle button to switch between default and overwritten value (see the screenshot below). The label is the name of the variable set by the user in the data context of the workflow model. By default, the value of each variable is the default one set in the data context. If the default value field is left empty in the data context, then the value of the variable is set to undefined. If the toggle button is set on overwritten value mode, then a wizard is made available on the right of the input field which allows to select a reserved variable. When overwriting the value of a data context variable, two options are possibles: the override value may be a constant or a reserved variable. If a constant is used, then the value of the data context variable will not depend on the entity selection at launch time. However, if a reserved variable is used, such as \$(dataspace), the value of the corresponding variable will be mapped to the current entity selection, for example the data context variable which is assigned to \$(dataspace) will be initialized with the current dataspace at launch time. Note If there are no input variables in the data context of the workflow model, then this field should be hidden. If the workflow publication name field changes then this input is updated automatically and displays a new list of data context.
Documentation	Une documentation est composée d'un label et d'une description. Cette documentation est utilisée pour définir le titre du bouton qui permet de lancer un workflow à partir d'une table, un formulaire d'un enregistrement ou une hiérarchie. Quand à la description, elle s'affiche au survol de ce bouton. La valeur par défaut du label est héritée du nom de la publication. Alors que la valeur par défaut de la description est vide. Ce champs (label et description) peut être surchargé ici ou dans l'activation. Il faut noter que la valeur définie dans l'activation est prioritaire, i.e. si les deux champs sont surchargés alors c'est le contenu du champs de l'activation qui est utilisé.

At the bottom of a launcher record, a table of all activations of the current launcher is displayed (see the section [Launcher activations](#) in the screenshot below).

The screenshot displays the TIBCO EBX interface for configuring a workflow launcher named 'createRecord'. The left-hand navigation pane shows the 'Workflows launcher' section with 'Launchers' and 'Activations' sub-sections. The 'createRecord' launcher is selected. The main configuration area includes the following details:

- Launcher Name:** createRecord
- Workflow Publication Name:** testCP-13244
- Documentation:** English (United States) testCP-13244
- Data Context Variables:**

Variable	Value	Wizard	Toggle Button
dataspace	<code>\${dataspace}</code>		
dataset	<code>\${dataset}</code>		
table	<code>\${xpath(table)}</code>		

Below the configuration fields is the 'Launcher activations' section, which contains a table with one activation record:

Publication	Action	Table	Details
Publication: test	Create a record	table	[see details]

At the bottom of the configuration area are four buttons: 'Save', 'Save and close', 'Revert', and 'Close'.

67.4 Activating workflow launcher

An activation is the entity which controls the availability of a workflow launcher on a table or record form. In order to display a workflow launcher in a toolbar of a particular table, first [create a launcher](#) [p 442], then navigate to the administration area, select the workflow management section, select the

workflow launchers dataset, select the activations table, and then add a new record. The second step, is to setup the following fields of the record:

Field name	Description
Lanceur	A unique identifier of a launcher which is associated to a publication name of the target workflow publication [p 443]. This field allows to select the launcher which will be displayed in the data section.
Model de données	Référence de schéma d'un modèle de données publié. Seuls les schémas publiés et utilisés dans les jeux de données sont disponibles.
Tables	The identifier of a table which will display the workflow launcher. It is possible to select one particular table or "All tables". This selector displays all tables which are contained in the field Data Model [p 445].
Profils	La liste des profils qui peuvent executer lancer le workflow sélectionné.
Confirmation	This field allows displaying or not a dialog box to confirm launching of a data workflow (see screenshot about confirmation message [p 447]). By default this feature is deactivated, in order to display the dialog box, the value "Enabled" should be checked (see screenshot below).
Documentation	A documentation is composed of a label and a description. The default value of the label and description are inherited from the documentation field of the launcher. This field is used to display the title of the button of the user service in the toolbar. The description is displayed when the user hovers over that button. The documentation of the activation has a higher priority over the one of the launcher. Note that, if the description of the documentation is left empty then the following one is displayed "This user service will launch a data workflow. In the case of the custom toolbar, the value of this field is also used for action button and action menu item if the documentation field of the action button or action menu item is left empty.

TIBCO EBX

Administration Add new activation

Workflows laun

Actions ▾

Launchers

▾ Activations

New record

Launcher * createRecord

Data Model * Publication: test

Tables * table

Profiles * 1. [all profiles]

Confirmation ☒ Enable ☐ Disable

Documentation * English (United States)
Create a record
▸ French (France)
▸ lang
▸ mxm

Save Save and close Close

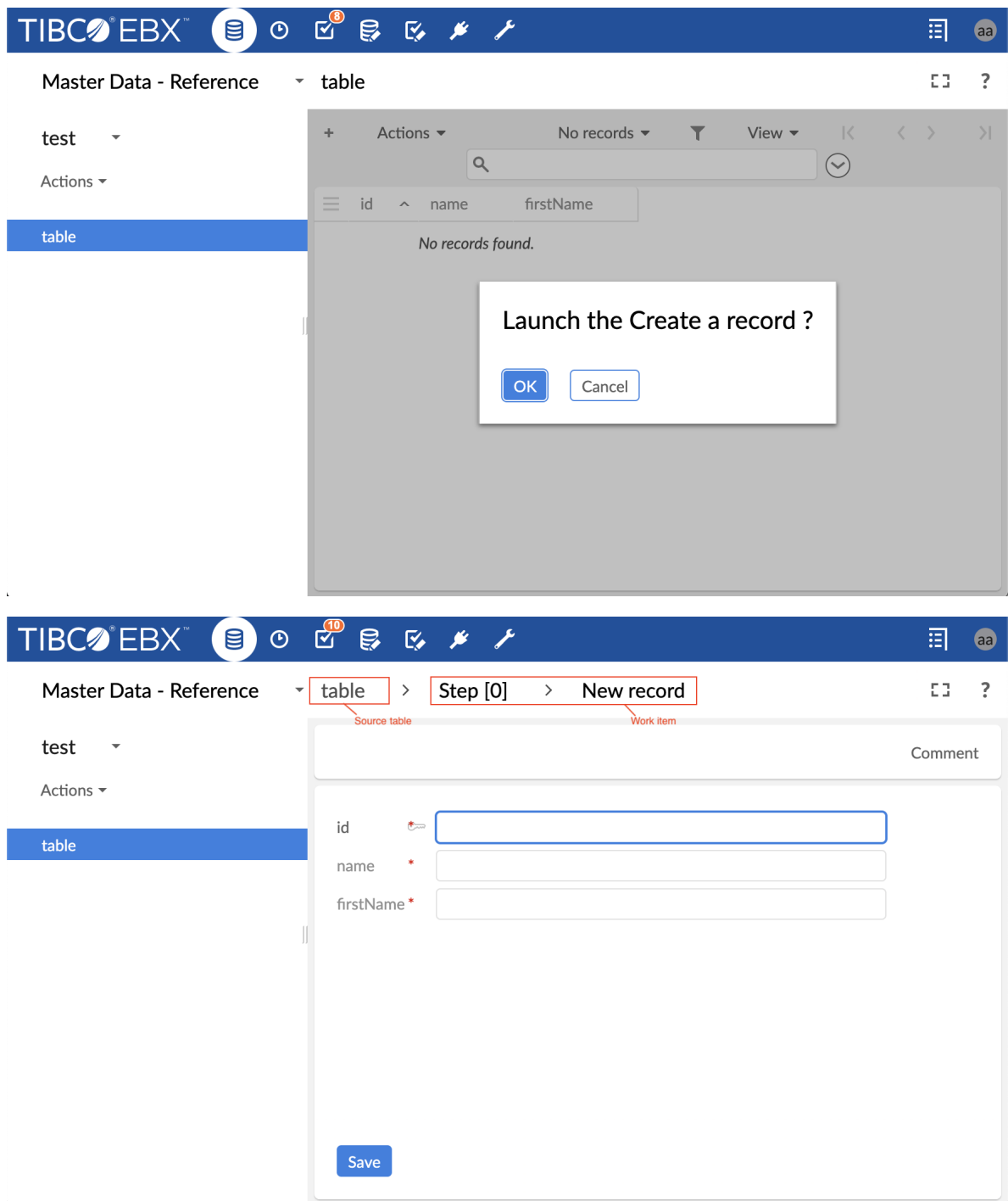
67.5 Launching a workflow

Once a launcher [created](#) [p 442] and [activated](#) [p 444] the corresponding workflow launcher becomes available on the toolbar's action menu. To launch the workflow click on the button of the workflow launchers (see [screenshot about default display of a workflow launcher](#) [p 441]).

If the option [Confirmation](#) [p 445] is enabled, then a dialog box displays to ask the user the confirmation before launching the selected data workflow (see [screenshot about data workflow launch confirmation](#) [p 447]). Otherwise, the dialog box does not display and different outcomes are possible:

- If the option [automatically open the first step](#) [p 187] is activated and the user has the [rights to execute the first work item](#) [p 173] of the workflow, then the workflow launches and the first work item displays directly in the workspace of the Data section. While displaying the work item in the workspace, the origin data selection still displayed on the breadcrumb which allows to the user to maintain an overall contextual awareness (see [screenshot about displaying a work item on the data section](#) [p 447]).
- If the option [automatically open the first step](#) [p 187] is deactivated and the user has the [rights to launch the workflow](#) [p 191], then the workflow is launched and added to the [data workflow inbox](#) [p 201]. An information message informs the user that the workflow has been launched successfully and that it is necessary to go to the [inbox of the data workflow section](#) [p 201] to display and execute the first work item.
- If the workflow requires record selection (see [note in the section workflow launcher in data area](#) [p 442]), the workflow is launched and displayed if and only if a record is selected otherwise an error message is displayed informing the user that a record should be selected. A record can be selected manually from a table view or automatically when displaying a record form.

- If more than one record are selected, then a warning message notifies that only one record should be selected.



67.6 Adding workflow launcher on a custom toolbar

The workflow launchers can be made available not only on default toolbar but also on [custom toolbar](#) [p 78]. A workflow launcher can be added on a custom toolbar as an [action button](#) [p 80] or as an [action menu item](#) [p 83] in a [custom menu group](#) [p 82]. Adding an action button or an action menu item to launch a workflow on a custom toolbar follows a similar procedure as for common user services:

first, set the **target** field to **current context** otherwise the user service **workflow launcher** will not be available, then select the user service **workflow launcher**. When the service workflow launcher is selected, a new field **Launcher**, exclusive for workflow launcher user service, appears below the service input field (see [screenshot about adding a workflow launcher on custom toolbar](#) [p 448]).

The field **Launcher** displays the list of launchers that have already been created in the table "Activation/workflow management/workflow launcher/Launchers" (see [section about creating and configuring launchers](#) [p 442]). Therefore, before adding a workflow launcher in a custom toolbar the launcher should be created and configured for this workflow launcher.

As for default toolbar, in order to show the workflow launcher on a custom toolbar its launcher should be activated in "Activation/workflow management/workflow launcher/activations". A quick link to access the activations table of the launchers is displayed on the right of the field **Launcher** (see [screenshot about adding a workflow launcher on custom toolbar](#) [p 448])

The field Label and description displays the label and description inherited from the launcher (see [screenshot about adding a workflow launcher on custom toolbar](#) [p 448]), then if the launcher field changes, the label and description should update automatically. This field can be overridden to customize the label and description of the action button on the toolbar. The label and description that will be displayed on the toolbar, are computed in the following order of priority: the custom value of the field Label and description of the action button is used, if this field is left empty or contains the default label of the launcher, then the label and description of the launcher activation are displayed. If this last one is left empty, then the following description is used "This user service will launch a data workflow."

Note

The label and description which are displayed on the toolbar in the data section and the label of the action button in the toolbar tree in the DMA could be different. This is the case when the field "Label and description" of the "action button" is left empty (neither default nor custom label is defined). The label of the "action button" in the toolbar tree in the DMA is inherited from the label of the launcher, however, the label and description on the toolbar in the data section are set to the label and description of the activation of the launcher.

The screenshot shows the TIBCO EBX interface with the 'New record' toolbar configuration window open. The 'Parent element' is 'toolbar For Table'. The 'Toolbar item type' is 'Action button'. The 'Service' is 'Workflow Launcher' and the 'Launcher' is 'createRecord'. The 'Label and description' field is highlighted with a red box, and a tooltip is displayed explaining the priority of labels. The 'Layout' is 'Text with icon on the left' and the 'Icon' type is 'Standard'. The 'Language' is set to 'English (United States)'.

67.7 Access a launcher after workflow model modification

After [creating a launcher](#) [p 442] which points to a [workflow publication](#) [p 443], the workflow model of this publication may be changed and republished. For instance, the data context of the workflow model may be changed: one or more variable data context can be added; one or more variable data context can be removed; the name of a variable can be changed. In this particular case, the launchers which points to this workflow publication should be reviewed and validated (saved) by the user because the data context used by these launchers is no more valid and it should be updated to match the one of the up-to-date workflow publication.

After republishing a workflow model, the user is notified if any of the workflow launcher points to the current workflow publication and if it should be reviewed. For that purpose, after publishing a workflow model, a preview button is displays allowing a quick access to the launchers in question . Note that if the user has no rights to access the workflow launchers list, then the names of those launchers are displayed .

When the user access to one of these launchers, via the preview button in the workflow model section or via the Administration section, the persisted data context is replaced with the up-to-date data context of the workflow publication, however the new data context is not yet persisted for the current launcher. In order to update the data context of the launcher, the user should first review the values and then save to accept the new changes (see [screenshot about accessing a launcher after modification of the data context of the corresponding workflow publication](#) [p 449]).

The screenshot shows the TIBCO EBX Administration interface. The left sidebar contains the following menu items: Administration, Workflows launchers (expanded), Actions, Launchers (expanded), WFY1 (selected), and Activations. The main content area is titled 'WFY1' and contains the following sections:

- Launcher name:** WFY1
- Workflow publication name:** WFY1
- Documentation:** English (United States) WFY1
- Data context variables:** A table with the following data:

Variable	Value
dataspace	[not defined]
record	[not defined]
dataset	[not defined]
table	[not defined]
- Launcher activations:** A table with the following data:

Activation
No records found.

At the bottom of the interface, there are four buttons: Save, Save and close, Revert, and Close.

CHAPITRE 68

Users and roles directory

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Concepts](#)
3. [Default directory](#)
4. [Custom directory](#)

68.1 Overview

TIBCO EBX® uses a directory for user authentication and user role definition.

A default directory is provided and integrated into the EBX® repository; the 'Directory' administration section allows defining which users can connect and what their roles are.

It is also possible to integrate another type of enterprise directory.

Voir aussi

[*Configuring the user and roles directory*](#) [p 374]

[*Custom directory*](#) [p 454]

68.2 Concepts

In EBX®, a user can be a member of several roles, and a role can be shared by several users. Moreover, a role can be included into another role. The generic term *profile* is used to describe either a user or a role.

In addition to the directory-defined roles, EBX® provides the following *built-in roles*:

Role	Definition
Profile.ADMINISTRATOR	Built-in Administrator role. Allows performing general administrative tasks.
Profile.READ_ONLY	Built-in read-only role. A user associated with the read-only role can only view the EBX® repository, and has no right to perform modifications in the repository.
Profile.OWNER	Dynamic built-in owner role. This role is checked dynamically depending on the current element. It is only activated if the user belongs to the profile defined as owner of the current element.
Profile.EVERYONE	All users belong to this role.

Information related to profiles is primarily defined in the directory.

Attention

Associations between users and the built-in roles *OWNER* and *EVERYONE* are managed automatically by EBX®, and thus must not be modified through the directory.

User permissions are managed separately from the directory. See [Permissions](#) [p 293].

Voir aussi

[profil](#) [p 25]

[rôle](#) [p 26]

[utilisateur](#) [p 25]

[administrateur](#) [p 26]

[annuaire des utilisateurs et des rôles](#) [p 26]

Policy

These properties configure the policies of the user and roles directory, for example, whether or not users can edit their own profiles.

Users

This table lists all the users defined in the internal directory. New users can be added from there.

Roles

This table lists all the users defined in the internal directory. New roles can be created in this table.

68.3 Default directory

Directory content

The default directory is represented by the dataset 'Directory', in the 'Administration' area.

This dataset contains tables for users and roles, as well as users' roles table, roles' inclusions table and salutations table.

Note

If a role inclusion cycle is detected, the role inclusion is ignored at the permission resolution. Refresh and check the directory validation report for cycle detection.

Note

Users' roles, roles' inclusions and salutations tables are [hidden by default](#) [p 604].

Depending on the policies defined, users can modify information related to their own accounts, regardless of the permissions defined on the directory dataset.

Note

It is not possible to delete or duplicate the default directory.

Password recovery procedure

In the default directory, passwords are encrypted (by default with a SHA256-like algorithm), and stored in this state. Consequently, it is impossible to retrieve lost passwords. A new password must be generated and sent to the user.

There are two options for this procedure:

1. A notification email is sent to the administrator, the administrator manually changes the password and sends the new password to the user.
2. A procedure automatically generates a new password and sends it to the user.

By default, the first option is used. To activate the second option, specify the property `ebx.password.remind.auto=true` in the [TIBCO EBX® main configuration file](#) [p 371].

Note

For security reasons, the password recovery procedure is not available for administrator profiles. If required, use the administrator recovery procedure instead.

Administrator recovery procedure

If all the 'login/password' credentials of the administrators are lost, a special procedure must be followed. A specific directory class redefines an administrator user with login 'admin' and password 'admin'.

To activate this procedure:

- Specify the following property in the [TIBCO EBX® main configuration file](#) [p 371]:
`ebx.directory.factory=
com.orchestranetworks.service.directory.DirectoryDefaultRecoverFactory`

- Start EBX® and wait until the procedure completes.
- Reset the 'ebx.directory.factory' property.
- Restart EBX® and connect using the 'admin' account.

Note

While the 'ebx.directory.factory' property is set for the recovery procedure, authentication of users will be denied.

68.4 Custom directory

As an alternative to the default directory, it is possible to integrate a specific company directory. For example, an LDAP instance, a relational database or a specific directory model instantiated into EBX®.

Voir aussi *DirectoryFactory^{API}*

CHAPITRE 69

Data model administration

Ce chapitre contient les sections suivantes :

1. [Administrating publications and versions](#)
2. [Migration of previous data models in the repository](#)
3. [Schema evolutions](#)

69.1 Administrating publications and versions

Technical data related to data model publications and versions can be accessed in the *Administration* section by an administrator.

Data Modeling contains the following two tables:

- *Publications*. Stores the publications available in the repository.
- *Versions*. Stores the versions of the data models available in the repository.

These tables are 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 datasets will become unavailable. So this technical data must be deleted with caution.

It is possible to spread this technical data to other TIBCO EBX® repositories exporting an archive from an EBX® repository and importing it to another one. It may be useful for propagating the evolutions of data models to other repositories.

69.2 Migration of previous data models in the repository

In versions before 5.2.0, published data models not depending on a module were generated in the file system directory `${ebx.repository.directory}/schemas/`, with the name of the data model (*product.xsd* for example if the data model is named *Product*). Since the 5.2.0 version, this kind of data model is now fully managed within EBX® through *Publications*. That is, republishing an existing data model migrates it as a *Publication* and redirects linked datasets to the new embedded data model. The previous XML Schema Document located in `${ebx.repository.directory}/schemas/` is renamed and suffixed with *toDelete*, meaning that the document is no longer used and can be safely deleted.

69.3 Schema evolutions

It is crucial to evaluate the impact of data model changes on the administration side. The following points are to be considered:

Impacts on data persistence

Administration tasks can be related to the database cleanup after a modification of the models. The following link describes how the evolutions of data models are managed at the persistence level: [Purging master tables in the database](#) [p 459].

Impacts on side features

Some components rely heavily on the data models and can be impacted by their evolutions. Some examples are: the user interface, the WSDL documents, existing archives, etc.

The 'Administration' section offers the possibility to manage some of these components (such as the views), whereas other components fall out of the administrator's scope, such as archives, WSDL files, etc.

CHAPITRE 70

Database mapping administration

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Renaming columns in the database](#)
3. [Purging columns in the database](#)
4. [Renaming master tables in the database](#)
5. [Renaming auxiliary tables in the database](#)
6. [Purging master tables in the database](#)

70.1 Overview

Information and services relative to database mapping can be found in the *Administration* area.

Voir aussi

[Mapped modes](#) [p 267]

DatabaseMapping^{API}

70.2 Renaming columns in the database

This feature is available on the 'Columns' table records, under the 'Actions' menu. It allows renaming a column in the database.

The administrator can specify the name of each column of the data model in the database for mapped modes.

Once the service is selected on a record, a summary screen displays information regarding the selected column and the administrator is prompted to enter a new name for the column in the database.

Note

It is required that the new identifier begins with a letter.

Besides, the new name must be a valid column identifier, which depends on the naming rules of the underlying RDBMS.

Voir aussi *DatabaseMapping*^{API}

70.3 Purging columns in the database

This feature is available on the 'Columns' table records, under the 'Actions' menu. It allows purging columns in mapped structures.

A column can be purged if it has been disabled for mapped modes.

A column is disabled for mapped modes when:

- the corresponding field has been removed from the data model, or
- the corresponding field has been changed in the data model, in a way that is not compatible (for example: its data type has been modified), or
- the defined mapped modes have been disabled locally on the corresponding fields, using the elements `osd:history` and `osd:replication`.

Voir aussi

[*Disabling history on a specific field or group*](#) [p 270]

[*Disabling replication on a specific field or group*](#) [p 279]

Note that this behavior will change for aggregated lists:

- when deactivating a complex aggregated list, its inner fields will still be in the `LIVING` state, whereas the list node is disabled. As lists are considered as auxiliary tables in the mapping system, this information can be checked in the 'Tables' table,
- on the other hand, when the deactivation is just for inner nodes of the list, then the list will remain `LIVING`, while its children will be `DISABLED IN MODEL`.

A column can be purged only if its own state is `DISABLED IN MODEL`, or if it is an inner field of a `DISABLED IN MODEL` list.

70.4 Renaming master tables in the database

This feature allows renaming master tables for history tables in the database. It is not available for replicated tables since their names are specified in the data model.

Both features are available on the 'Tables' table records, under the 'Actions' menu.

Master tables are database tables used for persisting the tables of the data model.

The administrator can specify in the database the name of each master table corresponding to a table of the data model.

Once the service is selected on a record, a summary screen displays information regarding the selected master table and the administrator is prompted to enter a new name for the master table in the database.

Note

It is required that the new identifier begins with a letter and with the repository prefix.

For history tables, it is also required for the repository prefix to be followed by the history tables prefix.

Besides, the new name must be a valid table identifier, which depends on the naming rules of the underlying RDBMS.

70.5 Renaming auxiliary tables in the database

This feature allows renaming history auxiliary tables in the database. This feature is not available for replicated tables since their names are specified in the data model.

This feature is available on the 'Tables' table records, under the 'Actions' menu.

Auxiliary tables are database tables used for persisting aggregated lists.

The administrator can specify in the database the name of each auxiliary table corresponding to an aggregated list of the data model.

Once the service is selected on a record, a summary screen displays information regarding the selected auxiliary table and the administrator is prompted to enter a new name for the auxiliary table in the database.

Note

It is required for the new identifier to begin with a letter.

It is required for the new identifier to begin with the repository prefix.

It is also required for the repository prefix to be followed by the history tables prefix.

Besides, the new name must be a valid table identifier, which depends on the naming rules of the underlying RDBMS.

70.6 Purging master tables in the database

This feature allows purging history in the database if it is no longer used.

It is available on the 'Tables' table records, under the 'Actions' menu, and is only available for master tables. This feature only applies to master tables. When a master table is purged, all its auxiliary tables are purged as well.

A mapped table can be purged in the database only if it has been disabled for the corresponding mapped mode.

To disable the mapped mode for a table, follow the procedure hereafter.

- Deactivate historization of the table in the data model, or
- Remove the table from the data model

CHAPITRE 71

Workflow management

Ce chapitre contient les sections suivantes :

1. [Workflows](#)
2. [Interactions](#)
3. [Workflow history](#)

71.1 Workflows

To define general parameters for the execution of data workflows, the management of workflow publications, or to oversee data workflows in progress, navigate to the 'Administration' area. Click on the down arrow in the navigation pane and select *Workflow management > Workflows*.

Note

In cases where unexpected inconsistencies arise in 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 'Actions' menu in the navigation pane under *Administration > Workflow Management > Workflows*.

Execution of workflows

Various tables can be used to manage the data workflows that are currently in progress. These tables are accessible in *Workflow management > Workflows* in the navigation pane.

Voir aussi [Administration de workflows de données](#) [p 213]

Workflows table

The 'Workflows' table contains instances of all data workflows in the repository, including those invoked as sub-workflows. A data workflow is a particular execution instance of a workflow model publication. This table provides access to the data context variables for all data workflows. It can be used to access the status of advancement of the data workflow in terms of current variable values, and in case of a data workflow suspension, to modify the variable values.

From the 'Actions' menu of the 'Workflows' table, it is possible to clear the completed data workflows that are older than a given date, by selecting the 'Clean from a date' service. This service automatically ignores the active data workflows.

Tokens table

The 'Tokens' table allows managing the progress of data workflows. Each token marks the current step being executed in a running data workflow, as well as the current state of the data workflow.

Voir aussi [token](#) [p 33]

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. 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](#) [p 33]

Waiting workflows table

The 'Waiting workflows' table contains all the workflows waiting for an event. If needed, a service is available to clean this table: this service deletes all lines associated with a deleted workflow.

Voir aussi [tâche d'attente](#) [p 32]

Comment table

The 'Comments' table contains the user's comments for main workflows and their sub-workflows.

Workflow publications

The 'Workflow publications' table is a technical table that contains all the workflow model publications of 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

Email configuration

In order for email notifications to be sent during the data workflow execution, the following settings must be configured under 'Email configuration':

- The Définition de l'URL field is used to build links and value mail variables in the workflow.
- The 'From email' field must be completed with the email address that will be used to send email notifications.

Interface customization

Modeling default values

The default value for some properties can be customized in this section.

The administrator has the possibility to define the default values to be used when a new workflow model or workflow step is created in the 'Workflow Modeling' section.

Work items views

Specific columns are available in the inbox and in the monitoring work items tables, in the 'Data workflows' section.

10 specific columns are available. For each specific column, a customized label can be defined.

Priorities configuration

The property 'Default priority' defines how data workflows and their work items across the repository display if they have no 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 'Normal' priority.

The 'priorities' table defines all priority levels available to data workflows in the repository. As many integer priority levels as needed can be added, along with their labels, which will appear when users hover over the priority icon in the work item tables. The icons that correspond to each priority level can also be selected, either from the set provided by TIBCO EBX®, or by specifying a URL to an icon image file.

Temporal tasks

Under 'Temporal tasks', the polling interval for time-dependent tasks in the workflow can be set, such as deadlines and reminders. If no interval value is set, the 'in progress' steps are checked every hour.

Workflow inbox counter configuration

The workflow inbox counter is refreshed asynchronously, even if the end-user does not launch any action. To adjust it, two parameters need to be set:

Durée d'expiration du cache (secondes)	Durée d'expiration (en secondes) avant une nouvelle mise à jour du cache de la boîte de réception. À noter que ce paramètre peut avoir des conséquences sur la charge CPU et les performances pour de gros volumes de données car le temps de calcul peut être assez coûteux. Si aucune valeur n'est renseignée, la valeur par défaut est 600.
Périodicité de rafraîchissement de l'interface utilisateur (secondes)	Durée de rafraîchissement (en secondes) entre deux mises à jour du compteur de la boîte de réception au niveau de l'interface graphique. À noter que ce rafraîchissement concerne tous les compteurs de boîtes de réception contenus dans l'interface ; à savoir l'en-tête et le compteur de la boîte de réception de la perspective avancée. Si aucune valeur n'est renseignée, la valeur par défaut est 5. Si la valeur renseignée est 0 (ou négative), le rafraîchissement est désactivé. Aussi, la modification de cette valeur ne sera effective qu'après reconnexion de l'utilisateur.

Also, please note that some actions can force the inbox counter to refresh:

- access on **Data workflows**
- access on any subdivision of the **Data workflows section**
- accept or reject a work item

- launch a workflow

These parameters are accessible in *Workflow management > Workflows > Configuration > Temporal tasks* in the navigation pane.

71.2 Interactions

To manage workflow interactions, navigate to the Administration area. Click the down arrow in the navigation pane and select the entry *Workflow management > 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 EBX® session. When a work item is executed, the user performs the assigned actions based upon its interaction, independently 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 user service.

71.3 Workflow history

To view the history data workflow execution, browse the 'Administration' area. Click on the down arrow in the navigation pane and select *Workflow management > Workflow history*.

The 'Workflows' table contains all actions that have been performed during the execution of workflows.

This data can be viewed graphically or textually. It 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 an error, a technical log is available.

Clean history

From the 'Actions' menu of the 'Workflows' table, the history of completed data workflows older than a given date can be cleared by selecting the 'Clear from a date' service.

Only the history of workflows that have been previously cleaned (e.g. their execution data deleted) is cleared. This service automatically ignores the history associated with existing workflows. It is necessary to clear data workflows before clearing the associated history, by using the dedicated service 'Clear from a date' from the 'Workflows' table. Also, a scheduled 'Clear from a date' can be used with the built-in scheduled task *SchedulerPurgeWorkflowMainHistory*.

Please note that only main processes are cleaned. In order to remove sub-processes and all related data, it will be necessary to run a 'standard EBX® purge'.

Voir aussi [How to clean workflow history](#) [p 419]

Note

An API is available to fetch the history of a workflow. Direct access to the underlying workflow history SQL tables is not supported. See **WorkflowEngine.getProcessInstanceHistory** WorkflowEngine.getProcessInstanceHistory^{API}.

CHAPITRE 72

Task scheduler

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Configuration from EBX®](#)
3. [Cron expression](#)
4. [Task definition](#)
5. [Task configuration](#)

72.1 Overview

TIBCO EBX® offers the ability to schedule programmatic tasks.

Note

In order to avoid conflicts and deadlocks, tasks are scheduled in a single queue.

72.2 Configuration from EBX®

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the 'Administration' area.

- **Schedules:** defines scheduling using "cron expressions".
- **Tasks:** configures tasks, including parametrizing task instances and user profiles for their execution.
- **Scheduled tasks:** current schedule, including task scheduling activation/deactivation.
- **Execution reports:** 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.

72.3 Cron expression

(An extract of the [Quartz Scheduler](#) documentation)

The task scheduler uses "cron expressions", which can 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 composed of 6 or 7 fields separated by a 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	empty, 1970-2099	, - * /

A cron expression can be as simple as this: "0 * * * * ?",

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 composed of 6 or 7 fields separated by a 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 Minutes field means "every minute".
- ? ("no specific value") - useful when you need to specify something in one of the two fields in which the character is allowed, but not the other. For example, if I want my trigger to fire on a particular day of the month (say, the 10th), but don't care what day of the week that happens to be, I would put "10" in the day-of-month field, and "?" in the day-of-week field. See the examples below for clarification.
- - - used to specify ranges. For example, "10-12" in the hour field means "the hours 10, 11 and 12".
- , - used to specify additional values. For example, "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".
- / - used to specify increments. For example, "0/15" in the seconds field means "the seconds 0, 15, 30, and 45". And "5/15" in the seconds field means "the seconds 5, 20, 35, and 50". You can also

specify '/' after the "**character - in this case**" is equivalent to having '0' before the '/'. '1/3' in the day-of-month field means "fire every 3 days starting on the first day of the month".

- **L** ("last") - has different meaning in each of the two fields in which it is allowed. For example, the value "L" in the day-of-month field means "the last day of the month" - day 31 for January, day 28 for February on non-leap years. If used in the day-of-week field by itself, it simply means "7" or "SAT". But if used in the day-of-week field after another value, it means "the last xxx day of the month" - for example "6L" means "the last friday of the month". When using the 'L' option, it is important not to specify lists, or ranges of values, as you'll get confusing results.
- **W** ("weekday") - used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify "15W" as the value for the day-of-month field, the meaning is: "the nearest weekday to the 15th of the month". So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However if you specify "1W" as the value for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not 'jump' over the boundary of a month's days. The 'W' character can only be specified when the day-of-month is a single day, not a range or list of days.

Note

The 'L' and 'W' characters can also be combined in the day-of-month field to yield 'LW', which translates to "last weekday of the month".

- **#** - used to specify "the nth" day-of-week day of the month. For example, the value of "6#3" in the day-of-week field means "the third Friday of the month" (day 6 = Friday and "#3" = the 3rd one in the month). Other examples: "2#1" = the first Monday of the month and "4#5" = the fifth Wednesday of the month. Note that if you specify "#5" and there is not 5 of the given day-of-week in the month, then no firing will occur that month.

Examples

Expression	Meaning
0 0 12 * * ?	Fire at 12pm (noon) every day.
0 15 10 ? * *	Fire at 10:15am every day.
0 15 10 * * ?	Fire at 10:15am every day.
0 15 10 * * ? *	Fire at 10:15am every day.
0 15 10 * * ? 2005	Fire at 10:15am every day during the year 2005.
0 * 14 * * ?	Fire every minute starting at 2pm and ending at 2:59pm, every day.
0 0/5 14 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day.
0 0/5 14,18 * * ?	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at 6pm and ending at 6:55pm, every day.
0 0-5 14 * * ?	Fire every minute starting at 2pm and ending at 2:05pm, every day.
0 10,44 14 ? 3 WED	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
0 15 10 ? * MON-FRI	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday.
0 15 10 15 * ?	Fire at 10:15am on the 15th day of every month.
0 15 10 L * ?	Fire at 10:15am on the last day of every month.
0 15 10 ? * 6L	Fire at 10:15am on the last Friday of every month.
0 15 10 ? * 6L 2002-2005	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005.
0 15 10 ? * 6#3	Fire at 10:15am on the third Friday of every month.
0 0 12 1/5 * ?	Fire at 12pm (noon) every 5 days every month, starting on the first day of the month.
0 11 11 11 11 ?	Fire every November 11th at 11:11am.

Note

Pay attention to the effects of '?' and '*' in the day-of-week and day-of-month fields.

Support for specifying both a day-of-week and a day-of-month value is not complete (you must currently use the '?' character in one of these fields).

Be careful when setting fire times between the hours of the morning when "daylight savings" changes occur in your locale (for US locales, this would typically be the hour before and after 2:00 AM - because the time shift can cause a skip or a repeat depending on whether the time moves back or jumps forward).

72.4 Task definition

EBX® scheduler comes with some predefined tasks.

Custom scheduled tasks can be added by the means of **scheduler** Package `com.orchestranetworks.schedulerAPI` Java API.

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the 'Administration' area.

72.5 Task configuration

A user must be associated with a task definition; this user will be used to generate the **session** `SessionAPI` 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 a 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.

CHAPITRE 73

Audit trail

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Update details and disk management](#)
3. [File organization](#)

73.1 Overview

Attention

XML audit trail is a feature that allows logging updates to XML files. This legacy feature, now deprecated, will be removed in a future version. As an alternative, please consider using the history feature, which registers table updates in the relational database; see [History](#) [p 269].

Any persistent updates performed in the TIBCO EBX® repository are logged to an audit trail XML file. Procedure executions are also logged, even if they do not perform any updates, as procedures are always considered to be transactions. The following information is logged:

- Transaction type, such as dataset creation, record modification, record deletion, specific procedure, etc.
- Dataspace or snapshot on which the transaction is executed.
- Transaction source. If the action was initiated by EBX®, this source is described by the user identity, HTTP session identifier and client IP address. If the action was initiated programmatically, only the user's identity is logged.
- Optional "trackingInfo" value regarding the session
- Transaction date and time (in milliseconds);
- Transaction UUID (conform to the Leach-Salz variant, version 1);
- Error information; if the transaction has failed.
- Details of the updates performed. If there are updates and if history detail is activated, see next section.

73.2 Update details and disk management

The audit trail is able to describe all updates made in the EBX® repository, at the finest level. Thus, the XML files can be quite large and the audit trail directory must be carefully supervised. The following should be taken into account:

1. If an archive import is executed in non-interactive mode (without a change set), the audit trail does not detail the updates; it only specifies the archive that has been imported. In this case, if it is important to keep a fine trace of the import-replace, the archive itself must be preserved.
2. If an archive import is executed in interactive mode (with a change set), or if a dataspace is merged to its parent, the resulting log size will nearly triple the unzipped size of the archive. Furthermore, for consistency concerns, each transaction is logged to a temporary file (in the audit trail directory) before being moved to the main file. Therefore, EBX® requires *at least six times the unzipped size of the largest archive that may be imported*.
3. In the context of a custom procedure that performs many updates not requiring auditing, it is possible for the developer to disable the detailed history using the method `ProcedureContext.setHistoryActivationAPI`.

Voir aussi [EBX® monitoring](#) [p 419]

73.3 File organization

All audit trail files are stored in the directory `${ebx.repository.directory}/History`.

"Closed" audit files

Each file is named as follows:

```
<yyyy-mm-dd>-part<nn>.xml
```

where `<yyyy-mm-dd>` is the file date and `<nn>` is the file index for the current day.

Writing to current audit files

When an audit file is being written, the XML structure implies working in an "open mode". The XML elements of the modifications are added to a text file named:

```
<yyyy-mm-dd>-part<nn>Content.txt
```

The standard XML format is still available in an XML file that references the text file. This file is named:

```
<yyyy-mm-dd>-part<nn>Ref.xml
```

These two files are then re-aggregated in a "closed" XML file when the repository has been cleanly shut down, or if EBX® 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 74

Other

Ce chapitre contient les sections suivantes :

1. [Lineage](#)
2. [Event broker](#)

74.1 Lineage

To administer lineage, three tables are accessible:

- **Authorized profiles:** Profiles must be added to this table to be used for data lineage WSDL generation.
- **History:** Lists the general data lineage WSDLs and their configuration.
- **JMS location:** Lists the JMS URL locations.

74.2 Event broker

Overview

TIBCO EBX® offers the ability to receive notifications and information related to specific events using the event broker feature. This feature consists in sending notifications related to EBX® core events to the subscriber according to their chosen topics.

Terminology

Event broker	Notification component for loosely-coupled event handling. Consists of dispatching fired events from EBX® core to concerned subscribers. The event broker is mainly used for monitoring and statistical purposes.
Topic	Corresponds to the EBX® event type that contains messages. The number of subscribers registered to a topic is unlimited.
Subscriber	Client implementation in the modules that receive the events related to the subscribed topic(s).

Topics

Dataspace and snapshot	Corresponds to operations in the dataspace and in the snapshot, such as: create, close, reopen, delete, archive export and archive import (only for dataspace merge).
Repository	Corresponds to operations in the repository, such as: start-up and purge.
User session	Corresponds to the operations related to user authentication, such as: login and logout.

Administration

The management console is located under 'Event broker' in the 'Administration' area. It contains three tables: 'Topics', 'Subscribers' and 'Subscriptions'.

All content is read-only, except for the following operations:

- Topics and subscribers can be manually activated or deactivated using dedicated services.
- Subscribers that are no longer registered to the broker can be deleted.

The event broker is based on a thread pool mechanism. The maximum number of threads can be defined in the properties file as follows:

```
# Defines the number of thread pool executors to
# guarantee the publication of asynchronous events.
# The default value is 2
ebx.eventBroker.threadPool.size=2
```

Distributed Data Delivery (D3)

CHAPITRE 75

Introduction to D3

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [D3 terminology](#)
3. [Known limitations](#)

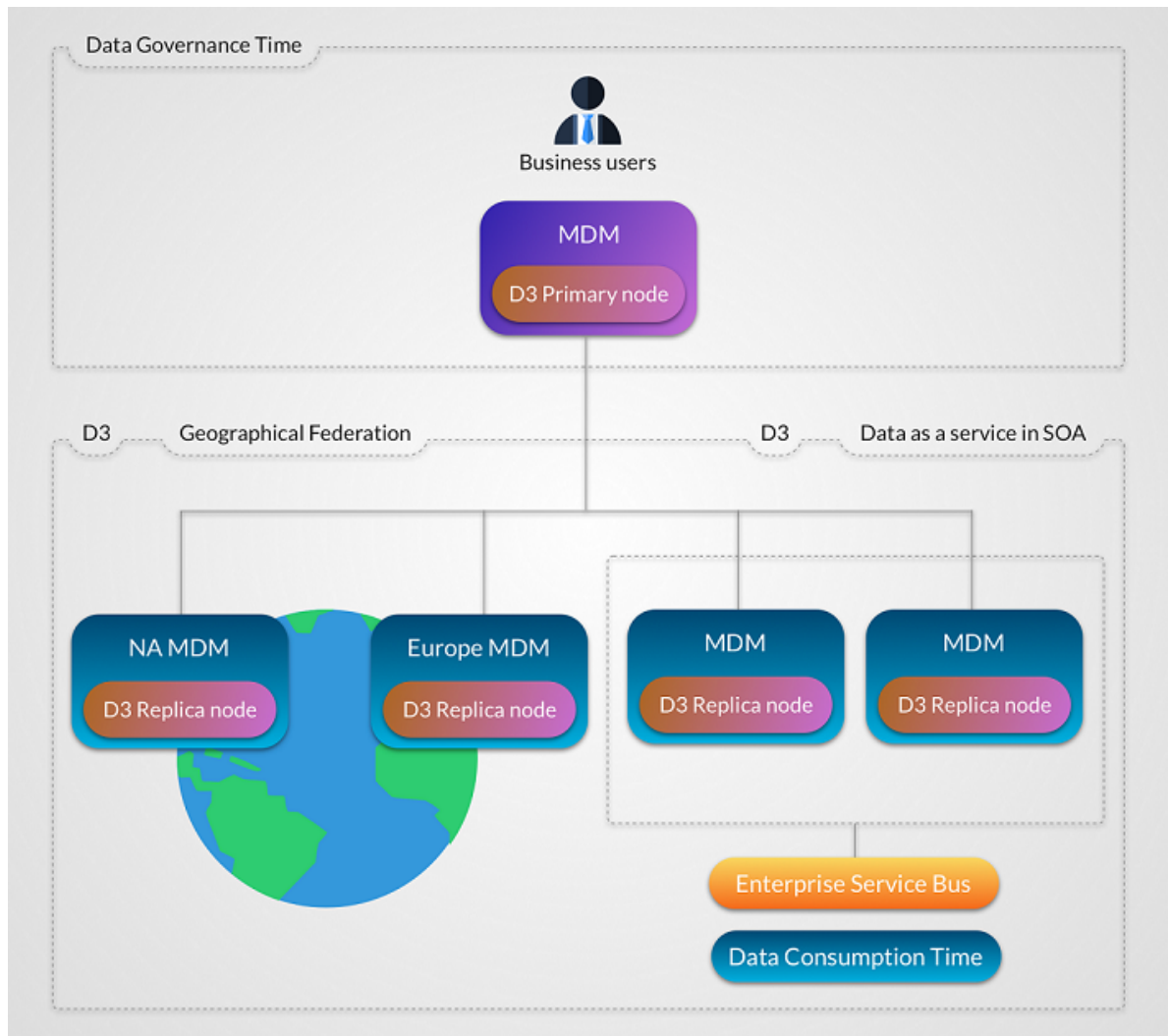
75.1 Overview

TIBCO EBX® offers the ability to send data from an EBX® instance to other instances. Using a broadcast action, it also provides an additional layer of security and control to the other features of EBX®. It is particularly suitable for situations where data governance requires the highest levels of data consistency, approvals and the ability to rollback.

D3 architecture

A typical D3 installation consists of one primary node and multiple replica nodes. In the primary node, a Data Steward declares which dataspace must be broadcast, as well as which user profile is allowed to broadcast them to the replica nodes. The Data Steward also defines delivery profiles, which are groups of one or more dataspace.

Each replica node must define from which delivery profile it receives broadcasts.



Involving third-party systems

The features of D3 also allow third-party systems to access the data managed in EBX® through data services. Essentially, when a system consumes the data of a delivery dataspace, the data is transparently redirected to the last broadcast snapshot. This ensures a more controlled and reliable view of the managed data.

Third-party systems can either access data directly through the primary node or through a replica node. Thus, a physical architecture consisting of a primary node and no replica nodes is possible.

Protocols

If JMS is activated, the conversation between a primary node and a replica node is based on SOAP over JMS, while archive transfer is based on JMS binary messages.

If JMS is not activated, conversation between a primary node and a replica node is based on SOAP over HTTP(S), while binary archive transfer is based on TCP sockets. If HTTPS is used, make sure that the target node connector is correctly configured by enabling SSL with a trusted certificate.

Voir aussi [*JMS for distributed data delivery \(D3\)*](#) [p 487]

75.2 D3 terminology

broadcast	Send a publication of an official snapshot of data from a primary node to replica nodes. The broadcast transparently and transactionally ensures that the data is transferred to the replica nodes.
delivery dataspace	A delivery dataspace is a dataspace that can be broadcast to authenticated and authorized users using a dedicated action. By default, when a data service accesses a delivery dataspace on any node, it is redirected to the last snapshot that was broadcast. See Data services [p 485].
delivery profile	A delivery profile is a logical name that groups one or more delivery dataspaces. Replica nodes subscribe to one or more delivery profiles.
cluster delivery mode	Synchronization with subscribed replica 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 replica nodes and their primary node delivery dataspace. Primary and replica nodes use the same last broadcast snapshots.
federation delivery mode	Synchronization is performed in a single phase, and with each registered replica 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, replica nodes can be at different last broadcast snapshots. The synchronization processes are thus independent of one another and replay of individual replica nodes are performed for certain broadcast failures.
Primary node	An instance of EBX® that can define one or more delivery dataspace, and to which replica nodes can subscribe. A primary node can also act as a regular EBX® server.
Replica node	An instance of EBX® attached to a primary node, in order to receive delivery dataspace broadcasts. Besides update restrictions on delivery dataspace, the replica node acts as a regular EBX® server.

Hub node

An instance of EBX® acting as both a primary node and a replica node. Primary delivery dataspace and replica node delivery dataspace **must** be disjoint.

75.3 Known limitations

General limitations

- Each replica node must have only one primary node.
- Embedded data models cannot be used in D3 dataspace. Therefore, it is not possible to create a dataset based on a publication in a D3 dataspace.
- The compatibility is not assured if at least one replica node product version is different from the primary node.

Broadcast and delivery dataspace limitations

- Access rights on dataspace are not broadcast, whereas access rights on datasets are.
- Dataspace information is not broadcast.
- If a dataspace and its parent are broadcast, their parent-child relationship will be lost in the replica nodes.
- Once a snapshot has been broadcast to a replica, subsequent broadcasts of *any* snapshot with the same name will result in restoring the originally broadcast version of that same name on the replica node. That is, if the original snapshot on the primary node is purged and a new one is created with the same name and subsequently broadcast, then the content of the replica will be restored to that of the previously broadcast snapshot, and not to the latest one of the same name.
- To guarantee dataspace consistency between D3 nodes, the data model (embedded or packaged in a module) on which the broadcast contents are based, must be the same between the primary node and its replica nodes.
- On a replica delivery dataspace, if several replica nodes are registered, and if replication is enabled in data models, it will be effective for all nodes. No setting is available to activate/deactivate replication according to D3 nodes.
- Replication on replica nodes does not take part in the distributed transaction: it is automatically triggered after commit.

Administration limitations

Technical dataspace cannot be broadcast, thus the EBX® default user directory cannot be synchronized using D3.

CHAPITRE 76

D3 broadcasts and delivery dataspace

Ce chapitre contient les sections suivantes :

1. [Broadcast](#)
2. [Replica node registration](#)
3. [Accessing delivery dataspace](#)

76.1 Broadcast

Scope and contents of a broadcast

A D3 broadcast occurs at the dataspace or snapshot level. For dataspace broadcasts, D3 first creates a snapshot to capture the current state, then broadcasts this newly created snapshot.

A broadcast performs one of the following procedures depending on the situation:

- An update of the differences computed between the new broadcast snapshot and the current 'commit' one on the replica node.
- A full synchronization containing all datasets, tables, records, and permissions. This is done on the first broadcast to a given replica node, if the previous replica node commit is not known to the primary node, or on demand using the user service in '[D3] Primary node configuration'.

Voir aussi [Services on primary nodes](#) [p 500]

Performing a broadcast

The broadcast can be performed:

- By the end-user, using the **Broadcast** action available in the dataspace or snapshot (this action is available only if the dataspace is registered as a delivery dataspace)
- Using custom Java code that uses `D3NodeAsMasterAPI`.

Conditions

In order to be able to broadcast, the following conditions must be fulfilled:

- The authenticated user profile has permission to broadcast.

- The dataspace or snapshot to be broadcast has no validation errors.
Note: Although it is not recommended, it is possible to force a broadcast of a delivery dataspace that contains validation errors. In order to do this, set the maximum severity threshold allowed in a delivery dataspace validation report under '[D3] Primary node configuration' in the 'Administration' area.
- The D3 primary node configuration has no validation errors on the following scope: the technical record of the concerned delivery dataspace and all its dependencies (dependent delivery mappings, delivery profiles and registered replica nodes).
- There is an associated delivery profile.
- If broadcasting a dataspace, the dataspace is not locked.
- If broadcasting a snapshot, the snapshot belongs to a dataspace declared as delivery dataspace and is not already the current broadcast snapshot (though a rollback to a previously broadcast snapshot is possible).
- The dataspace or snapshot contains differences compared to the last broadcast snapshot.

Persistence

When a primary node shuts down, all waiting or in progress broadcast requests abort, then they will be persisted on a temporary file. On startup, all aborted broadcasts are restarted.

Voir aussi [Temporary files](#) [p 502]

Destination

On the target replica or hub node side:

- The ebx-d3-reference dataspace identifier is the common parent of all the delivery dataspace.
- The delivery dataspace has the same identifier in primary, replica or hub nodes.
- If the delivery dataspace is missing, it will be created on the first or on the full synchronization broadcast.
- If the delivery dataspace already exists on the first broadcast or full synchronization, it will be overridden.
- If an existing dataspace with the same identifier as the delivery one is detected outside of the ebx-d3-reference, an error will be raised.

Voir aussi [Known limitations](#) [p 482]

Note

Broadcasts are performed asynchronously. Therefore, no information is displayed in the user interface about the success or failure of a broadcast. Nevertheless, it is possible to monitor the broadcast operations inside '[D3] Primary node configuration'. See [Supervision](#) [p 501].

76.2 Replica node registration

Scope and contents

An initialization occurs at the replica node level according to the delivery profiles registered in the TIBCO EBX® main configuration file of the replica node. When the primary node receives that initialization request, it creates or updates the replica node entry, then sends the last broadcast snapshot of all registered delivery dataspaces.

Note

If the registered replica node repository ID or communication layer already exists, the replica node entry in the 'Registered replica nodes' technical table is updated, otherwise a new entry is created.

Performing an initialization

The initialization can be done:

- Automatically at replica node server startup.
- Manually when calling the replica node service 'Register replica node'.

Conditions

To be able to register, the following conditions must be fulfilled:

- The D3 mode must be 'hub' or 'slave'.
- The primary and replica node authentication parameters must correspond to the primary node administrator and replica node administrator defined in their respective directories.
- The delivery profiles defined on the replica node must exist in the primary node configuration.
- All data models contained in the registered dataspaces must exist in the replica node. If embedded, the data model names must be the same. If packaged, they must be located at the same module name and the schema path in the module must be the same in both the primary and replica nodes.
- The D3 primary node configuration has no validation error on the following scope: the technical record of the registered replica node and all its dependencies (dependent delivery profiles, delivery mappings and delivery dataspaces).

Note

To set the parameters, see the replica or hub EBX® properties in [Configuring primary, hub and replica nodes](#) [p 497].

76.3 Accessing delivery dataspaces

Data services

By default, when a data service accesses a delivery dataspace, it is redirected to the current snapshot, which is the last broadcast one. However, this default behavior can be modified either at the request level or in the global configuration.

Voir aussi [Common parameter 'disableRedirectionToLastBroadcast'](#) [p 716]

Access restrictions

On the primary node, a delivery dataspace can neither be merged nor closed. Other operations are available depending on permissions. For example, modifying a delivery dataspace directly, creating a snapshot independent from a broadcast, or creating and merging a child dataspace.

On the replica node, aside from the broadcast process, no modifications of any kind can be made to a delivery dataspace, whether by the end-user, data services, or a Java program. Furthermore, any dataspace-related operations, such as merge, close, etc., are forbidden on the replica node.

D3 broadcast Java API

The last broadcast snapshot may change between two calls if a broadcast has taken place in the meantime. If a fully stable view is required for several successive calls, these calls need to specifically refer to the same snapshot.

To get the last broadcast snapshot, see `D3Node.getBroadcastVersionAPI`.

CHAPITRE 77

D3 JMS Configuration

Ce chapitre contient les sections suivantes :

1. [JMS for distributed data delivery \(D3\)](#)

77.1 JMS for distributed data delivery (D3)

To configure D3 to use JMS instead of the default HTTP and TCP protocols, you must configure the [JMS connection factory](#) [p 346] and the following queues declared in the `WEB-INF/web.xml` deployment descriptor of the 'ebx' web application.

Note

If the TIBCO EBX® 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 EBX® runtime. See [JMS](#) [p 379] and [Distributed data delivery \(D3\)](#) [p 379] in the EBX® main configuration properties for more information on these properties.

Common declarations on primary and replica nodes (for shared queues)

Reserved resource name	Default JNDI name	Description
jms/EBX_D3MasterQueue	Weblogic: EBX_D3MasterQueue JBoss: java:/jms/EBX_D3MasterQueue	D3 primary JMS queue (only for D3 mode 'slave' or 'hub'). It specifies the queue name used to send SOAP requests to the D3 primary node. The message producer sets the primary node repository ID as a value of the header field JMSType. Java type: javax.jms.Queue
jms/EBX_D3ReplyQueue	Weblogic: EBX_D3ReplyQueue JBoss: java:/jms/EBX_D3ReplyQueue	D3 Reply JMS queue (for all D3 modes except the 'single' mode). It specifies the name of the reply queue for receiving SOAP responses. The consumption is filtered using the header field JMSCorrelationID. Java type: javax.jms.Queue
jms/EBX_D3ArchiveQueue	Weblogic: EBX_D3ArchiveQueue JBoss: java:/jms/EBX_D3ArchiveQueue	D3 JMS Archive queue (for all D3 modes except the 'single' mode). It specifies the name of the transfer archive queue used by the D3 node. The consumption is filtered using the header field JMSCorrelationID. If the archive weight is higher than the threshold specified in the property ebx.jms.d3.archiveMaxSizeInKB, the archive will be divided into several sequences. Therefore, the consumption is filtered using the header fields JMSXGroupID and JMSXGroupSeq instead. Java type: javax.jms.Queue
jms/EBX_D3CommunicationQueue	WebLogic: EBX_D3CommunicationQueue JBoss: java:/jms/EBX_D3CommunicationQueue	D3 JMS Communication queue (for all D3 modes except 'single' mode). It specifies the name of the communication queue where the requests are received. The consumption is filtered using the header field JMSType which corresponds to the current repository ID. Java type: javax.jms.Queue

Note

These JNDI names are set by default, but can be modified inside the web application archive ebx.war, included in EBXForWebLogic.ear (if using Weblogic) or in EBX.ear (if using JBoss, Websphere or other application servers).

Optional declarations on primary nodes (for replica-specific queues)

Note

Used for ascending compatibility prior to 5.5.0 or for mono-directional queues topology.

The deployment descriptor of the primary node must be manually modified by declaring specific communication and archive queues for each replica node. It consists in adding resource names in 'web.xml' inside 'ebx.war'. The replica-specific node queues can be used by one or more replica nodes. Resources can be freely named, but the physical names of their associated queue must correspond to the definition of replica nodes for resources `jms/EBX_D3ArchiveQueue` and `jms/EBX_D3CommunicationQueue`.

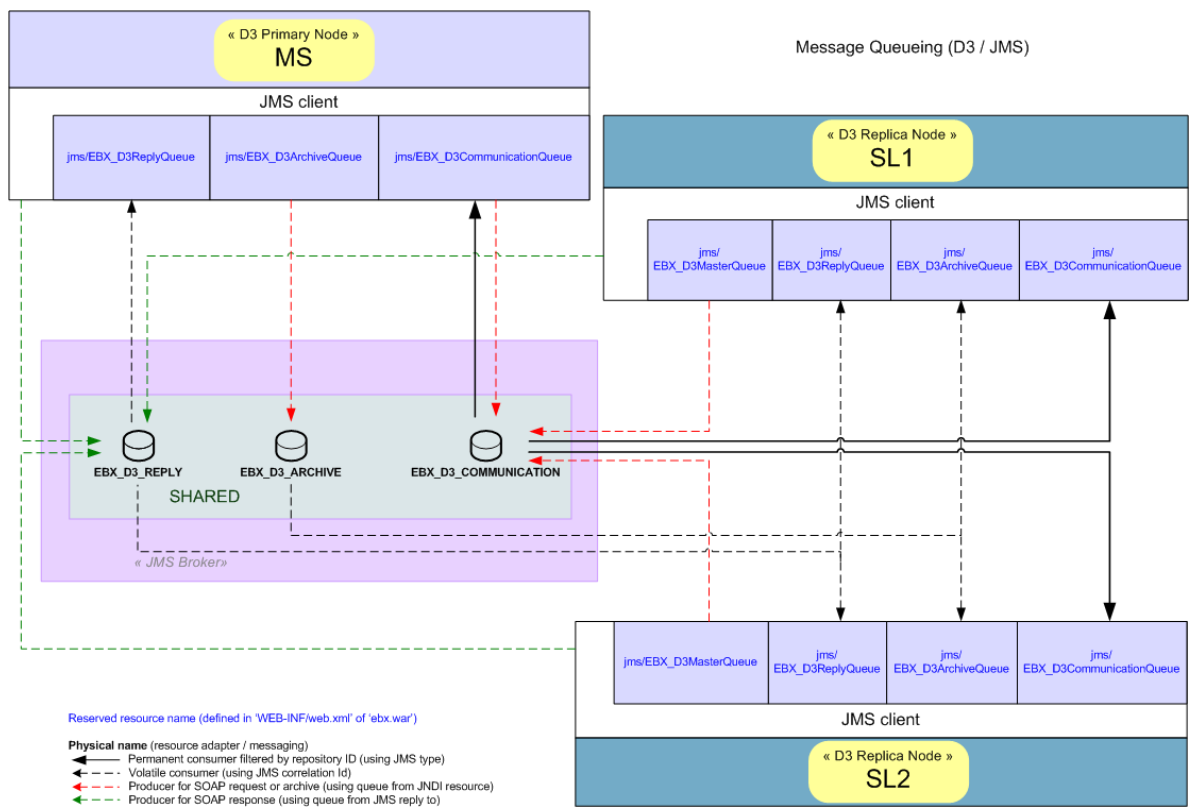
Note

Physical queue names matching: on registration, the replica node sends the communication and archive physical queue names. These queues are matched by physical queue name among all resources declared on the primary node. If unmatched, the registration fails.

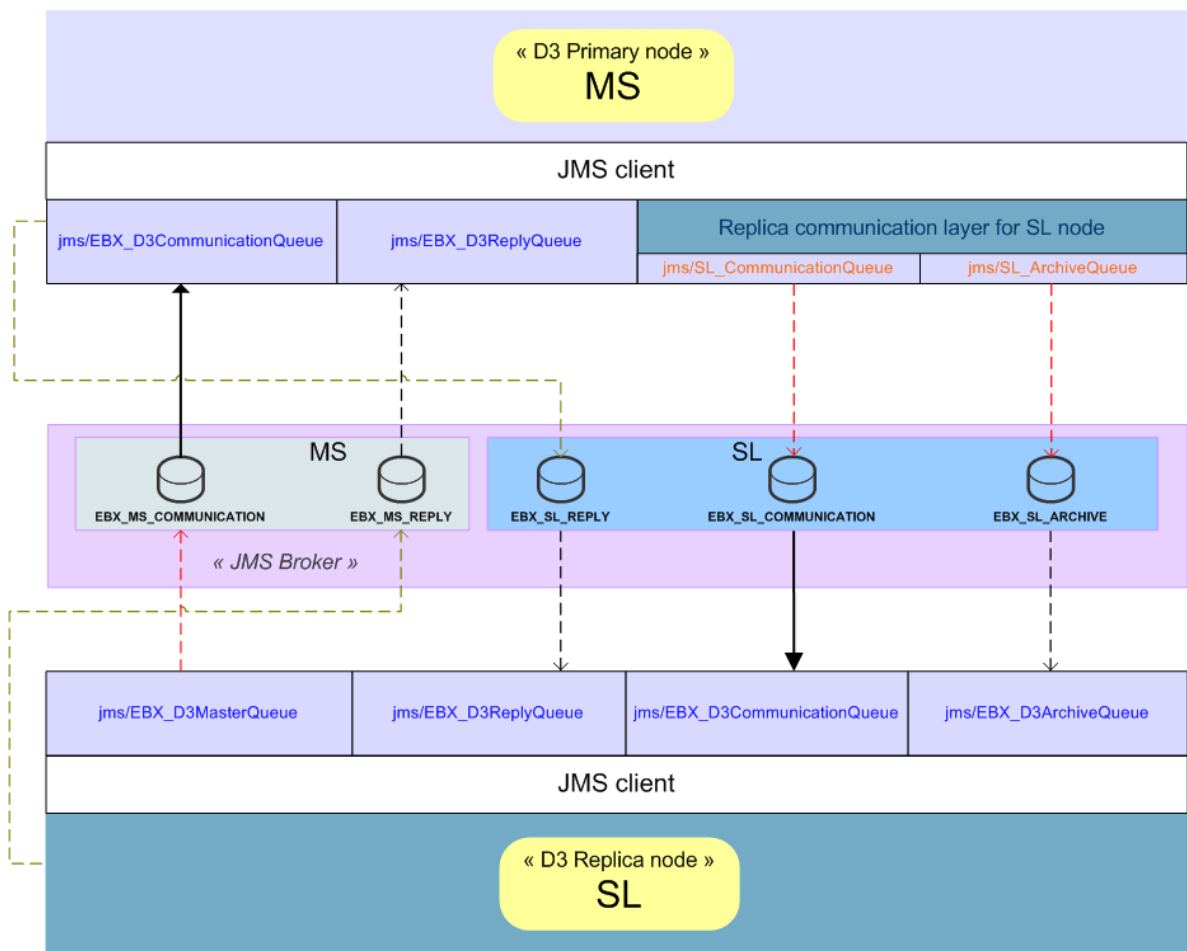
Examples of JMS configuration

	Shared queues	Specific queues
Primary-Replica nodes architecture	Between a primary node and two replica nodes with shared queues [p 490]	Between a primary node and a replica node with replica-specific queues [p 491]
Hub-Hub architecture	Between two hub nodes with shared queues [p 492]	Between two hub nodes with replica-specific queues [p 493]

Between a primary node and two replica nodes with shared queues



Between a primary node and a replica node with replica-specific queues

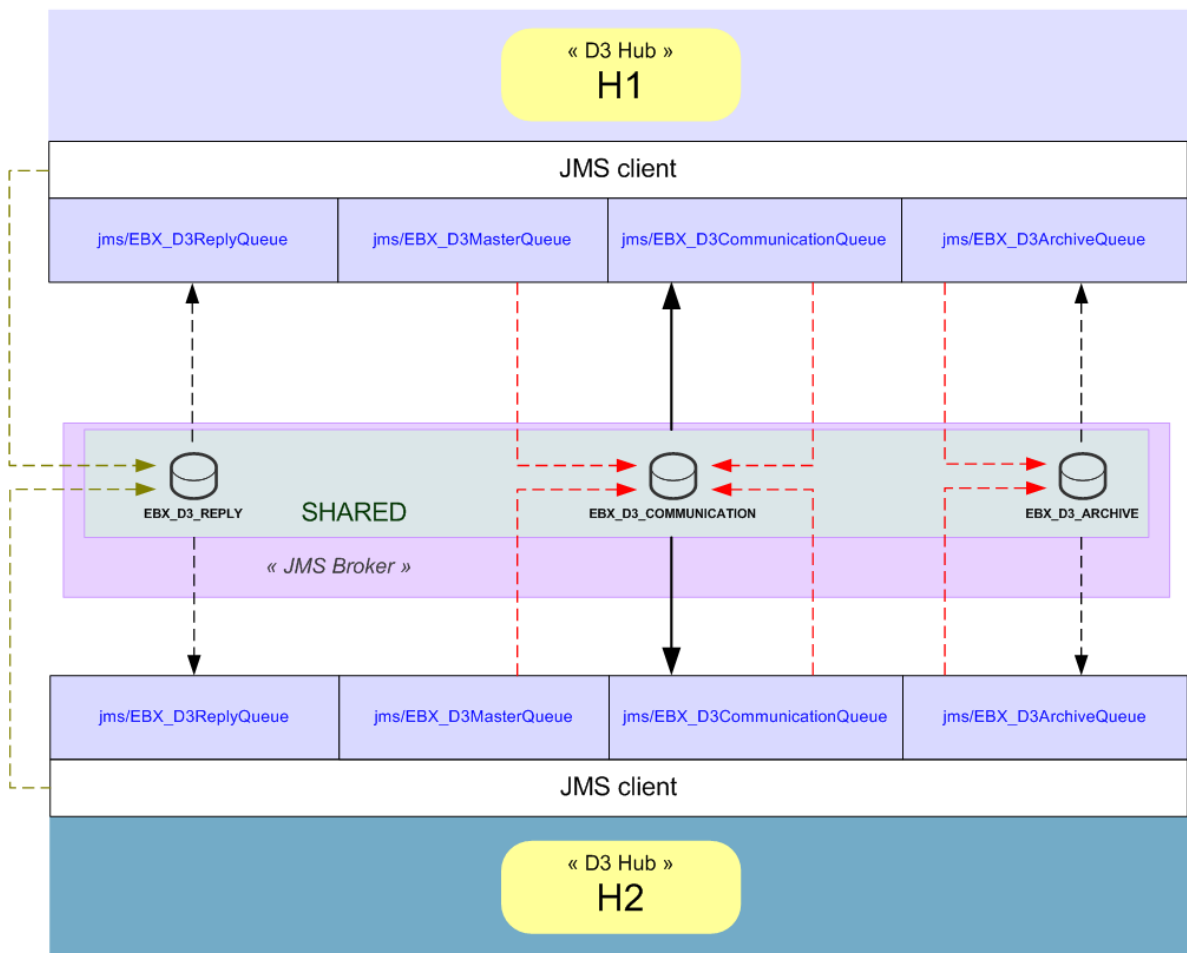


Reserved or custom resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

Physical name (resource adapter / messaging)

- ← Permanent consumer filtered by repository ID (using JMS type)
- ← -- Volatile consumer (using JMS correlation Id)
- ← - - - Producer for SOAP request or archive (using queue from JNDI resource)
- ← - - - Producer for SOAP response (using queue from JMS reply to)

Between two hub nodes with shared queues

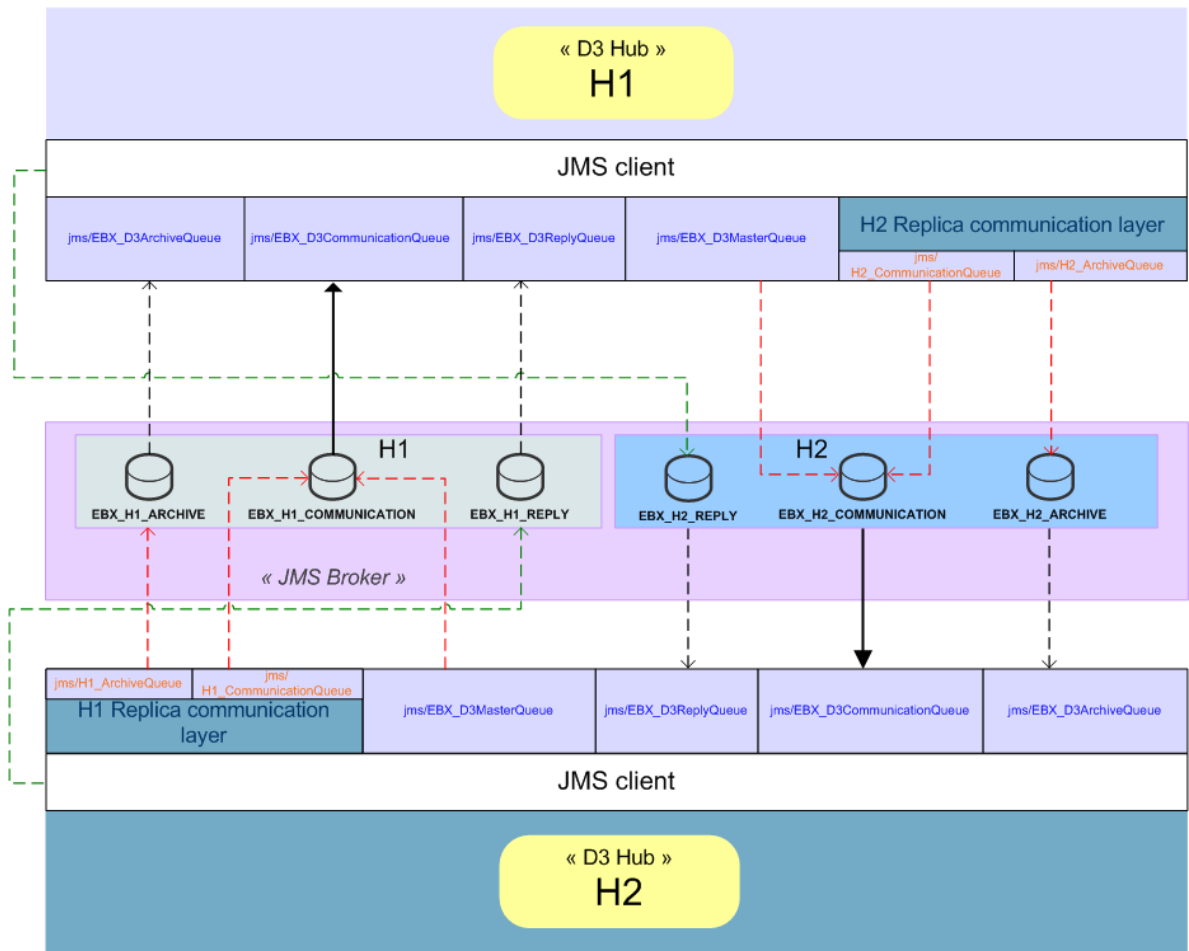


Reserved resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

Physical name (resource adapter / messaging)

- Permanent consumer filtered by repository ID (using JMS type)
- - - Volatile consumer (using JMS correlation Id)
- - - Producer for SOAP request or archive (using queue from JNDI resource)
- - - Producer for SOAP response (using queue from JMS reply to)

Between two hub nodes with replica-specific queues



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

Physical name (resource adapter / messaging)

- ← Permanent consumer filtered by repository ID (using JMS type)
- ← -- Volatile consumer (using JMS correlation Id)
- ← - - Producer for SOAP request or archive (using queue from JNDI resource)
- ← - - Producer for SOAP response (using queue from JMS reply to)

CHAPITRE 78

D3 administration

Ce chapitre contient les sections suivantes :

1. [Quick start](#)
2. [Configuring D3 nodes](#)
3. [Supervision](#)

78.1 Quick start

This section introduces the configuration of a basic D3 architecture with two TIBCO EBX® instances. Before starting, please check that each instance can work properly with its own repository.

Note

Deploy EBX® on two different web application containers. If both instances are running on the same host, ensure that all communication TCP ports are distinct.

Declare an existing dataspace on the primary node

The objective is to configure and broadcast an existing dataspace from a *primary* node.

This configuration is performed on the entire D3 infrastructure ([primary](#) [p 481] and [replica](#) [p 481] nodes included).

Update the `ebx.propertiesprimary` node configuration file with:

1. Define D3 mode as *primary* in key `ebx.d3.mode`.

Note

The *primary* node can be started after the configuration.

After authenticating as a built-in administrator, navigate within the administration tab:

1. Prerequisite: Check that the node is configured as a *primary* node (in the 'Actions' menu use 'System information' and check 'D3 mode').
2. Open the '[D3] Primary configuration' administration feature.
3. Add the dataspace to be broadcast to the 'Delivery dataspaces' table, and declare the allowed profile.
4. Add the [delivery profile](#) [p 481] to the 'Delivery profiles' table (it must correspond to a logical name) and declare the delivery mode. Possible values are: [cluster mode](#) [p 481] or [federation mode](#) [p 481].

5. Map the delivery dataspace with the delivery profile into the 'Delivery mapping' table.

Note

The *primary* node is now ready for the replica node(s) registration on the delivery profile. Check that the D3 broadcast menu appears in the 'Actions' menu of the dataspace or one of its snapshots.

Configure replica node for registration

The objective is to configure and register the *replica* node based on a delivery profile and communications settings.

Update the `ebx.properties` replica node configuration file with:

1. Define D3 mode as replica in key `ebx.d3.mode`.
2. Define the [delivery profile](#) [p 481] set on the *primary* node in key `ebx.d3.delivery.profiles` (delivery profiles must be separated by a comma and a space).
3. Define the *primary* node user authentication (must have the built-in administrator profile) for node communications in `ebx.d3.master.username` and `ebx.d3.master.password`.
4. Define [HTTP/TCP protocols](#) [p 499] for *primary* node communication, by setting a value for the property key `ebx.d3.master.url`
(for example `http://localhost:8080/ebx-dataservices/connector`).
5. Define the *replica* node user authentication (must have the built-in administrator profile) for node communications in `ebx.d3.slave.username` and `ebx.d3.slave.password`.
6. Define [HTTP/TCP protocols](#) [p 499] for *replica* node communication, by setting a value for the property key `ebx.d3.slave.url`
(for example `http://localhost:8090/ebx-dataservices/connector`).

Note

The *replica* node can be started after the configuration.

After authenticating as a built-in administrator, navigate inside the administration tab:

1. Prerequisite: Check that the node is configured as the *replica* node (in the 'Actions' menu use 'System information' and check 'D3 mode').
2. Open the '[D3] Replica configuration' administration feature.
3. Check the information on the 'Primary information' screen: No field should have the 'N/A' value.

Note

Please check that the model is available before broadcast (from data model assistant, it must be published).

The *replica* node is then ready for broadcast.

78.2 Configuring D3 nodes

Runtime configuration of primary and hub nodes through the user interface

The declaration of delivery dataspace and delivery profiles is done by selecting the '[D3] Primary configuration' feature from the 'Administration' area, where you will find the following tables:

Delivery dataspace	Declarations of the dataspace that can be broadcast.
Delivery profiles	Profiles to which replica nodes can subscribe. The delivery mode must be defined for each delivery profile.
Delivery mapping	The association between delivery dataspace and delivery profiles.

Note

The tables above are read-only while some broadcasts are pending or in progress.

Configuring primary, hub and replica nodes

This section details how to configure a node in its EBX® main configuration file.

Voir aussi [Overview](#) [p 371]

Primary node

In order to act as a *primary* node, an instance of EBX® must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=master` node:

```
#####
## D3 configuration
#####
# Configuration for master, hub and slave
#####
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=master
```

Voir aussi [primary node](#) [p 481]

Hub node

In order to act as a *hub* node (combination of primary and replica node configurations), an instance of EBX® must declare the following property in its main configuration file.

Sample configuration for `ebx.d3.mode=hub` node:

```
#####
## D3 configuration
#####
# Configuration for master, hub and slave
```

```
#####
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=hub

#####
# Configuration dedicated to hub or slave
#####
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

Voir aussi [hub node](#) [p 482]

Replica node

In order to act as a *replica* node, an instance of EBX® must declare the following property in its main configuration file.

Sample configuration for ebx.d3.mode=slave node:

```
#####
## D3 configuration
#####
# Configuration for master, hub and slave
#####
# Optional property.
# Possibles values are single, master, hub, slave
# Default is single meaning the server will be a standalone instance.
ebx.d3.mode=slave

#####
# Configuration dedicated to hub or slave
#####
# Profiles to subscribe to
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.delivery.profiles=

# User and password to be used to communicate with the master.
# Mandatory properties if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.master.username=
ebx.d3.master.password=

# User and password to be used by the master to communicate with the hub or slave.
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave
ebx.d3.slave.username=
ebx.d3.slave.password=
```

Voir aussi [replica node](#) [p 481]

Configuring the network protocol of a node

This section details how to configure the network protocol of a node in its EBX® main configuration file.

Voir aussi [Overview](#) [p 371]

HTTP(S) and socket TCP protocols

Sample configuration for `ebx.d3.mode=hub` or `ebx.d3.mode=slave` node with HTTP(S) network protocol:

```
#####
# HTTP(S) and TCP socket configuration for D3 hub and slave
#####
# URL to access the data services connector of the master
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: [protocol]://[master_host]:[master_port]/ebx-dataservices/connector
# Where the possible values of 'protocol' are 'http' or 'https'.
ebx.d3.master.url=

# URL to access the data services connector of the slave
# Mandatory property if ebx.d3.mode=hub or ebx.d3.mode=slave and JMS for D3 is not activated.
# This property will be ignored if JMS for D3 is activated.
# The URL must follow this pattern: [protocol]://[slave_host]:[slave_port]/ebx-dataservices/connector
# Where the possible values of 'protocol' are 'http' or 'https'.
ebx.d3.slave.url=

# Minimum port to use to transfer archives on TCP mode.
# Must be a positive integer above zero and below 65535.
# If not set, a random port will be used.
#ebx.d3.slave.socket.range.min=

# Max port to use on TCP mode to transfer archives.
# Must be a positive integer above ebx.d3.slave.socket.range.min and below 65535.
# Mandatory if ebx.d3.slave.socket.range.min is set.
#ebx.d3.slave.socket.range.max=
```

JMS protocol

If JMS is activated, the following properties can be defined in order to enable JMS functionalities for a D3 node.

Sample configuration for all D3 nodes with JMS network protocol:

```
#####
## JMS configuration for D3
#####
# Taken into account only if Data Services JMS is configured properly
#####
# Configuration for master, hub and slave
#####
# Default is false, activate JMS for D3
## If activated, the deployer must ensure that the entries
## 'jms/EBX_D3ReplyQueue', 'jms/EBX_D3ArchiveQueue' and 'jms/EBX_D3CommunicationQueue'
## are bound in the operational environment of the application server.
## On slave or hub mode, the entry 'jms/EBX_D3MasterQueue' must also be bound.
ebx.jms.d3.activate=false

# Change the default timeout when using reply queue.
# Must be a positive integer that does not exceed 3600000.
# Default is 10000 milliseconds.
#ebx.jms.d3.reply.timeout=10000

# Time-to-live message value expressed in milliseconds.
# This value will be set on each message header 'JMSExpiration' that defines the
# countdown before the message deletion managed by the JMS broker.
# Must be a positive integer equal to 0 or above the value of 'ebx.jms.d3.reply.timeout'.
# The value 0 means that the message does not expire.
# Default is 3600000 (one hour).
#ebx.jms.d3.expiration=3600000

# Archive maximum size in KB for the JMS body message. If exceeds, the message
# is transferred into several sequences messages in a same group, where each one does
# not exceed the maximum size defined.
# Must be a positive integer equals to 0 or above 100.
# Default is 0 that corresponds to unbounded.
#ebx.jms.d3.archiveMaxSizeInKB=

#####
# Configuration dedicated to hub or slave
#####
# Master repository ID, used to set a message filter for the concerned master when sending JMS message
# Mandatory property if ebx.jms.d3.activate=true and if ebx.d3.mode=hub or ebx.d3.mode=slave
```

#ebx.jms.d3.master.repositoryId=

Voir aussi [JMS for distributed data delivery \(D3\)](#) [p 487]

Services on primary nodes

Services to manage a primary node are available in the 'Administration' area of the replica node under '[D3] Primary node configuration' and also in the 'Delivery dataspace' and 'Registered replica nodes' tables. The services are:

Relaunch replays	Immediately relaunch all replays for waiting federation deliveries.
Delete replica node delivery dataspace	Delete the delivery dataspace on chosen replica nodes and/or unregister it from the configuration of the D3 primary node. To access the service, select a delivery dataspace from the 'Delivery dataspace' table on the primary node, then launch the wizard.
Fully resynchronize	Broadcast the full content of the last broadcast snapshot to the registered replica nodes.
Subscribe a replica node	Subscribe a set of selected replica nodes.
Deactivate replica nodes	Remove the selected replica nodes from the broadcast scope and switch their states to 'Unavailable'. Note The "in progress" broadcast contexts are rolled back.
Unregister replica nodes	Disconnects the selected replica nodes from the primary node. Note The "in progress" broadcast contexts are rolled back.

Note
The primary node services above are hidden while some broadcasts are pending or in progress.

Services on replica nodes

Services are available in the 'Administration' area under *[D3] Configuration of replica node* to manage its subscription to the primary node and perform other actions:

Register replica node	Re-subscribes the replica node to the primary node if it has been unregistered.
Unregister replica node	Disconnects the replica node from the primary node.
Note The "in progress" broadcast contexts are rolled back.	
Close and delete snapshots	Clean up a replica node delivery dataspace. To access the service, select a delivery dataspace from the 'Delivery dataspace' table on the replica node, then follow the wizard to close and delete snapshots based on their creation dates. Note: The last broadcast snapshot is automatically excluded from the selection.

78.3 Supervision

The last broadcast snapshot is highlighted in the snapshot table of the dataspace, it is represented by an icon displayed in the first column.

Primary node management console

Several tables make up the management console of the primary node, located in the 'Administration' area of the primary node, under '[D3] Primary node configuration'. They are as follows:

Registered replica nodes	Replica nodes registered with the primary node. From this table, several services are available on each record.
Broadcast history	History of broadcast operations that have taken place.
Replica node 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.

Primary node supervision services

Available in the 'Administration' area of the primary node under '[D3] Primary node configuration'. The services are as follows:

Check replica node information	Lists the replica nodes and related information, such as the replica node's state, associated delivery profiles, and delivered snapshots.
Clear history content	Deletes all records in all history tables, such as 'Broadcast history', 'Replica node registration log' and 'Detailed history'.

Replica node monitoring through the Java API

A replica node monitoring class can be created to implement actions that are triggered when the replica node's status switches to either 'Available' or 'Unavailable'. To do so, it must implement the `NodeMonitoring` interface. This class must be outside of any EBX® module and accessible from the class-loader of 'ebx.jar' and its full class name must be specified under '[D3] Replica node configuration'.

Voir aussi `NodeMonitoringAPI`

Primary node notification

A D3 administrator can set up mail notifications to receive broadcast events:

- On broadcast failure,
- On federation broadcast, if replays exceed a given threshold.

The mail contains a table of events with optional links to further details.

To enable notifications, open the '[D3] Primary node configuration' dataspace from the 'Administration' area and configure the 'Notifications' group under 'Global configuration'.

The 'From email' and 'URL definition' options should also be configured by using the 'Email configuration' link.

Log supervision

The technical supervision can be done through the log category 'ebx.d3', declared in the EBX® main configuration file. For example:

```
ebx.log4j.category.log.d3= INFO, Console, ebxFile:d3
```

Voir aussi [Configuring the EBX® logs](#) [p 376]

Temporary files

Some temporary files, such as exchanged archives, SOAP messages, broadcast queue, (...), are created and written to the EBX® temporary directory. This location is defined in the EBX® main configuration file:

```
#####
```

```
## Directories for temporary resources.
#####
# When set, allows specifying a directory for temporary files different from java.io.tmpdir.
# Default value is java.io.tmpdir
ebx.temp.directory = \\${java.io.tmpdir}

# Allows specifying the directory containing temporary files for cache.
# If unset, the used directory is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.cache.directory = ${ebx.temp.directory}/ebx.platform

# When set, allows specifying the directory containing temporary files for import.
# If unset, the used directory is ${ebx.temp.directory}/ebx.platform.
#ebx.temp.import.directory = ${ebx.temp.directory}/ebx.platform
```

Guide de sécurité (en anglais)

CHAPITRE 79

Security Best Practices

Here is a list of best practices that are considered useful to enforce a good security level for the EBX® setup. These best practices apply to EBX® and to other environments, their configuration, protocols and policies. While these practices are commonly regarded as beneficial, they may not be relevant to your particular infrastructure and security policy.

Ce chapitre contient les sections suivantes :

1. [Encryption algorithms](#)
2. [HTTPS](#)
3. [Installation](#)
4. [Web Server](#)
5. [Application Server](#)
6. [Java](#)
7. [Database](#)
8. [Archive directory](#)
9. [User directory and Administration rights](#)
10. [Permissions](#)

79.1 Encryption algorithms

The Web Server or Application Server may specify encryption algorithms when setting HTTPS parameters. Some recommendations on these algorithms are provided in section [HTTPS](#) [p 506]. Password and fields having `osd:password` as a type store a hash of their value, using the SHA_512 algorithm. This is, notably, the case for the password of users of the default directory.

79.2 HTTPS

It is recommended to use HTTPS for communication with clients (GUI and REST or SOAP). All HTTP traffic should be redirected to HTTPS.

A secure [cipher suite](#) and protocols should be used whenever possible. This applies, for example, to Web Servers, Application Servers, and JDBC connections.

TLS v1.2 should be the main protocol, because it is the only version that offers modern authenticated encryption (also known as AEAD).

Several obsolete cryptographic primitives must be avoided:

- Anonymous Diffie-Hellman (ADH) suites do not provide authentication,
- NULL cipher suites provide no encryption,
- Export cipher suites are insecure when negotiated in a connection, but they can also be used against a server that prefers stronger suites (the FREAK attack),
- Suites with weak ciphers (typically of 40 and 56 bits) use encryption that can easily be broken,
- RC4 is insecure,
- 3DES is slow and weak,

On the other hand, being too restrictive on allowed cyphers may prevent some clients from connecting, as they may not be able to negotiate a HTTPS connection.

The following configuration is compatible with browsers supported by EBX®.

- Cipher suites: ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
- Versions: TLSv1.2

79.3 Installation

Deployed components, such as as Web Server and Application Server, should be installed using a non-root or unprivileged user, and following the [principle of least privilege](#) whenever possible. For example, only necessary ports and protocols should be opened.

79.4 Web Server

If you have to expose web applications on the internet, it is a good practice to protect them with a Web Server in a [demilitarized zone](#), while EBX® and the database server can be in a production zone. Consider the following practices for your configuration.

The secure cipher suite and protocols should be set according to the above section [HTTPS](#) [p 506].

Do not use the default configuration, and remove any banner that might also expose the version and type of web server.

For example, on Apache2, to remove the banner (default page returned at the root), just remove the folder `/var/www/html`.

Also, on Apache2, to remove headers identifying the Web Server, the value of [ServerTokens](#) and [ServerSignature](#) from the file `security.conf` should have the following values:

```
# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of: Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
ServerTokens Prod

# Optionally add a line containing the server version and virtual host
# name to server-generated pages (internal error documents, FTP directory
# listings, mod_status and mod_info output etc., but not CGI generated
# documents or custom error documents).
# Set to "Email" to also include a mailto: link to the ServerAdmin.
# Set to one of: On | Off | Email
```

```
ServerSignature Off
```

Use the Web Server to set restrictions with HTTP security headers. Note that headers related to the origin impact authorized URLs for all resources returned by EBX®. That includes the content of fields of the URL type (example: image of avatar).

Here is a list of security headers and how to set them for EBX®. First, configure EBX® to not set any HTTP security headers. To do so, set the property `ebx.security.headers.activated` to false or unset.

X-XSS-Protection

The `x-xss-protection` header is designed to enable the cross-site scripting (XSS) filter built into modern web browsers. Here is what the header should look like.

```
x-xss-protection: 1; mode=block
```

Enable in Nginx

```
header always unset x-xss-protection
header always set x-xss-protection "1; mode=block"
```

Enable in Apache2

```
proxy_hide_header x-xss-protection;
add_header x-xss-protection "1; mode=block" always;
```

x-Frame-Options

The `x-frame-options` header provides clickjacking protection by not allowing iframes to load on the site. Be aware, this may not be compatible with your configuration if EBX® is integrated through frames for example. Here is what the header should look like:

```
x-frame-options: SAMEORIGIN
```

Enable in Nginx

```
add_header x-frame-options "SAMEORIGIN" always;
```

Enable in Apache2

```
header always sets x-frame-options "SAMEORIGIN"
```

X-Content-Type-Options

The `x-content-type-options` header prevents Internet Explorer and Google Chrome from sniffing a response away from the declared content-type. This helps reduce the danger of drive-by downloads and helps treat the content properly. Here is what the header looks like.

```
x-content-type-options: nosniff
```

Enable in Nginx

```
add_header X-Content-Type-Options "nosniff" always;
```

Enable in Apache2

```
header always sets X-Content-Type-Options "nosniff"
```

Strict-Transport-Security

The `strict-transport-security` header is a security enhancement that restricts web browsers to access web servers solely over HTTPS. This ensures the connection cannot be established through an insecure HTTP connection which could be vulnerable to attacks. Here is what the header should look like:

```
strict-transport-security: max-age=31536000; includeSubDomains
```

Enable in Nginx

```
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;
```

Enable in Apache2

```
header always sets Strict-Transport-Security "max-age=31536000; includeSubDomains"
```

Content-Security-Policy

The content-security-policy HTTP header provides an additional layer of security. This policy helps prevent attacks such as Cross Site Scripting (XSS) and other code injection attacks by defining content sources which are approved and thus allowing the browser to load them. Here is what the header should look like. Make sure to adapt it with your domain name (server.company.com in the example).

```
content-security-policy: default-src 'self'; font-src * data: server.company.com; img-src * data: server.company.com; script-src * 'unsafe-inline' 'unsafe-eval'; style-src * 'unsafe-inline';
```

Enable in Nginx

```
add_header Content-Security-Policy "default-src 'self'; font-src * data: server.company.com; img-src * data: server.company.com; script-src * 'unsafe-inline' 'unsafe-eval'; style-src * 'unsafe-inline';" always;
```

Enable in Apache2

```
header always sets Content-Security-Policy "default-src 'self'; font-src * data: server.company.com; img-src * data: server.company.com; script-src * 'unsafe-inline' 'unsafe-eval'; style-src * 'unsafe-inline';"
```

Referrer-Policy

The Referrer-Policy HTTP header governs which referrer information should be included with requests made. The Referrer-Policy tells the web browser how to handle referrer information that is sent when a user clicks on a link that leads to another page. Here is what it should look like:

```
Referrer-Policy: strict-origin
```

Enable in Nginx

```
add_header Referrer-Policy: "strict-origin" always;
```

Enable in Apache2

```
header always sets Referrer-Policy "strict-origin"
```

79.5 Application Server

As for Web Servers, the same best practice applies: do not expose technical information on the Application Server. For example, for Tomcat, it is recommended to fill the attribute server of connector in server.xml with a generic value as AppServer.

```
<Connector port="8080" enableLookups="false" protocol="HTTP/1.1" useBodyEncodingForURI="true" server="AppServer"/>
```

If the Application Server is exposed through HTTPS, the secure cipher suite and Protocols should be set according to the above section [HTTPS](#) [p 506].

If there is a Web Server, it is also recommended to use ports higher than 1024 and let the Web Server do proxy.

If there is no Web Server, security headers should be set by the Application Server as described above.

79.6 Java

It is recommended to follow the [security best practices from Oracle](#). Last supported patches should also be applied as soon as they are available, especially when they include security patches. Consider using the Server JRE for server systems, such as application servers or other long-running back-end

processes. The Server JRE is the same as the regular JRE except that it does not contain the web-browser plugins.

EBX® allows a very high level of customization through custom code. All integrated Java modules are considered by EBX® as trusted. Hence, all development on top of EBX® should be reviewed and validated. As an example, developers should not generate HTML from values coming from the database without proper escaping. For more details on this, see the [Cross Site Scripting prevention on the OWASP site](#). Here is a proper escaping example: the name of a store is encoded before being displayed in an HTML form. The `StringEscapeUtils` class included in Apache Commons Lang is used for string encoding.

```
public class StoreMainPane implements UIFormPane
{
    public static final String STORE_NAME_STYLE = "font-weight: bold; padding-top:20px; padding-bottom:20px";

    @Override
    public void writePane(final UIFormPaneWriter writer, final UIFormContext context)
    {
        String storeName = (String) context.getValueContext().getValue(Paths._Store._Name);

        writer.add("<div").addSafeAttribute("style", STORE_NAME_STYLE).add(">");
        writer.add("Data stored for " + StringEscapeUtils.escapeHtml(storeName));
        writer.add("</div>");

        // ...
    }
}
```

79.7 Database

Databases should be encrypted at rest and in transit. If there is a private key for encryption, it should not be stored in the same location as the data files. Regarding the JDBC connection, consider configuring the JDBC driver to use SSL/TLS. Contact your database administrator for detailed instructions. You should always use the last supported version or RDBMS, including drivers.

79.8 Archive directory

On the server, the [archive directory](#) [p 416] must be properly secured and/or encrypted. Indeed, any archive exported from the EBX® instance will be created there, and these archives are neither encrypted nor protected by password. As a consequence, any user with an access to these files will be able to see the content regardless of any permission defined in EBX®.

79.9 User directory and Administration rights

For production and test platforms, EBX® must be integrated with a [custom directory](#) [p 454] to enforce the password policy of your company. The default directory can be used only for development platforms.

According to the [Separation of Duties](#) best practice, administrators can manage users and grant access but should not have any functional rights.

79.10 Permissions

Special care is required when defining permissions in EBX®. Persons in charge of this are expected to be aware of the content of the [permission documentation](#) [p 293], and especially the information provided in the [Important considerations about permissions](#) [p 295] section.

Guide du développeur (en anglais)

Introduction

CHAPITRE 80

Packaging TIBCO EBX® modules

An EBX® module is a standard Java EE web application, packaging various resources such as XML Schema documents, Java classes and static resources.

Since EBX® modules are web applications they benefit from features such as class-loading isolation, WAR or EAR packaging, and Web resources exposure.

Ce chapitre contient les sections suivantes :

1. [Module structure](#)
2. [Module declaration](#)
3. [Module registration](#)
4. [Packaged resources](#)

80.1 Module structure

An EBX® module contains the following files:

/WEB-INF/ebx/module.xml	This mandatory document defines the main properties and services of the module. See Module declaration [p 514].
/WEB-INF/web.xml	This is the standard Java EE deployment descriptor. It can perform the registration of the EBX® module when the application server is launched. See Module registration [p 514].
/META-INF/MANIFEST.MF	Optional. If present, EBX® reports the 'Implementation-Title' and 'Implementation-Version' values to <i>Administration > Configuration technique > Modules et modèles de données</i> .
/www/	This optional directory contains all packaged resources, which are accessible via public URL. See Packaged resources [p 516].

Required files for Oracle WebLogic server:

/WEB-INF/weblogic.xml

WebLogic deployment descriptor file which activates the prefer-web-inf-classes policy, such as the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app xmlns="http://xmlns.oracle.com/weblogic/
weblogic-web-app">
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
</weblogic-web-app>
```

See [weblogic.xml Deployment Descriptor Elements](#) for more information.

80.2 Module declaration

A module is declared using the document /WEB-INF/ebx/module.xml. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:ebx-schemas:module_2.4"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ebx-schemas:module_2.4 http://schema.orchestranetworks.com/module_2.4.xsd">
  <name>moduleTest</name>
</module>
```

See the [associated schema](#) for documentation about each property. The main properties are as follows:

Element	Description	Required
name	Defines the unique identifier of the module in the server instance. The module name usually corresponds to the name of the web application (the name of its directory).	Yes.
publicPath	Defines a path other than the module's name identifying the web application in public URLs. This path is added to the URL of external resources of the module when computing absolute URLs. If this field is not defined, the public path is the module's name, defined above.	No.
services	Declares user services using the legacy API. See <i>Declaration and configuration</i> of legacy user services. From the version 5.8.0, it is strongly advised to use the new user services [p 661].	No.
beans	Declares reusable Java bean components. See the workflow package [p 647].	No.
ajaxComponents	Declares Ajax components. See Declaring an Ajax component in a module <code>UIAjaxComponent.declareInModule^{apt}</code> in the Java API.	No.

80.3 Module registration

In order to be identifiable by EBX®, a module must be registered at runtime when the application server is launched. For a web application, every EBX® module must:

- contain a Java class with the annotation `@WebListener` extending the class `ModuleRegistrationListenerAPI`.

Attention

When using the `@WebListener` annotation, ensure that the application server is configured to activate the servlet 3.0 annotation scanning for the web application. See [JSR 315: Java™ Servlet 3.0 Specification](#) for more information.

or:

- contain a Servlet extending the class `ModuleRegistrationServletAPI`;
- make a standard declaration of this servlet in the deployment descriptor `/WEB-INF/web.xml`;
- ensure that this servlet will be registered at server startup by adding the following standard element to the deployment descriptor: `<load-on-startup>1</load-on-startup>`.

Additional recommendations and information:

- The method `handleRepositoryStartup` in `ModuleRegistrationServletAPI` allows setting the logger associated with the module and defining additional behavior such as common JavaScript and CSS resources.
- The specific class extending `ModuleRegistrationServlet` must be located in the web application (under `/WEB-INF/classes` or `/WEB-INF/lib`; due to the fact that this class is internally used as a hook to the application's class-loader, to load Java classes used by the data models associated with the module).
- The application server startup process is asynchronous and web applications / EBX® modules are discovered dynamically. The EBX® repository initialization depends on this process and will wait for the registration of all used modules up to an unlimited amount of time. As a consequence, if a used module is not deployed for any reason, it must be declared in the EBX® main configuration file. For more information, see the property [Declaring modules as undeployed](#) [p 385].
- All module registrations and unregistrations are logged in the `log.kernel` category.
- If an exception occurs while loading a module, the cause is written in the application server log.
- Once the servlet is out of service, the module is unregistered and the data models and associated datasets become unavailable. Note that hot deployment/undeployment is [not supported](#) [p 334].

Deployment descriptor example

Here is an example of a Java EE deployment descriptor (`/WEB-INF/web.xml`):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    https://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-app_3_0.xsd"
  version="3.0">
  <servlet>
    <servlet-name>InitEbxServlet</servlet-name>
    <servlet-class>com.foo.RegisterServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
</web-app>
```

Registration example

Here is an implementation example of the `ModuleRegistrationServlet`:

```
package com.foo;
import javax.servlet.*;
import javax.servlet.http.*;
import com.onwbp.base.repository.*;
/**
 */
public class RegisterServlet extends ModuleRegistrationServlet
{

    public void handleRepositoryStartup(ModuleContextOnRepositoryStartup aContext)
    throws OperationException
    {
        // Perform module-specific initializations here
        ...

        // Declare custom resources here
        aContext.addExternalStyleSheetResource(MyCompanyResources.COMMON_STYLESHEET_URL);
        aContext.addExternalJavaScriptResource(MyCompanyResources.COMMON_JAVASCRIPT_URL);

        aContext.addPackagedStyleSheetResource("myModule.css");
        aContext.addPackagedJavaScriptResource("myModule.js");

    }

    public void handleRepositoryShutdown()
    {
        // Release resources of the current module when the repository is shut down here
        ...
    }

    public void destroyBeforeUnregisterModule()
    {
        // Perform operations when this servlet is being taken out of service here
        ...
    }
}
```

80.4 Packaged resources

The packaged resources are files and documents that can be directly accessed from client browsers and can be managed and specified either as `osd:resource` fields or via the Java API. They have various types and can also be localized.

Voir aussi

ResourceType^{APT}

[Type `osd:resource` \[p 538\]](#)

Directory structure

The packaged resources must be located under the following directory structure:

1. On the first level, the directory `/www/` must be located at the root of the module (web application).
2. On the second level, the directory must specify the localization. It can be:
 - `common/` should contain all the resources to be used by default, either because they are locale-independent or as the default localization (in EBX®, the default localization is `en`, namely English);
 - `{lang}/` when localization is required for the resources located underneath, with `{lang}` to be replaced by the actual locale code; it should correspond to the locales supported by EBX®; for more information, see [Configuring EBX® localization](#) [p 375].

3. On the third level, the directory must specify the resource type. It can be:

- `jscripts/` for JavaScript resources;
- `stylesheets/` for Cascading Style Sheet (CSS) resources;
- `html/` for HTML resources;
- `icons/` for icon typed resources;
- `images/` for image typed resources.

Example

In this example, the image `logowithText.jpg` is the only resource that is localized:

```
/www
├── common
│   ├── images
│   │   ├── myCompanyLogo.jpg
│   │   └── logowithText.jpg
│   ├── jscripts
│   │   └── myCompanyCommon.js
│   └── stylesheets
│       └── myCompanyCommon.css
├── de
│   └── images
│       └── logowithText.jpg
└── fr
    └── images
        └── logowithText.jpg
```


CHAPITRE 81

Mapping to Java

Ce chapitre contient les sections suivantes :

1. [How to access data from Java?](#)
2. [Transactions and concurrency](#)
3. [Mapping of data types](#)
4. [Java bindings](#)

81.1 How to access data from Java?

Read access

Data can be read from various generic Java classes, mainly `AdaptationAPI` and `ValueContextAPI`. The getter methods for these classes return objects that are typed according to the mapping rules described in the section [Mapping of data types](#) [p 521].

Write access

Data updates must be performed in a well-managed context:

- In the context of a procedure execution, by calling the methods `setValue...` of the interface `ValueContextForUpdateAPI`, or
- During the user input validation, by calling the method `setNewValue` of the class `ValueContextForInputValidationAPI`.

Modification of mutable objects

According to the mapping that is described in the [Mapping of data types](#) [p 521] 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 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.

81.2 Transactions and concurrency

Concurrency

At the dataspace level	In a single dataspace, the system supports running only one single read-write Procedure and multiple concurrent ReadOnlyProcedures. Concurrent accesses outside any Procedure are also supported.
At the repository level	At the repository level, concurrency is limited for only some specific operations. For example (non-exhaustive list): • A data model publication excludes many operations. • A dataspace merge excludes write operations on the two dataspaces involved in the merge.

Queries snapshot isolation

The following table defines the properties related to queries isolation. Note that all of the rules applying to `QueryResult` also apply to `RequestResult`:

Queries outside of a Procedure	Data is frozen at the time of fetching the <code>QueryResult</code> . More precisely, a query result accesses only committed data as of the last committed transaction at the time of fetching this result. The content of this result never changes afterwards. A query outside of a Procedure can be considered as a self-containing <code>ReadOnlyProcedure</code> .
Queries inside of a Procedure, in the same dataspace as the Procedure underlying dataspace	The <code>QueryResult</code> reflects the last committed state before the Procedure starts and the changes that occurred in the Procedure previously to the <code>QueryResult</code> fetch. The content of this result never changes afterwards, whatever happens in the Procedure.
Queries inside of a Procedure, in another dataspace	The consistency is guaranteed at the repository level, so the <code>QueryResult</code> reflects the last committed state before the Procedure starts. The content of this result never changes after the query is fetched, whatever happens in the whole repository.

Adaptation objects

In Java, a persistent dataset or a persistent record are both represented by an instance of the `Adaptation` class.

The following table defines the properties related to Adaptation objects.

Immutability	An <code>Adaptation</code> object instance is immutable. Therefore, the client code should not "hold" an <code>Adaptation</code> object for a long time (in particular beyond a transaction boundaries). However, it is possible to invoke the method <code>Adaptation.getUpToDateInstance^{API}</code>.
Fetch	If an <code>Adaptation</code> is fetched from a <code>QueryResult</code>, then the snapshot isolation rules described in the previous section apply. Otherwise, if an <code>Adaptation</code> is fetched from a running <code>Procedure</code>, it reflects the last committed state before the <code>Procedure</code> starts. Otherwise, outside of a <code>QueryResult</code> or a running <code>Procedure</code>, the <code>Adaptation</code> reflects the state of the record on its fetch-time. Voir aussi <code>AdaptationHome.findAdaptationOrNull^{API}</code> <code>AdaptationTable.</code> <code>lookupAdaptationByPrimaryKey^{API}</code>

81.3 Mapping of data types

This section describes how XML Schema type definitions and element declarations are mapped to Java types.

Simple data types

Basic rules for simple data types

Each XML Schema simple type corresponds to a Java class; the mapping is documented in the table [XML Schema built-in simple types](#) [p 534].

Voir aussi `SchemaNode.createNewOccurrenceAPI`

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 that is determined from the mapping of the simple type (see previous section).

Complex data types

Complex type definitions without a class declaration

By default (no attribute `osd:class`), a terminal node of a complex type is instantiated using an internal class. This class provides a generic JavaBean implementation. However, if a custom client Java code

must access these values, use a custom JavaBean. To do so, use the `osd:class` declaration described in the next section.

You can transparently instantiate, read and modify the mapped Java object, with or without the attribute `osd:class`, by invoking the methods `SchemaNode.createNewOccurrenceAPI`, `SchemaNode.executeReadAPI` and `SchemaNode.executeWriteAPI`.

Mapping of complex types to custom JavaBeans

You can map an XML Schema complex type to a custom Java class. This is done by adding the attribute `osd:class` to the complex node definition. Unless the element has `xs:maxOccurs > 1`, you must also specify the attribute `osd:access` for the node to be considered a *terminal* node. If the element has `xs:maxOccurs > 1`, it is automatically considered to be terminal.

The custom Java class must conform to the JavaBean protocol. This means that each child of the complex type must correspond to a JavaBean property of the class. Additionally, each JavaBean property must be a read-write property, and its implementation must ensure that the value set by the setter method is returned, as-is, by the getter method. Contextual computations are not allowed in these methods.

Example

In this example, the Java class `com.carRental.Customer` must define the methods `getFirstName()` and `setFirstName(String)`.

A JavaBean can have a custom user interface within TIBCO EBX®, by using a `UIBeanEditorAPI`.

```
<xs:element name="customer" osd:access="RW">
  <xs:complexType name="subscriber" osd:class="com.carRental.Customer">
    <xs:sequence>
      <xs:element name="firstName" type="xs:string"/>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Multiple cardinality on a complex element

If the attribute `maxOccurs` is greater than 1, then the corresponding instance in Java is:

- An instance of `java.util.List` for an aggregated list, where every element in the list is an instance of the Java class determined by the [mapping of simple types](#) [p 534], or
- An instance of `AdaptationTableAPI`, if the property `osd:table` is specified.

81.4 Java bindings

Java bindings support generating Java types that reflect the structure of the data model. The Java code generation can be done in the user interface. See [Generating Java bindings](#) [p 525].

Benefits

Ensuring the link between XML Schema structure and Java code provides a number of benefits:

- **Development assistance:** Auto-completion when you type an access path to parameters, if it is supported by your IDE.
- **Access code verification:** All accesses to parameters are verified at code compilation.
- **Impact verification:** Each modification of the data model impacts the code compilation state.

- **Cross-referencing:** By using the reference tools of your IDE, you can easily verify where a parameter is used.

Consequently, it is strongly encouraged that you use Java bindings.

XML declaration

The specification of the Java types to be generated from the data model is included in the main schema. Each binding element defines a generation target. It must be located at, in XPath notation, `xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding`, where the prefix `ebxbnd` is a reference to the namespace identified by the URI `urn:ebx-schemas:binding_1.0`. Several binding elements can be defined if you have different generation targets.

The attribute `targetDirectory` of the element `ebxbnd:binding` defines the root directory used for Java type generation. Generally, it is the directory containing the project source code, `src`. A relative path is interpreted based on the current runtime directory of the VM, as opposed to the XML schema.

See [bindings XML Schema](#).

XML bindings 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/">
        <jpathConstants typeName="com.creditonline.RulesPaths">
          <nodes root="/rules" prefix="" />
        </jpathConstants>
        <jpathConstants typeName="com.creditonline.StylesheetConstants">
          <nodes root="/stylesheet" prefix="" />
        </jpathConstants>
      </ebxbnd:binding>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

Java constants can be defined for XML schema paths. To do so, generate one or more interfaces from a schema node, including the root node `/`. The example generates two Java path constant interfaces, one from the node `/rules` and the other from the node `/stylesheet` in the schema. Interface names are described by the element `jpathConstants` with the attribute `typeName`. The associated node is described by the element `nodes` with the attribute `root`.

CHAPITRE 82

Tools for Java developers

TIBCO EBX® provides Java developers with tools to facilitate use of the EBX® API, as well as integration with development environments.

Ce chapitre contient les sections suivantes :

1. [Activating the development tools](#)
2. [Data model refresh tool](#)
3. [Generating Java bindings](#)
4. [Path to a node](#)
5. [Web component link generator](#)

82.1 Activating the development tools

To activate the development tools, run EBX® in *development mode*. This is specified in the EBX® main configuration file [EBX® run mode](#) [p 387] using the property `backend.mode=development`.

82.2 Data model refresh tool

When editing the data model directly as an XML Schema document without using the data-modeling tool provided by EBX®, you can refresh it without restarting the application server.

In the 'Administration' area, select **Select > Configuration technique > Outils de développement > Synchroniser les schémas mis à jour** (or **Synchroniser tous les schémas**).

Attention

Since the operation is critical regarding data consistency, refreshing the data models acquires a global exclusive lock on the repository. This means that most other operations (data access and update, validation, etc.) will wait until the completion of the data model refresh.

82.3 Generating Java bindings

The Java types specified by Java bindings can be generated from a dataset or a data model, by selecting **Actions > Generate Java** in the navigation pane.

Voir aussi [Java bindings](#) [p 522]

82.4 Path to a node

The field 'Data path' is displayed in the documentation pane of a node. This field indicates the path to the node, which can be useful when writing XPath formulas.

Note

This field is always available to administrators.

82.5 Web component link generator

The 'Web component link generator' service is a user interface designed to create HTTP requests that call EBX® web components. To launch this service, select **Actions > Web component link generator** in the navigation pane.

CHAPITRE 83

Terminology changes

A new TIBCO EBX® release can introduce new vocabulary for users. To preserve the backward compatibility, these terminology changes do not usually impact the API. Consequently, Java class names, method names, data services operation names, etc. still use the older version terminology. This chapter purpose is to facilitate the correspondence of the old term in the API to the new terms.

Voir aussi [Glossaire](#) [p 25]

Ce chapitre contient les sections suivantes :

1. [Terminology changes in version 5.9](#)
2. [Terminology changes in version 5.0](#)

83.1 Terminology changes in version 5.9

New term	Term prior to version 5.9.0
D3 primary node	D3 master node
D3 replica node	D3 slave node

83.2 Terminology changes in version 5.0

The following table summarizes the mappings between the version 5.0.0 terminology and previous terminology:

New term	Term prior to version 5.0.0
Dataset	Adaptation instance
Child dataset	Child adaptation instance
Data model	Data model
Dataspace	Branch
Snapshot	Version
Dataspace 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

Data model

CHAPITRE 84

Introduction

A data model is a structural definition of the data to be managed in the TIBCO EBX® repository. Data models contribute to EBX®'s ability to guarantee the highest level of data consistency and to facilitate data management.

Specifically, the data model is a document that conforms to the XML Schema standard (W3C recommendation). Its main features are as follows:

- A rich library of well-defined [simple data types](#) [p 533], such as integer, boolean, decimal, date, time;
- The ability to define additional [simple types](#) [p 535] and [complex types](#) [p 535];
- The ability to define simple lists of items, called [aggregated lists](#) [p 545];
- [Validation constraints](#) [p 571] (facets), for example: enumerations, uniqueness constraints, minimum/maximum boundaries.

EBX® also uses the extensibility features of XML Schema for other useful information, such as:

- [Predefined types](#) [p 536], for example: locale, resource, html;
- Definition of [tables](#) [p 549] and [foreign key constraints](#) [p 554];
- Mapping data in EBX® to Java beans;
- [Advanced validation constraints](#) [p 571] (extended facets), such as dynamic enumerations;
- Extensive [presentation information](#) [p 591], such as labels, descriptions, and error messages.

Note

EBX® supports a subset of the W3C recommendations, as some features are not relevant to Master Data Management.

Ce chapitre contient les sections suivantes :

1. [Editing the data model](#)
2. [References](#)
3. [Relationship between datasets and data models](#)
4. [Pre-requisite for XML Schemas](#)
5. [Conventions](#)
6. [Schemas with reserved names](#)

84.1 Editing the data model

There are two different ways to define a data model:

- The data model can be defined using an XML Schema editor or through the data model assistant. The data model assistant has the advantage of being integrated into the EBX® user interface, abstracting the verbose underlying XML. For more information, see [Introduction aux modèles de données](#) [p 36]. The data model assistant allows using features that are not documented to be used outside of the DMA; e.g. Toolbars and Widgets.
- By using an external XML Schema document editor.

84.2 References

For an introduction to XML Schema, see the W3Schools [XML Schema Tutorial](#).

Voir aussi

[XML Schema Part 0: Primer](#)

[XML Schema Part 1: Structures](#)

[XML Schema Part 2: Datatypes](#)

84.3 Relationship between datasets and data models

Each root dataset is associated with a single data model. At the dataspace creation, an associated data model is selected, on which to base the dataset.

Voir aussi [Création du jeu de données](#) [p 125]

84.4 Pre-requisite for XML Schemas

In order for an XML Schema to be accepted by EBX®, it must include a global element declaration that includes the attribute `osd:access="- -"`.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

84.5 Conventions

By convention, namespaces are always defined as follows:

Prefix	Namespace
xs:	http://www.w3.org/2001/XMLSchema
osd:	urn:ebx-schemas:common_1.0
fmt:	urn:ebx-schemas:format_1.0
usd:	urn:ebx-schemas:userServices_1.0
emd:	urn:ebx-schemas:entityMappings_1.0

84.6 Schemas with reserved names

Several data models in EBX® have reserved names.

All references to other data models (using the attribute `schemaLocation` for an import, include or redefine) that end with one of the following strings are reserved:

- common_1.0.xsd
- org_1.0.xsd
- coreModel_1.0.xsd
- session_1.0.xsd
- userServices_1.0.xsd
- entityMappings_1.0.xsd

These XSD files correspond to the schemas provided for the module `ebx-root-1.0`, at the path `/WEB-INF/ebx/schemas`. The attribute `schemaLocation` can reference the files at this location or a copy, if the file names are identical. This is useful if you want to avoid a module dependency on `ebx-root-1.0`.

For security reasons, EBX® uses an internal definition for these schemas to prevent any modification.

CHAPITRE 85

Data types

This chapter details the data types supported by TIBCO EBX®.

Voir aussi [Tables and relationships](#) [p 549]

Ce chapitre contient les sections suivantes :

1. [XML Schema built-in simple types](#)
2. [XML Schema named simple types](#)
3. [XML Schema complex types](#)
4. [Extended simple types defined by EBX®](#)
5. [Complex types defined by EBX®](#)
6. [Aggregated lists](#)
7. [Including external data models](#)

85.1 XML Schema built-in simple types

The table below lists all the simple types defined in XML Schema that are supported by EBX®, along with their corresponding Java types.

XML Schema type	Java class	Notes
xs:string	<code>java.lang.String</code>	The default strategies defined for string fields are detailed in the section Default strategy for string fields [p 317].
xs:boolean	<code>java.lang.Boolean</code>	
xs:decimal	<code>java.math.BigDecimal</code>	A <code>totalDigits</code> facet with a value equal to 15 is added by default to decimal fields that are contained in a mapped table (historized or replicated table). However, this facet can be overwritten with a greater value in the data model.
xs:dateTime	<code>java.util.Date</code>	
xs:time	<code>java.util.Date</code>	The date portion of the returned Date is always set to '1970/01/01'.
xs:date	<code>java.util.Date</code>	The time portion of the returned Date is always the beginning of the day, that is, '00:00:00'.
xs:anyURI	<code>java.net.URI</code>	
xs:Name (xs:string restriction)	<code>java.lang.String</code>	
xs:int (xs:decimal restriction)	<code>java.lang.Integer</code>	
xs:integer (xs:decimal restriction)	<code>java.lang.Integer</code>	This mapping does not comply with the XML Schema recommendation. Although the XML Schema specification states that <code>xs:integer</code> has no value space limitation, this value space is, in fact, restricted by the Java specifications of the <code>java.lang.Integer</code> object.

The mapping between XML Schema types and Java types are detailed in the section [Mapping of data types](#) [p 521].

85.2 XML Schema named simple types

Named simple types can be defined when designing a data model for redefining an existing built-in simple type. A *named simple type* can be reused in the data model.

Restrictions:

- In the data model, only the element restriction is allowed in a named simple type, and even then, only derivation by restriction is supported. Notably, the elements `list` and `union` are not supported.
- Facet definition is not cumulative. That is, if an element and its named type both define the same kind of facet, then the facet defined in the type is overridden by the local facet definition. However, this restriction does not apply to programmatic facets defined by the element `osd:constraint`. For `osd:constraint`, if an element and its named type both define a programmatic facet with different Java classes, the definition of these facets will be cumulative. Contrary to the XML Schema Specification, EBX® is not strict regarding the definition of a facet of the same kind in an element and its named type. That is, the value of a same kind of facet defined in an element is not checked according to the one defined in the named type. However, in the case of static enumerations defined both in an element and its type, the local enumeration will be replaced by the intersection between these enumerations.
- It is not possible to define different types of enumerations on both an element and its named type. For instance, you cannot specify a static enumeration in an element and a dynamic enumeration in its named type.
- It is not possible to simultaneously define a pattern facet in both an element and its named type.

85.3 XML Schema complex types

Complex types can be defined when designing a data model. A *named complex type* can be reused in the data model.

Restrictions:

- In the data model, only the element sequence is allowed. Notably, attribute definition is not supported.
- Type extensions are not supported in the current version of EBX®.

85.4 Extended simple types defined by EBX®

EBX® provides pre-defined simple data types:

XML Schema type	Java class
osd:text (xs:string restriction)	java.lang.String
osd:html (xs:string restriction)	java.lang.String
osd:email (xs:string restriction)	java.lang.String
osd:password (xs:string restriction)	java.lang.String
osd:color (xs:string restriction)	java.lang.String
osd:resource (xs:anyURI restriction)	internal class
osd:locale (xs:string restriction)	java.util.Locale
osd:dataspaceKey (xs:string restriction)	java.lang.String
osd:datasetName (xs:string restriction)	java.lang.String

The above types are defined by the internal schema `common-1.0.xsd`. They are defined as follows:

osd:text

This type represents textual information. Its default user interface in EBX® consists of a dedicated editor with several lines for input and display.

The use of this data type is intended for very long texts; this is why the 'Text' search strategy is associated with this type, and cannot be modified. As a consequence, fields of type `osd:text` cannot benefit from sorting; moreover, their indexing strategy and search results differ from those of the `xs:string` type. See [Default strategy for string fields](#) [p 317] for more information.

```
<xs:simpleType name="text">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

osd:html

This represents a character string with HTML formatting. A WYSIWYG editor is provided in EBX®.

```
<xs:simpleType name="html">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

osd:email

This represents an email address as specified by the [RFC822](#) standard.

```
<xs:simpleType name="email">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

osd:password

This represents a hashed or encrypted password. A specific editor is provided in EBX®.

```
<xs:element name="password" type="osd:password" />
```

The default editor performs a hash computation using the SHA-512 algorithm. This encryption function is also available from a Java client using the method `DirectoryDefault.encryptStringAPI`.

It is also possible for the default editor to use a different encryption mechanism by specifying a class that implements the interface `EncryptionAPI`.

```
<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>
```

It is possible to specify some salt by referencing a path to another field, and by using a class the implements the interface `HashComputationAPI`.

```
<xs:element name="password" type="osd:password">
  <xs:annotation>
    <xs:appinfo>
      <osd:uiBean class="com.orchestranetworks.ui.UIPassword">
        <encryptionClass>package.HashClassName</encryptionClass>
        <saltPath>../login</saltPath>
      </osd:uiBean>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

osd:locale

This represents a geographical, political or cultural location. The locale type is translated into Java by the class `java.util.Locale`.

```
<xs:simpleType name="locale">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ar" osd:label="Arabic" />
    <xs:enumeration value="ar_AE" osd:label="Arabic (United Arab Emirates)" />
    <xs:enumeration value="ar_BH" osd:label="Arabic (Bahrain)" />
    <xs:enumeration value="ar_DZ" osd:label="Arabic (Algeria)" />
    <xs:enumeration value="ar_EG" osd:label="Arabic (Egypt)" />
    <xs:enumeration value="ar_IQ" osd:label="Arabic (Iraq)" />
    ...
    <xs:enumeration value="vi_VN" osd:label="Vietnamese (Vietnam)" />
  >
  <xs:enumeration value="zh" osd:label="Chinese" />
  <xs:enumeration value="zh_CN" osd:label="Chinese (China)" />
  <xs:enumeration value="zh_HK" osd:label="Chinese (Hong Kong)" />
  <xs:enumeration value="zh_TW" osd:label="Chinese (Taiwan)" />
</xs:restriction>
</xs:simpleType>
```

osd:color

This represents a character string with hexadecimal RGB color formatting. A color picker `UIComponent` is provided in EBX®.

```
<xs:simpleType name="color">
  <xs:restriction base="xs:string" />
</xs:simpleType>
```

osd:resource

This represents a resource packaged in a module. For more information, see [Packaged resources](#) [p 516]. This type requires the definition of the facet [FacetOResource](#) [p 576].

```
<xs:simpleType name="resource">
  <xs:restriction base="xs:anyURI" />
</xs:simpleType>
```

osd:dataspaceKey

This type represents a reference to a dataspace.

```
<xs:element name="dataspaceField" type="osd:dataspaceKey" />
```

A specific editor is provided in EBX® that displays the dataspaces that can be referenced.

It is possible to specify the dataspaces that can be referenced using the element `osd:dataspaceSet` under `xs:annotation/`
`xs:appInfo`. If the element `osd:dataspaceSet` is not defined, then by default, only open branches can be referenced.

```
<xs:element name="dataspaceField" type="osd:dataspaceKey">
  <xs:annotation>
    <xs:appinfo>
```

```

<osd:dataspaceSet>
  <include>
    <pattern>a pattern</pattern>
    <type>all | branch | version</type>
    <includeDescendants>none | allDescendants |
allBranchDescendants | allSnapshotDescendants | branchChildren |
snapshotChildren</includeDescendants>
  </include>
  <exclude>
    <pattern>a pattern</pattern>
    <type>all | branch | version</type>
    <includeDescendants>none | allDescendants |
allBranchDescendants | allSnapshotDescendants | branchChildren |
snapshotChildren</includeDescendants>
  </exclude>
  <filter osd:class="com.foo.MyDataspaceFilter">
    <param1>...</param1>
    <param2>...</param2>
  </filter>
</osd:dataspaceSet>
</xs:appinfo>
</xs:annotation>
</xs:element>

```

- includes

Définit les espaces de données qui peuvent être référencés par ce champ. An include must at least be defined.

pattern: Définit une expression régulière à laquelle le nom d'un espace de données doit être conforme. This property is mandatory.

type: Définit le type d'espaces de données (branches ou versions) qui peuvent être référencés par ce champ. Si non définie, cette restriction s'appliquera aux espaces de données de type branche. If all then branches and snapshots are included. If branch then only branches are included. If snapshot then only snapshots are included. If not set, this property is branch by default.

includeDescendants: Specifies if children or descendants of the dataspace that match the specified pattern are included in the set. If none then neither children nor descendants of the dataspace that match the specified pattern are included. If allDescendants then all descendants of the dataspace that match the specified pattern are included. If allBranchDescendants then all descendant branches of the dataspace that match the specified pattern are included. If allSnapshotDescendants then all descendant snapshots of the dataspace that match the specified pattern are included. If directBranchChildren then only direct branches of the dataspace that match the specified pattern are included. If directSnapshotChildren then only direct snapshots of the dataspace that match the specified pattern are included. If not set, this property is none by default.

- excludes

Définit les espaces de données qui ne peuvent pas être référencés par ce champ. Les exclusions sont ignorées si la configuration ne définit pas d'inclusion.

pattern: Définit une expression régulière à laquelle le nom d'un espace de données doit être conforme. This property is mandatory.

type: Définit le type d'espaces de données (branches ou versions) qui peuvent être référencés par ce champ. Si non définie, cette restriction s'appliquera aux espaces de données de type branche. If all then branches and snapshots are excluded. If branch then only branches are excluded. If snapshot then only snapshots are excluded. If not set, this property is branch by default.

includeDescendants: Specifies if children or descendants of the datasets that match the specified pattern are excluded from the set. If none then neither children nor descendants of the dataspace that match the specified pattern are excluded. If allDescendants then all descendants of the dataspace that match the specified pattern are excluded. If allBranchDescendants then all descendant branches of the dataspace that match the specified pattern are excluded. If allSnapshotDescendants then all descendant snapshots of the dataspace that match the specified pattern are excluded. If directBranchChildren then only direct branches of the dataspace that match the specified pattern are excluded. If directSnapshotChildren then only direct snapshots of the dataspace that match the specified pattern are excluded. If not set, this property is none by default.

- filter

Définit un filtre pour accepter ou refuser des espaces de données dans le contexte d'un jeu de données ou d'un enregistrement. Ce filtre est uniquement utilisé par le composant de saisie dédié à ce type de champ. Ce filtre n'est pas utilisé lors de la validation de ce champ. Des contrôles additionnels peuvent être définis en utilisant une contrainte spécifique (composant).

The attribute `osd:class` specifies a Java bean that implements the interface `DataspaceSetFilterAPI`.

It is also possible to customize validation messages and the control policy associated with this type using the element validation under `xs:annotation/xs:appInfo/osd:dataspaceSet`. See [Facet validation message with severity](#) [p 594] and [Control policy](#) [p 581] for more information.

osd:datasetName

This type represents a reference to a dataset.

```
<xs:element name="dataset" type="osd:datasetName" />
```

A specific editor provided in EBX® displays the datasets that can be referenced.

It is also possible to specify the datasets that can be referenced using the element `osd:datasetSet` under `xs:annotation/xs:appInfo`:

```
<xs:element name="datasetField" type="osd:datasetName">
  <xs:annotation>
    <xs:appinfo>
      <osd:datasetSet>
        <branch>productsBranch</branch>
        <version>productsVersion</version>
        <dataspaceSelector>../dataspaceField</dataspaceSelector>
        <pattern>a pattern</pattern>
        <filter osd:class="com.foo.MyDatasetFilter">
          <param1>...</param1>
          <param2>...</param2>
        </filter>
      </osd:datasetSet>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

- **branch**

Spécifie la branche source. Seuls les jeux de données contenus dans cette branche pourront être référencés par un champ de type Identifiant de jeux de données (`osd:datasetName`).

- **version**

Spécifie la version source. Seuls les jeux de données contenus dans cette version pourront être référencés par un champ de type Identifiant de jeux de données (`osd:datasetName`).

- **dataspaceSelector**

Specifies a field in the same data model that defines the dataspace containing the datasets that can be referenced. The specified field must be of type `xs:string` or `osd:dataspaceKey`. The value of this field must comply with the representation of a persistent identifier of a dataspace or snapshot. See `HomeKey`. format^{API} for more information.

The referred node must respect the restrictions existing for dynamic facets, see [Dynamic constraints](#) [p 575].

- **includes**

Définit les jeux de données qui peuvent être référencés par ce champ.

pattern: Définit une expression régulière à laquelle le nom d'un jeu de données doit être conforme. This property is mandatory.

includeDescendants: Définit si les jeux de données enfants ou descendants sont inclus dans cet ensemble. Si non définie, cette restriction ne s'appliquera pas aux jeux de données enfants. If none then neither children nor descendants of the datasets that match the specified pattern are excluded. If directChildren then only direct children of the datasets that match the specified pattern are excluded. If allDescendants then all descendants of the datasets that match the specified pattern are excluded. If not set, this property is none by default.

- **excludes**

Définit les jeux de données qui ne peuvent pas être référencés par ce champ. Les exclusions sont ignorées si la configuration ne définit pas d'inclusion.

pattern: Définit une expression régulière à laquelle le nom d'un jeu de données doit être conforme. This property is mandatory.

includeDescendants: Définit si les jeux de données enfants ou descendants sont inclus dans cet ensemble. Si non définie, cette restriction ne s'appliquera pas aux jeux de données enfants. If none then neither children nor descendants of the datasets that match the specified pattern are excluded. If directChildren then only direct children of the datasets that match the specified pattern are excluded. If allDescendants then all descendants of the datasets that match the specified pattern are excluded. If not set, this property is none by default.

- **filter**

Définit un filtre pour accepter ou refuser des jeux de données dans le contexte d'un jeu de données ou d'un enregistrement. Ce filtre est uniquement utilisé par le composant de saisie dédié à ce type de champ. Ce filtre n'est pas utilisé lors de la validation de ce champ. Des contrôles additionnels peuvent être définis en utilisant une contrainte spécifique (composant).

The attribute `osd:class` specifies a Java bean that implements the interface `DatasetSetFilterAPI`. A validation message is added to the associated field if an input dataspace reference does not match this filter.

One of the elements `branch`, `version` or `dataspaceSelector` must be defined.

It is also possible to customize validation messages and the control policy associated with this type using the element `validation` under `xs:annotation/xs:appInfo/osd:datasetSet`. See [Facet validation message with](#)

[severity](#) [p 594] and [Control policy](#) [p 581] for more information.

85.5 Complex types defined by EBX®

EBX® provides pre-defined complex data types:

XML Schema type	Description
osd:UDA	User Defined Attribute: This type allows any user, according to their access rights, to define a value associated with an attribute defined in a dictionary called a UDA Catalog.
osd:UDACatalog	Catalog of User Defined Attributes: This type consists of a table in which attributes can be specified. This catalog is used by all osd:UDA elements declared in the same data model.

osd:UDA

A User Defined Attribute (UDA) supports both the minOccurs and maxOccurs attributes, as well as the attribute osd:UDACatalogPath, which specifies the path of the corresponding catalog.

```
<xs:element name="firstUDA" type="osd:UDA" minOccurs="0"
maxOccurs="unbounded" osd:UDACatalogPath="//insuranceCatalog" />
<xs:element name="secondUDA" type="osd:UDA" minOccurs="1"
maxOccurs="1"
osd:UDACatalogPath="/root/userCatalog" />
<xs:element name="thirdUDA" type="osd:UDA" minOccurs="0"
maxOccurs="1"
osd:UDACatalogPath="//userCatalog" />
```

In the manager, when working with a UDA, the editor will adapt itself to the type of the selected attribute.

osd:UDACatalog

Internally, a catalog is represented as a table. The parameters minOccurs and maxOccurs must be specified.

Several catalogs can be defined in the same data model.

```
<xs:element name="insuranceCatalog" type="osd:UDACatalog"
minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation xml:lang="en-US">Insurance Catalog.</
xs:documentation>
    <xs:documentation xml:lang="fr-FR">Catalog assurance.</
xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="userCatalog" type="osd:UDACatalog" minOccurs="0"
maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation xml:lang="en-US">User catalog.</
xs:documentation>
    <xs:documentation xml:lang="fr-FR">Catalogue utilisateur.</
xs:documentation>
  </xs:annotation>
</xs:element>
```

Only the following types are available for creating new attributes:

- xs:string
- xs:boolean
- xs:decimal

- xs:dateTime
- xs:time
- xs:date
- xs:anyURI
- xs:Name
- xs:int
- osd:html
- osd:email
- osd:password
- osd:locale
- osd:text

Restrictions on User Defined Attributes and Catalogs

The following features are unsupported on UDA elements:

- Facets
- Functions using the osd:function property
- UI bean editors using the osd:uiBean property
- The osd:checkNullInput property
- History features
- Replication
- Inheritance features, using the osd:inheritance property

As UDA catalogs are internally considered to be tables, the restrictions that apply to tables also exist for UDACatalog elements.

85.6 Aggregated lists

In XML Schema, the maximum number of times an element can occur is determined by the value of the maxOccurs attribute in its declaration. If this value is strictly greater than 1 or is unbounded, the data can have multiple occurrences. If no osd:table declaration is included, this element is called an *aggregated list*. In Java, it is then represented as an instance of the class `java.util.List`.

The following is an example of an aggregated list that defines the pricing of a loan product, depending on the amount borrowed.

```
<xs:element name="pricing" minOccurs="0" maxOccurs="unbounded"
  osd:access="RW">
  <xs:annotation>
    <xs:documentation>
      <osd:label>Pricing</osd:label>
      <osd:description>Pricing grid </osd:description>
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="amount" type="xs:int">
        <xs:annotation>
          <xs:documentation>
            <osd:label>Amount borrowed</osd:label>
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

</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 EBX®. This editor allows you to add or to delete occurrences.

Attention

The addition of an `osd:table` declaration to an element with `maxOccurs > 1` is a very important consideration that must be taken into account during the design process. An aggregated list is severely limited with respect to the many features that are supported by tables. Some features unsupported on aggregated lists that are supported on tables are:

- Performance and memory optimization;
- Lookups, filters and searches;
- Sorting, view and display in hierarchies;
- Identity constraints (primary keys and uniqueness constraints);
- Detailed permissions for creation, modification, deletion and particular permissions at the record level;
- Detailed comparison and merge.

Thus, *aggregated lists should be used only for small volumes of simple data (one or two dozen occurrences), with no advanced requirements*. For larger volumes of data or more advanced functionalities, it is strongly advised to use an `osd:table` declaration.

For more information on table declarations, see [Tables and relationships](#) [p 549].

85.7 Including external data models

Including another data model in your current model allows you to use the reusable types that are defined in that data model. You can thus use the inclusion of external data models to share data types between multiple XML Schema Documents.

To include another XML Schema Document in your model, thereby including the data types that it defines, specify the `xs:include` element as follows:

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="./schemaToInclude.xsd"/>
  ...
</xs:schema>

```

The attribute `schemaLocation` is mandatory and must specify either an absolute or 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.

EBX® includes extensions with specific URNs for including embedded data models and data models packaged in modules.

To include an embedded data model in a model, specify the URN defined by EBX®. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="urn:ebx:publication:myPublication"/>
  ...
</xs:schema>
```

To include a data model packaged in a module, specify the specific URN defined by EBX®. For example:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:osd="urn:ebx-schemas:common_1.0" xmlns:fmt="urn:ebx-schemas:format_1.0">
  <xs:include schemaLocation="urn:ebx:module:aModuleName:/WEB-INF/ebx/schema/myDataModel.xsd"/>
  ...
</xs:schema>
```

See SchemaLocation^{API} for more information about specific URNs supported by EBX®.

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.

CHAPITRE 86

Tables and relationships

Ce chapitre contient les sections suivantes :

1. [Tables](#)
2. [Foreign keys](#)
3. [Associations](#)
4. [Linked fields](#)

86.1 Tables

Overview

TIBCO EBX® supports the features of relational database tables, including the handling of large volumes of records, and identification by primary key.

Tables provide many benefits that are not offered by [aggregated lists](#) [p 545]. Beyond relational capabilities, some features that tables provide are:

- filters and searches;
- sorting, views and hierarchies;
- identity constraints: primary keys, [foreign keys](#) [p 554] and [uniqueness constraints](#) [p 573];
- specific permissions for creation, modification, and deletion;
- dynamic and contextual permissions at the individual record level;
- detailed comparison and merge;
- ability to have inheritance at the record level (see [dataset inheritance](#) [p 290]);
- performance and memory optimization.

Voir aussi

[Foreign keys](#) [p 554]

[Associations](#) [p 558]

[Linked fields](#) [p 568]

[Actions sur les jeux de données existants](#) [p 157]

[Vue tabulaire simple](#) [p 133]

[Vues hiérarchiques](#) [p 133]

[History](#) [p 269]

Declaration

A table element, which is an element with *maxOccurs* > 1, is declared by adding the following annotation:

```
<xs:annotation>
  <xs:appinfo>
    <osd:table>
      <primaryKeys>/pathToField1 /pathToField...n</primaryKeys>
    </osd:table>
  </xs:appinfo>
</xs:annotation>
```

Common properties

Element	Description	Required
primaryKeys	Specifies the primary key fields of the table. Each field of the primary key must be denoted by its absolute XPath notation that starts just under the root element of the table. If there are multiple fields in the primary key, the list is delimited by whitespace. Note: Whitespaces in primary keys of type <code>xs:string</code> are handled differently. See Whitespace handling for primary keys of type string [p 585].	Yes.
defaultLabel	Defines the end-user display of records. Multiple variants can be specified: - A static non-localized expression is defined using the <code>defaultLabel</code> element, for example: <pre><defaultLabel>Product: \${./productCode}</defaultLabel></pre> - Static localized expressions are specified using the <code>defaultLabel</code> element with the attribute <code>xml:lang</code>, for example: <pre><defaultLabel xml:lang="fr-FR">Produit : \${./productCode}</defaultLabel> <defaultLabel xml:lang="en-US">Product: \${./productCode}</defaultLabel></pre> - A JavaBean that implements the interface <code>UILabelRenderer^{API}</code> and/or the interface <code>UILabelRendererForHierarchy^{API}</code>. The JavaBean is specified by means of the attribute <code>osd:class</code>, for example: <pre><defaultLabel osd:class="com.wombat.MyLabel"></defaultLabel></pre> Attention Since the end-user display should be sortable, only fields that use sortable search strategies are allowed in static expressions. Note: The priority of the tags when displaying the user interface is the following: <code>defaultLabel</code> tags with a JavaBean (but it is not allowed to define several renderers of the same type); <code>defaultLabel</code> tags with a static localized expression using the <code>xml:lang</code> attribute; <code>defaultLabel</code> tags with a static non-localized expression. Attention: Les droits d'accès définis dans les jeux de données associés ne sont pas appliqués lors de l'affichage des libellés. Les champs habituellement cachés à cause de droits d'accès seront affichés dans les libellés des enregistrements.	No.
recordForm	Defines a specific component for customizing the record form in a dataset. This component is defined using a JavaBean that extends <code>UIForm^{API}</code> or implements <code>UserServiceRecordFormFactory^{API}</code>. The JavaBean is specified by means of the attribute <code>osd:class</code>, for example:	No.

Element	Description	Required
	<recordForm osd:class="com.wombat.MyRecordForm"/>	

Example

Below is an example of a product catalog:

```
<xs:element name="Products" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>
      <osd:label>Product Table </osd:label>
      <osd:description>List of products in Catalog </osd:description>
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:annotation>
      <xs:appinfo>
        <osd:table>
          <primaryKeys>./productRange /productCode</primaryKeys>
          <index name="indexProductCode">/productCode</index>
        </osd:table>
      </xs:appinfo>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="productRange" type="xs:string"/><!-- key -->
      <xs:element name="productCode" type="xs:string"/><!-- key -->
      <xs:element name="productLabel" type="xs:string"/>
      <xs:element name="productDescription" type="xs:string"/>
      <xs:element name="productWeight" type="xs:int"/>
      <xs:element name="productType" type="xs:string"/>
      <xs:element name="productCreationDate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="Catalogs" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>
      <osd:label>Catalog Table</osd:label>
      <osd:description>List of catalogs</osd:description>
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:annotation>
      <xs:appinfo>
        <osd:table>
          <primaryKeys>/catalogId</primaryKeys>
        </osd:table>
      </xs:appinfo>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="catalogId" type="xs:string"/><!-- key -->
      <xs:element name="catalogLabel" type="xs:string"/>
      <xs:element name="catalogDescription" type="xs:string"/>
      <xs:element name="catalogType" type="xs:string"/>
      <xs:element name="catalogPublicationDate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```


Properties related to dataset inheritance

The following properties are only valid in the context of dataset inheritance:

Element	Description	Required
onDelete- deleteOccultingChildren	Indique si lors de la suppression d'un enregistrement les enregistrements occultants doivent aussi être supprimés dans les jeux de données enfants. Valid values are: never or always.	No, default is never.
mayCreateRoot	Specifies whether root record creation is allowed. The expression must follow the syntax below. See definition modes [p 290].	No, default is always.
mayCreateOverwriting	Indique si un enregistrement peut être surchargé dans des jeux de données enfants. The expression must follow the syntax below. See definition modes [p 290].	No, default is always.
mayCreateOcculting	Indique si un enregistrement occultant peut être créé dans des jeux de données enfants. The expression must follow the syntax below. See definition modes [p 290].	No, default is always.
mayDuplicate	Indique si un enregistrement peut être dupliqué. The expression must follow the syntax below.	No, default is always.
mayDelete	Indique si un enregistrement peut être supprimé. The expression must follow the syntax below.	No, default is always.

The may... expressions specify when the action is possible, though the ultimate availability of the action also depends on the user access rights. The expressions have the following syntax:

expression ::= always | never | <condition>*

condition ::= [root:yes | root:no]

"always": the operation is "always" possible (but user rights may restrict this).

"never": the operation is never possible.

"root:yes": the operation is possible if the record is in a root instance.

"root:no": the operation is not possible if the record is in a root instance.

If the record does not define any specific conditions, the default is used.

Voir aussi [Dataset inheritance](#) [p 289]

Using toolbars

It is possible to define the toolbars to display in the user interface using the element `defaultView/toolbars` under `xs:annotation/appinfo/osd:table`. A toolbar allows to customize the buttons and menus to display when displaying a table view, a hierarchical view, or a record form.

The table below presents the elements that can be defined under `defaultView/toolbars`.

Element	Description	Required
<code>tabularViewTop</code>	Définit la barre d'outils à afficher en entête de la vue table par défaut.	No.
<code>tabularViewRow</code>	Définit la barre d'outils à afficher pour chaque ligne de la vue table par défaut.	No.
<code>recordTop</code>	Définit la barre d'outils à afficher en entête du formulaire d'un enregistrement.	No.
<code>hierarchyViewTop</code>	Définit la barre d'outils à afficher en entête de la hiérarchie par défaut de la table.	No.

Voir aussi [Toolbars](#) [p 607]

Example

Below is an example of custom toolbars used by a product catalog:

```
<xs:element name="Products" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:documentation>
      <osd:label>Product Table </osd:label>
      <osd:description>List of products in Catalog </osd:description>
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:annotation>
      <xs:appinfo>
        <osd:table>
          <primaryKeys>./productRange /productCode</primaryKeys>
          <defaultView>
            <toolbars>
              <tabularViewTop>toolbar_name_for_tabularViewTop</tabularViewTop>
              <tabularViewRow>toolbar_name_for_tabularViewRow</tabularViewRow>
              <recordTop>toolbar_name_for_recordTop</recordTop>
              <hierarchyViewTop>toolbar_name_for_hierarchyViewTop</hierarchyViewTop>
            </toolbars>
          </defaultView>
        </osd:table>
      </xs:appinfo>
    </xs:annotation>
  </xs:complexType>
</xs:element>
```

Note

If a toolbar does not exist or is not available for a specific location then no toolbar will be displayed in the user interface in the corresponding location.

86.2 Foreign keys

Declaration

A reference to a [table](#) [p 549] is defined using the extended facet `osd:tableRef`. The node holding the `osd:tableRef` declaration must be of type `xs:string`. At the instantiation, any value of the node identifies a record in the target table using its **primary key syntax** `PrimaryKey`.

syntax^{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 dataset that contains the target table.	Only if the dataspace element is defined. Otherwise, default is the current dataset.
branch	Reference of the dataspace that contains the container dataset.	No, default is the current dataspace or snapshot.
display	Custom display for presenting the selected foreign key in the current record and the sorted list of possible keys. Two variants can be specified, either pattern-based expressions, or a JavaBean if the needs are very specific: - Static expressions are specified using the <code>display</code> and <code>pattern</code> elements. These static expressions can be localized using the additional attribute <code>xml:lang</code> on the <code>pattern</code> element, for example: <pre> <display> <pattern>Product : \${./productCode}</pattern> <pattern xml:lang="fr-FR">Produit : \${./productCode}</pattern> <pattern xml:lang="en-US">Product: \${./productCode}</pattern> </display> </pre> Attention Since the display pattern should be sortable, only fields that use sortable search strategies are allowed. - A JavaBean that implements the interface <code>TableRefDisplay</code>^{API}. It is specified using the attribute <code>osd:class</code>. For example: <pre> <display osd:class="com.wombat.MyLabel"></display> </pre> Attention Quick search and sort operations in the user interface will use the raw value instead of the label of the records if a JavaBean is defined or if the target table defines a programmatic label <code>UILabelRenderer</code>^{API} for its records. A static expression must be defined to use in these operations the label of the records. It is not possible to define both variants on the same foreign key element. Attention: Les droits d'accès définis dans les jeux de données associés ne sont pas appliqués lors de l'affichage des libellés. Les champs habituellement cachés à cause de droits d'accès seront affichés dans les libellés des enregistrements.	No, if the <code>display</code> property is not specified, the table's record rendering (p 551) is used.

Element	Description	Required
filter	Specifies an additional constraint that filters the records of the target table. Two types of filters are available: - An XPath filter is an XPath predicate in the target table context. It is specified using the predicate element. For example: <pre><filter><predicate>type = \${../refType}</predicate></filter></pre> A localized validation message can be specified using the element <code>validationMessage</code>, which will be displayed to the end-user at the validation time if a record is not accepted by the filter. A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'. Each localized message variant is defined in a nested message element with its locale in an <code>xml:lang</code> attribute. To specify a default message for unsupported locales, define a message element with no <code>xml:lang</code> attribute. In the user interface, the XPath filter is applied to filter a table according to the value of a foreign key field. That is, if a foreign key field specifies an XPath filter in a data model, then it will be reused in the filter pane to restrict the set of values in the associated combo-box displayed in the filter pane. However, the predicate used by the filter pane will only take into account the non-contextual parts of the predicate. - A programmatic filter is a JavaBean that implements the interface <code>TableRefFilter^{API}</code>. It is specified using the attribute <code>osd:class</code>. For example: <pre><filter osd:class="com.wombat.MyFilter"></filter></pre> Additional validation messages can be specified during the setup of the programmatic filter. In the user interface, programmatic filters are not applied to filter the set of values in the associated combo-box displayed in the filter pane. However, it is possible to specify an additional XPath predicate that will be used in the filter pane of the user interface. This XPath predicate is specified during the setup of the programmatic filter using the method <code>TableRefFilterContext.setFilterForSearch^{API}</code>. Note The attributes <code>osd:class</code> and the property <code>predicate</code> cannot be set simultaneously. The validation search XPath functions are forbidden on a <code>tableRef</code> filter. Voir aussi JavaBean specifications Package <code>com.orchestranetworks.schema.JavaBeans^{API}</code> <code>JavaBeanVersion^{API}</code>	No.
validation	Specifies localized validation messages for the <code>osd:tableRef</code> and error management policy. A specific severity level can be defined in a nested <code>severity</code> element. The default severity is 'error'. An error management policy can be defined in a nested <code>blocksCommit</code> element. The error management policy that	No.

Element	Description	Required
	blocks all operations does not apply to filters. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected, and a validation error will be reported. Each localized message variant is defined in a nested message element with its locale in an <code>xml:lang</code> attribute. To specify a default message for unsupported locales, define a message element with no <code>xml:lang</code> attribute.	

Attention

You can create a dataset which has a foreign key to a container that does not exist in the repository. However, the content of this dataset will not be available until the container is created. After the creation of the container, a data model refresh is required to make the dataset available. When creating a dataset that refers to a container that does not yet exist, the following limitations apply:

- Triggers defined at the dataset level are not executed.
- Default values for fields that are not contained in tables are not initialized.
- During an archive import, it is not possible to create a dataset that refers to a container that does not exist.

Example

The example below specifies a foreign key in the 'Products' table to a record of the 'Catalogs' table.

```
<xs:element name="catalog_ref" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:tableRef>
          <tablePath>/root/Catalogs</tablePath>
          <display>
            <pattern xml:lang="en-US">Catalog: ${./catalogId}</pattern>
            <pattern xml:lang="fr-FR">Catalogue : ${./catalogId}</pattern>
          </display>
          <validation>
            <severity>error</severity>
            <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
            <message>A default error message</message>
            <message xml:lang="en-US">A localized error message</message>
            <message xml:lang="fr-FR">Un message d'erreur localisé</message>
          </validation>
        </osd:tableRef>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Voir aussi

[Table definition](#) [p 549]

Primary key syntax *PrimaryKey.syntax*^{API}

[Extraction of foreign keys \(XPath predicate syntax\)](#) [p 256]

[Associations](#) [p 558]

[View for advanced selection](#) [p 601]

SchemaNode.getFacetOnTableReference^{API}

SchemaFacetTableRef^{API}

86.3 Associations

Overview

An association provides an abstraction over an existing relationship in the data model, and allows an easy model-driven integration of *associated objects* in the user interface and in data services.

Several types of associations are supported:

- 'By foreign key' specifies the inverse relationship of an existing [foreign key field](#) [p 554].
- 'Over a link table' specifies a relationship based on an intermediate link table (such tables are often called "join tables"). This link table has to define two foreign keys, one referring to the 'source' table (the table holding the association element) and another one referring to the 'target' table.
- 'By an XPath predicate' specifies a relationship based on an XPath predicate.

For an association, it is also possible to:

- Filter associated objects by specifying an additional XPath filter.
- Configure a tabular view to define the fields that must be displayed in the associated table.
- Define how associated objects are to be rendered in forms.
- Hide/show associated objects in the data service 'select' operation. See [Hiding a field in Data Services](#) [p 600].
- Specify the minimum and maximum number of associated objects that are required.
- Add validation constraints using XPath predicates for restricting associated objects.

Voir aussi

`SchemaNode.getAssociationLink`^{API}

`SchemaNode.isAssociationNode`^{API}

`AssociationLink`^{API}

Declaration

Associations are defined in the data model using the XML Schema element `osd:association` under `xs:annotation/appInfo`.

Restrictions:

- An association must be a simple element of type `xs:string`.
- An association can only be defined inside a table.

Note

The "official" cardinality constraints (`minOccurs="0"` `maxOccurs="0"`) are required because, from an instance of XML Schema, the corresponding node is absent. In other words, an association has no value and is considered as a "virtual" element as far as XML and XML Schema is concerned.

The table below presents the elements that can be defined under `xs:annotation/appInfo/osd:association`.

Element	Description	Required
tableRefInverse	Defines the properties of an association that is the inverse relationship of a <i>foreign key</i>. The element <code>fieldToSource</code> defines the foreign key that refers to the source table of the association. The element <code>fieldToSource</code> is mandatory and must specify a foreign key field that refers to the table containing the association. The element <code>fieldToSource</code> can be defined on a foreign key list (<code>maxOccurs > 1</code>), in that case, the list should be declared as unique with a blocking uniqueness constraint (<code>onInsertUpdateOrDelete</code>). Voir aussi Blocking and non-blocking constraints [p 581] Uniqueness constraints [p 573]	Yes if the association is the inverse relationship of a <i>foreign key</i> , otherwise no.
linkTable	Defines the properties of an association over a link table. The element <code>table</code> specifies the link table used by the association. The element <code>table</code> is mandatory and must refer to an existing table. Important: In order to be used by an association, a link table must define a primary key that is composed of auto-incremented fields and/or the foreign key to the source or target table of the association. The element <code>fieldToSource</code> defines the foreign key that refers to the source table of the association. The element <code>fieldToSource</code> is mandatory and must specify a foreign key field that refers to the table containing the association. The element <code>fieldToTarget</code> defines the foreign key that refers to the target table of the association. The element <code>fieldToTarget</code> is mandatory and must specify a foreign key field.	Yes if the association is over a link table, otherwise no.
xpathLink	Defines the properties of an association that is based on an <i>XPath predicate</i>. The predicate element specifies the criteria of the association, relative to the current node. Examples: <code>/root/Products[catalog_ref = \${../catalogId}]</code> or <code>//Products[catalog_ref = \${../catalogId}]</code> or <code>../Products[catalog_ref = \${../catalogId}]</code>. The path to the predicate, for example <code>../Products</code>, specifies the target table of the association. This part of the path is resolved with respect to the current table. It is not possible to refer to a table using a relative path if the association targets a table in another dataset. If the association depends on fields of the source table, the XPath expression predicate must include references to the elements on which it depends	Yes if the association is based on an <i>XPath predicate</i> , otherwise no.

Element	Description	Required
	using the notation <code>\${<relative-path>}</code> where <code>relative-path</code> is a path that identifies the element relative to the association node. See EBX® XPath supported syntax [p 251]. Note The validation search XPath functions are forbidden on an XPath link.	
filter	Defines an XPath predicate to filter associated objects using the predicate element. For example: <pre><filter><predicate>type = \${../refType}</predicate></filter></pre> It is only possible to use fields from the source and the target tables when defining an XPath filter. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath filter. Error message on creation: in the user interface, the record creation is blocked when a user submits a new associated record that does not comply with the filter. The error message can be customized using the element <code>checkOnAssociatedRecordCreation/message</code>. Each localized message variant is defined in a nested message element with its locale in an <code>xml:lang</code> attribute. To specify a default message for unsupported locales, define a message element with no <code>xml:lang</code> attribute. See Examples [p 566] for more information on this property. Note The validation search XPath functions are forbidden for association filter.	No.
xpathFilter	Note: Deprecated. This property has been replaced by the property <code>filter</code>. Defines an XPath predicate to filter associated objects.	No.
recordForm	Defines a specific component for customizing the form of an associated record. This component is defined using a JavaBean that implements <code>UserServiceAssociationRecordFormFactory^{API}</code>. The JavaBean is specified by means of the attribute <code>osd:class</code>, for example: <pre><recordForm osd:class="com.wombat.MyRecordFormFactory"/></pre>	No.

It is possible to refer to another dataset. For that, the following properties must be defined either under the element `tableRefInverse`, `linkTable` or `xpathLink` depending on the type of the association:

Element	Description	Required
<code>schemaLocation</code>	Defines the data model containing the fields used by the association. The data model is defined using a specific URN that allows referring to embedded data models and data models packaged in modules. See <code>SchemaLocation^{APT}</code> for more information about specific URNs supported by EBX®.	Yes.
<code>dataSet</code>	Defines the dataset used by the association. This dataset must use the data model specified by the element <code>schemaLocation</code> .	Yes.
<code>dataSpace</code>	Defines the dataspace containing the dataset used by the association.	No.

Important:

- When creating a dataset, you can create a dataset that defines an association to a container that does not yet exist in the repository. However, the content of this dataset will not be available immediately upon creation. After the absent container is created, a data model refresh is required in order to make the dataset available. When creating a dataset that refers to a container that does not yet exist, the following limitations apply:
 - Triggers defined at the dataset level are not executed.
 - Default values on fields outside tables are not initialized.
 - During an archive import, it is not possible to create a dataset that refers to a container that does not exist.
- When using an association defined with a container that is in an explicitly defined dataspace, the container will be in severe error if the current dataspace does not match the association's container dataspace. For instance, let's consider a table in a dataspace "A" that has a foreign key to another table in a dataspace "B" and that the table in dataspace "B" has an association based on that foreign key. When creating a child dataspace of B, the association definition becomes inconsistent because it does not comply to the foreign key definition in A. The dataset in the child dataspace will then be in severe error.

User interface integration

It is possible to define how associated objects are to be rendered in forms, using the element `osd:defaultView/displayMode` under `xs:annotation/appinfo`.

Possible values are:

- `inline`, specifies that associated records are to be rendered in the form at the same position of the association in the data model.
- `tab`, specifies that associated records are to be rendered in a specific tab.
- `link`, specifies that associated records are to be rendered in a modal window.

By default, associated records are rendered `inline` if this property is not defined.

The following example specifies that associated objects are to be rendered inline in the form:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
      </osd:association>
      <osd:defaultView>
        <displayMode>inline</displayMode>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

The following example specifies that associated objects are to be rendered in a specific tab:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
      </osd:association>
      <osd:defaultView>
        <displayMode>tab</displayMode>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Using toolbars

It is possible to define the toolbars to display in the user interface using the element `osd:defaultView/toolbars` under `xs:annotation/appinfo`. A toolbar allows to customize the buttons and menus to display when displaying the tabular view of an association.

The table below presents the elements that can be defined under `osd:defaultView/toolbars`.

Element	Description	Required
tabularViewTop	Définit la barre d'outils à afficher en entête de la vue table par défaut de cette association.	No.
tabularViewRow	Définit la barre d'outils à afficher pour chaque ligne de la vue table par défaut de cette association.	No.

The following example shows how to use toolbars from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
      </osd:association>
      <osd:defaultView>
        <toolbars>
          <tabularViewTop>toolbar_name_for_tabularViewTop</tabularViewTop>
          <tabularViewRow>toolbar_name_for_tabularViewRow</tabularViewRow>
        </toolbars>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
```

</xs:element>

Note

It is only possible to use the toolbars defined in the data model containing the target table of the association. That is, if the target table of the association is defined in another data model, then it is only possible to reference a toolbar defined in this data model and not in the one holding the association.

Voir aussi [Toolbars](#) [p 607]

Customized view of associated objects

A specific tabular view can be specified to define the fields that must be displayed in the target table. If a tabular view is not defined, all columns that a user is allowed to view, according to the granted access rights, are displayed. A tabular view is defined using the element `osd:defaultView/tabularView` under `xs:annotation/appinfo`.

The table below shows the elements that can be defined under `osd:defaultView/tabularView`.

Element	Description	Required
column	Define a field of the target table to display. The specified path must be absolute from the target table and must refer to an existing field. Several column elements can be defined to specify the fields that are to be displayed.	No.
sort	Define a field that can be used to sort associated objects. Several sort elements can be defined to specify the fields that can be used to sort associated objects. The element <code>nodePath</code> defines the path of the field that can be used to sort associated objects. The element <code>isAscending</code> specifies whether the sort order is ascending (true) or descending (false).	No.

The following example shows how to define a tabular view from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
      </osd:association>
      <osd:defaultView>
        <tabularView>
          <column>/productRange</column>
          <column>/productCode</column>
          <column>/productLabel</column>
          <column>/productDescription</column>
          <sort>
            <nodePath>/productLabel</nodePath>
            <isAscending>true</isAscending>
          </sort>
        </tabularView>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Actions in the user interface

In the user interface, it is possible to perform the following actions:

- **Create:** it allows directly creating an object in the target table of the association. When a new object is created, it is automatically associated with the current record.
- **Duplicate:** allows to duplicate an object in the target table of the association. When a new object is created, it is automatically associated with the current record.
- **Associate:** associates an existing object with the current record. In the case of an association over a link table, a record in the link table is automatically created to materialize the link between the current record and the existing object.
- **Move:** associates the selected objects to a different record than the current one. In the case of an association over a link table, the previous link record is automatically deleted and a new record in the link table is automatically created to materialize the link between the selected objects and their new parent record.
- **Delete:** deletes selected associated objects in the target table of the association.
- **Detach:** breaks the semantic link between the current record and the selected associated objects. In the case of an association over a link table, the records in the link table are automatically deleted, to break the links between the current record and associated objects.

Note

The actions *associate* and *detach* are not available when the association is defined using an **XPath predicate** (element *xpathLink*).

Customized view for actions

A published view, tabular or hierarchical, can be specified to define how objects should be displayed when performing an action through the user interface. A published view is defined using the element `osd:defaultView/associationViews` under `xs:annotation/appinfo`.

The table below shows the elements that can be defined under `osd:defaultView/associationViews`.

Element	Description	Required
<code>viewForAssociateAction</code>	Define a published view to be used when displaying the objects in the target table to be associated with the current record. The specified view must be published and created upon the target table of the association.	No.
<code>viewForMoveAction</code>	Define a published view to be used when moving an associated object to another record of the current table. The specified view must be published and created upon the current table.	No.

The following example shows how to define views from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
      </osd:association>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

```

</osd:association>
<osd:defaultView>
  <associationViews>
    <viewForAssociateAction>view_name_for_catalogs</viewForAssociateAction>
    <viewForMoveAction>view_name_for_products</viewForMoveAction>
  </associationViews>
</osd:defaultView>
</xs:appinfo>
</xs:annotation>
</xs:element>

```

Validation

Some controls can be defined on associations, in order to restrict associated objects. These controls are defined under the element `osd:association`.

The table below presents the controls that can be defined under `xs:annotation/appInfo/osd:association`.

Element	Description	Required
<code>minOccurs</code>	Specifies the minimum number of associated objects that are required for this association. This minimum number is defined using the element value and must be a positive integer.	No, by default the minimum is not restricted.
<code>maxOccurs</code>	Specifies the maximum number of associated objects that are allowed for this association. This maximum number is defined by the element value and must be either a positive integer or the raw string unbounded which indicates that this maximum is not restricted. The maximum number of associated objects must be greater than the minimum number of associated objects.	No, by default the maximum is not restricted.
<code>constraint</code>	Defines an XPath predicate for restricting associated records. It is only possible to use fields from the source and the target table when defining an XPath predicate. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath predicate. In associated datasets, a validation message of the specified severity is added and displayed to the end-user at the validation time when an associated record does not comply with the specified constraint.	No.
<code>validation</code>	A validation message can be defined under the elements <code>minOccurs</code>, <code>maxOccurs</code> and <code>constraint</code>, using the element <code>validation</code>. The severity of the validation message is specified using the element <code>severity</code>. Possible severities are: <code>error</code>, <code>warning</code> and <code>info</code>. If the severity is not specified then, by default, the severity <code>error</code> is used. A localized validation message can be specified using the element <code>message</code>, which will be displayed to the end-user at the validation time if an association does not comply with this constraint. Each localized message variant is defined in a nested message element with its locale in an <code>xml:lang</code> attribute. To specify a default message for unsupported locales, define a message element with no <code>xml:lang</code> attribute.	No.

Data services integration

It is possible to define whether associated objects must be hidden in the Data service select operation. For this, the property `osd:defaultview/hiddenInDataServices` under `xs:annotation/xs:appinfo` can be set on the association. Setting the property to 'true' will hide associated objects in the Data service select operation. If this property is not defined then, by default, associated objects will be shown in the Data service select operation.

Voir aussi

[Hiding a field in Data Services](#) [p 600]

[Association field](#) [p 714]

Examples

For example, the product catalog data model defined [previously](#) [p 552] specifies that a product belongs to a catalog (explicitly defined by a foreign key in the 'Products' table). The reverse relationship (that a catalog has certain products) is not easily represented in XML Schema, unless the 'Catalogs' table includes the following association that is the inverse of a foreign key:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
      </osd:association>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

For an association over a link table, we can consider the previous example and bring some updates. For instance, the foreign key in the 'Products' table is deleted and the relation between a product and a catalog is redefined by a link table (named 'Catalogs_Products') that has a primary key composed of two foreign keys: one that refers to the 'Products' table (named 'productRef') and another to the 'Catalogs' table (named 'catalogRef'). The following example shows how to define an association over a link table from this new relationship:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <linkTable>
          <table>/root/Catalogs_Products</table>
          <fieldToSource>./catalogRef</fieldToSource>
          <fieldToTarget>./productRef</fieldToTarget>
        </linkTable>
      </osd:association>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

The following example shows an association that refers to a foreign key in another dataset. In this example, the 'Products' and 'Catalogs' tables are not in the same dataset:

```
<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <schemaLocation>urn:ebx:module:aModuleName:/WEB-INF/ebx/schema/products.xsd</schemaLocation>
          <dataSet>Products</dataSet>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
      </osd:association>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

```

</xs:appinfo>
</xs:annotation>
</xs:element>

```

The following example defines an XPath filter to associate only products of the 'Technology' type:

```

<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
        <filter>
          <predicate>./productType = 'Technology'</predicate>
          <checkOnAssociatedRecordCreation>
            <message>A default message</message>
            <message xml:lang="en-US">A localized message</message>
            <message xml:lang="fr-FR">Un message localisé</message>
          </checkOnAssociatedRecordCreation>
        </filter>
      </osd:association>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

The following example specifies the minimum number of products that are required for a catalog:

```

<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
        <minOccurs>
          <value>1</value>
          <validation>
            <severity>warning</severity>
            <message xml:lang="en-US">One product should at least be associated to this catalog.</message>
            <message xml:lang="fr-FR">Un produit doit au moins être associé à ce catalogue.</message>
          </validation>
        </minOccurs>
      </osd:association>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

The following example specifies that a catalog must contain at most ten products:

```

<xs:element name="products" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:association>
        <tableRefInverse>
          <fieldToSource>/root/Products/catalog_ref</fieldToSource>
        </tableRefInverse>
        <maxOccurs>
          <value>10</value>
          <validation>
            <severity>warning</severity>
            <message xml:lang="en-US">Too much products for this catalog.</message>
            <message xml:lang="fr-FR">Ce catalogue a trop de produits.</message>
          </validation>
        </maxOccurs>
      </osd:association>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

86.4 Linked fields

Overview

Linked fields provide the ability to simulate a multi-table view by aggregating the fields of a main table with some fields of another table over an existing relationship. It allows an easy model-driven integration of some fields from records that are referred by other ones using relationships.

Voir aussi *SchemaLinkedField*^{API}

Following relationships can be traversed:

- [Foreign key constraints](#) [p 554].
- [Associations 'By foreign key'](#) [p 558].

Important: these relationships must be "single valued relationships". That is, a record of the table that defines the relationship must refer to only one record in the target table of the relationship.

In details:

- A foreign key is considered as a single valued relationship if the foreign key field defines a *maxOccurs* equal to 1.
- An association by foreign key is considered as a single valued relationship if the referred foreign key field is the unique primary key field of the target table or if an uniqueness constraint is defined on this foreign key field.

Declaration

A linked field, which is an element with *minOccurs* = 0 and *maxOccurs* = 0, is declared by adding the following annotation:

```
<xs:element name="catalog_ref_label" minOccurs="0" maxOccurs="0" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:function linkedField="../../catalog_ref/catalogLabel"/>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Attribute *linkedField* defined in element *osd:function* defines a path composed of steps which refer to a single-valued relationship in the container table and of steps to a field in the target table of the relationship.

Important:

- If the path is absolute (starts with a "/") then it will be resolved from the container table node.
- If the path is relative then it will be resolved from the current field.

In the context of the product catalog data model defined [previously](#) [p 552], the field *catalog_ref_label* is added in the *Products* table. The first step *catalog_ref* refers to the foreign key that indicates that a product belongs to a catalog. The last step *catalogLabel* refers to the label of the referred catalog. This field belongs to the *Catalog* table that is the target table of the foreign key constraint.

Restrictions:

- Currently, linked fields only allow to target one level of single valued relationships. As a consequence:

- The single-valued relationship cannot be defined by another linked field.
- A linked field cannot target another linked field.
- A linked field must define properties *minOccurs* and *maxOccurs*. equal to 0.
- A linked field must define a data type that is compatible with the one of the target field.
- A linked field cannot be a part of the primary key of the container table.
- A linked field must target a terminal field.
- A linked field cannot target an aggregated list or a field under an aggregated list.
- A linked field cannot refer to a single-valued relationship that is also a linked field.
- Targeting a field which is a static enumeration with localized labels (defines *xs:enumeration* in the data model) has some limitations. That is, only the raw value of the linked field will be available for display and search, instead of the enumeration label of the target field. It is still possible to define the same static enumerations on both the linked field and its target to benefit of the labelling features for display and search.
- If a linked field and its target define both a foreign key constraint then these constraints must define exactly the same display pattern (property *osd:tableRef/display/pattern* in the data model). Otherwise the display pattern defined locally on the linked field will be ignored during a search operation. That is, the search operation works only the display pattern defined by the targeted foreign key field.

CHAPITRE 87

Constraints, triggers and functions

Facets allow you to define data constraints in your data models. TIBCO EBX® supports XML Schema constraining facets and provides extended and programmatic facets for advanced data controls.

Ce chapitre contient les sections suivantes :

1. [XML Schema supported facets](#)
2. [Extended facets](#)
3. [Programmatic facets](#)
4. [Control policy](#)

87.1 XML Schema supported facets

The tables below show the facets that are supported by different data types.

Key:

- **X** - Supported
- **1** - The `whiteSpace` facet can be defined, but is not interpreted by EBX®
- **2** - In XML Schema, boundary facets are not allowed on the type `string`. Nevertheless, EBX® allows such facets as extensions.
- **3** - The `osd:resource` type only supports the facet `FacetOResource`, which is required. See [Extended Facets](#) [p 575].

- **4** - `osd:dataspaceKey`, `osd:datasetName` and `osd:color` types do not support facets. Only [Programmatic constraints](#) [p 578] are supported on these types.

	length	minLength	max Length	pattern	enumeration	white Space
xs:string	X	X	X	X	X	1
xs:boolean				X		1
xs:decimal				X	X	1
xs:dateTime				X	X	1
xs:time				X	X	1
xs:date				X	X	1
xs:anyURI	X	X	X	X	X	1
xs:Name	X	X	X	X	X	1
xs:integer				X	X	1
osd:resource [p 536] ³						1
osd:dataspaceKey [p 538] ⁴						1
osd:datasetName [p 541] ⁴						1
osd:color [p 538] ⁴						1

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
xs:string			2	2	2	2
xs:boolean						
xs:decimal	X	X	X	X	X	X
xs:dateTime			X	X	X	X
xs:time			X	X	X	X
xs:date			X	X	X	X

	fraction Digits	total Digits	max Inclusive	max Exclusive	min Inclusive	min Exclusive
xs:anyURI						
xs:Name			2	2	2	2
xs:integer	X	X	X	X	X	X
osd:resource [p 536] ³						
osd:dataspaceKey [p 538] ⁴						
osd:datasetName [p 541] ⁴						
osd:color [p 538] ⁴						

Example:

```
<xs:element name="loanRate">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="4.5" />
      <xs:maxExclusive value="17.5" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Uniqueness constraint

It is possible to define a uniqueness constraint, using the standard XML Schema element [xs:unique](#). This constraint indicates that a value or a set of values has to be unique inside a table.

Example:

In the example below, a uniqueness constraint is defined on the 'publisher' table, for the target field 'name'. This means that no two records in the 'publisher' table can have the same name.

```
<xs:element name="publisher">
  ...
  <xs:complexType>
    <xs:sequence>
      ...
      <xs:element name="name" type="xs:string" />
      ...
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="uniqueName">
    <xs:annotation>
      <xs:appinfo>
        <osd:validation>
          <severity>error</severity>
          <message>Name must be unique in table.</message>
          <message xml:lang="en-US">Name must be unique in table.</message>
          <message xml:lang="fr-FR">Le nom doit être unique dans la table.</message>
        </osd:validation>
      </xs:appinfo>
    </xs:annotation>
    <xs:selector xpath="." />
    <xs:field xpath="name" />
  </xs:unique>
</xs:element>
```

A uniqueness constraint has to be defined within a table and has the following properties:

Property	Description	Mandatory
name attribute	Identifies the constraint in the data model.	Yes
xs:selector element	Indicates the table to which the uniqueness constraint applies using a restricted XPath expression ('..' is forbidden). It can also indicate an element within the table (without changing the meaning of the constraint).	Yes
xs:field element	Indicates the field in the context whose values must be unique, using a restricted XPath expression. It is possible to indicate that a set of values must be unique by defining multiple xs:field elements.	Yes

Note

Undefined values (null values) are ignored on uniqueness constraints applied to single fields. On multiple fields, undefined values are taken into account. That is, sets of values are considered as being duplicated if they have the same defined and undefined values.

Additional localized validation messages can be defined using the element `osd:validation` under the elements `annotation/appinfo`. If no custom validation messages are defined, a built-in validation message will be used.

The uniqueness constraint can also be applied on an simple aggregated list: in this case, each value of the list has to be unique in the scope of the list and not in the scope of the table.

Example:

In the example below, a uniqueness constraint is defined on the 'title' table, for the target field 'printedEditions'. This means that an edition can appear only once in the list.

```
<xs:element name="title">
  ...
  <xs:complexType>
    <xs:sequence>
      ...
      <xs:element name="printedEditions" type="xs:string" minOccurs="0" maxOccurs="5"/>
      ...
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="uniquePrintedEditions">
    <xs:annotation>
      <xs:appinfo>
        <osd:validation>
          <severity>error</severity>
          <message xml:lang="en-US">An edition must be referenced only once by this title</message>
          <message xml:lang="fr-FR">Une édition ne peut être référencée qu'une seule fois par ce livre</message>
        </osd:validation>
      </xs:appinfo>
    </xs:annotation>
    <xs:selector xpath="." />
    <xs:field xpath="printedEditions"/>
  </xs:unique>
</xs:element>
```

Limitations:

1. The target of the `xs:field` element must be in a table.
2. The uniqueness constraint does not apply to computed fields.
3. The uniqueness constraint cannot be applied on multiple fields that contains an aggregated list.

4. The uniqueness constraint cannot be applied on embedded lists.

Voir aussi *Uniqueness constraint in the Java API* `UniquenessConstraintAPI`

87.2 Extended facets

EBX® provides additional constraints that are not specified in XML Schema, but that are useful for managing master data.

In order to guarantee XML Schema conformance, these extended facets are defined under the element `annotation/appinfo/otherFacets`.

Foreign keys

EBX® allows to create a reference to an existing table by means of a specific facet. See [Foreign keys](#) [p 554] for more information.

Dynamic constraints

Dynamic constraint facets retain the semantics of XML Schema, but the value attribute is replaced with a path attribute that allows fetching the value from another element. The available dynamic constraints are:

- `length`
- `minLength`
- `maxLength`
- `maxInclusive`
- `maxExclusive`
- `minInclusive`
- `minExclusive`

Using these facets, the data model can be modified dynamically.

Example:

```
<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:minInclusive path="/domain/Loan/Pricing/AmountMini/amount" />
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

In this example, the boundary of the facet `minInclusive` is not statically defined. The value of the boundary comes from the node `/domain/Loan/Pricing/AmountMini/amount`.

Restrictions:

- Target field cannot be an aggregated list. That is, it cannot define `maxOccurs = 1`.
- Data type of the target field must be compatible with the facet. That is, it must be:
 - of type integer for facets `length`, `minLength` and `maxLength`.
 - compatible with the data type of the field holding the facet for facets `maxInclusive`, `maxExclusive`, `minInclusive` and `minExclusive`.

- Target field cannot be in a table if the field holding the facet is not in a table.
- Target field must be in the same table or outside a table if the field holding the facet is in a table.
- If the target field is under one or more aggregated lists, the field holding the facet must also be under these aggregated lists. That is: the field holding the facet must be in the same list occurrence as the target field, or in a parent occurrence, so that the target field refers to a single value, from an XPath perspective.

FacetOResource constraint

This facet must be defined for every definition using the type `osd:resource`, to specify the subset of available packaged resource files as an enumeration. For more information on this type, see [osd:resource type](#) [p 538]. It has the following attributes:

moduleName	Indicates, using an alias, the EBX® module that contains the resource. If the resource is contained in the current module, the alias must be preceded by "wbp". Otherwise, the alias must be one of the values defined in the element <code><dependencies></code> in the file <code>module.xml</code> .
resourceType	Définit le type de ressource parmi les valeurs suivantes : "Image", "JavaScript", "Feuille de style", "HTML".
relativePath	Précise dans quel répertoire sont contenues les ressources. Le répertoire utilisé doit être situé sous celui correspondant au type de ressource. Par exemple, pour une ressource de type "Image", le répertoire <code>www/common/images/</code> , situé au même niveau que le répertoire <code>WEB-INF/</code> du module cible, sera utilisé et le chemin relatif devra donc être défini à partir de celui-ci. De plus, si une ressource est située dans un répertoire internationalisé (<code>www/fr/</code> par exemple), elle sera utilisée uniquement si une ressource du même nom est définie dans le répertoire <code>www/common/</code> .

This facet has the same behavior as an enumeration facet: the values are collected by recursively listing all the files in the local path in the specified resource type directory in the specified module.

Example:

```
<xs:element name="promotion" type="osd:resource">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:FacetOResource osd:moduleName="wbp"
          osd:resourceType="ext-images" osd:relativePath="promotion/" />
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

For an overview of the standard directory structure of an EBX® module (Java EE web application), see [Module structure](#) [p 513].

Excluding values

excludeValue constraint

This facet verifies that a value is not the same as the specified excluded value.

In this example, the empty string is excluded from the allowed values.

Example:

```
<xs:element name="roleName">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:excludeValue value="">
          <osd:validation>
            <severity>error</severity>
            <message>Please select address role(s).</message>
          </osd:validation>
        </osd:excludeValue>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleType type="xs:string" />
</xs:element>
```

excludeSegment constraint

This facet verifies that a value is not included in a range of values. Boundaries are excluded.

Example:

In this example, values between 20000 and 20999 are not allowed.

```
<xs:element name="zipCode">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:excludeSegment minValue="20000" maxValue="20999">
          <osd:validation>
            <severity>error</severity>
            <message>Postal code not valid.</message>
          </osd:validation>
        </osd:excludeSegment>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleType type="xs:string" />
</xs:element>
```

Enumeration constraint defined using another node

Attention

This kind of constraint is obsolete. You should use [foreign key constraint](#) [p 554]. It has limitations, in particular Quick search and sort operations in the user interface will use the raw value of the field instead of the labels defined through the constraint.

By default, an enumeration facet is described statically in XML Schema.

The content of an enumeration facet can also be provided dynamically by a list of simple elements in the data model.

Example:

In this example, the content of an enumeration facet is sourced from the node CountryList.

```
<xs:annotation>
```

```
<xs:appinfo>
  <osd:otherFacets>
    <osd:enumeration osd:path="../../CountryList" />
  </osd:otherFacets>
</xs:appinfo>
</xs:annotation>
```

The referred node CountryList:

- Must be an aggregated list, that is, maxOccurs > 1.
- Must be a list of elements of the same type as the node with the enumeration facet.
- Must be a node outside a table if the node with the enumeration facet is not inside a table.
- Must be a node outside a table or in the same table as the node with the enumeration facet if the node with this enumeration is inside a table.
- If the target field is under one or more aggregated lists, the field holding the facet must also be under these aggregated lists. That is: the field holding the facet must be in the same list occurrence as the target field, or in a parent occurrence, so that the target field refers to a single value, from an XPath perspective.

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>
```

87.3 Programmatic facets

A programmatic constraint can be added to any XML element declaration for a simple type.

In order to guarantee XML Schema conformance, programmatic constraints are specified under the element annotation/appinfo/otherFacets.

Programmatic constraints

A programmatic constraint is defined by a Java class that implements the interface `ConstraintAPI`.

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:constraint class="com.foo.CheckAmount">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraint>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Voir aussi

JavaBean specifications Package `com.orchestranetworks.schema.JavaBeans`^{API}

`JavaBeanVersion`^{API}

Programmatic enumeration constraints

An enumeration constraint adds an ordered list of values to a basic programmatic constraint. This facet allows selecting a value from a list. It is defined by a Java class that implements the interface `ConstraintEnumeration`^{API}.

Example:

```
<xs:element name="amount">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:constraintEnumeration class="com.foo.CheckAmountInEnumeration">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraintEnumeration>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Attention

Quick search and sort operations in the user interface will use the raw value of the field instead of its label. Consider whether this enumeration can be replaced by a [foreign key constraint](#) [p 554] to a table that defines the same set of values.

Constraint on 'null' values

In some cases, a value is only mandatory if some conditions are satisfied, for example, if another field has a given value. In this case, the standard XML Schema attribute `minOccurs` is insufficient because it is static.

In order to check if a value is mandatory according to its context, the following requirements must be satisfied:

1. A programmatic constraint must be defined by a Java class (see above).
2. This class must implement the interface `ConstraintOnNull`^{API}.

3. The XML Schema cardinality attributes must specify that the element is optional (`minOccurs="0"` and `maxOccurs="1"`).

Note

By default, constraints on 'null' values are not checked upon user input. In order to enable a check at the input, the '[checkNullInput](#)' [property](#) [p 584] must be set. Also, if the element is terminal, the dataset 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>
```

Voir aussi

JavaBean specifications Package `com.orchestranetworks.schema.JavaBeans`^{API}

`JavaBeanVersion`^{API}

Constraints on table

A constraint on table is defined by a Java class that implements the interface `ConstraintOnTable`^{API}. It can only be defined on table nodes.

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <osd:table>
        <primaryKeys>/key</primaryKeys>
      </osd:table>
      <osd:otherFacets>
        <osd:constraint class="com.foo.checkTable">
          <param1>...</param1>
          <param...n>...</param...n>
        </osd:constraint>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
```

</xs:element>

Attention

For performance reasons, constraints on tables are only checked when getting the validation report of a dataset or table. This means that these constraints are not checked when updates, such as record insertions, deletions or modifications, occur on tables. However, the internal incremental validation framework will optimize the validation cost of these constraints if dependencies are defined. For more information, see [Validation performance](#) [p 322].

Voir aussi

JavaBean specifications Package `com.orchestranetworks.schema.JavaBeansAPI`

`JavaBeanVersionAPI`

87.4 Control policy

Blocking and non-blocking constraints

When an update in the repository is performed, and this update adds a validation error according to a given constraint, it is possible to specify whether the new error blocks the update (and cancels the transaction) or if it is considered as non-blocking (so that the update can be committed and the error

can be corrected later). The element `blocksCommit` within the element `osd:validation` allows this specification, with the following supported values:

<code>onInsertUpdateOrDelete</code>	Specifies that the constraint must always remain valid after an operation (dataset update, dataset deletion, record creation, update or deletion). In this case, any operation that would violate the constraint is rejected and the values remain unchanged. This is the default and mandatory policy for primary key constraints, data type conversion constraints (an integer or a date must be well-written) and also structural constraints in mapped tables.
<code>onUserSubmit - checkModifiedValues</code>	Specifies that the constraint must remain valid whenever a user modifies the associated value and submits a form. In this case, any form input that would violate the constraint is rejected and the values remain unchanged. This is the default policy for all blocking constraints mentioned in the previous case. For example, a foreign key constraint is by default not blocking (a record referred to by other records can be deleted, etc.), except in the context of a form submit.
<code>never</code>	Specifies that the constraint must never block operations. In this case, any operation that would violate the constraint is allowed. In the context of the user interface, this constraint does not block the form submission if the user sets a value that violates this constraint.

On foreign key constraints, the control policy that blocks all operations does not apply to filtered records. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected and a validation error occurs.

It is not possible to specify a control policy on structural constraints that are defined in mapped tables. That is, this property is not available for fixed length, maximum length, maximum number of digits, and decimal place constraints due to the validation policy of the underlying RDBMS blocking constraints.

This property does not apply to archive imports and when merging dataspace. That is, all blocking constraints, except structural constraints, are always disabled when importing archives and merging dataspace.

Voir aussi

[Facet validation message with severity](#) [p 594]

[Foreign keys](#) [p 554]

XML Schema facet

The control policy is described by the element `osd:validation` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minInclusive value="1000">
        <xs:annotation>
          <xs:appinfo>
            <osd:validation>
              <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
            </osd:validation>
          </xs:appinfo>
        </xs:annotation>
      </xs:minInclusive>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet

The control policy is described by the element `osd:enumerationValidation` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
  <xs:annotation>
    <xs:appinfo>
      <osd:enumerationValidation>
        <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
      </osd:enumerationValidation>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="0" osd:label="male" />
      <xs:enumeration value="1" osd:label="female" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

EBX® facet

The control policy is described by the element `osd:validation` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

The control policy with values `onInsertUpdateOrDelete` and `onUserSubmit-checkModifiedValues` is only available on `osd:excludeSegment`, `osd:excludeValue` and `osd:tableRef` EBX® facets.

The control policy with the value `never` can be defined on all EBX® facets. On programmatic constraints, the control policy with the value `never` can only be set directly during the setup of the corresponding constraint. See `ConstraintContext.setBlocksCommitToNeverAPI` and `ConstraintContextOnTable.setBlocksCommitToNeverAPI` in the Java API for more information.

Example:

```
<xs:element name="price" type="xs:decimal">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:minInclusive path="../../priceMin">
          <osd:validation>
            <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
          </osd:validation>
        </osd:minInclusive>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Check 'null' input

According to the EBX® default validation policy, in order to allow temporarily incomplete input, a mandatory element is not checked for completion upon user input. Rather, it is verified at the dataset validation only. If completion must be checked immediately upon user input, the element must additionally specify the attribute `osd:checkNullInput="true"`. This property is ignored if defined on an aggregated list (`maxOccurs > 1`).

Note

A value is mandatory if the data model specifies a mandatory element, either statically, using `minOccurs="1"`, or dynamically, using a constraint on 'null'. For terminal elements, mandatory values are only checked for an activated dataset. For non-terminal elements, the dataset does not need to be activated.

Example:

```
<xs:element name="amount" osd:checkNullInput="true" minOccurs="1">
  ...
</xs:element>
```

Voir aussi

[Constraint on 'null'](#) [p 579]

[Whitespace management](#) [p 584]

[Empty string management](#) [p 586]

EBX® whitespace management for data types

According to XML Schema (see <https://www.w3.org/TR/xmlschema-2/#rf-whiteSpace>), whitespace handling must follow one of the procedures *preserve*, *replace* or *collapse*:

preserve	No normalization is performed, the value is unchanged.
replace	All occurrences of #x9 (tab), #xA (line feed) and #xD (carriage return) are replaced with #x20 (space).
collapse	After the processing according to the replace procedure, contiguous sequences of #x20 are then collapsed to a single #x20, and any leading or trailing #x20s are removed.

General whitespace handling

EBX® complies with the XML Schema recommendation:

- For fields of type `xs:string`, whether a primary key element or not, whitespaces are always preserved and an empty string is never converted to `null`.

- For other fields (non-`xs:string` type), whitespaces are always collapsed and empty strings are converted to `null`.

Attention

Exceptions:

- For fields of type `osd:html` or `osd:password`, whitespaces are always preserved and empty strings are converted to `null`.
- For fields of type `xs:string` that define the property `osd:checkNullInput="true"`, an empty string is interpreted as `null` at user input by EBX®.

Whitespace handling upon user input

The rules described in the previous section are applied in the user interface, but leading and trailing whitespaces are removed upon user input. That is, in the user interface, whitespaces are by default always trimmed upon user input. Other input methods (Import XML/CSV, Data services, API updates) are not trimmed from the user interface.

Attention

Exceptions:

- For fields of type `osd:password`, whitespaces are not trimmed upon user input.
- For foreign key fields, whitespaces are not trimmed upon user input.

It is possible to indicate in a data model that whitespaces should not be trimmed upon user input. The attribute `osd:trim="disable"` can be set on the fields that allow leading and trailing whitespaces upon user input.

Example:

```
<xs:element name="field" osd:trim="disable" type="xs:string">
  ...
</xs:element>
```

Whitespace handling for primary keys of type string

For primary key columns of type `xs:string`, a default EBX® constraint is defined. This constraint forbids empty strings and non-collapsed whitespace values when creating a record. That is, any record creation that would violate this constraint is rejected.

However, if the primary key node specifies its own `xs:pattern` facet, this facet overrides the default EBX® constraint. For example, the specific pattern `".*"` would accept any string, although this is not recommended.

The default constraint allows handling certain ambiguities. For example, it would be difficult for a user to distinguish between the following strings: `"12 34"` and `"12 34"`. For generic values, this would not create conflicts, however, errors would occur for primary keys.

Voir aussi [Tables and relationships](#) [p 549]

Empty string management

Default conversion

For nodes of type `xs:string`, no distinction is made at user input between an empty string and a `null` value. That is, an empty string value is automatically converted to `null` at user input.

Distinction between empty strings and 'null' value

There are certain cases where the distinction is made between an empty string and the `null` value, such as when:

- A primary key defines a pattern that allows empty strings.
- An element defines a foreign key constraint and the target table has a single primary key defining a pattern that allows empty strings.
- An element defines a static enumeration that contains an empty string.
- An element defines a dynamic enumeration to another element with one of the aforementioned cases.

If the distinction is made between an empty string and a `null` value, this implies the following behaviors:

- An empty string will not be converted to `null` at user input,
- Input fields for nodes of type `xs:string` will display an additional button for setting the value of the node to `null`,
- At validation time, an empty string will be considered to be a compliant value with regard to the `minOccurs="1"` property.

Validation message threshold

It is possible to specify at the data model level the maximum number of validation messages allowed per constraint when performing a validation.

Example:

```
<xs:schema ...>
  ...
  <xs:annotation>
    <xs:appinfo>
      <osd:validation>
        <validationMessageThreshold>250</validationMessageThreshold>
      </osd:validation>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

The threshold is considered for each constraint defined in a data model and in each dataset validation report. When the threshold is reached by a constraint, the validation of the constraint is stopped and an error message indicating that the threshold has been reached is added to the validation report.

The validation message threshold is set by default to 1000 if it is not defined in the data model. It is not allowed to set an unlimited number of validation messages. Also, the specified validation message threshold must be greater or equal than 100.

Voir aussi `ValidationReport.hasConstraintsWithTooManyMessages`^{API}

CHAPITRE 88

Triggers and functions

EBX® data model allows to define triggers and computed fields. It also provides auto-incremented fields

Ce chapitre contient les sections suivantes :

1. [Computed values](#)
2. [Triggers](#)
3. [Auto-incremented values](#)

88.1 Computed values

By default, data is read and persisted in the XML repository. Nevertheless, data may be the result of a computation and/or external database access, for example, an RDBMS or a central system.

EBX® allows taking into account other data in the current dataset context.

This is made possible by defining *computation rules*.

A computation rule is specified in the data model using the `osd:function` element (see example below).

- The value of the *class* attribute must be the qualified name of a Java class that implements the Java interface `ValueFunctionAPI`
- Additional parameters may be specified at the data model level, in which case the JavaBean convention is applied.

Example:

```
<xs:element name="computedValue">
  <xs:annotation>
    <xs:appinfo>
      <osd:function class="com.foo.ComputeValue">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:function>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Disabling validation

In some cases, it can be useful to disable the validation of computed values if the execution of a function is time-consuming. Indeed, if the function is attached to a table with N records, then it will

be called N times when validating this table. The property `osd:disableValidation= "true"` specified in the data model allows to disable the validation of a computed value (see example below).

Example:

```
<xs:element name="computedValue" osd:disableValidation="true">
  <xs:annotation>
    <xs:appinfo>
      <osd:function class="com.foo.ComputeValue">
        ...
      </osd:function>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

88.2 Triggers

Datasets or table records can be associated with methods that are automatically executed when some operations are performed, such as creations, updates, or deletions.

In the data model, these triggers must be declared under the `annotation/appinfo` element using the `osd:trigger` element.

Trigger on dataset

For dataset triggers, a Java class that extends the abstract class `InstanceTriggerAPI` must be declared inside the element `osd:trigger`.

In the case of dataset triggers, it is advised to define `annotation/appinfo/osd:trigger` tags just under the root element of the data model.

Example:

```
<xs:element name="root" osd:access="--">
  ...
  <xs:annotation>
    <xs:appinfo>
      <osd:trigger class="com.foo.MyInstanceTrigger">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:trigger>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:element>
```

Trigger on table

For the definition of table record triggers, a Java class that extends the abstract class `TableTriggerAPI` must be defined inside the `osd:trigger` element. It is advised to define the `annotation/appinfo/osd:trigger` elements just under the element describing the associated table or table type.

Examples:

On a table element:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <osd:table>
        <primaryKey>/key</primaryKey>
      </osd:table>
      <osd:trigger class="com.foo.MyTableTrigger" />
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

On a table type element:

```
<xs:complexType name="MyTableType">
  ...
  <xs:annotation>
    <xs:appinfo>
      <osd:trigger class="com.foo.MyTableTrigger">
        <param1>...</param1>
        <param...n>...</param...n>
      </osd:trigger>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:complexType>
```

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol. In the example above, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

88.3 Auto-incremented values

It is possible to define auto-incremented values. Auto-incremented values are only allowed inside tables, and they must be of the type `xs:int` or `xs:integer`.

An auto-increment is specified in the data model using the element `osd:autoIncrement` under the element `annotation/appinfo`.

Example:

```
<xs:element name="autoIncrementedValue" type="xs:int">
  <xs:annotation>
    <xs:appinfo>
      <osd:autoIncrement />
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Also, there are two optional elements, `start` and `step`:

- The `start` attribute specifies the first value for this auto-increment. If this attribute is not specified, then the value 1 is set by default.
- The `step` attribute specifies the step for the next value to be generated by the auto-increment. If this attribute is not specified, then the value 1 is set by default.

Example:

```
<xs:element name="autoIncrementedValue" type="xs:int">
  <xs:annotation>
    <xs:appinfo>
      <osd:autoIncrement>
        <start>100</start>
        <step>5</step>
      </osd:autoIncrement>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

A field specifying an `osd:autoIncrement` has the following behavior:

- The computation and allocation of the field value are performed whenever a new record is inserted and the field value is undefined.
- No allocation is performed if a programmatic insertion already specifies a non-null value. For example, if an archive import or an XML import specifies the value, that value is preserved.

Consequently, the allocation is not performed for a record insertion in occulting or overwriting modes.

- A newly allocated value is, whenever possible, unique in the scope of the repository. More precisely, the uniqueness of the allocation spans over all the datasets of the data model, and it also spans over all the dataspace. The latter case allows the merge of a dataspace into its parent with a reasonable guarantee that there will be no conflict if the `osd:autoIncrement` is part of the records' primary key.

This principle has a very specific limitation: when a mass update transaction that specifies values is performed at the same time as a transaction that allocates a value on the same field, it is possible that the latter transaction will allocate a value that will be set by the first transaction (there is no locking between different dataspace).

Internally, the auto-increment value is stored in the 'Auto-increments' table of the repository. In the user interface, it can be accessed by administrators in the 'Administration' area. This field is automatically updated so that it defines the greatest value ever set on the associated `osd:autoIncrement` field, in any instance or dataspace in the repository. This value is computed, taking into account the max value found in the table being updated.

In certain cases, for example when multiple environments have to be managed (development, test, production), each with different auto-increment ranges, it may be required to avoid this "max value" check. This particular behavior can be achieved using the `disableMaxTableCheck` property. It is generally not recommended to enable this property unless it is absolutely necessary, as this could generate conflicts in the auto-increment values. However, this property can be set in the following ways:

- Locally, by setting a parameter element in the auto-increment declaration:
`<disableMaxTableCheck>true</disableMaxTableCheck>`,
- For the whole data model, by setting `<osd:autoIncrement disableMaxTableCheck="true"/>` in the element `xs:appinfo` of the data model declaration, or
- Globally, by setting the property `ebx.autoIncrement.disableMaxTableCheck=true` in the EBX® main configuration file.

See [TIBCO EBX® main configuration file](#) [p 371].

Note

When this option is enabled globally, it becomes possible to create records in the table of auto-increments, for example by importing from XML or CSV. If this option is not selected, creating records in the table of auto-increments is prohibited to ensure the integrity of the repository.

CHAPITRE 89

Labels and messages

TIBCO EBX® allows to have custom labels and error messages for data models to be displayed in the interface.

Ce chapitre contient les sections suivantes :

1. [Label and description](#)
2. [Enumeration labels](#)
3. [Mandatory error message \(osd:mandatoryErrorMessage\)](#)
4. [Conversion error message](#)
5. [Facet validation message with severity](#)

89.1 Label and description

A label and a description can be added to each node in an adaptation model.

In EBX®, each adaptation node is displayed with its label. If no label is defined, the name of the element is used.

Two different notations can be used:

Full	The label and description are defined by the elements <code><osd:label></code> and <code><osd:description></code> respectively.
Simple	The label is extracted from the text content, ending at the first period ('.'), with a maximum of 60 characters. The description uses the remainder of the text.

The description may also have a hyperlink, either a standard HTML `href` to an external document, or a link to another node of the adaptation within EBX®.

- When using the `href` notation or any other HTML, it must be properly escaped.
- EBX® link notation is not escaped and must specify the path of the target, for example:
`<osd:link path="../../../misc1">Link to another node in the adaptation</osd:link>`

Example:

```
<xs:element name="misc1" type="xs:string">
  <xs:annotation>
    <xs:documentation>
```

```

Miscellaneous 1. This is the description of miscellaneous element #1.
Click <a href="https://www.tibco.com" target="_blank">here</a>
to learn more.
</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="misc2" type="xs:string">
  <xs:annotation>
    <xs:documentation>
      <osd:label>
        Miscellaneous 2
      </osd:label>
      <osd:description>
        This is the miscellaneous element #2 and here is a
        <osd:link path="../misc1"> link to another node in the
        adaptation</osd:link>.
      </osd:description>
    </xs:documentation>
  </xs:annotation>
</xs:element>

```

If a node points to a named type, then the label of the node replaces the label of the named type. The same mechanism applies to the description of the node (element `osd:description`).

Note

Regarding whitespace management, the label of a node is always *collapsed* when displayed. That is, contiguous sequences of blanks are collapsed to a single blank, and leading and trailing blanks are removed. In descriptions, however, whitespaces are always *preserved*.

Dynamic labels and descriptions

As an alternative to statically defining the localized labels and descriptions for each node, it is possible to specify a Java class that programmatically determines the labels and descriptions for the nodes of the data model. To define the class, include the element `osd:documentation`, with the attribute `class` in the data model. It is possible to pass JavaBean properties using nested parameter elements.

Example:

```

<xs:schema ...>
  <xs:annotation>
    <xs:appinfo>
      <osd:documentation class="com.foo.MySchemaDocumentation">
        <param1>...</param1>
        <param2>...</param2>
      </osd:documentation>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema ...>

```

The labels and descriptions that are provided programmatically take precedence over the ones defined locally on individual nodes.

Voir aussi *SchemaDocumentation*^{APT}

89.2 Enumeration labels

In an enumeration, a simple, non-localized label can be added to each enumeration element, using the attribute `osd:label`.

Attention

Labels defined for an enumeration element are always collapsed when displayed.

Example:

```
<xs:element name="Service" maxOccurs="unbounded">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="1" osd:label="Blue" />
      <xs:enumeration value="2" osd:label="Red" />
      <xs:enumeration value="3" osd:label="White" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

It is also possible to fully localize the labels using the standard `xs:documentation` element. If both non-localized and localized labels are added to an enumeration element, the non-localized label will be displayed in any locale that does not have a label defined.

Example:

```
<xs:element name="access" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="readOnly">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            read only
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            lecture seule
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="readWrite">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            read/write
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            lecture écriture
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="hidden">
        <xs:annotation>
          <xs:documentation xml:lang="en-US">
            hidden
          </xs:documentation>
          <xs:documentation xml:lang="fr-FR">
            masqué
          </xs:documentation>
        </xs:annotation>
      </xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

89.3 Mandatory error message (`osd:mandatoryErrorMessage`)

If the node specifies the attribute `minOccurs="1"` (default behavior), then an error message, which must be provided, is displayed if the user does not complete the field. This error message can be defined specifically for each node using the element `osd:mandatoryErrorMessage`.

Example:

```
<xs:element name="birthDate" type="xs:date">
  <xs:annotation>
    <xs:documentation>
      <osd:mandatoryErrorMessage>
        Please give your birth date.
      </osd:mandatoryErrorMessage>
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

The mandatory error message can be localized:

```
<xs:documentation>
  <osd:mandatoryErrorMessage xml:lang="en-US">
    Name is mandatory
  </osd:mandatoryErrorMessage>
  <osd:mandatoryErrorMessage xml:lang="fr-FR">
    Nom est obligatoire
  </osd:mandatoryErrorMessage>
</xs:documentation>
```

Note

Regarding whitespace management, the enumeration labels are always *collapsed* when displayed.

89.4 Conversion error message

For each predefined XML Schema element, it is possible to define a specific error message if the user entry has an incorrect format.

Example:

```
<xs:element name="email" type="xs:string">
  <xs:annotation>
    <xs:documentation>
      <osd:ConversionErrorMessage xml:lang="en-US">
        Please enter a valid email address.
      </osd:ConversionErrorMessage>
      <osd:ConversionErrorMessage xml:lang="fr-FR">
        Saisissez un e-mail valide.
      </osd:ConversionErrorMessage>
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

89.5 Facet validation message with severity

The validation message that is displayed when the value of a field does not comply with a constraint can define a custom severity, a default non-localized message, and localized message variants. If no severity is specified, the default level is error. Blocking constraints *must* have the severity error.

XML Schema facet (*osd:validation*)

The validation message is described by the element `osd:validation` in `annotation/appinfo` under the definition of the facet.

Example:

```
<xs:element name="zipCode">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <!--facet is not localized, but validation message is localized-->
      <xs:minInclusive value="01000">
        <xs:annotation>
          <xs:appinfo>
            <osd:validation>
              <severity>error</severity>
              <message>Non-localized message.</message>
              <message xml:lang="en-US">English error message.</message>
              <message xml:lang="fr-FR">Message d'erreur en français.</message>
            </osd:validation>
          </xs:appinfo>
        </xs:annotation>
      </xs:minInclusive>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML Schema enumeration facet (osd:enumerationValidation)

The validation message is described by the element `osd:enumerationValidation` in `annotation/appinfo` under the definition of the field.

Example:

```
<xs:element name="Gender">
  <xs:annotation>
    <xs:appinfo>
      <osd:enumerationValidation>
        <severity>error</severity>
        <message>Non-localized message.</message>
        <message xml:lang="en-US">English error message.</message>
        <message xml:lang="fr-FR">Message d'erreur en français.</message>
      </osd:enumerationValidation>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="0" osd:label="male" />
      <xs:enumeration value="1" osd:label="female" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

EBX® facet (osd:validation)

The validation message is described by the element `osd:validation` under the definition of the facet (which is defined in `annotation/appinfo/otherFacets`).

Example:

```
<xs:element name="price" type="xs:decimal">
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:minInclusive path="../priceMin">
          <osd:validation>
            <severity>error</severity>
            <message>Non-localized message.</message>
            <message xml:lang="en-US">English error message.</message>
            <message xml:lang="fr-FR">Message d'erreur en français.</message>
          </osd:validation>
        </osd:minInclusive>
      </osd:otherFacets>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```


CHAPITRE 90

Additional properties

Ce chapitre contient les sections suivantes :

1. [Default values](#)
2. [Access properties](#)
3. [Information](#)
4. [Default view](#)
5. [Comparison mode](#)
6. [Apply last modifications policy](#)
7. [Categories](#)

90.1 Default values

In a data model, it is possible to specify a default value for a field using the attribute `default`. This property is used to assign a default value if no value is defined for a field.

The default value is displayed in the user input field at the creation time. That is, the default value will be displayed when creating a new record or adding a new occurrence to an aggregated list.

Example:

In this example, the element specifies a default string value.

```
<xs:element name="fieldwithDefaultValue" type="xs:string" default="aDefaultValue" />
```

90.2 Access properties

The attribute `osd:access` defines the access mode, that is, whether the data of a particular data model node can be read and/or written. This attribute must have one of the following values: `RW`, `R-`, `CC` or `--`.

For each XML Schema node, three types of adaptation are possible:

1. *Adaptation terminal node*

This node is displayed with an associated value in TIBCO EBX®. When accessed using the method `Adaptation.get()`, it uses the adaptation search algorithm.

2. *Adaptation non-terminal node*

This node is a complex type that is only displayed in EBX® if it has one child node that is also an adaptation terminal node. It has no value of its own. When accessed using the method `Adaptation.get()`, it returns `null`.

3. Non-adaptable node

This node is not an adaptation terminal node and has no child adaptation terminal nodes. This node is never displayed in EBX®. When accessing using the method `Adaptation.get()`, it returns the node default value if one is defined, otherwise it returns `null`.

Voir aussi *Adaptation*^{API}

Access mode	Behavior
RW	<i>Adaptation terminal node</i> : value can be read and written in EBX®.
R-	<i>Adaptation terminal node</i> : value can only be read in EBX®.
CC	<i>Cut</i> : This is not an adaptation terminal node and none of its children are adaptation terminal nodes. This "instruction" has priority over any child node regardless of the value of their <i>access</i> attribute.
--	If the node is a simple type, it is not adaptable. If the node is a complex type, it is not an adaptation terminal node and does not define any child nodes. The root node of a data model must specify this access mode.
Default	If the <i>access</i> attribute is not defined: • If the node is a computed value, it is considered to be R- • If the node is a simple type and its value is not computed, it is considered to be RW • If the node is an aggregated list, it is then a terminal value and is considered to be RW • Otherwise, it is not an adaptation terminal node and it does not define anything about its child nodes.

Example:

In this example, the element is adaptable because it is an adaptation terminal node.

```
<xs:element name="proxyIpAddress" type="xs:string" osd:access="RW"/>
```

90.3 Information

The element `osd:information` allows specifying additional information. This information can then be used by the integration code, for any purpose, by calling the method `SchemaNode.getInformation`^{API}.

Example:

```
<xs:element name="misc" type="xs:string">
  <xs:annotation>
    <xs:appinfo>
      <osd:information>
        This is the text information of miscellaneous element.
      </osd:information>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

```
</xs:annotation>
</xs:element>
```

90.4 Default view

Hiding a field or a table in the default view

It is possible for a table or field inside a table to be hidden by default in EBX® by using the element `osd:defaultView/hidden`. This property is used to hide elements from the default view of a dataset without defining specific access permissions. That is, elements hidden by default will not be visible in any default forms and views, whether tabular or hierarchical, generated from the structure of the associated data model.

Attention

- If an element is configured to be hidden in the default view of a dataset, 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 dataset by defining a view. Only in this case will the access permissions associated with this field be evaluated to determine whether the field will be displayed or not.
- It is not possible to display a table that is hidden in the default view of a dataset (in the navigation pane).

Example:

In this example, the element is hidden in the default view of a dataset.

```
<xs:element name="hiddenField" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:defaultView>
        <hidden>true</hidden>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Hiding groups and fields in views

It is possible for a field or a group to be hidden in all views of a table by using the element `osd:defaultView/hiddenInViews`. This property is used to hide elements from the tabular (including the default tabular view) and hierarchical views of a dataset without defining specific access permissions. That is, hidden elements will not be visible in any views, whether tabular or hierarchical, created from the structure of the associated data model. However, hidden elements in views will be displayed in forms.

To specify whether or not to hide an element in all views, use the `osd:defaultView/hiddenInViews="true|false"` element.

If this property is set to `true`, then the element will not be selectable when creating a custom view. As a consequence, the element will not be displayed in all views of a table in a dataset.

If a group is configured as hidden in views, then all the fields nested under this group will not be displayed respectively in the views of the table.

Hiding a field in structured search tools

To specify whether or not to hide an element in structured search tools, use the element `osd:defaultView/hiddenInSearch="true|false|textSearchOnly"`.

If this property is set to `true`, then the field will not be selectable in the text and typed search tools of a dataset.

If this property is set to `textSearchOnly`, then the field will not be selectable only in the text search of a dataset but will be selectable in the typed search.

Note

If a group is configured as hidden in search tools or only in the text search, then all the fields nested under this group will not be displayed respectively in the search tools or only in the text search.

In all cases, the field will remain searchable in the quick search tool. A field can be excluded from all search tools, including the quick search, by defining a specific search strategy.

Voir aussi [Excluding a field from search \('Void' indexing\)](#) [p 318]

Example:

```
<xs:element name="hiddenFieldInSearch" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:defaultView>
        <hiddenInSearch>true</hiddenInSearch>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the text and typed search tools of a dataset.

```
<xs:element name="hiddenFieldOnlyInTextSearch" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:defaultView>
        <hiddenInSearch>textSearchOnly</hiddenInSearch>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

In this example, the element is hidden only in the text search tool of a dataset.

Hiding a field in Data Services

To specify whether or not to hide an element in data services, use the element `osd:defaultView/hiddenInDataServices`. For more information, see [Disabling fields from data model](#) [p 713].

Note

- If a group is configured as being hidden, then all the fields nested under this group will be considered as hidden by data services.

Example:

```
<xs:element name="hiddenFieldInDataService" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:defaultView>
        <hiddenInDataServices>true</hiddenInDataServices>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```



```

</osd:defaultView>
</xs:appinfo>
</xs:annotation>
</xs:element>

```

In this example, the element is hidden in the Data Service select operation.

Defining a view for the combo box selector of a foreign key

It is possible to specify a published view that will be used to display the target table or the hierarchical view of a foreign key for a smoother selection. If a view has been defined, the selector will be displayed in the user interface in the combo box of this foreign key. The definition of a view can be done by using the XML Schema element `osd:defaultView/widget/viewForAdvancedSelection`.

Note

- This property can only be defined on foreign key fields.
- The published view must be associated with the target table of the foreign key.
- If the published view does not exist, then the advanced selection is not available in the foreign key field.

Example:

In this example, the name of a published view is defined to display the target table of a foreign key in the advanced selection.

```

<xs:element name="catalog_ref" type="xs:string" minOccurs="0"/>
  <xs:annotation>
    <xs:appinfo>
      <osd:otherFacets>
        <osd:tableRef>
          <tablePath>/root/Catalogs</tablePath>
        </osd:tableRef>
      </osd:otherFacets>
      <osd:defaultView>
        <widget>
          <viewForAdvancedSelection>catalogView</viewForAdvancedSelection>
        </widget>
      </osd:defaultView>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

```

See [Combo-box selector](#) [p 58] for more information.

Customizing a default widget

A widget can be defined using the data model assistant. See [Default view > Widget](#) [p 58] for more information.

Customizing REST data services

Default view configuration is managed in REST data services through the [session channel](#) [p 750].

90.5 Comparison mode

The attribute `osd:comparison` can be included on a terminal node element in order to set its comparison mode. This mode controls how differences are detected for the element during comparisons. The possible values for the attribute are:

default	Element is visible during comparisons of its data.
ignored	No changes will be detected when comparing two versions of the content in records or datasets. During a merge, the data values of an ignored element are not merged when contents are updated, even if the values are different. For new content, the values of ignored elements are merged. During an archive import, values of ignored elements are not imported when contents are updated. For new content, the values of ignored elements are imported.

Note

- If a group is configured as being ignored during comparisons, then all the fields nested under this group will also be ignored.
- If a terminal field does not include the attribute `osd:comparison`, then it will be included by default during comparisons.

Restrictions:

- This property cannot be defined on non-terminal fields.
- Primary key fields cannot be ignored during comparison.

Example:

In this example, the first element is explicitly ignored during comparison, the second element is explicitly included.

```
<xs:element name="fieldExplicitlyIgnoredInComparison"
  type="xs:string" minOccurs="0" osd:comparison="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredInComparison"
  type="xs:string" minOccurs="0" osd:comparison="default"/>
```

90.6 Apply last modifications policy

The attribute `osd:applyLastModification` can be defined on a terminal node element in order to specify if this element must be included or not in the 'apply last modifications' service that can be executed in a table of a dataset.

The possible values for the attribute are:

default	Last modifications can be applied to this element.
ignored	This element is ignored from the apply last modifications service. That is, the last modification that has been performed on this element cannot be applied to other records.

Note

- If a group is configured as being ignored by the 'apply last modifications' service, then all fields nested under this group will also be ignored.
- If a terminal field does not include the attribute `osd:applyLastModification`, then it will be included by default in the apply last modifications service.

Restriction:

- This property cannot be defined on non-terminal fields.

Example:

In this example, the first element is explicitly ignored in the 'apply last modifications' service, the second element is explicitly included.

```
<xs:element name="fieldExplicitlyIgnoredInApplyLastModification"
  type="xs:string" minOccurs="0" osd:applyLastModification="ignored"/>
<xs:element name="fieldExplicitlyNotIgnoredApplyLastModification"
  type="xs:string" minOccurs="0" osd:applyLastModification="default"/>
```

90.7 Categories

Categories can be used for "filtering", by restricting the display of data model elements.

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 dataset in the user interface, select **Actions > Categories > <category name>** from the navigation pane.

Predefined categories

Two categories with localized labels are predefined:

- Hidden

An instance node, including a table node itself, is hidden in the default view, but can be revealed by selecting **Actions > Categories > [hidden nodes]** from the navigation pane.

A table record node is always hidden.

- Constraint (deprecated)

Restriction

Categories do not apply to table record nodes, except the category 'Hidden'.

CHAPITRE 91

Data services

This chapter details how WSDL operations' names related to a table are defined and managed by TIBCO EBX®.

Ce chapitre contient les sections suivantes :

1. [Definition](#)
2. [Configuration](#)
3. [Publication](#)
4. [WSDL and table operations](#)
5. [Limitations](#)

91.1 Definition

EBX® generates a WSDL that complies with the [W3C Web Services Description Language 1.1](#) standard. By default, WSDL operations refer to a table using the last element of the table path. A WSDL operation name is composed of the action name (prefix) and the table name (suffix). It is possible to refer to tables in WSDL operations using unique names instead of the last element of their paths by overriding the suffix operations' names.

Voir aussi [Data services using the Data Model Assistant](#) [p 85]

91.2 Configuration

Embedded data model

WSDL suffix operations' names are embedded in EBX®'s repository and linked to a publication. That is, when publishing an embedded data model, the list of WSDL suffix operations' names can be defined in the data model definition, under the 'Configuration > Data services' table and managed by EBX®.

Packaged data model

WSDL suffix operations' names are defined in a dedicated XML document file and must be named as the data model and end with the keyword `_entities`. For instance, if a data model is named `catalog.xsd`, then the XML document containing the configuration of the WSDL operations' names overridden will be named `catalog_entities.xml`. This XML document must also be located in the same location as the

data model. The XML document is automatically loaded by EBX® if a file that matches this pattern is found when compiling a data model.

91.3 Publication

The suffix operations' names are validated at compilation time and contain a list of couples containing Path with a unique table name. Checked validation rules are:

- The path is not unique,
- The table name contains a syntax error,
- The table name is not unique in the XML document.

91.4 WSDL and table operations

WSDL Generator

An additional validation rule has been added: a unicity check is systematically applied to table names. The SOAP operation name is composed of the operation type as a prefix and, by default, of the table name (last step of the table path) as a suffix. A dataset can contain several identical table names but with different paths. It is possible to override table names that are not unique in order to guarantee the unicity.

SOAP operations

When an operation request on table has been invoked from the SOAP connector, the target table is retrieved by priority, the name corresponds to:

1. an overridden table name,
2. the last step of the table path.

Voir aussi [Data services](#) [p 694]

91.5 Limitations

WSDL operations' names are not available with external data models.

CHAPITRE 92

Toolbars

This chapter details how toolbars are defined and managed by TIBCO EBX®.

Ce chapitre contient les sections suivantes :

1. [Definition](#)
2. [Using toolbars](#)

92.1 Definition

Toolbars allow to customize the buttons and menus to display when accessing a table view, a hierarchical view, or a record form.

Toolbars can only be created and published using the *Data Model Assistant* and are available only on embedded and packaged data models.

For embedded data models, toolbars are embedded in EBX®'s repository and linked to a publication. That is, when publishing an embedded data model, the toolbars defined in the data model are embedded with the publication of the data model and managed by EBX®.

For packaged data models, toolbars are defined in a dedicated XML document and must be named as the data model and end with the keyword `_toolbars`. For instance, if a data model is named `catalog.xsd` then the XML document containing the definition of the toolbars must be named `catalog_toolbars.xml`. This XML document must also be placed in the same location as the data model. The toolbar document is automatically loaded by EBX® if a file complying with this pattern is found when compiling a data model.

Voir aussi

[*Configuring toolbars using the Data Model Assistant*](#) [p 79]

[*Using toolbars in data models*](#) [p 553]

Toolbar API *ToolbarFactory*^{API}

92.2 Using toolbars

Toolbars can be used on tables and associations.

On tables, it is possible to specify the toolbar to display:

- On the top of a tabular view
- On each row of a tabular view

- On the top of a record form
- On the top of a hierarchical view.

On associations, it is possible to specify the toolbar to display:

- On top of the tabular view of the association
- On each row of the tabular view of the association

Voir aussi

[*Using toolbars*](#) [p 553]

[*Associations*](#) [p 558]

CHAPITRE 93

Custom forms

This chapter details how custom forms are defined and managed by TIBCO EBX®.

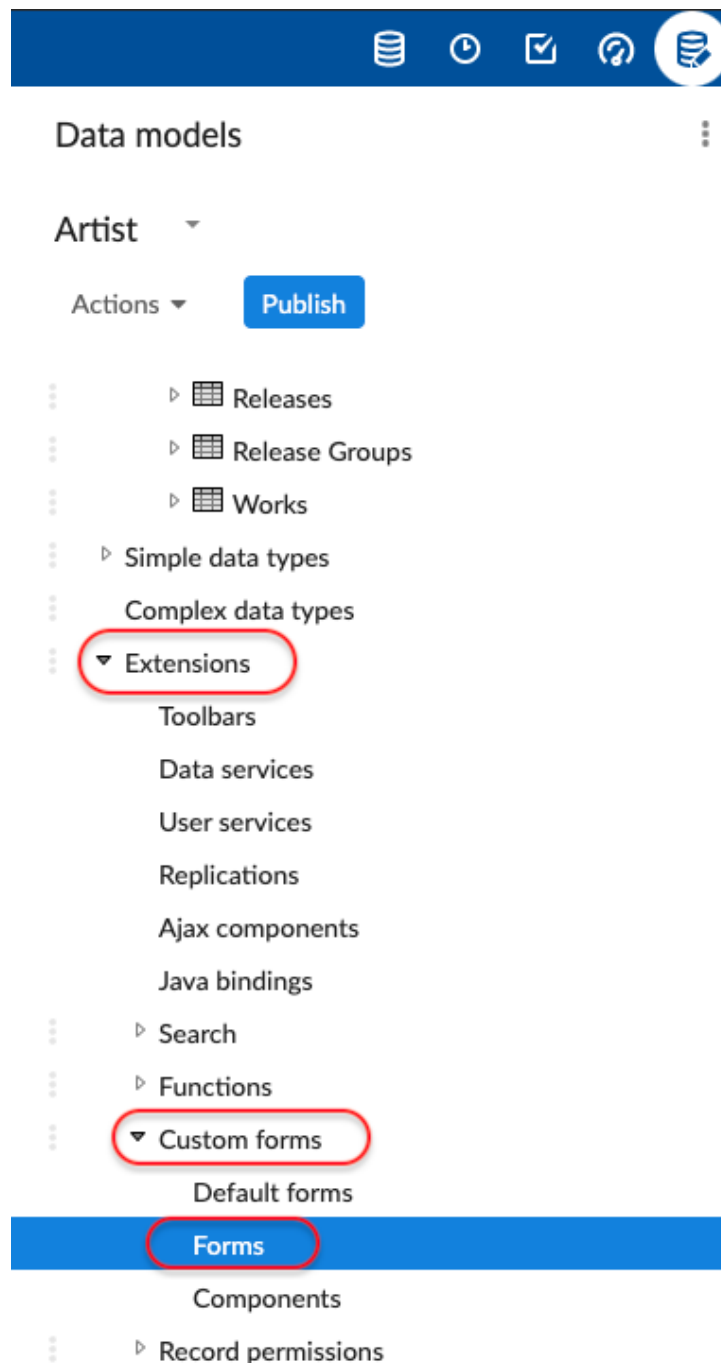
Concepts apparentés [Interface customization](#) [p 658]

Ce chapitre contient les sections suivantes :

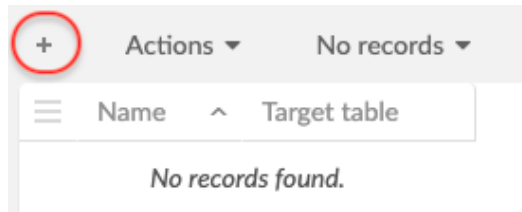
1. [Access](#)
2. [Forms and components](#)
3. [The editor](#)
4. [Blocks](#)

93.1 Access

To access it, go to: Data Models > Extensions > Custom forms > Forms

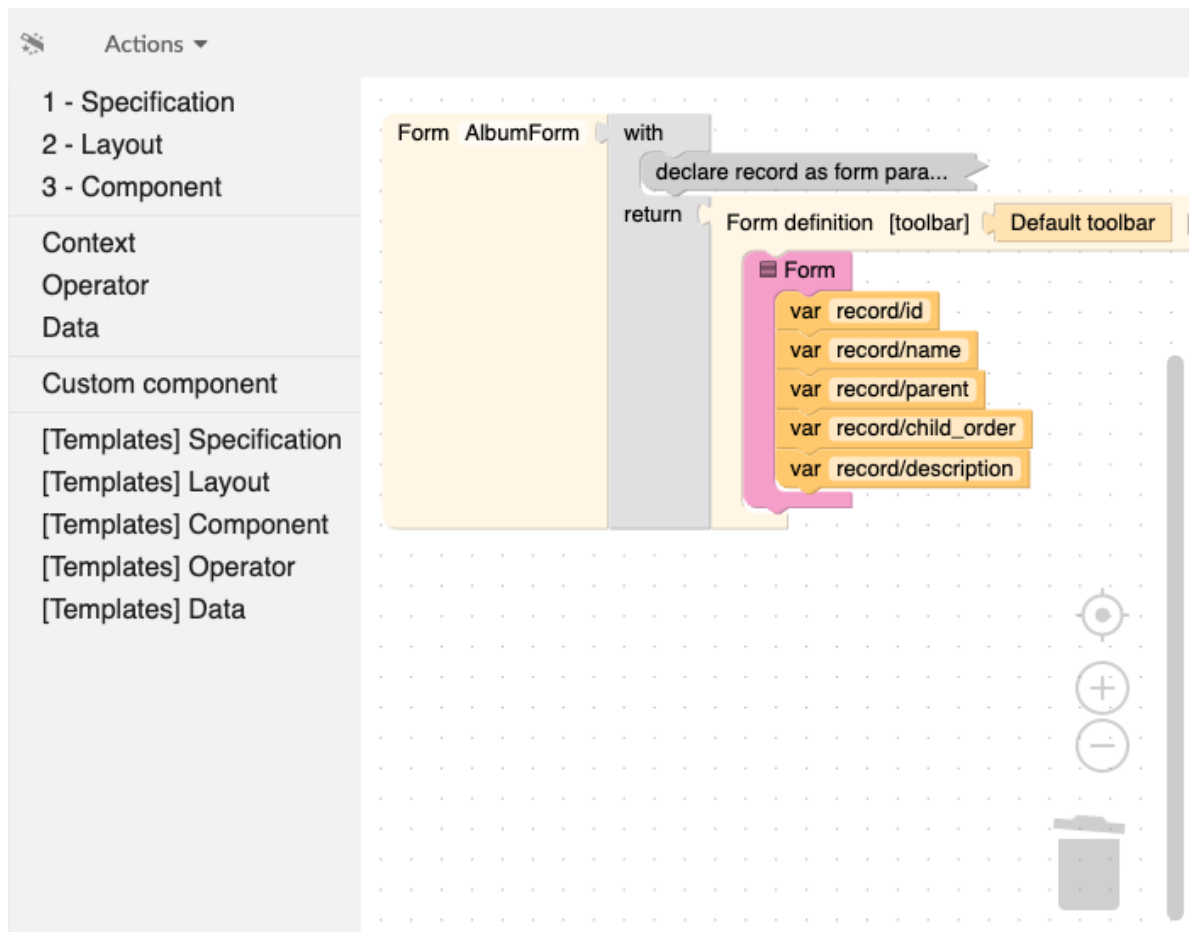


Click on the [+] button to create a form.



Name your new form and indicate the table on which it will be available.

Pressing the "Save" button will redirect you to the layout designer.



93.2 Forms and components

Both forms and components can be created in the 'Custom forms' data model extension.

- **A form describes the layout of a record.** It can access contextual information such as the record or the input parameters. When creating a form, the user is asked to provide its target table. Once created, **it can be declared as the default form of the table** either in the 'Default form' table of the extension, or in the 'Extensions' tab of the target table. Forms that are not the default for their table will not be used.
- **A component is a reusable fragment** that can be shared between forms and other components. Unlike forms, components don't have access to contextual information. If such information is needed, it must be provided explicitly by its caller.

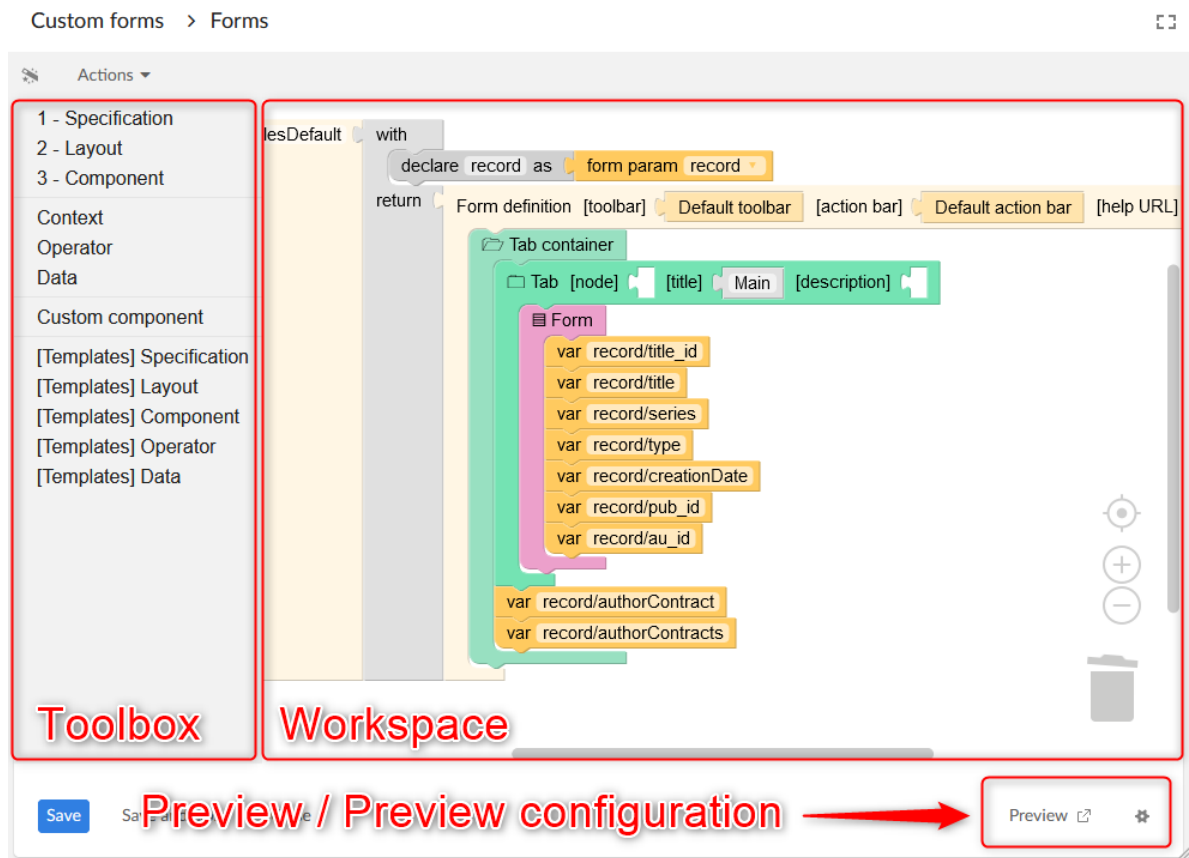
93.3 The editor

A form or component is just an imbrication of blocks. Blocks can represent a concrete graphical element or some piece of logic, allowing to have different layouts based on conditions like permissions, language, etc.

The workspace contains the description of the form, inside a predefined root block. Any block that is not connected to this root is grayed out, to mark it as inactive. **To move a block**, it must be dragged and dropped. Dragging a block also drags the blocks connected below it. If only one block has to be moved, hold the control key before clicking on it. **Right-clicking** on a block in the workspace also shows a list of options such as expand/collapse, comment, help, etc.

The toolbox on the left displays a list of categories. By clicking on a category, the blocks it contains are displayed. Some categories have a related 'Template' section. This section provides some combinations of the blocks of the section and can be considered as useful shortcuts or samples.

On the bottom right of the screen are the 'Preview' and 'Configure preview' buttons. These can be used to see what the form will look like.



93.4 Blocks

Here is the list of all the blocks that appear in the toolbox.

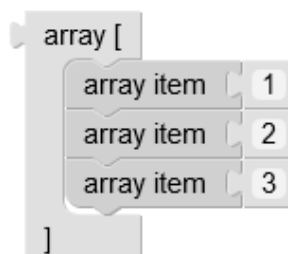
Arithmetic operator

Operations on two integers.



Array

A list of items.



Voir aussi [Array item](#) [p 614]

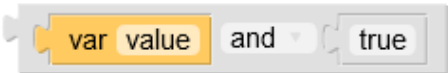
Array item

Adapter to make expression blocks into arrays.

Voir aussi [Array](#) [p 613]

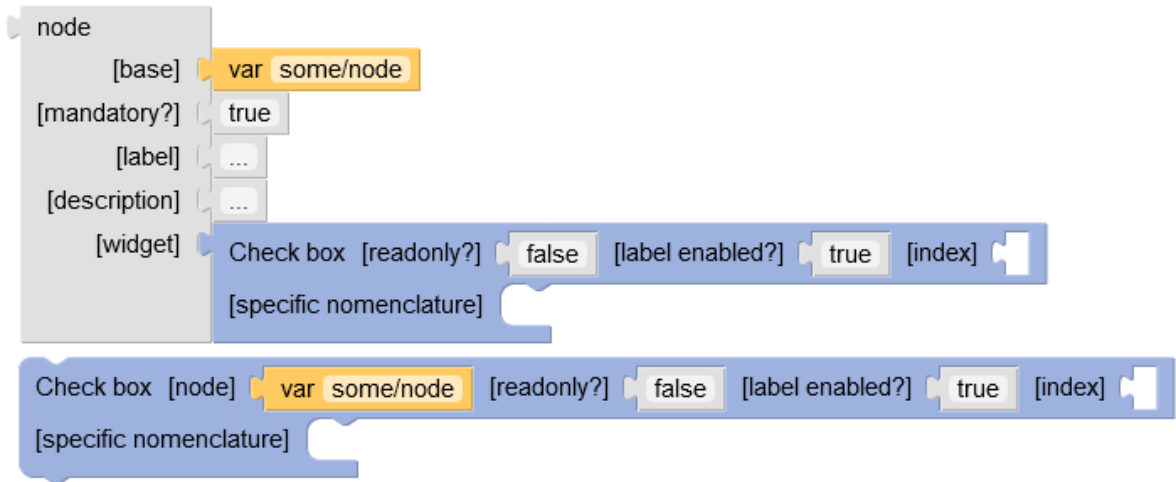
Boolean operator

Operations on two booleans.



Check box

Displays the checkbox widget.

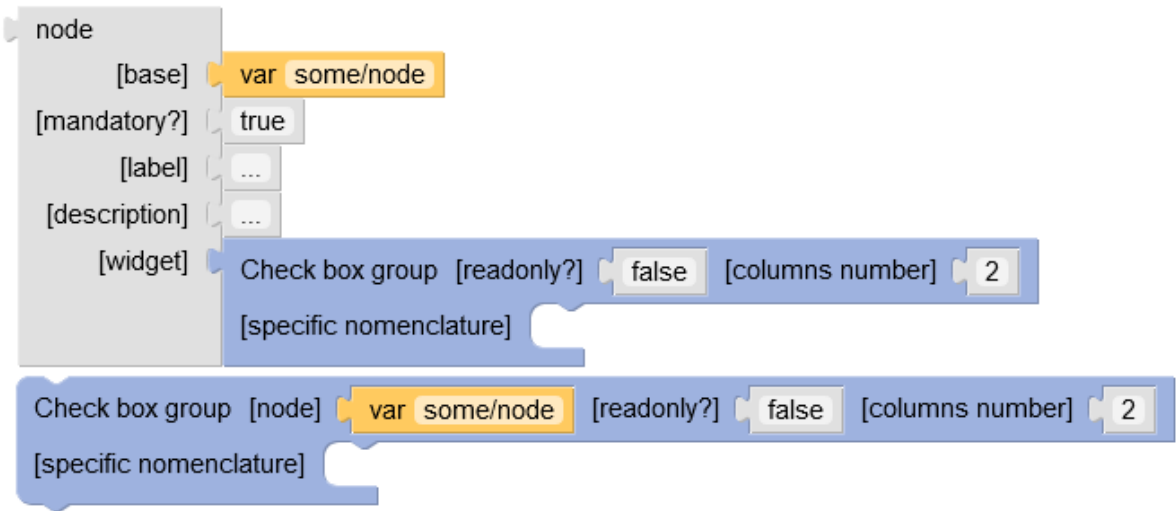


Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.
label enabled?	If set, indicates if the item label is to be added next to the widget.
index	The index for this enumeration item.
specific nomenclature	If set, overrides the model-driven nomenclature.

Check box group

Displays the checkbox group widget.



Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.
columns number	If set, defines the number of columns to use for the layout of the checkboxes.
specific nomenclature	If set, overrides the model-driven nomenclature.

Children

Returns the list of the children of the given node.

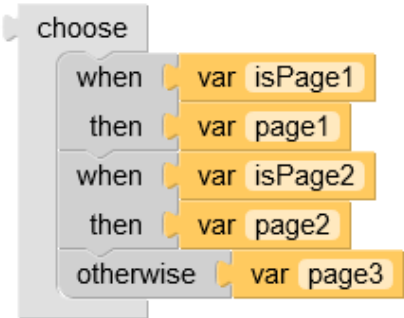


Parameters

Input	Description
node	The complex node from which to extract children.

Choose

A block returning the content of one of its inner 'when'/'otherwise' blocks.



Parameters

Input	Description
content	A list of 'when' blocks and optionally a final 'otherwise' block.

Voir aussi

[When](#) [p 645]

[Otherwise](#) [p 636]

Close button

Standard 'Close' button.

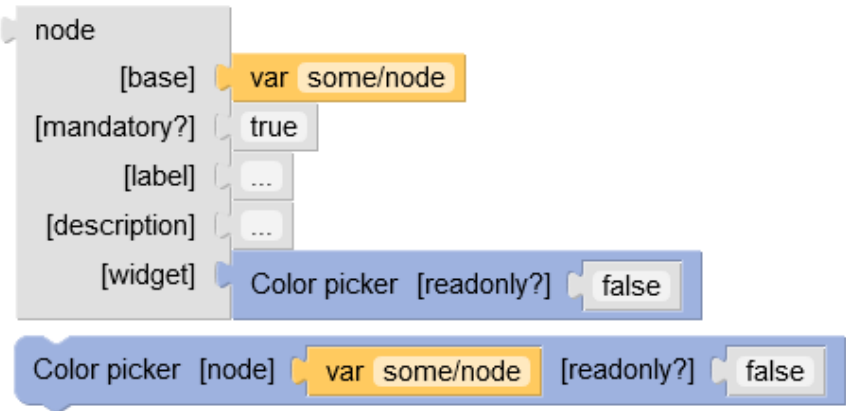


Parameters

Input	Description
label	If set, overrides the default label.
default?	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Color picker

Displays the color picker widget.

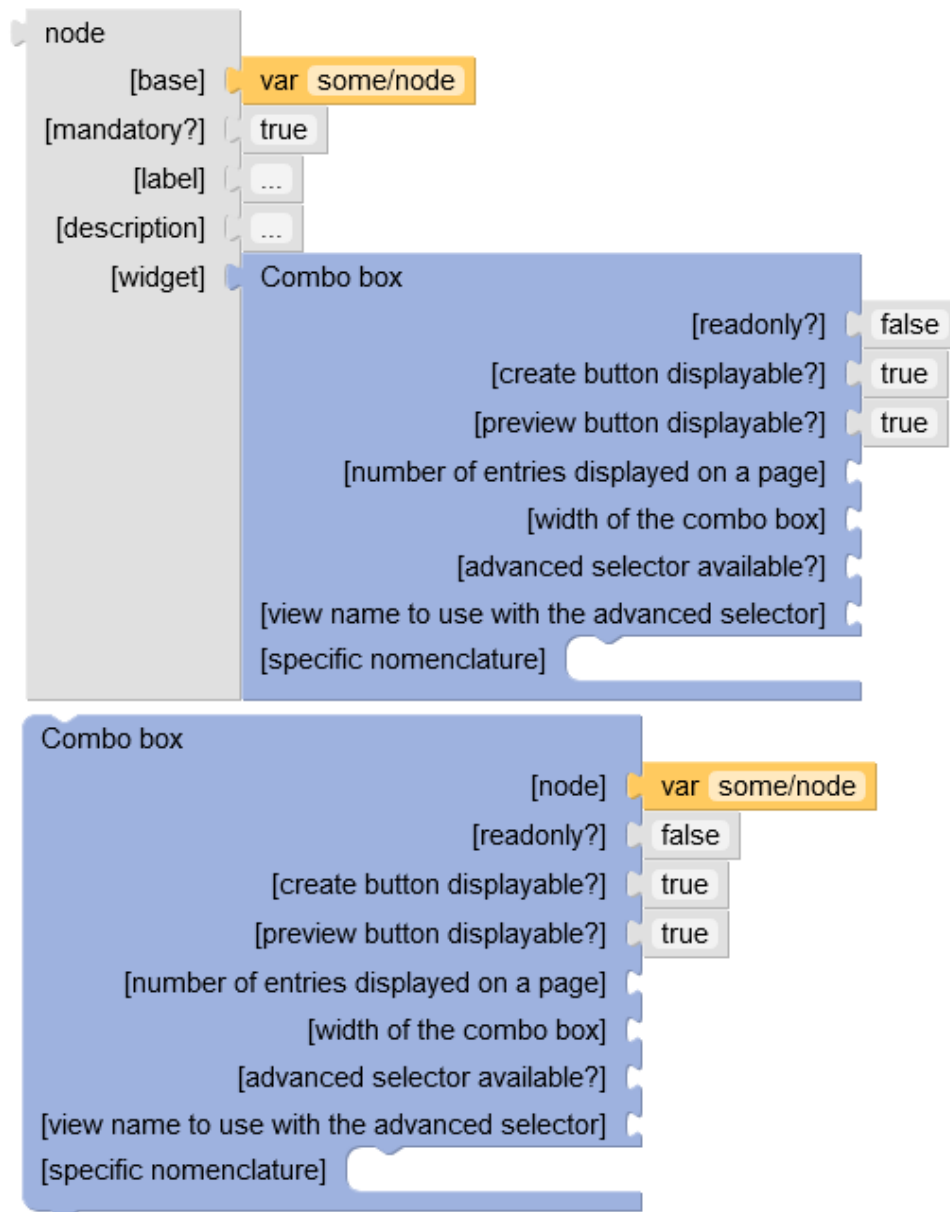


Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.

Combo box

Displays the combo box widget.

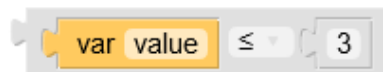


Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.
create button displayable?	If set, defines if the create button should be displayed when the underlying node is a foreign key.
preview button displayable?	If set, defines if the preview button should be displayed when the underlying node is a foreign key.
number of entries displayed on a page	If set, defines the number of entries on each page of the drop-down list.
width of the combo box	If set, defines the width of the combo box.
advanced selector available?	If set, defines if the advanced selector should be displayed when the underlying node is a foreign key.
view name to use with the advanced selector	If set, defines the name of the published view that will be used in the combo-box selection of the associated foreign key field.
specific nomenclature	If set, overrides the model-driven nomenclature.

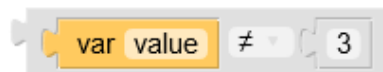
Comparator

Compares two integers.



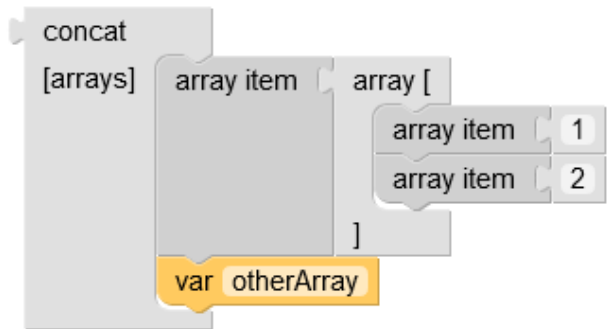
Comparison

Compares strings, numbers or booleans.



Concatenate arrays

Returns the concatenation of the given arrays.

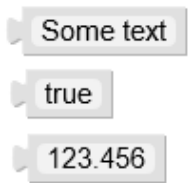


Parameters

Input	Description
arrays	A list of arrays.

Constant

A constant value. Depending on the context, this value may be interpreted as text, boolean or number.



Parameters

Input	Description
value	Text, boolean or number.

Content title

Display the workspace content title.

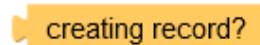


Parameters

Input	Description
label	The text to display.
horizontal alignment	The horizontal alignment of the element. If set, accepted values are: 'start', 'end', 'center'.

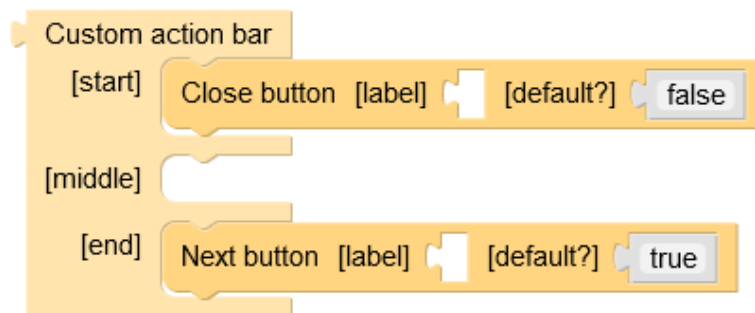
Creating record?

Returns a boolean indicating if the form is displayed in the context of a record creation or duplication.



Custom action bar

A custom action bar.

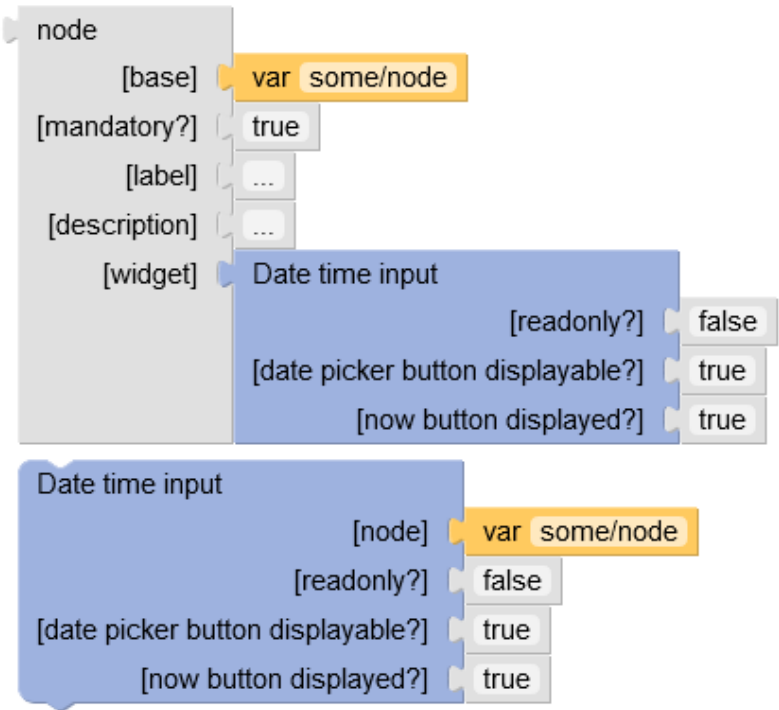


Parameters

Input	Description
start	A list of 'Button's to display on the left of the action bar.
middle	A list of 'Button's to display on the center of the action bar.
end	A list of 'Button's to display on the right of the action bar.

Date time input

Displays the date/time input.



Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.
date picker button displayable?	Indicates if the button opening the date picker is displayed (read/write mode only).
now button displayed?	Indicates if the button setting the date to the current time is displayed (read/write mode only).

Declare

The declaration of a variable, in a 'with' block.



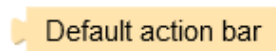
Parameters

Input	Description
name	The name of the variable. Must be unique for a given 'with' block.
content	The value of the variable, which will be returned by 'var' blocks referencing this.

Voir aussi [With](#) [p 646]

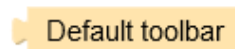
Default action bar

The default action bar.



Default toolbar

The model-driven toolbar.



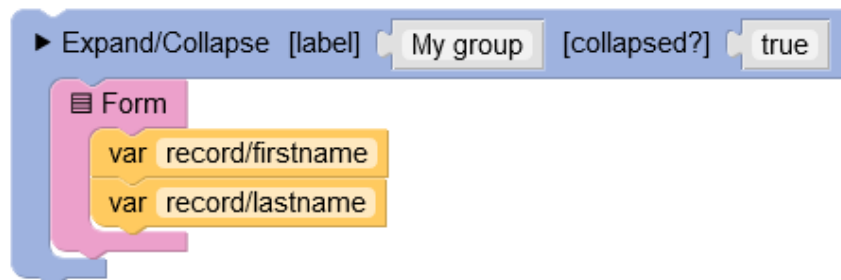
Duplicating record?

Returns a boolean indicating if the form is displayed in the context of a record duplication.



Expand/Collapse

Displays its content inside an expand/collapse block.

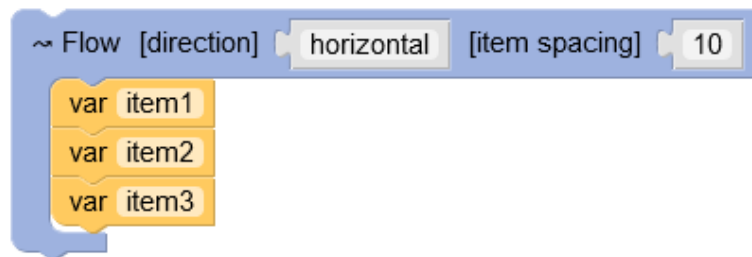


Parameters

Input	Description
label	Label displayed next to the arrow.
collapsed?	Indicates if the group is initially collapsed or not.
content	A list of components to be collapsible.

Flow

Displays its content in a fluid manner. If the elements don't fit in one row or column, they will wrap to start a new one.

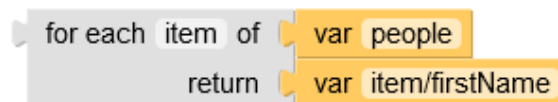


Parameters

Input	Description
direction	'horizontal' or 'vertical', indicates in which direction to queue its content.
item spacing	The space between elements, in pixels.
content	A list of elements to display either horizontally or vertically.

For each

Returns an array that is made of the result of the 'body' function, applied on each item of the 'of' array.

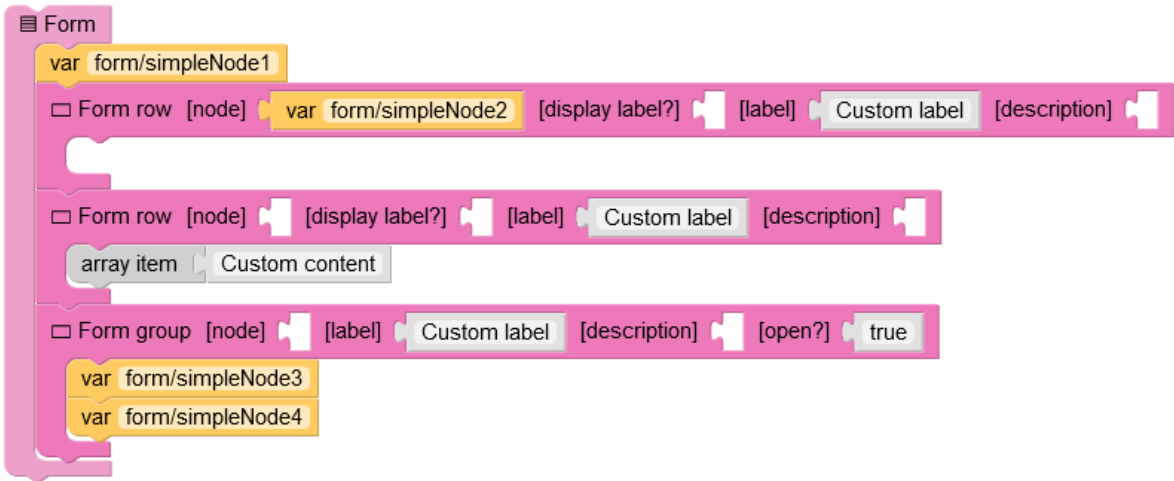


Parameters

Input	Description
item	The name of the variable containing the current item.
array	The array containing the items to transform.
return	What to return for the current item.

Form

Standard table-like layout with labels on the left side and values on the right side.



Parameters

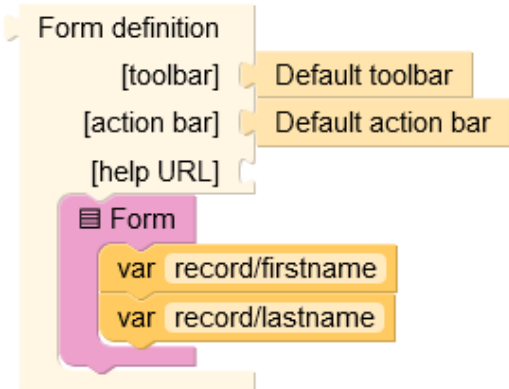
Input	Description
content	A list of 'Form row', 'Form group' or nodes. Nodes will result in model driven display, 'Form row' and 'Form group' allow to display custom content.

Voir aussi

- [Form row](#) [p 627]
- [Form group](#) [p 626]

Form definition

Actual definition of a form.

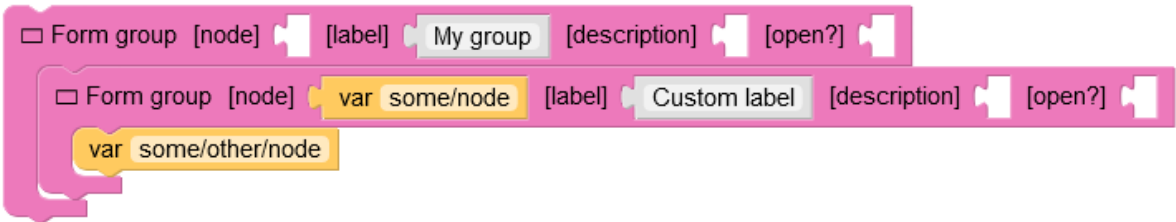


Parameters

Input	Description
toolbar	Definition of the toolbar displayed on top of the form.
action bar	Definition of the action bar displayed below the form.
help URL	URL of the help page.
content	The content of the layout.

Form group

Inside a 'Form', displays a collapsible group of items.



Parameters

Input	Description
node	If set, the group label, description and content will be model driven.
label	The label of the group. May override the model driven label if a node and a label are set.
description	The description of the group. May override the model driven description if a node and a description are set.
open?	Indicates if the group is initially open.
content	A list of 'Form row', 'Form group' or nodes. Nodes will result in model driven display, 'Form row' and 'Form group' allow to display custom content.

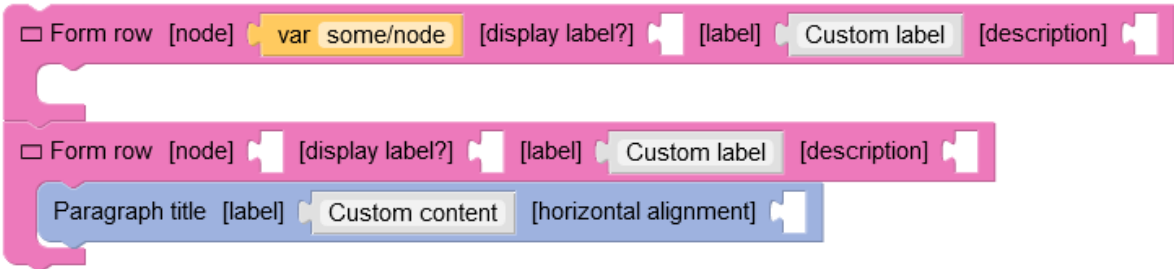
Voir aussi

[Form](#) [p 625]

[Form row](#) [p 627]

Form row

Inside a 'Form' or a 'Form group', displays a row, that is basically a label and a value.



Parameters

Input	Description
node	If set, the label, description and content will be model driven.
display_label?	Indicates if the label should be displayed. If set to false in a 'Form' the content will span over both the label and content area.
label	The label of the row. May override the model driven label if a node and a label are set.
description	The description of the row. May override the model driven label if a node and a label are set.
content	The content to be displayed next to the label.

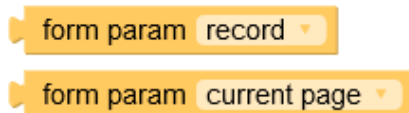
Voir aussi

[Form](#) [p 625]

[Form group](#) [p 626]

Form parameter

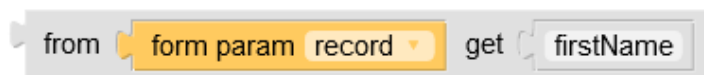
The 'record' is the node representing the displayed record. 'current page' is a number (1 by default) representing the displayed page. This value can be changed by the 'Previous' and 'Next' buttons.



From...get

Extract a value from an object or a node.

<from "node" get "path/to/value"> is equivalent to <var "node/path/to/value">

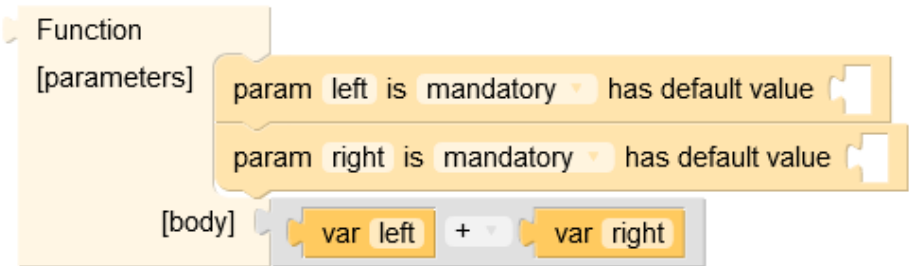


Parameters

Input	Description
from	Base object or node, from which to extract data.
get	Path of the data to extract, inside the object in 'from'.

Function

The definition of an anonymous function. A function makes use of its parameters in its body to return a result.



Parameters

Input	Description
parameters	List of 'param' blocks, defining the parameters accepted by this function.
body	Body describing the result of the function, using its parameters.

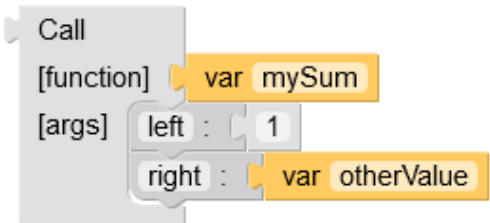
Voir aussi

[Function parameter](#) [p 630]

[Function call](#) [p 629]

Function call

Calls a function by setting actual values to its parameters.



Parameters

Input	Description
function	The function to call.
args	The arguments to pass to the function.

Voir aussi

[Function call argument](#) [p 630]

[Function](#) [p 629]

Function call argument

Actual value to pass to the parameter of the called function.



Parameters

Input	Description
name	The name of the parameter.
value	The actual value to pass the the parameter.

Voir aussi [Function call](#) [p 629]

Function parameter

This is the declaration of a parameter of a function.



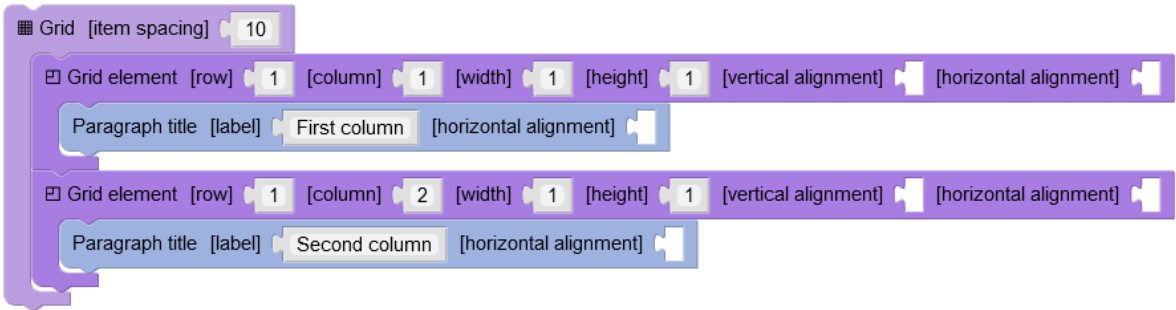
Parameters

Input	Description
name	The name of the parameter.
mandatory?	Indicates if this parameter must be provided by the caller.
default value	If set, this is the default value taken by this parameter if the caller does not provide one.

Voir aussi [Function](#) [p 629]

Grid

A layout allowing to place components in a grid with coordinates.



Parameters

Input	Description
item spacing	Space between elements, in pixels.
content	A list of 'Grid element's, which specify where to display their content.

Voir aussi [Grid element](#) [p 631]

Grid element

Inside a 'Grid', specifies the location of its content.

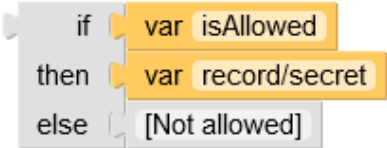
Parameters

Input	Description
row	The row where the element starts. Minimum value is 1.
column	The column where the element starts. Minimum value is 1.
width	The width of the element, in grid cells. Minimum value is 1.
height	The height of the element, in grid cells. Minimum value is 1.
vertical alignment	The vertical alignment of the element. If set, accepted values are: 'start', 'end', 'center'.
horizontal alignment	The horizontal alignment of the element. If set, accepted values are: 'start', 'end', 'center'.
content	The content inside the cell.

Voir aussi [Grid](#) [p 631]

If

A block returning either the content of 'then', or the content 'else', based on the result of the test.



Parameters

Input	Description
test	A test determining which value to return. This must be a boolean value.
then	The value returned if the test is true.
else	The value returned if the test is false.

Input parameter

Returns the raw value of an input parameter.

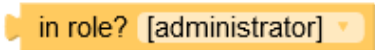


Parameters

Input	Description
name	The parameter name.

In role?

Returns a boolean indicating if the current user has the requested role.



Label

Displays the label of the given node.



Parameters

Input	Description
node	The node for which to display the label.

Localized

Block which result depends on the user locale.

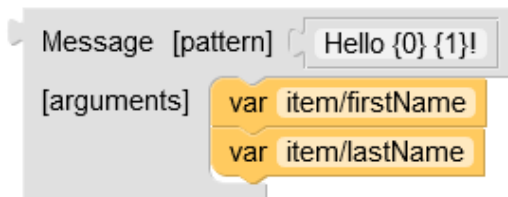


Parameters

Input	Description
[locale]	Each parameter relates to a locale. The result of the block will be the one corresponding to the locale of the current user.

Message

Formats the given text by replacing java-like {0}, {1}, etc. placeholders by the argument at the given index.



Parameters

Input	Description
pattern	Text with placeholders ({0}, {1}, etc.) which will be replaced by the given arguments.
arguments	The list of values which will replace the placeholders in the pattern.

Next button

A button which will increment the 'current page' value by one.

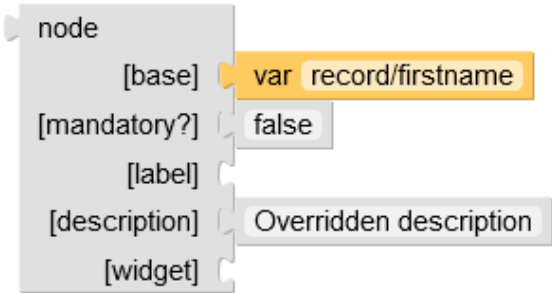


Parameters

Input	Description
label	If set, overrides the default label.
default?	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Node

Overrides the definition of an existing node.

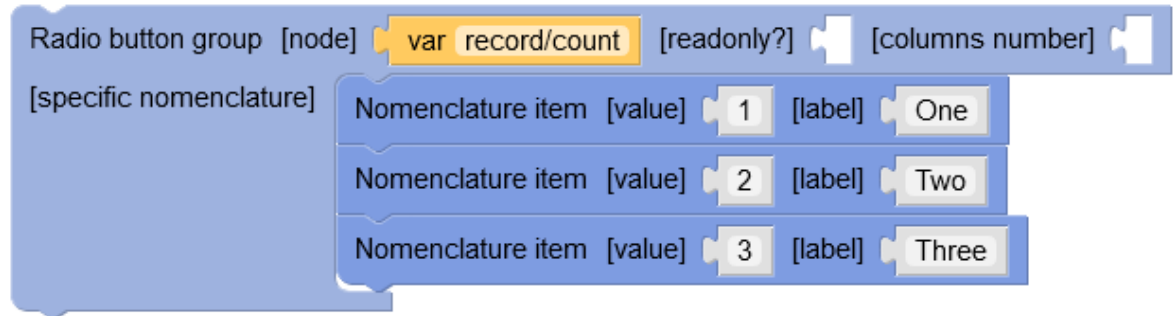


Parameters

Input	Description
base	The node to override.
mandatory?	If set, overrides the mandatory indicator of the base node.
label	If set, overrides the label of the node.
description	If set, overrides the label of the node.
widget	If set, overrides the default widget of the node.

Nomenclature item

A nomenclature item is a (key, label) pair.

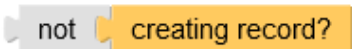


Parameters

Input	Description
value	The value of the item.
label	The label of the item.

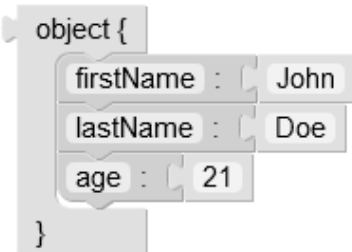
Not

Returns the inverse of the given boolean.



Object

An object composed of a list of (key, value) pairs.



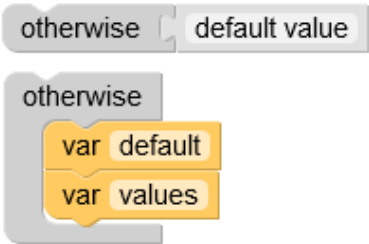
Parameters

Input	Description
content	A list of 'property' blocks.

Voir aussi [Property](#) [p 637]

Otherwise

Final choice in a 'choose' block. Will be returned if no 'when' case matched.



Parameters

Input	Description
content	The value to return if no 'when' case matched.

Voir aussi [Choose](#) [p 616]

Paragraph title

Display a paragraph title.

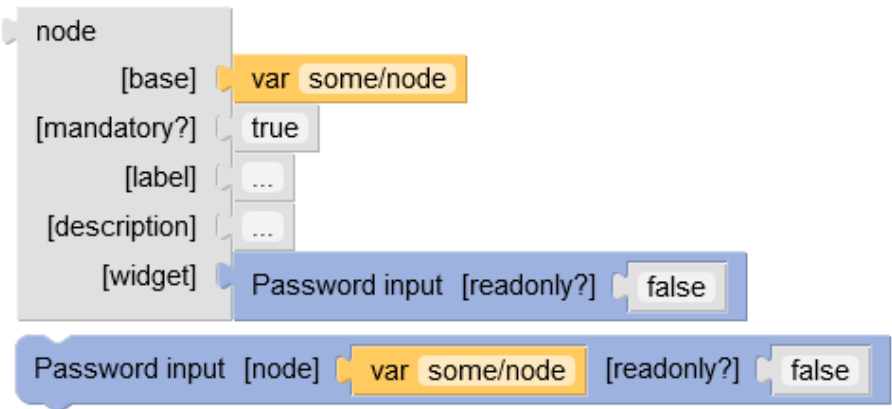


Parameters

Input	Description
label	The text to display.
horizontal alignment	The horizontal alignment of the element. If set, accepted values are: 'start', 'end', 'center'.

Password

Displays the input password widget.



Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.

Previous button

A button which will decrement the 'current page' value by one.



Parameters

Input	Description
label	If set, overrides the default label.
default?	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Property

A property of an object.

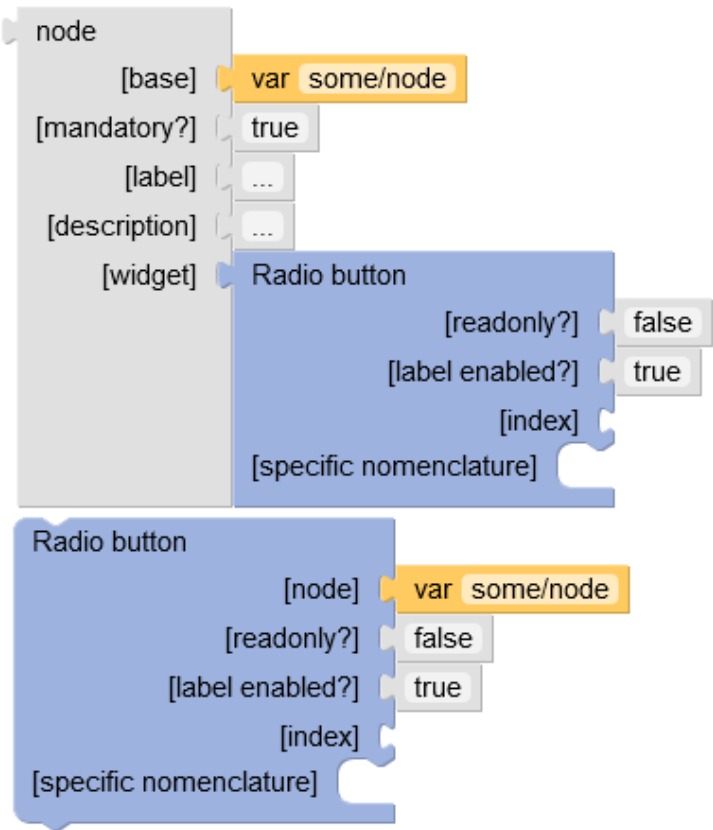


Parameters

Input	Description
name	Name of the property. Must be unique for a given object.
value	Value of the property.

Radio button

Displays a radio button widget.

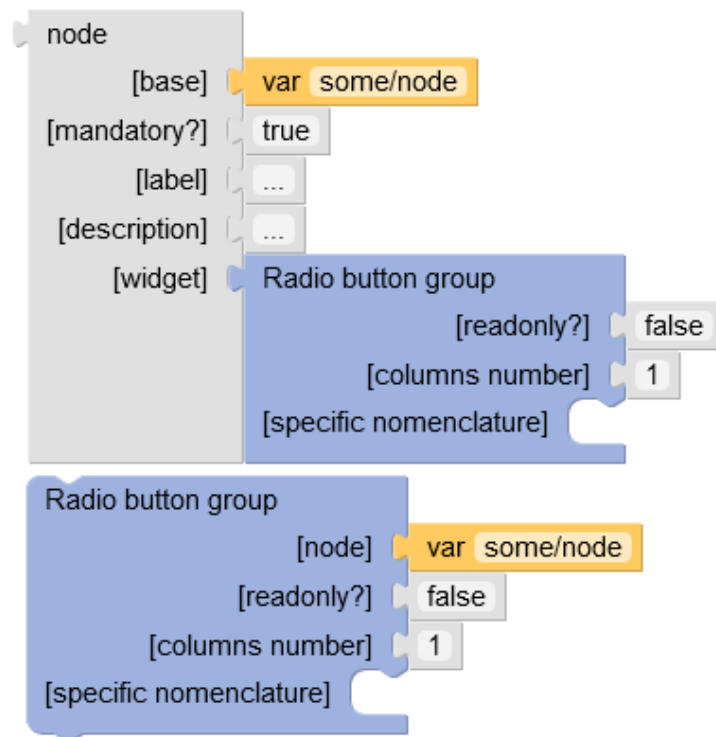


Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.
label enabled?	If set, indicates if the item label is to be added next to the widget.
index	The index for this enumeration item.
specific nomenclature	If set, overrides the model-driven nomenclature.

Radio button group

Displays the radio button group widget.



Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.
columns number	If set, defines the number of columns to use for the layout of the radio buttons.
specific nomenclature	If set, overrides the model-driven nomenclature.

Revert button

Standard 'Revert' button.

Revert button [label]  [default?]  false

Parameters

Input	Description
label	If set, overrides the default label.
default?	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Save button

Standard 'Save' button.

Save button [label]  [default?]  false

Parameters

Input	Description
label	If set, overrides the default label.
default?	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Save and close button

Standard 'Save and close' button.

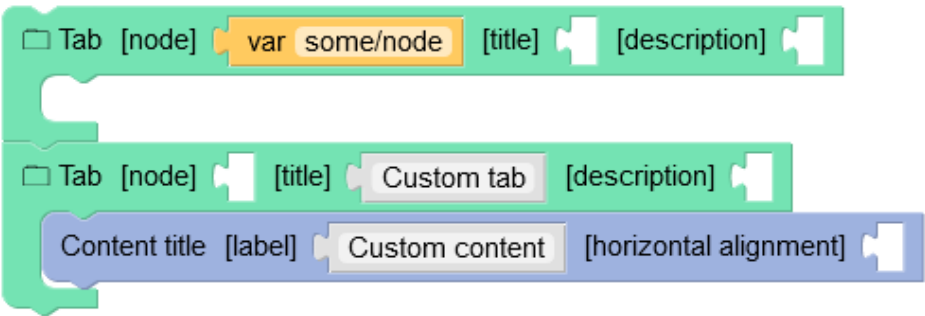
Save and close button [label] [default?] false

Parameters

Input	Description
label	If set, overrides the default label.
default?	Indicates if this is the action to trigger when pressing 'Enter'. Only one button should be the default one.

Tab

A tab, inside a 'Tab container'.



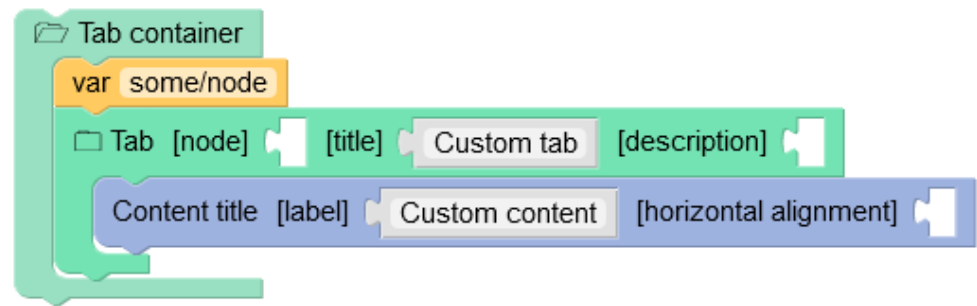
Parameters

Input	Description
node	If present, the tab title, description and content will be model driven.
title	The title of the tab. May override the model driven title if a node and a title are set.
description	The description of the tab. May override the model driven description if a node and a description are set.
content	The content of the tab. May override the model driven content if a node and a content are set.

Voir aussi [Tab container](#) [p 642]

Tab container

A container of tabs, which displays a list of tab names and one active tab.



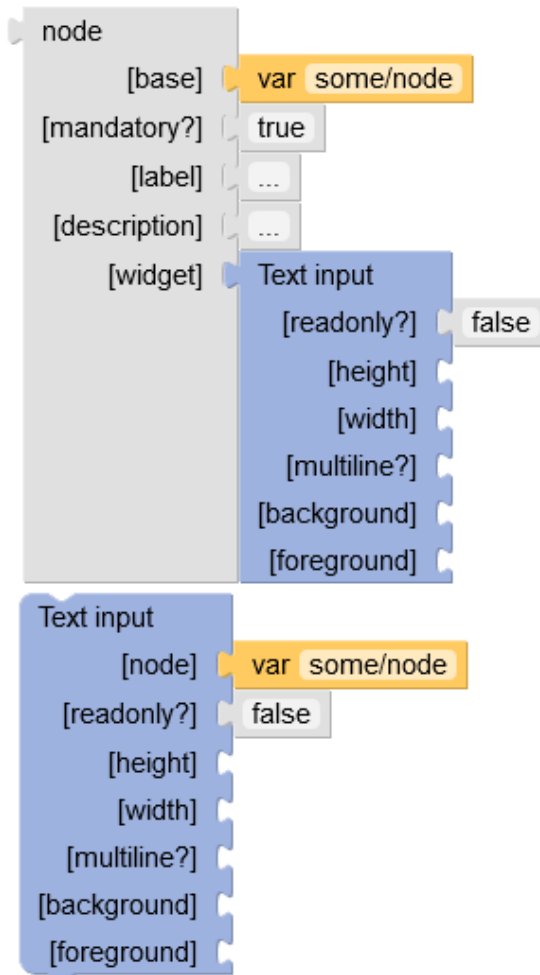
Parameters

Input	Description
content	A list of 'Tab's.

Voir aussi [Tab](#) [p 641]

Text input

Displays the text input widget.

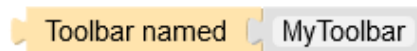


Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.
height	If set, defines the height of the widget, in pixels.
width	If set, defines the width of the widget, in pixels.
multiline?	If set, defines if the widget spans over multiple lines.
background	If set, defines the background color, in hexadecimal format.
foreground	If set, defines the foreground color, in hexadecimal format.

Toolbar named

The toolbar having the specified name.

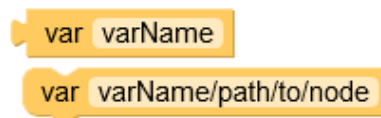


Parameters

Input	Description
name	The name of the toolbar to display.

Variable

A reference to a variable declared by a 'with...declare' block. A path is also accepted if the variable is an object or a node.

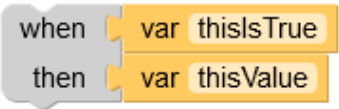


Parameters

Input	Description
expression	A reference to a variable declared by a 'with...declare' block. A path is also accepted if the variable is an object or a node.

When

A choice in a 'choose' block. If the test is true, the 'then' content will be returned, otherwise the next block will be tested.



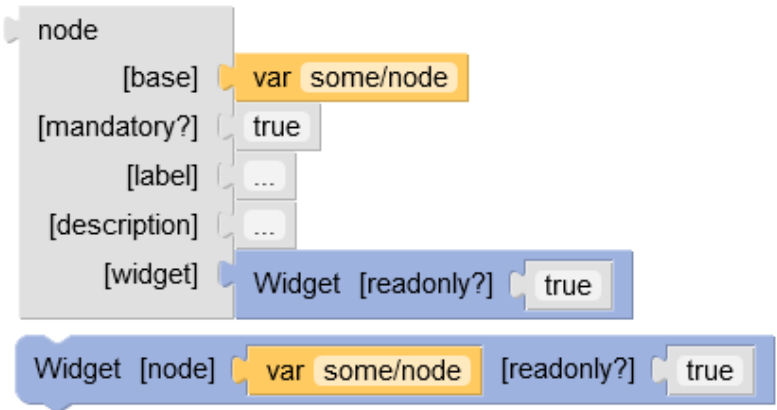
Parameters

Input	Description
when	A boolean test indicating if this case should be resolved.
then	If the test is true, this value will be returned.

Voir aussi [Choose](#) [p 616]

Widget

Displays the default widget.

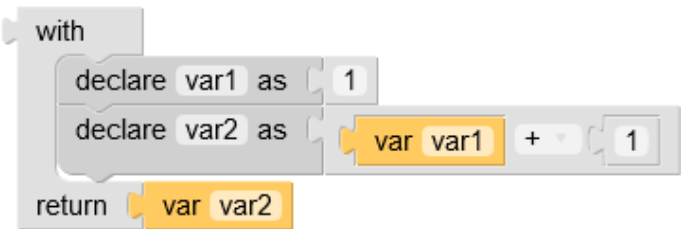


Parameters

Input	Description
node	The node for which to display the widget.
readonly?	Indicates if the widget should be in readonly mode.

With

Declares variables that can be used in its 'return' statement.



Parameters

Input	Description
variables	A list of 'declare' blocks.
return	The value to return. The variables declared in the 'variables' statement can be used here, with 'var' blocks.

Voir aussi [Declare](#) [p 622]

CHAPITRE 94

Workflow model

The workflow offers two types of steps: 'library' or 'specific'.

'Library' is a bean defined in `module.xml` and is reusable. Using the 'library' bean improves the ergonomics: parameters are dynamically displayed in the definition screens.

A 'specific' object is a bean defined only by its class name. In this case, the display is not dynamic.

Ce chapitre contient les sections suivantes :

1. [Bean categories](#)
2. [Sample of ScriptTask](#)
3. [Sample of ScriptTaskBean](#)
4. [Samples of UserTask](#)
5. [Samples of Condition](#)
6. [Sample of ConditionBean](#)
7. [Sample of SubWorkflowsInvocationBean](#)
8. [Sample of WaitTaskBean](#)
9. [Sample of ActionPermissionsOnWorkflow](#)
10. [Sample of WorkflowTriggerBean](#)
11. [Sample of trigger starting a process instance](#)

94.1 Bean categories

Step	Library	Specific
Scripts	ScriptTaskBean	ScriptTask
Conditions	ConditionBean	Condition
User task	UserTask	

94.2 Sample of ScriptTask

Java Code

A script task has to override the method execute as in the following example:

```
public class NppScriptTask_CreateWorkingBranch extends ScriptTask
{
    public void executeScript(ScriptTaskContext aContext) throws OperationException
    {
        Repository repository = aContext.getRepository();
        String initialBranchString = aContext.getVariableString("initialBranch");
        AdaptationHome initialBranch = repository.lookupHome(HomeKey.forBranchName(initialBranchString));
        if (initialBranch == null)
            throw OperationException.createError("Null value for initialBranch");

        HomeCreationSpec spec = new HomeCreationSpec();
        spec.setParent(initialBranch);
        spec.setKey(HomeKey.forBranchName("Name"));
        spec.setOwner(Profile.EVERYONE);
        spec.setHomeToCopyPermissionsFrom(initialBranch);
        AdaptationHome newHome = repository.createHome(spec, aContext.getSession());
        //feeds dataContext
        aContext.setVariableString("workingBranch", newHome.getKey().getName());
    }
}
```

Voir aussi [com.orchestranetworks.workflow.ScriptTask](#) *ScriptTask*^{API}

94.3 Sample of ScriptTaskBean

Java Code

A script task bean has to override the method executeScript as in the following example:

```
public class ScriptTaskBean_CreateBranch extends ScriptTaskBean
{
    private String initialBranchName;

    private String newBranch;

    public String getInitialBranchName()
    {
        return this.initialBranchName;
    }

    public void setInitialBranchName(String initialBranchName)
    {
        this.initialBranchName = initialBranchName;
    }

    public String getNewBranch()
    {
        return this.newBranch;
    }

    public void setNewBranch(String newBranch)
    {
        this.newBranch = newBranch;
    }

    public void executeScript(ScriptTaskBeanContext aContext) throws OperationException
    {
        final Repository repository = aContext.getRepository();

        String initialBranchName = this.getInitialBranchName();
        final AdaptationHome initialBranch = repository.lookupHome(HomeKey.forBranchName(initialBranchName));
        final HomeCreationSpec spec = new HomeCreationSpec();
        spec.setParent(initialBranch);
        spec.setKey(HomeKey.forBranchName(XsFormats.SINGLETON.formatDateTime(new Date())));
        spec.setOwner(Profile.EVERYONE);
        spec.setHomeToCopyPermissionsFrom(initialBranch);
        final AdaptationHome branchCreate = repository.createHome(spec, aContext.getSession());
    }
}
```



```

        this.setNewBranch(branchCreate.getKey().getName());
    }
}

```

Voir aussi *com.orchestranetworks.workflow.ScriptTaskBean* *ScriptTaskBean*^{API}

Configuration through module.xml

A script task bean must be declared in module.xml:

```

<module>
  <beans>
    <bean className="com.orchestranetworks.workflow.genericScriptTask.ScriptTaskBean_CreateBranch">
      <documentation xml:lang="fr-FR">
        <label>Créer une branche</label>
        <description>
          Ce script permet de créer une branche
        </description>
      </documentation>
      <documentation xml:lang="en-US">
        <label>Create a branch</label>
        <description>
          This script creates a branch
        </description>
      </documentation>
      <properties>
        <property name="initialBranchName" input="true">
          <documentation xml:lang="fr-FR">
            <label>Branche initiale</label>
            <description>
              Nom de la branche initiale.
            </description>
          </documentation>
          <documentation xml:lang="en-US">
            <label>Initial branch</label>
            <description>
              Initial branch name.
            </description>
          </documentation>
        </property>
        <property name="newBranch" output="true">
          <documentation xml:lang="fr-FR">
            <label>Nouvelle branche</label>
            <description>
              Nom de la branche créée
            </description>
          </documentation>
          <documentation xml:lang="en-US">
            <label>New branch</label>
            <description>
              Created branch name.
            </description>
          </documentation>
        </property>
      </properties>
    </bean>
  </beans>
</module>

```

94.4 Samples of UserTask

Service declaration via module.xml

A built-in service can be declared in module.xml to be used in the user task definition.

```

<services>
  <service name="ServiceModule">
    <resourcePath>/service.jsp</resourcePath>
    <type>branch</type>
    <documentation xml:lang="fr-FR">
      <label>Workflow service</label>
      <description>
        Ce service permet de ...
      </description>
    </documentation>
    <documentation xml:lang="en-US">

```

```

        <label>Service workflow</label>
        <description>
            The purpose of this service is ...
        </description>
    </documentation>
    <properties>
        <property name="param1" input="true">
            <documentation xml:lang="fr-FR">
                <label>Param1</label>
                <description>Param1 ...</description>
            </documentation>
        </property>
        <property name="param2" output="true">
        </property>
    </properties>
</service>
<serviceLink serviceName="adaptationService">
    <importFromSchema>
        /WEB-INF/ebx/schema/schema.xsd
    </importFromSchema>
</serviceLink>
</services>

```

A more complex UserTask

The GUI is quite similar as the example above. The field 'Rule' must be filled to define the class extending the 'UserTask' to invoke.

```

public class NppUserTask_ValidateProduct extends UserTask
{
    public void handleWorkItemCompletion(UserTaskWorkItemCompletionContext context)
        throws OperationException
    {
        if (context.getCompletedWorkItem().isRejected())
        {
            context.setVariableString(NppConstants.VAR_VALIDATION, "KO");
            context.completeUserTask();
        }
        else if (context.checkAllWorkItemMatchStrategy())
        {
            context.setVariableString(NppConstants.VAR_VALIDATION, "OK");
            context.completeUserTask();
        }
    }

    public void handleCreate(UserTaskCreationContext context) throws OperationException
    {
        CreationWorkItemSpec spec = CreationWorkItemSpec.forOfferring(NppConstants.ROLE_PVALIDATOR);
        spec.setNotificationMail("1");
        context.createWorkItem(spec);
        context.setVariableString(NppConstants.VAR_VALIDATION, "validating");
    }
}

```

Voir aussi [*com.orchestranetworks.workflow.UserTask*](#) *UserTask^{API}*

94.5 Samples of Condition

Java Code

The method evaluate has to be overridden:

```

public class NppCondition_IsValidationOK extends Condition
{
    public boolean evaluateCondition(ConditionContext context) throws OperationException
    {
        String validation = context.getVariableString("validationResult");
        boolean hasError = "KO".equals(validation);
        return !hasError;
    }
}

```

Voir aussi *com.orchestranetworks.workflow.Condition* ^{API} Condition

94.6 Sample of ConditionBean

Java Code

The method `evaluateCondition` has to be overridden as in the following sample:

```
public class ConditionBean_IsBranchValid extends ConditionBean
{
    private String branchName;

    public String getBranchName()
    {
        return this.branchName;
    }

    public void setBranchName(String branchName)
    {
        this.branchName = branchName;
    }

    public boolean evaluateCondition(ConditionBeanContext aContext) throws OperationException
    {
        final Repository repository = aContext.getRepository();
        Severity severityForValidation = Severity.ERROR;
        String branchToTestName = this.getBranchName();
        final AdaptationHome branchToTest = repository.lookupHome(HomeKey.forBranchName(branchToTestName));
        if (branchToTest.getValidationReportsMap(severityForValidation) != null
            && branchToTest.getValidationReportsMap(severityForValidation).size() > 0)
        {
            return false;
        }
        return true;
    }
}
```

Voir aussi *com.orchestranetworks.workflow.ConditionBean* ^{API} ConditionBean

Configuration through module.xml

The condition bean must be declared in `module.xml`:

```
<module>
  <beans>
    <bean className="com.orchestranetworks.workflow.genericScriptTask.ConditionBean_IsBranchValid">
      <documentation xml:lang="fr-FR">
        <label>Branche valide ?</label>
        <description>
          Ce script permet de tester si une branche est valide.
        </description>
      </documentation>
      <documentation xml:lang="en-US">
        <label>Branch valid ?</label>
        <description>
          This script allows to check if a branch is valid.
        </description>
      </documentation>
      <properties>
        <property name="branchName" input="true">
          <documentation xml:lang="fr-FR">
            <label>Branche à contrôler</label>
            <description>
              Nom de la branche à valider.
            </description>
          </documentation>
          <documentation xml:lang="en-US">
            <label>Branch to check</label>
            <description>
              Branch name to check.
            </description>
          </documentation>
        </property>
      </properties>
    </bean>
  </beans>
</module>
```

```
</beans>
</module>
```

94.7 Sample of SubWorkflowsInvocationBean

Java Code

```
public class MySubWorkflowsInvocationBean extends SubWorkflowsInvocationBean
{
    @Override
    public void handleCreateSubWorkflows(SubWorkflowsCreationContext aContext)
        throws OperationException
    {
        final ProcessLauncher subWorkflow1 = aContext.registerSubWorkflow(
            AdaptationName.forName("validateProduct"),
            "validateProduct1");
        subWorkflow1.setLabel(UserMessage.createInfo("Validate the new product by marketing"));
        subWorkflow1.setInputParameter("workingBranch", aContext.getVariableString("workingBranch"));
        subWorkflow1.setInputParameter("code", aContext.getVariableString("code"));
        subWorkflow1.setInputParameter("service", aContext.getVariableString("marketing"));

        final ProcessLauncher subWorkflow2 = aContext.registerSubWorkflow(
            AdaptationName.forName("validateProduct"),
            "validateProduct2");
        subWorkflow2.setLabel(UserMessage.createInfo("Validate the new product by direction"));
        subWorkflow2.setInputParameter("workingBranch", aContext.getVariableString("workingBranch"));
        subWorkflow2.setInputParameter("code", aContext.getVariableString("code"));
        subWorkflow2.setInputParameter("service", aContext.getVariableString("direction"));

        // Conditional launching.
        if (aContext.getVariableString("productType").equals("book"))
        {
            final ProcessLauncher subWorkflow3 = aContext.registerSubWorkflow(
                AdaptationName.forName("generateISBN"),
                "generateISBN");
            subWorkflow3.setLabel(UserMessage.createInfo("Generate ISBN"));
            subWorkflow3.setInputParameter(
                "workingBranch",
                aContext.getVariableString("workingBranch"));
            subWorkflow3.setInputParameter("code", aContext.getVariableString("code"));
        }

        aContext.launchSubWorkflows();
    }
    @Override
    public void handleCompleteAllSubWorkflows(SubWorkflowsCompletionContext aContext)
        throws OperationException
    {
        aContext.getCompletedSubWorkflows();
        final ProcessInstance validateProductMarketing = aContext.getCompletedSubWorkflow("validateProduct1");
        final ProcessInstance validateProductDirection = aContext.getCompletedSubWorkflow("validateProduct2");
        if (aContext.getVariableString("productType").equals("book"))
        {
            final ProcessInstance generateISBN = aContext.getCompletedSubWorkflow("generateISBN");
            aContext.setVariableString("isbn", generateISBN.getDataContext().getVariableString(
                "newCode"));
        }

        if (validateProductMarketing.getDataContext().getVariableString("Accepted").equals("true")
            && validateProductDirection.getDataContext().getVariableString("Accepted").equals(
                "true"))
            aContext.setVariableString("validation", "ok");
    }
}
```

Voir aussi *com.orchestranetworks.workflow.SubWorkflowsInvocationBean*
SubWorkflowsInvocationBean^{API}

Configuration through module.xml

SubWorkflowsInvocationBean bean must be declared in module.xml:

```
<module>
  <beans>
    <bean className="com.orchestranetworks.workflow.test.MySubWorkflowsInvocationBean"/>
  </beans>
```

```
</module>
```

94.8 Sample of WaitTaskBean

Java Code

```
public class MyWaitTaskBean extends WaitTaskBean
{
    @Override
    public void onStart(WaitTaskOnStartContext aContext)
    {
        Map<String, String> params = new HashMap<String, String>();
        params.put("resumeId", aContext.getResumeId());
        myMethod.callWebService(params);
    }

    @Override
    public void onResume(WaitTaskOnResumeContext aContext) throws OperationException
    {
        // Defines a specific mapping.
        aContext.setVariableString("code", aContext.getOutputParameters().get("isbn"));
        aContext.setVariableString("comment", aContext.getOutputParameters().get("isbnComment"));
    }
}
```

Voir aussi ***com.orchestranetworks.workflow.WaitTaskBean*** *WaitTaskBean*^{API}

Configuration through module.xml

WaitTaskBean bean must be declared in module.xml:

```
<module>
  <beans>
    <bean className="com.orchestranetworks.workflow.test.MyWaitTaskBean"/>
  </beans>
</module>
```

94.9 Sample of ActionPermissionsOnWorkflow

Java Code

```
package com.orchestranetworks.workflow.test;

import com.orchestranetworks.service.*;
import com.orchestranetworks.workflow.*;
import com.orchestranetworks.workflow.ProcessExecutionContext.*;

/**
 */
public class MyDynamicPermissions extends ActionPermissionsOnWorkflow
{
    public ActionPermission getActionPermission(
        WorkflowPermission aWorkflowAction,
        ActionPermissionsOnWorkflowContext aContext)
    {
        if (WorkflowPermission.VIEW.equals(aWorkflowAction)
            || WorkflowPermission.CREATE_PROCESS.equals(aWorkflowAction))
            return ActionPermission.getEnabled();
        return ActionPermission.getDisabled();
    }
}
```

Voir aussi ***com.orchestranetworks.workflow.ActionPermissionsOnWorkflow*** *ActionPermissionsOnWorkflow*^{API}

Configuration through module.xml

ActionPermissionsOnWorkflow bean must be declared in module.xml:

```
<module>
  <beans>
    <bean className="com.orchestranetworks.workflow.test.MyDynamicPermissions"/>
  </beans>
</module>
```

94.10 Sample of WorkflowTriggerBean

Java Code

```
public class MyWorkflowTriggerBean extends WorkflowTriggerBean
{
    @Override
    public void handleAfterProcessInstanceStart(
        WorkflowTriggerAfterProcessInstanceStartContext aContext) throws OperationException
    {
        final DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());
        final MailSpec spec = aContext.createMailSpec();
        spec.notify(NotificationType.TO, "supervisor@mail.com");

        spec.setSubject("[TRIGGER] After process instance start");
        spec.setBody("The workflow '"
            + policy.formatUserMessage(aContext.getProcessInstance().getLabel())
            + "' has been created.");

        spec.sendMail(Locale.US);
    }

    @Override
    public void handleBeforeProcessInstanceTermination(
        WorkflowTriggerBeforeProcessInstanceTerminationContext aContext) throws OperationException
    {
        final DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

        final MailSpec spec = aContext.createMailSpec();
        spec.notify(NotificationType.TO, "supervisor@mail.com");

        spec.setSubject("[TRIGGER] Before process instance termination");
        spec.setBody("The workflow '"
            + policy.formatUserMessage(aContext.getProcessInstance().getLabel())
            + "' has been completed. The created product is: '"
            + aContext.getVariableString(NppConstants.VAR_CODE) + "'.");

        spec.sendMail(Locale.US);
    }

    @Override
    public void handleAfterWorkItemCreation(WorkflowTriggerAfterWorkItemCreationContext aContext)
        throws OperationException
    {
        DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

        MailSpec spec = aContext.createMailSpec();
        spec.notify(NotificationType.TO, "supervisor@mail.com");

        spec.setSubject("[TRIGGER] After work item creation");
        WorkItem workItem = aContext.getWorkItem();
        State state = workItem.getState();
        String body = "The work item '" + policy.formatUserMessage(workItem.getLabel())
            + "' has been created. \n The step id is : " + aContext.getCurrentStepId()
            + ". \n The work item is in state : " + policy.formatUserMessage(state.getLabel());

        if (workItem.getOfferedTo() != null)
            body += "\n The role is : " + workItem.getOfferedTo().format();
        if (workItem.getUserReference() != null)
            body += "\n The user is : " + workItem.getUserReference().format();

        spec.setBody(body);

        spec.sendMail(Locale.US);
    }

    @Override
```

```

public void handleBeforeWorkItemStart(WorkflowTriggerBeforeWorkItemStartContext aContext)
    throws OperationException
{
    DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

    MailSpec spec = aContext.createMailSpec();
    spec.notify(NotificationType.TO, "supervisor@mail.com");

    spec.setSubject("[TRIGGER] Before work item start");
    spec.setBody("The work item '"
        + policy.formatUserMessage(aContext.getWorkItem().getLabel())
        + "' has been started. \n The current step id is : "
        + aContext.getCurrentStepId()
        + ". \n The work item user is: '"
        + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
            aContext.getWorkItem().getUserReference(),
            aContext.getSession().getLocale()) + "'");

    spec.sendMail(Locale.US);
}

@Override
public void handleBeforeWorkItemAllocation(
    WorkflowTriggerBeforeWorkItemAllocationContext aContext) throws OperationException
{
    DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

    MailSpec spec = aContext.createMailSpec();
    spec.notify(NotificationType.TO, "supervisor@mail.com");

    spec.setSubject("[TRIGGER] Before work item allocation");
    spec.setBody("The work item '"
        + policy.formatUserMessage(aContext.getWorkItem().getLabel())
        + "' has been allocated. \n The current step id is : "
        + aContext.getCurrentStepId()
        + ". \n The work item user is: '"
        + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
            aContext.getUserReference(),
            aContext.getSession().getLocale()) + "'");

    spec.sendMail(Locale.US);
}

@Override
public void handleBeforeWorkItemDeallocation(
    WorkflowTriggerBeforeWorkItemDeallocationContext aContext) throws OperationException
{
    DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

    MailSpec spec = aContext.createMailSpec();
    spec.notify(NotificationType.TO, "supervisor@mail.com");

    spec.setSubject("[TRIGGER] Before work item deallocation");
    spec.setBody("The work item '"
        + policy.formatUserMessage(aContext.getWorkItem().getLabel())
        + "' has been deallocated. \n The current step id is : "
        + aContext.getCurrentStepId()
        + ". \n The old work item user is: '"
        + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
            aContext.getWorkItem().getUserReference(),
            aContext.getSession().getLocale()) + "'");

    spec.sendMail(Locale.US);
}

@Override
public void handleBeforeWorkItemReallocation(
    WorkflowTriggerBeforeWorkItemReallocationContext aContext) throws OperationException
{
    DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

    MailSpec spec = aContext.createMailSpec();
    spec.notify(NotificationType.TO, "supervisor@mail.com");

    spec.setSubject("[TRIGGER] Before work item reallocation");
    spec.setBody("The work item '"
        + policy.formatUserMessage(aContext.getWorkItem().getLabel())
        + "' has been reallocated. \n The current step id is : "
        + aContext.getCurrentStepId()
        + ". \n The work item user is: '"
        + DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
            aContext.getUserReference(),
            aContext.getSession().getLocale())
        + "' The old work item user is: '"
    );
}

```

```

+ DirectoryHandler.getInstance(aContext.getRepository()).displayUser(
aContext.getWorkItem().getUserReference(),
aContext.getSession().getLocale()) + "'.");

spec.sendMail(Locale.US);

}
@Override
public void handleBeforeWorkItemTermination(
WorkflowTriggerBeforeWorkItemTerminationContext aContext) throws OperationException
{
DisplayPolicy policy = DisplayPolicyFactory.getPolicyForSession(aContext.getSession());

MailSpec spec = aContext.createMailSpec();
spec.notify(NotificationType.TO, "supervisor@mail.com");

spec.setSubject("[TRIGGER] Before work item termination");
spec.setBody("The work item '"
+ policy.formatUserMessage(aContext.getWorkItem().getLabel())
+ "' has been terminated. \n The current step id is: " + aContext.getCurrentStepId()
+ ". \n The work item has been accepted ? " + aContext.isAccepted());

spec.sendMail(Locale.US);
}
}

```

Voir aussi *com.orchestranetworks.workflow.WorkflowTriggerBean* *WorkflowTriggerBean*^{API}

Configuration through module.xml

WorkflowTriggerBean bean must be declared in module.xml:

```

<module>
  <beans>
    <bean className="com.orchestranetworks.workflow.test.MyWorkflowTriggerBean"/>
  </beans>
</module>

```

94.11 Sample of trigger starting a process instance

Sample

```

public class TriggerWorkflow extends TableTrigger
{
    public void handleAfterModify(AfterModifyOccurrenceContext aContext) throws OperationException
    {
        ValueContext currentRecord = aContext.getOccurrenceContext();
        String code = (String) currentRecord.getValue(Path.parse("/code"));

        //Get published process
        PublishedProcessKey processPublishedKey = PublishedProcessKey.forName("productProcess");
        //Defines process instance
        ProcessLauncher launcher = ProcessLauncherHelper.createLauncher(
            processPublishedKey,
            aContext.getProcedureContext());
        //initialize Data Context
        launcher.setInputParameter("code", "/root/Client[.code=\"\" + code + "\"]");
        launcher.setInputParameter("workingBranch", aContext.getAdaptationHome().getKey().getName());

        //Starts process
        launcher.launchProcess();
    }
    //...
}

```

User interface

CHAPITRE 95

Interface customization

The TIBCO EBX® graphical interface can be customized through various EBX® APIs.

Ce chapitre contient les sections suivantes :

1. [Using EBX® as a Web Component](#)
2. [Adding user services](#)
3. [Customizing forms](#)
4. [Customizing widgets](#)
5. [Customizing table filter](#)
6. [Customizing record label](#)
7. [Including CSS and JavaScript](#)

95.1 Using EBX® as a Web Component

EBX® can be integrated into any application that is accessible through a [supported web browser](#) [p 332], thanks to the Web Component API.

A typical use is to integrate EBX® views into an organization's intranet framework. Web Components can also be invoked from the EBX® user interface using [User services](#) [p 658].

To embed all or part of EBX® in a web page, the HTML tag <iframe> should be used by indicating the URL to EBX®. This URL can be specified either manually or by using the `UIHttpManagerComponent` API. A single web page may include several iframes that integrate EBX®. It is then possible to create a portal made of tables, forms, hierarchical views, etc., from EBX®.

Voir aussi [Using TIBCO EBX® as a Web Component](#) [p 229]

95.2 Adding user services

A user service is an extension of EBX® that provides a graphical user interface (GUI) that allows users to access specific or advanced functions.

Powerful custom user services can be developed using the same visual components and data validation mechanisms as standard EBX® user interfaces.

Voir aussi [User service overview](#) [p 661]

95.3 Customizing forms

It is possible to override the default layout and behavior of forms in the user interface by using various tools and API.

Custom forms editor

It is possible to override the default layout of forms in the user interface by using the 'Custom forms' extension in the DMA. This extension provides a graphical editor, which gives the possibility to customize the layout of a form, while having the same components and standard behavior as record forms using the default layout.

Voir aussi [Custom forms](#) [p 609]

Programmatic form layout

It is also possible to use the `UIForm` API to override the default layout of forms. This API provides the standard input components from EBX®, which give the possibility to customize the layout of a form, while having the same components and standard behavior as record forms using the default layout.

Voir aussi

`UIFormAPI`

`UIFormPanewriterAPI`

`UIWidgetAPI`

`UIFormHeaderAPI`

`UIFormBottomBarAPI`

User service as form layout

It is also possible to use user services to override the default layout of forms. This API gives the possibility to customize the layout of a form, while having the same components as record forms using the default layout, with a customizable behavior.

Voir aussi

`UserServiceRecordFormFactoryAPI`

[Overview](#) [p 661]

95.4 Customizing widgets

Custom widgets are included in the Java API to allow the development of user interface components for fields or groups of fields. A custom widget (`UICustomWidget`) allows, for a given node, to control the area where the input or display component is located. This allows having an input and display component that is fully customizable in HTML. The standard components (`UIwidgets`) are available and can be used. The custom widget can implement several display aspects: input component in the

form, display in the form, display in a table cell. If a custom widget writes its own HTML components, it has the possibility to save the value in the database when submitting the form.

Voir aussi *UICustomWidget*^{API}

95.5 Customizing table filter

A table filter allows, for a given table, to create a criteria input form in order to apply a filter to the table view. The `UITableFilter` API is used to implement a table filter with a custom UI. It provides methods to create a UI that automatically adapts to the underlying data format (for example, by displaying a combo box when applicable).

Voir aussi

UITableFilter^{API}

[*Propriétés des éléments du modèle de données*](#) [p 53]

UILabelRendererForHierarchy^{API}

95.6 Customizing record label

EBX® uses a label to display a reference to a given record (for example a foreign key). Labels are also used in the title of a record form and in hierarchical views. This label can be customized in the model using expressions. It is also possible to customize labels using the `UILabelRenderer` API.

Voir aussi *UILabelRenderer*^{API}

95.7 Including CSS and JavaScript

It is possible to integrate CSS and JavaScript files in each EBX® page by declaring them in the registration module. The inclusion of JavaScript files can be subject to conditions through development depending on the context.

Voir aussi

[*Module registration*](#) [p 514]

[*Development recommendations*](#) [p 689]

UIDependencyRegisterer^{API}

CHAPITRE 96

Overview

A user service is an extension to TIBCO EBX® that provides a graphical user interface (GUI) allowing users to access specific or advanced functionalities.

An API is available allowing the development of powerful custom user services using the same visual components and data validation mechanisms as standard EBX® user interfaces.

Ce chapitre contient les sections suivantes :

1. [Nature](#)
2. [Declaration](#)
3. [Display](#)
4. [Legacy user services](#)

96.1 Nature

User services exist in different types called *natures*. The nature defines the minimal elements (dataspace, dataset, table, record...) that need to be selected to execute the service. The following table lists the available natures.

Nature	Description
Dataspace	The nature of a user service that can be launched from the actions menu of a dataspace (branch or snapshot) or from any context where the current selection implies selecting a dataspace.
Dataset	The nature of a user service that can be launched from the actions menu of a dataset or from any context where the current selection implies selecting a dataset.
TableView	The nature of a user service that can be launched from the toolbar of a table, regardless of the selected view, or from any context where the current selection implies selecting a table.
Record	The nature of a user service that can be launched from the toolbar of a record form or from any context where the current selection implies selecting a <i>single</i> record.
Hierarchy	The nature of a user service that can be launched from the toolbar of a table when a hierarchy view is selected.
HierarchyNode	The nature of a user service that can be launched from the menu of a table hierarchy view node. Currently, only <i>record</i> hierarchy nodes are supported.
Association	The nature of a user service that can be launched from the target table view of an association or for any context where the current selection implies selecting the target table view of an association.
AssociationRecord	The nature of a user service that can be launched from the form of a target record of an association node or from any context where the current selection implies selecting a <i>single</i> association target record.

96.2 Declaration

A user service can be declared at two levels:

- Module,
- Data model.

A service declared by a data model can only be launched when the current selection includes a dataset of this model. The user service cannot be of the Dataspace nature.

A service declared by a module may be launched for any dataspace or dataset.

The declaration can add restrictions on selections that are valid for the user service.

96.3 Display

On the following figure are displayed the functional areas of a user service.

The screenshot shows a user service form for 'John Doe'. The form is divided into two tabs: 'Main' and 'Groups'. The 'Main' tab is active, showing fields for Last Name, First Name, Start Date, Country, Job, Email, Phone, and Mobile Phone. At the bottom are buttons for Save, Save and close, Revert, and Close. The form is annotated with red numbers 1 through 9 and letters A and B. A vertical red line separates the form into two sections, A and B.

Annotation	Description
1	Breadcrumb
2	Message box button
3	Top toolbar
4	Navigation buttons
5	Help button
6	Close button (pop-ups only)
7	Tabs
8	Form pane (one per tab)
9	Bottom buttons

A. Header	1. Breadcrumb
B. Form	2. Message box button
	3. Top toolbar
	4. Navigation buttons
	5. Help button
	6. Close button (pop-ups only)
	7. Tabs
	8. Form pane (one per tab)
	9. Bottom buttons

Most areas are optional and customizable. Refer to [Quick start](#) [p 665], [Implementing a user service](#) [p 669] and [Declaring a user service](#) [p 683] for more details.

96.4 Legacy user services

Before the 5.8.0 version, user services were declared in XML and based on Servlet/JSP. Although this type of declaration should no longer be used, the *legacy documentation* is still available.

CHAPITRE 97

Quick start

Ce chapitre contient les sections suivantes :

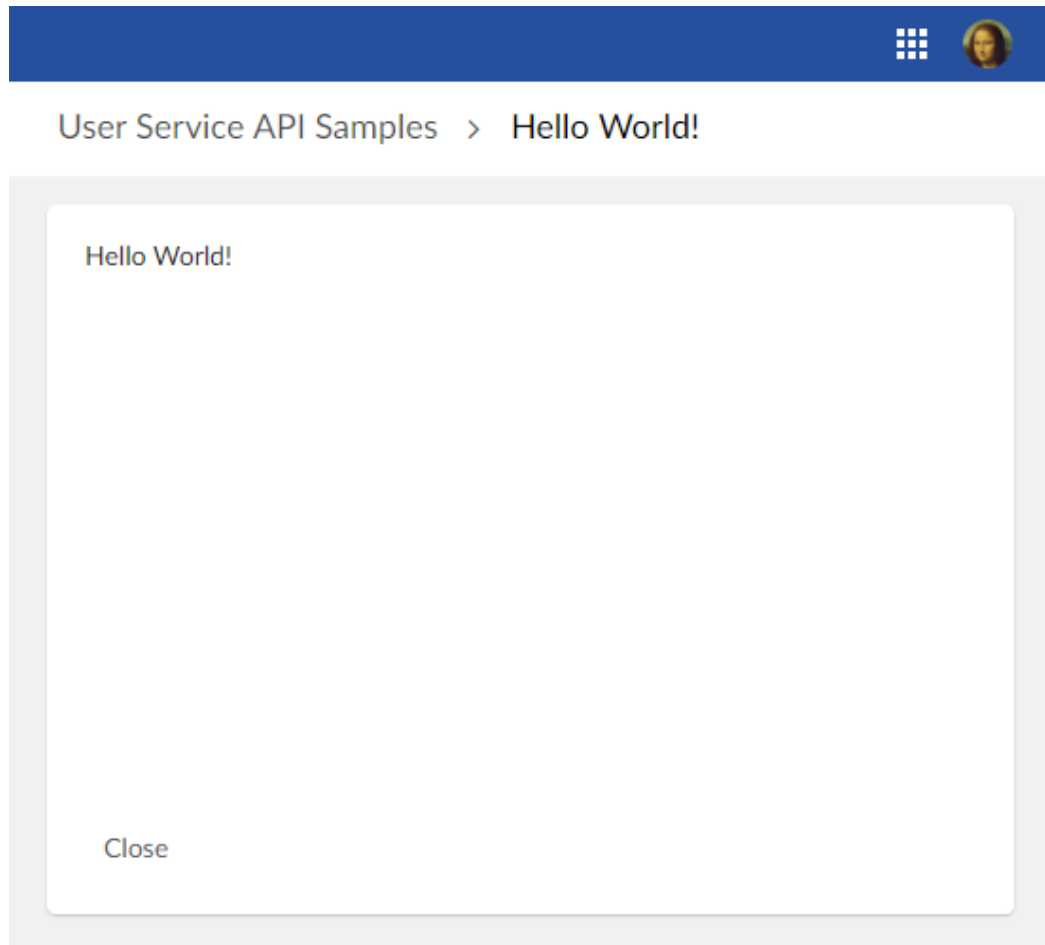
1. [Main classes](#)
2. [Hello world](#)

97.1 Main classes

The minimum requirement is to implement two classes, one for the service declaration and one for the implementation itself.

97.2 Hello world

The sample is a dataset user service that simply displays a "hello" message, it can be launched from the action menu of a dataset:



The service implementation class must implement the interface `UserService<DatasetEntitySelection>`:

```
/**
 * This service displays hello world!
 */
public class HelloWorldService implements UserService<DatasetEntitySelection>
{
    public HelloWorldService()
    {
    }

    @Override
    public void setupDisplay(
        UserServiceSetupDisplayContext<DatasetEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator)
    {
        // Set bottom bar
        UIButtonSpecNavigation closeButton = aConfigurator.newCloseButton();
        closeButton.setDefaultButton(true);
        aConfigurator.setLeftButtons(closeButton);

        // Set content callback
        aConfigurator.setContent(this::writeHelloWorld);
    }

    private void writeHelloWorld(
        UserServicePaneContext aPaneContext,
```

```

UserServicePaneWriter aWriter)
{
    // Display Hello World!

    aWriter.add("<div ");
    aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
    aWriter.add(">");
    aWriter.add("Hello World!");
    aWriter.add("</div>");
}

@Override
public void setupObjectContext(
    UserServiceSetupObjectContext<DatasetEntitySelection> aContext,
    UserServiceObjectContextBuilder aBuilder)
{
    // No context yet.
}

@Override
public void validate(UserServiceValidateContext<DatasetEntitySelection> aContext)
{
    // No custom validation is necessary.
}

@Override
public UserServiceEventOutcome processEventOutcome(
    UserServiceProcessEventOutcomeContext<DatasetEntitySelection> aContext,
    UserServiceEventOutcome anEventOutcome)
{
    // By default do not modify the outcome.
    return anEventOutcome;
}
}

```

The declaration class must implement the interface `UserServiceDeclaration.OnDataset`:

```

/**
 * Declaration for service hello world!
 */
public class HelloWorldServiceDeclaration implements UserServiceDeclaration.OnDataset
{
    // The service key identifies the user service.
    private static final ServiceKey serviceKey = ServiceKey.forName("HelloWorld");

    public HelloWorldServiceDeclaration()
    {
    }

    @Override
    public ServiceKey getServiceKey()
    {
        return serviceKey;
    }

    @Override
    public UserService<DatasetEntitySelection> createUserService()
    {
        // Creates an instance of the user service.
        return new HelloWorldService();
    }

    @Override
    public void defineActivation(ActivationContextOnDataset aContext)
    {
        // The service is activated for all datasets instantiated with
        // the associated data model (see next example).
    }

    @Override
    public void defineProperties(UserServicePropertiesDefinitionContext aContext)
    {
        // This label is displayed in menus that can execute the user service.
        aContext.setLabel("Hello World Service");
    }

    @Override
    public void declareWebComponent(WebComponentDeclarationContext aContext)
    {
    }
}

```

In this sample, the user service is registered by a data model. The data model needs to define a schema extension that implements the following code:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
    @Override
    public void defineExtensions(SchemaExtensionsContext aContext)
    {
        // Register the service.
        aContext.registerUserService(new HelloWorldServiceDeclaration());
    }
}
```

For details on the declaration of schema extensions, see `SchemaExtensionsAPI`.

CHAPITRE 98

Implementing a user service

Ce chapitre contient les sections suivantes :

1. [Implementation interface](#)
2. [Life cycle and threading model](#)
3. [Object Context](#)
4. [Display setup](#)
5. [Database updates](#)
6. [Ajax](#)
7. [REST data services](#)
8. [File upload](#)
9. [File download](#)
10. [User service without display](#)

98.1 Implementation interface

The following table lists, per nature, the interface to implement:

Nature	Interface
Dataspace	UserService<DataspaceEntitySelection>
Dataset	UserService<DatasetEntitySelection>
TableView	UserService<TableViewEntitySelection>
Record	UserService<RecordEntitySelection>
Hierarchy	UserService<HierarchyEntitySelection>
HierarchyNode	UserService<HierarchyNodeEntitySelection>
Association	UserService<AssociationEntitySelection>
AssociationRecord	UserService<AssociationRecordEntitySelection>

98.2 Life cycle and threading model

The user service implementation class is:

- Instantiated at the first HTTP request by a call to its declaration **createUserService()** `UserServiceDeclaration.createUserServiceAPI` method.
- Discarded when the current page goes out of scope or when the session times out.

Access to this class is synchronized by TIBCO EBX® to make sure that only one HTTP request is processed at a time. Therefore, the class does not need to be thread-safe.

The user service may have attributes. The state of these attributes will be preserved between HTTP requests. However, developers must be aware that these attributes should have moderate use of resources, such as memory, not to overload the EBX® server.

98.3 Object Context

The object context is a container for objects managed by the user service. This context is initialized and modified by the user service's implementation of the method `UserService.setupObjectContextAPI`.

An object of the object context is identified by an object key:

```
ObjectKey customerKey = ObjectKey.forName("customer");
```

An object can be:

- A record,
- A dataset,
- A new record not yet persisted,
- A dynamic object.

The object context is maintained between HTTP requests and usually only needs to be set up upon the first request.

Once persisted, a *new record* object is automatically changed to a *plain record* object.

As with **adaptations** `AdaptationAPI`, **path** `PathAPI` expressions are used to reference a sub-element of an object.

In the following sample, a pane writer adds a form input mapped to the attribute of an object:

```
// Add an input field for customer's last name.
aWriter.setCurrentObject(customerKey);
aWriter.addFormRow(Path.parse("lastName"));
```

In the following sample, an event callback gets the value of the attribute of an object:

```
// Get value of customer's last name.
ValueContext customerValueContext = aValueContext.getValueContext(customerKey);
String lastName = customerValueContext.getValue(Path.parse("lastName"));
```

A *dynamic object* is an object whose schema is defined by the user service itself. An API is provided to define the schema programmatically. This API allows defining only instance elements (instance nodes). Defining tables is not supported. It supports most other features available with standard EBX® data models, such as types, labels, custom widgets, enumerations and constraints, including programmatic ones.

The following sample defines two objects having the same schema:

```
public class SampleService implements UserService<TableViewEntitySelection>
{
    // Define an object key per object:
    private static final ObjectKey _PersonObjectKey = ObjectKey.forName("person");
    private static final ObjectKey _PartnerObjectKey = ObjectKey.forName("partner");

    // Define a path for each property:
    private static final Path _FirstName = Path.parse("firstName");
    private static final Path _LastName = Path.parse("lastName");
    private static final Path _BirthDate = Path.parse("birthDate");

    ...

    // Define and register objects:
    @Override
    public void setupObjectContext(
        UserServiceSetupObjectContext<DataspaceEntitySelection> aContext,
        UserServiceObjectContextBuilder aBuilder)
    {
        if (aContext.isInitialDisplay())
        {
            BeanDefinition def = aBuilder.createBeanDefinition();

            BeanElement firstName = def.createElement(_FirstName, SchemaTypeName.XS_STRING);
            firstName.setLabel("First name");
            firstName.setDescription("This is the given name");
            firstName.setMinOccurs(1);

            BeanElement lastName = def.createElement(_LastName, SchemaTypeName.XS_STRING);
            lastName.setLabel("Last name");
            lastName.setDescription("This is the family name");
            lastName.setMinOccurs(1);

            BeanElement birthDate = def.createElement(_BirthDate, SchemaTypeName.XS_DATE);
            birthDate.setLabel("Birth date");
            birthDate.addFacetMax(new Date(), false);

            aBuilder.registerBean(_PersonObjectKey, def);
            aBuilder.registerBean(_PartnerObjectKey, def);
        }

        ...
    }
}
```

98.4 Display setup

The display is set up by the user service's implementation of the method `UserService.setupDisplayAPI`. This method is called at each request and can set the following:

- The title (the default is the label specified by the user service declaration),
- The contextual help URL,
- The breadcrumbs,
- The toolbar,
- The bottom buttons.

If necessary, the header and the bottom buttons can be hidden.

The display setup is not persisted and, at each HTTP request, is reset to default before calling the method `UserService.setupDisplayAPI`.

Bottom buttons

Buttons may be of two types: *action* and *submit*.

An *action* button triggers an *action* event without submitting the form. By default, the user needs to acknowledge that, by leaving the page, the last changes will be lost. This behavior can be customized.

A *submit* button triggers a *submit* event that always submits the form.

More information on events can be found in the following sections.

Content callback

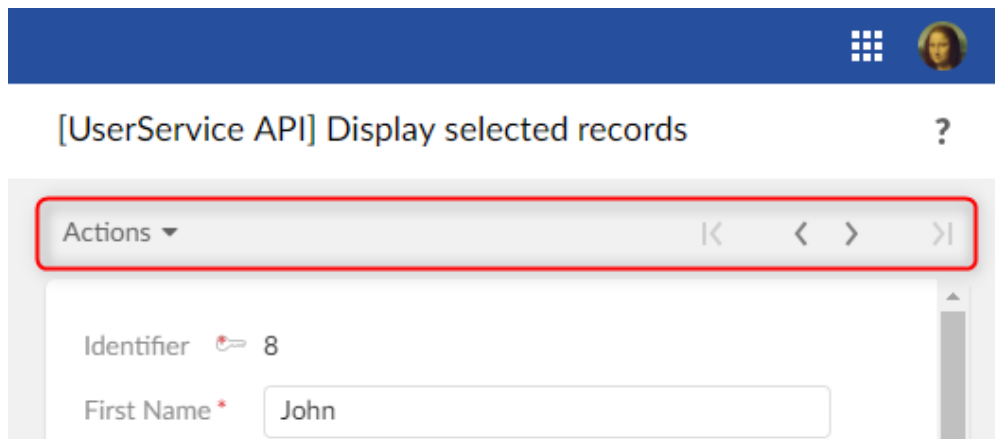
This callback usually implements the interface `UserServicePaneAPI` to render a plain EBX® form. The callback can also be an instance of `UserServiceTabbedPaneAPI` to render an EBX® form with tabs.

For specific cases, the callback can implement `UserServiceRawPaneAPI`. This interface has restrictions but is useful when one wants to implement an HTML form that is not managed by EBX®.

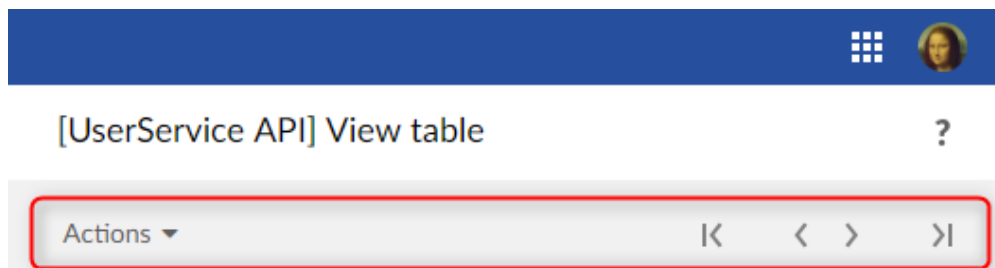
Toolbars

Toolbars are optional and come in two flavors.

The *form* style:



The *table* view style:



The style is automatically selected: toolbars defined for a *record* are of the form style and toolbars defined for a *table* are of the table view style.

Samples

The following sample implements a button that closes the current user service and redirects the user back to the current selection, only if saving the data was successful:

```
public class SampleService implements UserService<...>
{
    private static final ObjectKey _RecordObjectKey = ObjectKey.forName("record");

    ...

    @Override
    public void setupDisplay(
        UserServiceSetupDisplayContext<RecordEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator)
    {
        ...
        // Define a "save and close" button with callback onSave().
        aConfigurator.setLeftButtons(aConfigurator.newSaveCloseButton(this::onSave));
    }

    private UserServiceEventOutcome onSave(UserServiceEventContext anEventContext)
```



```

{
    ProcedureResult result = anEventContext.save(_RecordObjectKey);
    if (result.hasFailed())
    {
        // Save has failed. Redisplay the user message.
        return null;
    }

    // Save has succeeded. Close the service.
    return UserServiceNext.nextClose();
}
}

```

The following sample is compatible with the Java 6 syntax. Only differences with the previous code are shown:

```

public class SampleService implements UserService<...>
{
    ...

    @Override
    public void setupDisplay(
        UserServiceSetupDisplayContext<RecordEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator)
    {
        ...
        // Define a "save and close" button with callback onSave().
        aConfigurator.setLeftButtons(aConfigurator.newSaveCloseButton(new UserServiceEvent() {
            @Override
            public UserServiceEventOutcome processEvent(UserServiceEventContext anEventContext)
            {
                return onSave(anEventContext);
            }
        }));
    }
}

```

The following sample implements a URL that closes the service and redirects the current user to another user service:

```

public class SampleService implements UserService<...>
{
    ...
    private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
    {
        // Displays an URL that redirect current user.
        String url = aWriter.getURLForAction(this::goElsewhere);
        aWriter.add("<a ");
        aWriter.addSafeAttribute("href", url);
        aWriter.add(">Go elsewhere</a");
    }

    private UserServiceEventOutcome goElsewhere(UserServiceEventContext anEventContext)
    {
        // Redirects current user to another user service.
        ServiceKey serviceKey = ServiceKey.forModuleServiceName("CustomerModule", "CustomService");
        return UserServiceNext.nextService(serviceKey);
    }
}

```

The following code is an implementation of the method `UserService.processEventOutcome`^{API}, sufficient for simple user services:

```

public class HelloWorldService implements UserService<...>
{
    @Override
    public UserServiceEventOutcome processEventOutcome(
        UserServiceProcessEventOutcomeContext<DatasetEntitySelection> aContext,
        UserServiceEventOutcome anEventOutcome)
    {
        // By default do not modify the outcome.
        return anEventOutcome;
    }
}

```

The following sample is a more complex "wizard" service that includes three steps, each having its own `UserService.setupDisplay`^{API} method:

```

// Custom outcome values.

```

```

public enum CustomOutcome implements UserServiceEventOutcome {
    displayStep1, displayStep2, displayStep3
};

// All steps of the wizard service implement this interface.
public interface WizardStep
{
    public void setupDisplay(
        UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator);
}

// The user service implementation.
public class WizardService implements UserService<...>
{
    // Attribute for current step.
    private WizardStep step = new WizardStep1();

    ...

    @Override
    public void setupDisplay(
        UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator)
    {
        ...

        // Display current step.
        this.step.setupDisplay(aContext, aConfigurator);
    }

    @Override
    public UserServiceEventOutcome processEventOutcome(
        UserServiceProcessEventOutcomeContext<DataspaceEntitySelection> aContext,
        UserServiceEventOutcome anEventOutcome)
    {
        // Custom outcome value processing.

        if (anEventOutcome instanceof CustomOutcome)
        {
            CustomOutcome action = (CustomOutcome) anEventOutcome;
            switch (action)
            {
                case displayStep1:
                    this.step = new WizardStep1();
                    break;

                case displayStep2:
                    this.step = new WizardStep2();
                    break;

                case displayStep3:
                    this.step = new WizardStep3();
                    break;
            }

            // Redisplay the user service.
            return null;
        }

        // Let EBX® process the event outcome.
        return anEventOutcome;
    }
}

```

98.5 Database updates

An event callback may update the database.

The following sample saves two objects using a single transaction:

```

public class MultipleObjectsSampleService implements UserService<...>
{
    // This service defines a two objects having same schema.
    private static final ObjectKey _Person1_ObjectKey = ObjectKey.forName("person1");
    private static final ObjectKey _Person2_ObjectKey = ObjectKey.forName("person2");

    ...

    // Save button callback.

```

```
private UserServiceEventOutcome onSave(UserServiceEventContext aContext)
{
    ProcedureResult result = aContext.save(_Person1_ObjectKey, _Person2_ObjectKey);
    if (result.hasFailed())
    {
        //Save failed. Redisplay the service.
        //The user interface will automatically report error messages.
        return null;
    }

    // Save succeeded. Close the service.
    return UserServiceNext.nextClose();
}
}
```

The following sample updates the database using a **procedure** Procedure^{API}:

```
import com.orchestranetworks.service.*;
import com.orchestranetworks.userservice.*;

public class MultipleObjectsSampleService implements UserService<...>
{
    ...

    // Event callback.
    private UserServiceEventOutcome onUpdateSomething(UserServiceEventContext aContext)
    {
        Procedure procedure = new Procedure()
        {
            public void execute(ProcedureContext aContext) throws Exception
            {
                // Code that updates database should be here.
                ...
            }
        };

        UserServiceTransaction transaction = aContext.createTransaction();
        transaction.add(procedure);

        ProcedureResult result = transaction.execute();
        if (result.hasFailed())
        {
            aContext.addError("Procedure failed");
        }
        else
        {
            aContext.addInfo("Procedure succeeded");
        }

        return null;
    }
}
```

98.6 Ajax

A user service can implement Ajax callbacks. An Ajax callback must implement the interface UserServiceAjaxRequest^{API}.

The client calls an Ajax callback using the URL generated by: UserServiceResourceLocator.getURLForAjaxRequest^{API}.

To facilitate the use of Ajax components, EBX® provides the JavaScript prototype EBX_AJAXResponseHandler for sending the request and handling the response. For more information on EBX_AJAXResponseHandler see UserServiceAjaxRequest^{API}.

The following sample implements an Ajax callback that returns partial HTML:

```
public class AjaxSampleService implements UserService<DataspaceEntitySelection>
{
    ...

    @Override
    public void setupDisplay(
        UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator)
    {
        aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
        aConfigurator.setContent(this::writePane);
    }
}
```

```

}

/**
 * Displays an URL that will execute the callback
 * and display the returned partial HTML inside a <div> tag.
 */
private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
{
    // Generate the URL of the Ajax callback.
    String url = aWriter.getURLForAjaxRequest(this::ajaxCallback);

    // The id of the <div> that will display the partial HTML returned by the Ajax callback.
    String divId = "sampleId";

    aWriter.add("<div ");
    aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
    aWriter.add(">");

    // Display the URL that will execute the callback.
    aWriter.add("<a ");
    aWriter.addSafeAttribute("href", "javascript:sample_sendAjaxRequest('" + url + "', '"
        + divId + "')");
    aWriter.add(">");
    aWriter.add("Click to call a user service Ajax callback");
    aWriter.add("</a>");

    // Output the <div> tag that will display the partial HTML returned by the callback.
    aWriter.add("<div ");
    aWriter.addSafeAttribute("id", divId);
    aWriter.add("></div>");

    aWriter.add("</div>");

    // JavaScript method that will send the Java request.
    aWriter.addJS_cr();
    aWriter.addJS_cr("function sample_sendAjaxRequest(url, targetDivId) {");
    aWriter.addJS_cr("    var ajaxHandler = new EBX_AJAXResponseHandler();");

    aWriter.addJS_cr("    ajaxHandler.handleAjaxResponseSuccess = function(responseContent) {");
    aWriter.addJS_cr("        var element = document.getElementById(targetDivId);");
    aWriter.addJS_cr("        element.innerHTML = responseContent;");
    aWriter.addJS_cr("    }");

    aWriter.addJS_cr("    ajaxHandler.handleAjaxResponseFailed = function(responseContent) {");
    aWriter.addJS_cr("        var element = document.getElementById(targetDivId);");
    aWriter.addJS_cr("        element.innerHTML = '<span class='" + UICSSClasses.TEXT.ERROR
        + "'>Ajax call failed</span>");
    aWriter.addJS_cr("    }");

    aWriter.addJS_cr("    ajaxHandler.sendRequest(url);");
    aWriter.addJS_cr("}");
}

/**
 * The Ajax callback that returns partial HTML.
 */
private void ajaxCallback(
    UserServiceAjaxContext anAjaxContext,
    UserServiceAjaxResponse anAjaxResponse)
{
    UserServiceWriter writer = anAjaxResponse.getWriter();
    writer.add("<p style='color:green'>Ajax callback succeeded!</p>");
    writer.add("<p>Current data and time is: ");

    DateFormat format = DateFormat.getDateTimeInstance(
        DateFormat.FULL,
        DateFormat.FULL,
        Locale.US);
    writer.addSafeInnerHTML(format.format(new Date()));

    writer.add("</p>");
}
}

```

98.7 REST data services

A user service can access REST data services through HTTP requests.

The client should use the URL generated by: `UIResourceLocator.getURLForRestAPI`. This URL includes required information for the user authentication.

For more information on REST data services see the [Built-in RESTful services](#) [p 757].

The following sample implements a REST data service call whose response is printed in a textarea:

```
public class RestCallSampleService implements UserService<DataspaceEntitySelection>
{
    ...

    @Override
    public void setupDisplay(
        UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator)
    {
        aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
        aConfigurator.setContent(this::writePane);
    }

    private void writePane(UserServicePaneContext aPaneContext, UserServicePaneWriter aWriter)
    {
        // Generates the URL for REST data service call without additional parameters
        final String url = aWriter.getURLForRest("/ebx-dataseservices/rest/{specificPath}", null);

        final String resultAreaId = "restResult";

        // Displays a link for REST data service call
        aWriter.add("<div ");
        aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
        aWriter.add(">");
        aWriter.add("<p>This link will display the response after making a REST call</p>");
        aWriter.add("<a ");
        aWriter.addSafeAttribute("href",
            "javascript:sendRestRequest('" + url + "', '" + resultAreaId + "')");
        aWriter.add(">");
        aWriter.add("Make the call.");
        aWriter.add("</a>");
        aWriter.add("<textarea ");
        aWriter.addSafeAttribute("id", resultAreaId);
        aWriter.add(" readonly=\"readonly\" style=\"width: 100%;\" ></textarea>");
        aWriter.add("</div>");

        // JavaScript method that will send the HTTP REST request
        aWriter.addJS_cr("function sendRestRequest(url, targetId) {");
        aWriter.addJS_cr("    var xhttp = new XMLHttpRequest();");
        aWriter.addJS_cr("    xhttp.open('GET', url, true);");
        aWriter.addJS_cr("    xhttp.setRequestHeader('Content-type', 'application/json');");
        aWriter.addJS_cr("    xhttp.send();");
        aWriter.addJS_cr("    var element = document.getElementById(targetId);");
        aWriter.addJS_cr("    xhttp.onreadystatechange = function() {");
        aWriter.addJS_cr("        if (xhttp.readyState == 4)");
        aWriter.addJS_cr("            element.innerHTML = xhttp.responseText;");
        aWriter.addJS_cr("    }");
        aWriter.addJS_cr("}");
    }
}
```

98.8 File upload

A user service can display forms with file input fields.

The following sample displays a form with two input fields, a title and a file:

```
public class FileUploadService implements UserService<...>
{
    // This service defines a single object named "file".
    private static final ObjectKey _File_ObjectKey = ObjectKey.forName("file");

    // Paths for the "file" object.
    public static final Path _Title = Path.parse("title");
    public static final Path _File = Path.parse("file");

    ...

    @Override
    public void setupObjectContext(
        UserServiceSetupObjectContext<DataspaceEntitySelection> aContext,
        UserServiceObjectContextBuilder aBuilder)
    {
        if (aContext.isInitialDisplay())
        {
            // Create a definition for the "model" object.
        }
    }
}
```

```

BeanDefinition def = aBuilder.createBeanDefinition();
aBuilder.registerBean(_File_ObjectKey, def);

BeanElement element;

element = def.createElement(_Title, SchemaTypeName.XS_STRING);
element.setLabel("Title");
element.setMinOccurs(1);

// Type for a file must be BeanDefinition.OSD_FILE_UPLOAD.
element = def.createElement(_File, BeanDefinition.OSD_FILE_UPLOAD);
element.setLabel("File");
element.setMinOccurs(1);
}
}

@Override
public void setupDisplay(
    UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
    UserServiceDisplayConfigurator aConfigurator)
{
    aConfigurator.setTitle("File upload service");
    aConfigurator.setLeftButtons(aConfigurator.newSubmitButton("Upload", this::onUpload),
    aConfigurator.newCancelButton());

    // IMPORTANT: Following method must be called to enable file upload.
    // This will set form encryption type to "multipart/form-data".
    aConfigurator.setFileUploadEnabled(true);

    aConfigurator.setContent(this::writePane);
}

private void writePane(UserServicePaneContext aContext, UserServicePaneWriter aWriter)
{
    final UIWidgetFileUploadFactory fileUploadFactory = new UIWidgetFileUploadFactory();

    aWriter.setCurrentObject(_File_ObjectKey);

    aWriter.startTableFormRow();

    // Title input.
    aWriter.addFormRow(_Title);

    // File upload input.
    UIWidgetFileUpload widget = aWriter.newCustomWidget(_File, fileUploadFactory);
    // Default filter for file names.
    widget.setAccept(".txt");
    aWriter.addFormRow(widget);

    aWriter.endTableFormRow();
}

private UserServiceEventOutcome onUpload(UserServiceEventContext anEventContext)
{
    ValueContextForInputValidation valueContext = anEventContext.getValueContext(_File_ObjectKey);

    String title = (String) valueContext.getValue(_Title);
    UploadedFile file = (UploadedFile) valueContext.getValue(_File);

    InputStream in;
    try
    {
        in = file.getInputStream();
    }
    catch (IOException e)
    {
        // Should not happen.
        anEventContext.addError("Cannot read file.");
        return null;
    }

    // Do something with title and the input stream.
    return UserServiceNext.nextClose();
}
}

```

For more information, see `UIWidgetFileUploadAPT`.

98.9 File download

A user service can display URLs or buttons to download files. The actual downloading of a file is under the control of the user service.

The following sample displays a URL to download a file:

```
public class FileDownloadService implements UserService<DataspaceEntitySelection>
{
    ...

    @Override
    public void setupDisplay(
        UserServiceSetupDisplayContext<DataspaceEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator)
    {
        aConfigurator.setLeftButtons(aConfigurator.newCloseButton());
        aConfigurator.setContent(this::writePane);
    }

    private void writePane(UserServicePaneContext aContext, UserServicePaneWriter aWriter)
    {
        aWriter.add("<div ");
        aWriter.addSafeAttribute("class", UICSSClasses.CONTAINER_WITH_TEXT_PADDING);
        aWriter.add(">");

        // Generate and display the URL for the download.
        String downloadURL = aWriter.getURLForGetRequest(this::processDownloadRequest);

        aWriter.add("<a ");
        aWriter.addSafeAttribute("href", downloadURL);
        aWriter.add(">Click here to download a sample file</a>");

        aWriter.add("</div>");
    }

    private void processDownloadRequest(
        UserServiceGetContext aContext,
        UserServiceGetResponse aResponse)
    {
        // The file is plain text.
        aResponse.setContentType("text/plain;charset=UTF-8");
        // Remove the following statement to display the file directly in the browser.
        aResponse.setHeader("Content-Disposition", "attachment; filename=\"sample.txt\"");

        // Write a text file using UTF-8 encoding.
        PrintWriter out;
        try
        {
            out = new PrintWriter(new OutputStreamWriter(aResponse.getOutputStream(), "UTF-8"));
        }
        catch (IOException ex)
        {
            throw new RuntimeException(ex);
        }

        DateFormat format = DateFormat.getDateInstance(
            DateFormat.FULL,
            DateFormat.MEDIUM,
            Locale.US);
        Date now = new Date();

        out.println("Hello !");
        out.println("This is a sample text file downloaded on " + format.format(now)
            + ", from EBX®.");

        out.close();
    }
}
```

98.10 User service without display

A user service may be designed to execute a task without display and return to the previous screen or redirect the user to another screen.

This type of service must implement the interface **UserServiceExtended** `UserServiceExtendedAPI` and method `UserServiceExtended.initializeAPI`.

The following sample deletes selected records in the current table view:

```
public class DeleteRecordsService implements UserServiceExtended<TableViewEntitySelection>
{
    ...

    @Override
    public UserServiceEventOutcome initialize(
        UserServiceInitializeContext<TableViewEntitySelection> aContext)
    {
        final List<AdaptationName> records = new ArrayList<>();

        // Deletes all selected rows in a single transaction.
        RequestResult requestResult = aContext.getEntitySelection().getSelectedRecords().execute();
        try
        {
            for (Adaptation record = requestResult.nextAdaptation(); record != null; record =
requestResult.nextAdaptation())
            {
                records.add(record.getAdaptationName());
            }
        }
        finally
        {
            requestResult.close();
        }

        Procedure deleteProcedure = new Procedure()
        {
            @Override
            public void execute(ProcedureContext aContext) throws Exception
            {
                for (AdaptationName record : records)
                {
                    aContext.doDelete(record, false);
                }
            }
        };

        UserServiceTransaction transaction = aContext.createTransaction();
        transaction.add(deleteProcedure);

        // Adds an information messages for current user.
        ProcedureResult procedureResult = transaction.execute(true);
        if (!procedureResult.hasFailed())
        {
            if (records.size() <= 1)
            {
                aContext.addInfo(records.size() + " record was deleted.");
            }
            else
            {
                aContext.addInfo(records.size() + " records were deleted.");
            }
        }

        // Do not display the user service and return to current view.
        return UserServiceNext.nextClose();
    }

    @Override
    public void setupObjectContext(
        UserServiceSetupObjectContext<TableViewEntitySelection> aContext,
        UserServiceObjectContextBuilder aBuilder)
    {
        //Do nothing.
    }

    @Override
    public void setupDisplay(
        UserServiceSetupDisplayContext<TableViewEntitySelection> aContext,
        UserServiceDisplayConfigurator aConfigurator)
    {
        //Do nothing.
    }

    @Override
    public void validate(UserServiceValidateContext<TableViewEntitySelection> aContext)
    {
        //Do nothing.
    }
}
```



```
}  
  
@Override  
public UserServiceEventOutcome processEventOutcome(  
    UserServiceProcessEventOutcomeContext<TableViewEntitySelection> aContext,  
    UserServiceEventOutcome anEventOutcome)  
{  
    return anEventOutcome;  
}  
}
```

Known limitation

If such service is called in the context of a Web component, an association, a perspective action or a hierarchy node, The service will be launched, initialized and closed, but the service's target entity will still be displayed.

CHAPITRE 99

Declaring a user service

Ce chapitre contient les sections suivantes :

1. [Declaration interface](#)
2. [Life cycle and threading model](#)
3. [Registration](#)
4. [Service properties](#)
5. [Service activation scope](#)
6. [Web component declaration](#)
7. [User service groups](#)

99.1 Declaration interface

The following table lists, per nature, the interface that the declaration class of a user service must implement:

Nature	Declaration Interface
Dataspace	UserServiceDeclaration.OnDataspace
Dataset	UserServiceDeclaration.OnDataset
TableView	UserServiceDeclaration.OnTableView
Record	UserServiceDeclaration.OnRecord
Hierarchy	UserServiceDeclaration.OnHierarchy
HierarchyNode	UserServiceDeclaration.OnHierarchyNode
Association	UserServiceDeclaration.OnAssociation
AssociationRecord	UserServiceDeclaration.OnAssociationRecord

99.2 Life cycle and threading model

The user service declaration class is instantiated at the TIBCO EBX® startup and must be coded to be thread-safe. This is usually not an issue as most implementations should be immutable classes.

99.3 Registration

A user service declaration must be registered by a module or a data model.

Registration by a module is achieved by the module registration servlet by a code similar to:

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
    @Override
    public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
    {
        // Register custom user service declaration.
        aContext.registerUserService(new CustomServiceDeclaration());
    }
}
```

For more information on the module registration servlet, see [module registration](#) [p 514] and `ModuleRegistrationServlet`^{API}.

Registration by a data model is achieved by a code similar to:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
    @Override
    public void defineExtensions(SchemaExtensionsContext aContext)
    {
        // Register custom user service declaration.
        aContext.registerUserService(new CustomServiceDeclaration());
    }
}
```

For more information on data model extensions, see `SchemaExtensions`^{API}.

99.4 Service properties

The properties of a user service include its *label*, *description*, *confirmation message* and the *group* that owns the service. All are optional but it is a good practice to at least define the label.

For more information, see `UserServiceDeclaration.defineProperties`^{API}.

99.5 Service activation scope

The activation scope defines on which selection the service is available.

Example of a service activation definition:

```
public class CustomServiceDeclaration implements UserServiceDeclaration.OnTableView
{
    ...

    @Override
    public void defineActivation(ActivationContextOnTableView aContext)
    {
        // activates the service in all dataspace except the "Reference" branch.
        aContext.includeAllDataspace(DataspaceType.BRANCH);
        aContext.excludeDataspaceMatching(Repository.REFERENCE, DataspaceChildrenPolicy.NONE);

        // activates the service only on tables "table01" and "table03".
        aContext.includeSchemaNodesMatching(
            CustomDataModelPath._Root_Table01.getPathInSchema(),
            CustomDataModelPath._Root_Table03.getPathInSchema());

        // service will be enabled only when at least one record is selected.
    }
}
```

```

aContext.forbidEmptyRecordSelection();

// service will not be displayed in hierarchical views (neither in the
// top toolbar, nor in the hierarchy nodes' menu).
aContext.setDisplayForLocations(
    ActionDisplaySpec.HIDDEN,
    ToolbarLocation.HIERARCHICAL_VIEW_TOP,
    ToolbarLocation.HIERARCHICAL_VIEW_NODE);

// service will be considered as disabled if not explicitly enabled
// via the UI.
aContext.setDefaultPermission(UserServicePermission.getDisabled());
}
}

```

For more information about declaring the activation scope, see `UserServiceDeclaration.defineActivationAPI`.

For more information about the resolution of the user service availability, see [Resolving permissions on services](#) [p 307].

99.6 Web component declaration

Parameters declaration and availability in workflows and perspectives

User services are automatically available as web components with a set of built-in parameters depending on the service's nature. To define custom parameters and/or set the service web component as available when configuring a workflow user task, a perspective menu action or a toolbar web component action, `UserServiceDeclaration.declareWebComponentAPI` must be used.

Example of a web component declaration:

```

public class CustomServiceDeclaration implements UserServiceDeclaration.OnDataset
{
    ...

    @Override
    public void declareWebComponent(WebComponentDeclarationContext aContext)
    {
        // makes this web component available when configuring a workflow user task.
        aContext.setAvailableAsWorkflowUserTask(true);

        // adds a custom input parameter.
        aContext.addInputParameter(
            "source",
            UserMessage.createInfo("Source"),
            UserMessage.createInfo("Source of the imported data.));

        // modifies the built-in "instance" parameter to be "input/output" instead of "input".
        aContext.getBuiltInParameterForOverride("instance").setOutput(true);
    }
}

```

See [Using TIBCO EBX® as a Web Component](#) [p 229] for more information.

User service extension

It is possible to extend existing user services (built-in or custom) in order to add input/output parameters when using these services as web components.

In order to do so, a user service extension must first be registered by a module or a data model.

Registration by a module is achieved by the module registration servlet by code similar to:

```

public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
    ...

    @Override
    public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
    {
        ...
    }
}

```

```
{
    // Register user service extension declaration.
    aContext.registerUserServiceExtension(new ServiceExtensionDeclaration());
}
}
```

For more information on the module registration servlet, see [module registration](#) [p 514] and `ModuleRegistrationServlet`^{API}.

Registration by a data model is achieved by a code similar to:

```
public class CustomSchemaExtensions implements SchemaExtensions
{
    ...

    @Override
    public void defineExtensions(SchemaExtensionsContext aContext)
    {
        // Register user service extension declaration.
        aContext.registerUserServiceExtension(new ServiceExtensionDeclaration());
    }
}
```

For more information on the data model extension, see `SchemaExtensions`^{API}.

99.7 User service groups

User service groups are used to organize the display of user services in menus and permission management screens.

The following types of service groups are available:

- [Built-in User Service Groups](#) [p 686] provided by EBX®,
- [Custom User Service Groups](#) [p 687] declared in a module.

The link between groups and services is made upon service declaration. See [Associating a service to a group](#) [p 687].

Built-in User Service Groups

Available built-in service groups:

Service Group Key	Description
@ebx-importExport	Group containing all built-in import and export services provided by EBX®. In the default menus, these services are displayed in an "Import / Export" sub-menu.
@ebx-views	Group containing services to display in the 'Views' menu. Unlike other service groups, services associated with this group are not displayed in the default menus, but only in the 'Views' menu displayed in the non-customizable part of the table top toolbar. These services can still be added manually to a custom toolbar.

Declaring a User Service Group

User Service Groups must be declared while registering the module, using the method `ModuleServiceRegistrationContext.registerServiceGroup`^{API}:

```
public class CustomRegistrationServlet extends ModuleRegistrationServlet
{
    ...

    @Override
    public void handleServiceRegistration(ModuleServiceRegistrationContext aContext)
    {
        // In CustomModuleConstants,
        // CUSTOM_SERVICE_GROUP_KEY = ServiceGroupKey.forServiceGroupInModule("customModule", "customGroup")

        // registers CUSTOM_SERVICE_GROUP_KEY service group
        aContext.registerServiceGroup(
            CustomModuleConstants.CUSTOM_SERVICE_GROUP_KEY,
            UserMessage.createInfo("Custom group"),
            UserMessage.createInfo("This group contains services related to..."));
    }
}
```

Associating a service to a group

The association of a service with a group is made at its **declaration** `UserServiceDeclaration`^{API}, using the method `UserServicePropertiesDefinitionContext.setGroup`^{API}:

```
public class CustomServiceDeclaration implements UserServiceDeclaration.OnDataset
{
    ...

    @Override
    public void defineProperties(UserServicePropertiesDefinitionContext aContext)
    {
        // associates the current service to the CUSTOM_SERVICE_GROUP_KEY group
        aContext.setGroup(CustomModuleConstants.CUSTOM_SERVICE_GROUP_KEY);
    }
}
```

A service can be associated with either a built-in or a custom service group. In the latter case, this service will be displayed in this built-in group, just like other built-in services belonging to this group.

CHAPITRE 100

Development recommendations

Ce chapitre contient les sections suivantes :

1. [HTML](#)
2. [CSS](#)
3. [JavaScript](#)

100.1 HTML

It is recommended to minimize the inclusion of specific HTML styles and tags to allow the default styles of TIBCO EBX® to apply to custom interfaces. The approach of the API is to automatically apply a standardized style to all elements on HTML pages, while simplifying the implementation process for the developer.

XHTML

EBX® is a Rich Internet Application developed in XHTML 1.0 Transitional. It means that the structure of the HTML is strict XML file and that all tags must be closed, including "br" tags. This structure allows for greater control over CSS rules, with fewer differences in browser rendering.

iFrames

Using iFrame is allowed in EBX®, especially in collaboration with a URL of a `UIHttpManagerComponentAPI`. For technical reasons, it is advised to set the `src` attribute of an iFrame using JavaScript only. In this way, the iFrame will be loaded once the page is fully rendered and when all the built-in HTML components are ready.

Example

The following example, developed from any `UIComponentWriterAPI`, uses a `UIHttpManagerComponentAPI` to build the URL of an iFrame, and set it in the right way:

```
// using iFrame in the current page requires a sub session component
UIHttpManagerComponent managerComponent = writer.createWebComponentForSubSession();

// [...] managerComponent configuration

String iFrameURL = managerComponent.getURIWithParameters();

String iFrameId = "mySweetIFrame";

// place the iFrame in the page, with an empty src attribute
writer.add("<iframe id=\"\"").add(iFrameId).add("<\" src=\"\" >").add("</iframe>");

// launch the iFrame from JavaScript
```

```
writer.addJS("document.getElementById(\"").addJS(iFrameId).addJS(\"\".src = \"\").addJS(iFrameURL).addJS(\"\";");
```

100.2 CSS

Public CSS classes

The constant catalog `UICSSClasses`^{API} offers the main CSS classes used in the software to style the components. These CSS classes ensure a proper long-term integration into the software, because they follow the background colors, borders, customizable text in the administration; the floating margins and paddings fluctuate according to the variable density; to the style of the icons, etc.

Voir aussi `UICSSUtils`^{API}

Advanced CSS

EBX® allows to integrate to all its pages one or more external Cascading Style Sheet. These external CSS, considered as resources, need to be declared in the [Module registration](#) [p 514].

In order to ensure the proper functioning of your CSS rules and properties without altering the software, the following recommendations should be respected. Failure to respect these rules could lead to:

- Improper functioning of the software, both aesthetically and functionally: risk of losing the display of some of the data and some input components may stop working.
- Improper functioning of your CSS rules and properties, since the native CSS rules will impact the CSS implementation.

Reserved prefixes for CSS identifiers and class names

The following prefixes should not be used to create CSS #ids and .classes.

ebx_	Internal built-in
yui	Yahoo User Interface global
ygtv	Yahoo User Interface tree view
layout-doc	Yahoo User Interface layout
cke_	CK editor (used by HTML editor widget)
fa	Font Awesome (icons used by perspectives and toolbars)

CSS classes used internally by EBX®

The following CSS classes should never be included in a ruleset that has no contextual selector.

If you do not prefix your CSS selector using one of the CSS classes below, it will cause conflicts and corrupt the UI of EBX®.

selected	YUI selected tree node
hd	YUI floating pane header
bd	YUI floating pane body
ft	YUI floating pane footer
container-close	YUI inner popup close button
underlay	YUI inner popup shadow
hastitle	YUI menu group with title
topscrollbar	YUI menu top scroll zone
bottomscrollbar	YUI menu bottom scroll zone
withtitle	YUI calendar
link-close	YUI calendar close button
collapse	YUI layout closed pane indicator
pull-right	Font Awesome parameter
pull-left	Font Awesome parameter

Examples to avoid conflicts

Don't

```
.selected {
  background-color: red;
}
```

Do

```
#myCustomComponent li.selected {
  background-color: red;
}
```

100.3 JavaScript

Public JS functions

The catalog of JavaScript functions `JavaScriptCatalogAPI` offers a list of functions to use directly (through copy-paste) in the JS files.

JavaScript call during page generation in Java

When generating the HTML of a Java component, it is possible to add specific JavaScript code with the API `UIJavaScriptWriterAPI`.

This JavaScript is executed once the whole page is loaded. It is possible to instantly manage the HTML elements written with `UIBodyWriter.addAPI`. Setting on-load functions (such as `window.onload = myFunctionToCallOnload;`) is not supported because the execution context comes after the on-load event.

Advanced JavaScript

EBX® allows to include one or more external JavaScript files. These external JavaScript files, considered as resources, need to be declared in the [Module registration](#) [p 514]. For performance reasons, it is recommended to include the JavaScript resource only when necessary (in a User service or a specific form, for example). The API `UIDependencyRegistererAPI` allows a developer to specify the conditions for which the JavaScript resources will be integrated into a given page according to its context.

In order to ensure the proper functioning of your JavaScript resources without altering the software, the following recommendations should be respected. Failure to respect them could lead to:

- Improper functioning of the software: if functions or global variables of the software were to be erased, some input or display components (including the whole screen) may stop working.
- Improper functioning of your JavaScript instructions, since global variables or function names could be erased.

Reserved JS prefixes

The following prefixes are reserved and should not be used to create variables, functions, methods, classes, etc.

ebx_	Internal built-in API
EBX_	Internal built-in API
YAHOO	Yahoo User Interface API

SOAP data services

CHAPITRE 101

Introduction

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Activation and configuration](#)
3. [Interactions](#)
4. [Security](#)
5. [Monitoring](#)
6. [SOAP and REST comparative](#)
7. [Limitations](#)

101.1 Overview

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

In order to invoke [SOAP operations](#) [p 713], for an integration use case, a [WSDL](#) [p 705] must be generated from a data model. It will be possible to perform operations such as:

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

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

- Creating, merging, or closing a dataspace
- Creating or closing a snapshot
- Validating a dataset, dataspace, or a snapshot
- Starting, resuming or ending a data workflow

- Administrative operations to manage access to the UI or to system information

Note

See [SOAP and REST comparative](#) [p 702].

101.2 Activation and configuration

Data services are enabled by deploying the ebx-dataservices web application along with the other EBX® modules. See [Java EE deployment overview](#) [p 339] for more information.

In case of specific deployment, for example using reverse-proxy mode, see [URLs computing](#) [p 381] for more information.

The default method for accessing data services is over HTTP, although it is also possible to use JMS for the SOAP operations. See [JMS configuration](#) [p 379] and [Using JMS](#) [p 696] for more information.

101.3 Interactions

Input and output message encoding

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

Tracking information

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

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

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

For more information, see `Session.getTrackingInfoAPI` in the Java API.

Session parameters

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

Input parameters are available on custom Java components with a session object, such as: triggers, access rules, custom web services. They are also available on data workflow operations.

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

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

For more information, see `Session.getInputParameterValueAPI` in the Java API.

Exception handling

In case of unexpected server error upon execution of:

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

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

Using JMS

It is possible to access SOAP operations using JMS instead of HTTP. The JMS architecture relies on one JMS request queue (mandatory), on one JMS failure queue (optional), and on JMS response queues, see configuration [JMS](#) [p 379]. The mandatory queue is the input queue. Request messages must be put in the input queue, and response messages are put by EBX® 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 and activated 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 location points must be defined in the Lineage administration in order to specialize the generated WSDL. If no specific location point is given, the default value will be `jms:queue:jms/EBX_QueueIn`.

101.4 Security

Authentication

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

- 'Basic Authentication Scheme' method is based on the HTTP-Header Authorization in base 64 encoding, as described in [RFC 2617 \(Basic Authentication Scheme\)](#).

```
If the user agent wishes to send the userid "Alibaba" and password "open sesame",
it will use the following header field:
> Authorization: Basic QWxpYmFiYTpvYGVuIHNLc2FtZQ==
```

- 'Standard Authentication Scheme' is based on the HTTP Request. User and password are extracted from request parameters. For more information on request parameters, see [Parameters](#) [p 708] section.

For more information on this authentication scheme, see `Directory.authenticateUserFromLoginPasswordAPI`.

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

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

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

Note

Only available for [SOAP operations](#) [p 713].

- 'Specific authentication Scheme' is based on the HTTP Request. An implementation of this method can, for example, extract a password-digest or a ticket from the HTTP request. See `Directory.authenticateUserFromHttpRequestAPI` for more information.
- The 'SOAP Specific Header Authentication Scheme'.

For more information, see [Overriding the SOAP security header](#) [p 697].

Global permissions

Global access permissions can be independently defined for the SOAP and WSDL connector accesses. For more information see [Global permissions](#) [p 423].

Overriding the SOAP security header

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

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

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

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

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

The corresponding XML Schema header declaration would be as follows:

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

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

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

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

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

Note

Only available for [SOAP operations](#) [p 713].

Lookup mechanism

Because EBX® offers several authentication methods, a lookup mechanism based on conditions was set to know which method should be applied for a given request. The method application conditions are evaluated according to the authentication scheme priority. If the conditions are not satisfied, the server evaluates the next method. The following table presents the available authentication methods

for each supported protocol and their application conditions. They are ordered from the highest priority to the lowest.

Operation / Protocol	Authentication methods and application conditions
SOAP / JMS	SOAP Security Header [p 697] • The SOAP request is received over the JMS protocol. • The SOAP header content must contains a Security element. SOAP Specific Header [p 697] • The SOAP request is received over the JMS protocol. • The SOAP header content must not contain a Security element.
SOAP / HTTP	Basic [p 696] • The HTTP request must hold an Authorization header. • Authorization header value must start with the word Basic. • No login is provided in the URL parameters. Standard [p 696] • The HTTP request must not hold an Authorization header. • A login and a password are provided in the URL parameters. SOAP Security Header [p 697] • The SOAP header content must contain a Security element. • The HTTP request must not hold an Authorization header. • No login is provided in the URL parameters. Specific [p 697] • The HTTP request must not satisfy the conditions of the previous authentication methods. SOAP Specific Header [p 697] • The SOAP header content must not contain a Security element. • The HTTP request must not hold an Authorization header. • No login is provided in the URL parameters.
WSDL / HTTP	Basic [p 696] • The HTTP request must not hold an Authorization header. • Authorization header value must start with the word Basic. • No login is provided in the URL parameters. Standard [p 696] • The HTTP request must not hold an Authorization header. • A login and a password are provided in the URL parameters. Specific [p 697] • The HTTP request must not satisfy the conditions of the previous authentication methods.

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

101.5 Monitoring

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

Voir aussi

[*Configuring the EBX® logs*](#) [p 376]

[*TIBCO EBX® main configuration file*](#) [p 371]

101.6 SOAP and REST comparative

Operations	SOAP	REST
Data		
Select metadata		X
Select or count records (with filter and/or view publication)	X	X
Selector for possible enumeration values (with filter)		X
Prepare for creation or duplication		X
Insert, update or delete records	X	X
Select or count history records (with filter and/or view publication)		X
Select node values from dataset	X	X
Update node value from dataset		X
Get table or dataset changes between dataspace or snapshots	X	
Refresh a replication unit	X	
Get credentials for records	X	
Generate service contract	WSDL	OpenAPI
Views		
Look up published table views		X
Dataspaces		
Select dataspace or snapshot information		X
Select root or children dataspace, or select snapshots		X
Create, close, merge a dataspace	X	X
Create, close a snapshot	X	X
Validate a dataspace or a snapshot	X	

Operations	SOAP	REST
Validate a dataset	X	
Locking, unlocking a dataspace	X	X
Workflow		
Start, resume or end a workflow	X	
Administration		
Manage the default directory content 'Users', 'Roles'... tables.	X	X
Open, close the user interface	X	X
Select, insert, update, delete operations for administration dataset		X
Select the system information	X	X
Other		
Develop web services from the Java API		X (*)

(*) See [REST Toolkit](#) [p 897] for more information.

101.7 Limitations

Date, time & dateTime format

Data services only support the following date and time formats:

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

SOAP naming convention

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

CHAPITRE 102

WSDL generation

Ce chapitre contient les sections suivantes :

1. [Supported standard](#)
2. [Operation types](#)
3. [WSDL download methods](#)
4. [HTTP examples](#)

102.1 Supported standard

TIBCO EBX® generates a WSDL that complies with the [W3C Web Services Description Language 1.1](#) standard.

102.2 Operation types

A WSDL can be generated for different types of operations:

Operation type	WSDL description
custom	WSDL for EBX® add-ons.
dataset	WSDL for dataset and replication operations.
directory	WSDL for default EBX® directory operations. It is also possible to filter data using the tablePaths [p 709] or operations [p 709] parameters.
repository	WSDL for dataspace or snapshot management operations.
tables	WSDL for operations on the tables of a specific dataset.
userInterface	Deprecated since version 5.8.1. This operation type has been replaced by administration. While the user interface management operations are still available for backward compatibility reasons, it is recommended to no longer use this type. WSDL for user interface management operations (these operations can only be accessed by administrators).
administration	WSDL for administration operations like: • user interface management • system information retrieval These operations can only be accessed by administrators.
workflow	WSDL for EBX® workflow management operations.

102.3 WSDL download methods

EBX® supports the following methods:

- from the user interface
- from HTTP protocol

A WSDL can only be downloaded by authorized profiles:

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

WSDL download from the user interface

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

Note

See [Générer un WSDL pour accéder à un espace de données](#) [p 222] in the user guide for more information.

WSDL download from HTTP protocol

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

URL format

```
http[s]://<host>[:<port>]/<ebx-dataservices>/<type>[/<dataspace>[/<dataset>]]?<queryParameters>
```

Where:

- <ebx-dataservices> corresponds to the 'ebx-dataservices.war' web application's path. The path is composed by multiple, or none, URI segments followed by the web application's name.
- {type} corresponds to the [operation type](#) [p 706].
- {dataspace} corresponds to the dataspace or snapshot identifier.
- {dataset} corresponds to the dataset name.
- {queryParameters} corresponds to common or dedicated operation parameters passed through the URL.

Parameters

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

- a PathParam which corresponds to a path segment from the URL (recommended)

- a QueryParam which corresponds to a standard HTTP parameter with value.

Parameter name	PathParam	QueryParam	Required	Description
WSDL	no	yes	yes	Specifies the WSDL download operation. Empty value.
login	no	yes	no	Specifies the user identifier. Required when the standard authentication method is used. String type value.
password	no	yes	no	Specifies the user password. Required when the standard authentication method is used. String type value.
type	yes	no	yes	Specifies the operation type [p 706]. Possible values are: custom, dataset, directory, administration, userInterface, repository, tables or workflow. String type value.
branch version	yes	yes	(*)	Specifies the dataspace or the snapshot identifier. (*) required for tables and dataset types, ignored otherwise. String type value.
instance	yes	yes	(*)	Specifies the dataset name. String type value.
tablePaths	no	yes	no	Specifies the list of table paths. Optional for tables or directory types, ignored otherwise. If not defined, all tables are selected. Each table path is separated by a comma character. String type value.
operations	no	yes	no	Allows generating a WSDL for an operations subset. Optional for tables or directory operation types, ignored otherwise. If not defined, all operations for the given type are generated. This parameter's value is a concatenation of one or more of the following characters: • C = Count record(s) • D = Delete record(s) • E = Get credentials • G = Get changes • I = Insert record(s) • U = Update record(s)

Parameter name	PathParam	QueryParam	Required	Description
				- R = Read operations (equivalent to CEGS) - S = Select record(s) - W = Write operations (equivalent to DIU) String type value.
namespaceURI	yes	yes	(**)	Specifies the unique name space URI of the custom web service. (**)Is required when type parameter is set to custom, ignored otherwise. URI type value.
attachmentFilename	no	yes	(***)	Specifies the attachment file name. (***) optional if isContentInAttachment parameter is defined and set to true, ignored otherwise. String type value.
isContentInAttachment	no	yes	no	Specifies if the WSDL is downloaded as an attachment. Boolean type value. Default value is false.
targetNamespace	no	yes	no	Overrides the target namespace URI of the WSDL. URI type value. Default value is urn:ebx:ebx-dataservices.

Message body

No message body is required.

HTTP codes

An HTTP code is always returned. Errors are indicated by a code above 400.

Status code	Information
200 (<i>OK</i>)	The WSDL content was successfully generated and is returned by the request (optionally in an attachment [p 710]).
400 (<i>Bad request</i>)	The request is incorrect. This occurs when: - A request element is incorrect. - The unicity check on table names contains at least one error. Note See WSDL and table operations [p 606] for more information.
401 (<i>Unauthorized</i>)	Request requires an authenticated user.
403 (<i>Forbidden</i>)	Request is not allowed for the authenticated user.
405 (<i>Method not allowed</i>)	Request is not allowed in this configuration.
500 (<i>Internal error</i>)	Request generates an error (a stack trace and a detailed error message are returned).

Response body

The response body depends on the returned status code and on the requested WSDL content.

- 200 (*OK*): the HTTP header Content-Type is set to text/xml; charset=UTF-8.
If the content is in attachment, the HTTP header Content-Disposition is set to attachment; filename*=UTF-8' '<filename.wsdl>.
- 4xx: A detailed message is returned in the body. The HTTP header Content-Type is set to text/html; charset=utf-8.

102.4 HTTP examples

Some of the following examples are displayed in two methods: PathParam and QueryParam.

- The WSDL will contain all repository operations, using standard authentication method.
http[s]://<host>[:<port>]/<ebx-dataservices>/repository?WSDL&login=<login>&password=<password>
- The WSDL will contain all workflow operations.
http[s]://<host>[:<port>]/<ebx-dataservices>/workflow?WSDL
- The WSDL will contain all tables operations for the 'dataset1' dataset in 'dataspace1' dataspace.
PathParam

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

QueryParam

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
```

```
WSDL&branch=dataspace1&instance=dataset1
```

- The WSDL will contain all tables with only read operations for the 'dataset1' dataset in 'dataspace1' dataspace.

PathParam

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?
```

```
WSDL&operations=R
```

QueryParam

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
```

```
WSDL&branch=dataspace1&instance=dataset1&operations=R
```

- The WSDL will contain the two selected tables operations for the 'dataset1' dataset in 'dataspace1' dataspace.

PathParam

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables/dataspace1/dataset1?
```

```
WSDL&tablePaths=/root/table1,/root/table2
```

QueryParam

```
http[s]://<host>[:<port>]/<ebx-dataservices>/tables?
```

```
WSDL&branch=dataspace1&instance=dataset1&tablePaths=/root/table1,/root/table2
```

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

PathParam

```
http[s]://<host>[:<port>]/<ebx-dataservices>/custom/urn:ebx-
```

```
test:com.orchestranetworks.dataservices.WSDemo?WSDL
```

QueryParam

```
http[s]://<host>[:<port>]/<ebx-dataservices>/custom?WSDL&namespaceURI=urn:ebx-
```

```
test:com.orchestranetworks.dataservices.WSDemo
```


CHAPITRE 103

SOAP operations

Ce chapitre contient les sections suivantes :

1. [Operations generated from a data model](#)
2. [Operations on datasets and dataspace](#)
3. [Operations on data workflows](#)
4. [Administrative services](#)

103.1 Operations generated from a data model

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

Attention

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

Content policy

Access to the content of records, the presence or absence of XML elements, depend on the [resolved permissions](#) [p 293] of the authenticated user session. Additional aspects, detailed below, can impact the content.

Disabling fields from data model

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

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

- On request, if the data model property has been changed to `true`, and if the concerned field is present in the WSDL request, an exception will be thrown.

- On response, if the schema property has been changed to `false`, WSDL validation will return an error if it is activated.

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

Voir aussi

[*Hiding a field in Data Services*](#) [p 600]

[*Permissions*](#) [p 293]

Association field

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

Note

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

Common request parameters

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

Element	Description	Required
branch	The identifier of the dataspace to which the dataset belongs.	Either this parameter or the 'version' parameter must be defined. Required for the 'insert', 'update' and 'delete' operations.
version	The identifier of the snapshot to which the dataset belongs.	Either this parameter or the 'branch' parameter must be defined
instance	The unique name of the dataset which contains the table to query.	Yes
predicate	XPath predicate [p 251] defines the records on which the request is applied. If empty, all records will be retrieved, except for the 'delete' operation where this field is mandatory.	Only required for the 'delete' operation
data	Contains the records to be modified, represented within the structure of their data model. The whole operation is equivalent to an XML import. The details of the operations performed on data content are specified in the section Import [p 143].	Only required for the insert and update operations
viewPublication	This parameter can be combined with the predicate [p 715] parameter as a logical AND operation. The behavior of this parameter is described in the section EBX® as a Web Component [p 232]. It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical views.	No
viewId	<i>Deprecated since version 5.2.3.</i> This parameter has been replaced by the parameter 'viewPublication'. While it remains available for backward compatibility, it will eventually be removed in a future version. This parameter cannot be used if the 'viewPublication' parameter is used.	No
blockingConstraintsDisabled	This property is available for all table updates data service operations. If true, the validation process disables blocking constraints defined in the data model. If this parameter is not present, the default is false. See Blocking and non-blocking constraints [p 581] for more information.	No
details	The details element specifies the following option: The optional attribute locale (default 'en-US') defines the language of the blockingConstraintsDisabled [p 715] parameter in which the validation messages must be returned.	No

Element	Description	Required
disableRedirectionToLastBroadcast	This property is available for all data service operations. If true, access to a delivery dataspace on a D3 primary node is not redirected to the last broadcast snapshot. Otherwise, access to such a dataspace is always redirected to the last snapshot broadcast. If this parameter is not present, the default is false (redirection on a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default [p 379] has been set. If the specified dataspace is not a delivery dataspace on a D3 primary node, this parameter is ignored.	No

Select operations

Select request on table

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 715].	
version	See the description under Common parameters [p 715].	
instance	See the description under Common parameters [p 715].	
predicate	See the description under Common parameters [p 715]. This parameter can be combined with the viewPublication [p 715] parameter as a logical AND operation.	
viewPublication	See the description under Common parameters [p 715].	
includesTechnicalData	The response will contain technical data if true. See also the optimistic locking [p 732] 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	Specifies the pagination configuration, see child elements below.	No
pageSize (nested under the pagination element)	Specifies the number of records per page. Integer type, default value is 10. Voir aussi ebx.dataservices.pagination.maxSize.default [p 379]	No
previousPageLastRecordPredicate (nested under the pagination element)	When pagination is enabled, XPath predicate that defines the record after which the page must fetched, this value is provided by the previous response, as the element lastRecordPredicate. If the passed record is not found, the first page will be returned.	No
disableRedirectionToLastBroadcast	See the description under Common parameters [p 716].	

Select response on table

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

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

with:

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

See also the [optimistic locking](#) [p 732] section.

Select request on dataset

This operation returns dataset content without table.

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 715].	
version	See the description under Common parameters [p 715].	
instance	See the description under Common parameters [p 715].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 716].	

Select response on dataset

```
<ns1:selectInstanceResponse xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <data>
    <settings>
      <XX>
        <a>key1</a>
        <b>valueb</b>
        <c>1</c>
        <d>true</d>
      </XX>
    </settings>
  </data>
</ns1:selectInstanceResponse>
```

with:

Element	Description
data	Dataset content without table.

Delete operation

Deletes records or, for a child dataset, defines the record state as "occulting" or "inherited" according to the record context. Records are selected by the predicate parameter.

Delete request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 715].	
instance	See the description under Common parameters [p 715].	
predicate	See the description under Common parameters [p 715].	
includeOcculting	Includes the records in occulting mode. Default value is false.	No
inheritIfInOccultingMode	Inherits the record from its parent if it is in occulting mode. Default value is false.	No
occultIfInherit	<i>Deprecated since version 5.7.0</i> Occults the record if it is in inherit mode. Default value is false.	No
checkNotChangedSinceLastTime	Timestamp used to ensure that the record has not been modified since the last read. Also see the optimistic locking [p 732] section.	No
blockingConstraintsDisabled	See the description under Common parameters [p 715].	
details	See the description under Common parameters [p 715].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 716].	

Delete response

If one of the provided parameters is illegal, if a required parameter is missing, if the action is not authorized or if no record is selected, an exception is returned. Otherwise, the specific response is returned:

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

with:

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

Count operation

Count request

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

with:

Element	Description
branch	See the description under Common parameters [p 715].
version	See the description under Common parameters [p 715].
instance	See the description under Common parameters [p 715].
predicate	See the description under Common parameters [p 715].
disableRedirectionToLastBroadcast	See the description under Common parameters [p 716].

Count response

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


with:

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

Update operation

Update request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 715].	
instance	See the description under Common parameters [p 715].	
updateOrInsert	If <code>true</code> and the record does not currently exist, the operation creates the record. boolean type, the default value is <code>false</code> .	No
byDelta	If <code>true</code> and an element does not currently exist in the incoming message, the target value is not changed. If <code>false</code> and node is declared <code>hiddenInDataServices</code> , the target value is not changed. The complete behavior is described in the sections Opérations d'insertion et de mise à jour [p 144].	No
blockingConstraintsDisabled	See the description under Common parameters [p 715].	
details	See the description under Common parameters [p 715].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 716].	
data	See the description under Common parameters [p 715].	

Voir aussi [Optimistic locking](#) [p 732]

Update response

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

with:

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

Insert operation

Insert request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 715].	
instance	See the description under Common parameters [p 715].	
byDelta	If true and an element does not currently exist in the incoming message, the target value is not changed. If false and node is declared hiddenInDataServices, the target value is not changed. The complete behavior is described in the sections Opérations d'insertion et de mise à jour [p 144].	No
blockingConstraintsDisabled	See the description under Common parameters [p 715].	
details	See the description under Common parameters [p 715].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 716].	
data	See the description under Common parameters [p 715].	

Insert response

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

with:

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

Get changes operations

Returns changes according to the [Content policy](#) [p 713].

Get changes requests

Changes between two datasets:

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

Changes between two tables:

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 715].	
version	See the description under Common parameters [p 715].	
instance	See the description under Common parameters [p 715].	
compareWithBranch	The identifier of the dataspace with which to compare.	One of either this parameter or the ' compareWithVersion ' [p 725] parameter must be defined.
compareWithVersion	The identifier of the snapshot with which to compare.	One of either this parameter or the ' compareWithBranch ' [p 725] parameter must be defined.
compareWithInstance	The identifier of the dataset with which to compare. If it is undefined, instance [p 725] parameter is used.	No
resolvedMode	Defines whether or not the difference is calculated in resolved mode. Default is true. See Resolved mode <code>DifferenceHelper.resolvedMode</code> ^{opt} for more information.	No
includeInstanceUpdates	Defines if the content updates of the dataset are included. Default is false.	No
pagination	Enables pagination context for the operations <code>getChanges</code> and <code>getChangesOnDataSet</code> . Allows client to define pagination context size. Each page contains a collection of inserted, updated and/or deleted records of tables according to the maximum size. Get changes persisted context is built at first call according to the page size parameter in request. The pagination context is persisted on the server file system [p 419] and allows invoking the next page until last page or when a timeout is reached. For creation: Defines <code>pageSize</code> parameter. For next: Defines context element with <code>identifier</code> from previous response. Enables pagination, see child elements below.	No
pageSize (nested under pagination element)	Defines maximum number of records in each page. Minimal size is 50.	No (Only for creation)
context (nested under pagination element)	Defines content of pagination context.	No (Only for next)

Element	Description	Required
identifier (nested under context element)	Pagination context identifier.	Yes
disableRedirectionToLastBroadcast	See the description under Common parameters [p 716].	

Note

If none of the *compareWithBranch* or *compareWithVersion* parameters are specified, the comparison will be made with their parent:

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

Voir aussi *DifferenceHelper*^{API}

Get changes responses

Changes between two datasets:

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

Changes between two tables:

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

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

with:

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

Get changes operation with pagination enabled

Only pagination element and sub elements have been described.

For creation:

Extract of request:

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

Extract of response:

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

For next:

Extract of request:

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

Extract of response:

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

For last:

Extract of request:

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

Extract of response:

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

Get credentials operation

Get credentials request

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

with:

Element	Description	Required
branch	See the description under Common parameters (p 715).	
version	See the description under Common parameters (p 715).	
instance	See the description under Common parameters (p 715).	
predicate	See the description under Common parameters (p 715).	
viewPublication	See the description under Common parameters (p 715).	

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 dataset, while ensuring a consistent response. The operations are executed sequentially, according to the order defined on the client side.

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

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

See [Concurrency](#) [p 520].

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 715].	
version	See the description under Common parameters [p 715].	
instance	See the description under Common parameters [p 715].	
blockingConstraintsDisabled	See the description under Common parameters [p 715].	
details	See the description under Common parameters [p 715].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 716].	
request	This element contains one operation, like a single operation without branch, version and instance parameters. This element can be repeated multiple times for additional operations. Each request can be identified by an 'id' attribute. In a response, this 'id' attribute is returned for identification purposes. Operations such as count, select, getChanges, getCredentials, insert, delete or update.	Yes

Note:

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

Limitation:

- The multi operation applies to one model and one dataset (parameter instance).

Voir aussi

Procedure^{APT}

Repository^{API}

Multiple operations response

See each response operation for details.

```
<ns1:multi_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <response id="id1">
    <{operation}_{TableName}Response>
      ...
    </{operation}_{TableName}Response>
  </response>
  <response id="id2">
    <{operation}_{TableName}Response>
      ...
    </{operation}_{TableName}Response>
  </response>
</ns1:multi_Response>
```

with:

Element	Description
response	This element contains the response of one operation. It is be repeated multiple times for additional operations. Each response is identified by an 'id' attribute set in the request or automatically generated. The content of the element corresponds to the response of a single operation, such as count, select, getChanges, getCredentials, insert, delete or update.

Optimistic locking

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

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

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

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

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

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

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

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

Note

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

103.2 Operations on datasets and dataspace

Parameters for operations on dataspace and snapshots are as follows:

<i>Element</i>	<i>Description</i>	<i>Required</i>
branch	Identifier of the target dataspace on which the operation is applied. When not specified, the 'Reference' dataspace is used except for the merge dataspace operation where it is required.	One of either this parameter or the 'version' parameter must be defined. Required for the dataspace merge, locking, unlocking and replication refresh operations.
version	Identifier of the target snapshot on which the operation is applied.	One of either this parameter or the 'branch' parameter must be defined
versionName	Identifier of the snapshot to create. If empty, it will be defined on the server side.	No
childBranchName	Identifier of the dataspace child to create. If empty, it will be defined on the server side.	No
instance	The unique name of the dataset on which the operation is applied.	Required for the replication refresh operation.
ensureActivation	Defines if validation must also check whether this instance is activated.	Yes
details	Defines if validation returns details. The optional attribute <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. The optional attribute <code>locale</code> (default 'en-US') defines the language in which the validation messages are to be returned.	No. If not specified, no details are returned.
owner	Defines the owner. Must respect the inner format as returned by <code>Profile.format^{API}</code>.	No
branchToCopyPermissionFrom	Defines the identifier of the dataspace from which to copy the permissions.	No
documentation	Documentation for a dedicated language.	No

<i>Element</i>	<i>Description</i>	<i>Required</i>
	Multiple documentation elements may be used for several languages.	
locale (nested under the documentation element)	Locale of the documentation.	Only required when the documentation element is used
label (nested under the documentation element)	Label for the language.	No
description (nested under the documentation element)	Description for the language.	No

Validate a dataspace

Validate dataspace request

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

Validate dataspace response

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

Validate a dataset

Validate dataset request

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

Validate dataset response

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

```
<path>Path</path>
</subject>
</reportItem>
</details>
</validationReport>
</ns1:validateInstance_Response>
```

Create a dataspace

Create dataspace request

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

Create dataspace response

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

with:

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

Create a snapshot

Create snapshot request

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

Create snapshot response

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

with:

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

Locking a dataspace

Lock dataspace request

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

with:

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

Lock dataspace response

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

with:

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

Unlocking a dataspace

Unlock dataspace request

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

Unlock dataspace response

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

with:

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

Merge a dataspace

Merge dataspace request

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

with:

Element	Description	Required
deleteDataOnMerge	This parameter is available for the merge dataspace operation. Sets whether the specified dataspace and its associated snapshots will be deleted upon merge. 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 the EBX® main configuration file [p 379]. See Deleting data and history [p 418] for more information.	No
deleteHistoryOnMerge	This parameter is available for the merge dataspace operation. Sets whether the history associated with the specified dataspace will be deleted upon merge. Default value is false. 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 the EBX® main configuration file [p 379]. See Deleting data and history [p 418] for more information.	No

Note

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

Merge dataspace response

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

with:

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

Close a dataspace or snapshot

Close dataspace or snapshot request

Close dataspace request:

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

Close snapshot request:

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

with:

Element	Description	Required
deleteDataOnClose	This parameter is available for the close dataspace and close snapshot operations. Sets whether the specified snapshot, or dataspace and its associated snapshots, will be deleted upon closure. When this parameter is not specified in the request, the default value is false. It is possible to redefine this default value by specifying the property <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> in the EBX® main configuration file [p 379]. See Deleting data and history [p 418] for more information.	No
deleteHistoryOnClose	This parameter is available for the close dataspace operation. Sets whether the history associated with the specified dataspace will be deleted upon closure. Default value is false. 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 the EBX® main configuration file [p 379]. See Deleting data and history [p 418] for more information.	No

Close dataspace or snapshot response

Close dataspace response:

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

Close snapshot request:

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

Replication refresh

Replication refresh request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 734].	Yes
instance	See the description under Common parameters [p 734].	Yes
unitName	Name of the replication unit. Voir aussi Replication refresh information [p 278]	Yes

Replication refresh response

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

with:

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

103.3 Operations on data workflows

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

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

See [session parameters](#) [p 695] for more information.

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

Start a workflow

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

Note

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

Sample request:

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

with:

Element	Description	Required
publishedProcessKey	Identifier of the workflow launcher.	Yes
documentation	See the description under Common parameters [p 734].	No
parameters	<i>Deprecated since version 5.7.0</i> See the description under Common parameters [p 741].	No

Sample response:

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

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

with:

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

Resume a workflow

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

Sample request:

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

with:

Element	Description	Required
resumeId	Resume identifier of the waiting task.	Yes
parameters	<i>Deprecated since version 5.7.0</i> See the description under Common parameters [p 741].	No

Sample response:

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

with:

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

End a workflow

End a workflow from its identifier.

Sample request:

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

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

with:

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

Sample response:

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

with:

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

103.4 Administrative services

Directory services

The services on directory provide operations on the 'Users' and 'Roles' tables of the default directory. To execute an operation related to these services, the authenticated user must be a member of the built-in role 'Administrator'.

The technical dataspace and dataset must be set to ebx-directory. For all SOAP operation syntaxes, see [Operations generated from a data model](#) [p 713] for more information.

Create a user in the directory

This example of a SOAP insert request adds a user to the EBX® directory.

```
<m:insert_user xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <branch>ebx-directory</branch>
  <instance>ebx-directory</instance>
  <data>
    <directory>
      <users>
        <login>login</login>
        <lastName>lastname</lastName>
```

```
<firstName>firstname</firstName>
<email>firstname.lastname@email.com</email>
<password>***</password>
<passwordMustChange>true</passwordMustChange>
<builtInRoles>
  <administrator>false</administrator>
  <readOnly>false</readOnly>
</builtInRoles>
<comments>a comment</comments>
</users>
</directory>
</data>
</m:insert_user>
```

For the insert SOAP response syntax, see [insert response](#) [p 723] for more information.

User interface operations

See [Application locking](#) [p 428] for more information.

Parameters for operations on the user interface are as follows:

Element	Description	Required
closedMessage	Message to be displayed to users when the user interface is closed to access.	No

Close user interface request

The close operation removes all user sessions that are not acceptable in this mode.

```
<m:close xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <closedMessage>Access is temporarily forbidden.</closedMessage>
</m:close>
```

Close user interface response

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

Open user interface request

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

Open user interface response

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

System information operation

This operation returns the EBX® system information. The information returned is the same as the information contained in the log header `kernel.log` or in the UI tab 'Administration' > 'System Information'. The response contains several keys, labels, and values representing the configuration and status of EBX®. To execute this operation, the authenticated user must be a member of the built-in role 'Administrator'.

Parameters

The following parameter is applicable.

Parameter	Description	Required
details	Defines attributes that must be applied to response messages. The attribute <code>locale</code> (default: EBX® default locale) defines the language in which the system item messages must be returned.	No, but if specified, the <code>locale</code> attribute must be provided.

System information request

This SOAP request will return all EBX® instance's system information and format them using "en_US" locale.

```
<m:systemInformation xmlns:m="urn:ebx-schemas:dataservices_1.0">
  <details locale="en_US" />
</m:systemInformation>
```

System information response

```
<ns1:systemInformation_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
  <bootInfoEBX>
    <label>String</label>
    <infoItem>
      <key>String</key>
      <label>String</label>
      <content>String</content>
      <content>String</content>
      ...
    </infoItem>
    ...
  </bootInfoEBX>
  <repositoryInfo>
    <label>String</label>
    <infoItem>
      <key>String</key>
      <label>String</label>
      <content>String</content>
      <content>String</content>
      ...
    </infoItem>
    ...
  </repositoryInfo>
  <bootInfoVM>
    <label>String</label>
    <infoItem>
      <key>String</key>
      <label>String</label>
      <content>String</content>
      ...
    </infoItem>
    ...
  </bootInfoVM>
</ns1:systemInformation_Response>
```

REST data services

CHAPITRE 104

Introduction

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Activation and configuration](#)
3. [Interactions](#)
4. [Security](#)
5. [Monitoring](#)
6. [SOAP and REST comparative](#)
7. [Limitations](#)

104.1 Overview

REST data services allow external systems to interact with data governed in the TIBCO EBX® repository using the RESTful built-in services.

The request and response syntax for built-in services are described in the chapter [Built-in RESTful services](#) [p 757].

Built-in REST data services allow performing operations such as:

- Selecting, inserting, updating, deleting, or counting records
- Selecting or counting history records
- Selecting, updating, or counting dataset values
- Selecting or updating dataspace or snapshot information
- Selecting children dataspace or snapshots
- Creating, merging, or closing a dataspace
- Creating or closing a snapshot
- Administrative operations to manage access to the UI or to system information

Note

See [SOAP and REST comparative](#) [p 755].

104.2 Activation and configuration

REST and SOAP Data services are activated by deploying the `ebx-dataservices` web application along with the other EBX® modules. See [Java EE deployment overview](#) [p 339] for more information.

In case of specific deployment, for example using reverse-proxy mode, see [URLs computing](#) [p 381] for more information.

Currently only the HTTP(S) protocol is supported.

104.3 Interactions

Input and output message encoding

All input and output messages must be *exclusively* in UTF-8 for REST built-in.

Tracking information

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

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

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

For more information, see `Session.getTrackingInfoAPI` in the Java API.

Session parameters

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

Input parameters are available on custom Java components with a session object, such as: triggers, access rules, custom web services. They are also available on data workflow operations.

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

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

For more information, see `Session.getInputParameterValueAPI` in the Java API.

Session channel

The session channel allows to filter what can be selected or modified, from the EBX® repository, when using a REST built-in or REST toolkit service. The filter is based on table, group or field configuration where the visibility is defined through the data model, by specifying a [default view](#) [p 599].

It can be specified through the query parameter [ebx-channel](#) [p 761]. It's available values are:

- dataServices
- ui

The filter behavior is described by this combinatorial:

Data services channel

XML element	Value	Schema node type	View	Behaviour
<hiddenInDataServices>	true	field node	Default tabular view	Hidden for content and not sortable
			CustomView (tabular or hierarchical)	Hidden for meta, content, filter and sort
			Default form record	Hidden for meta and content
			Default form record field	Not found

User Interface channel

XML element	Value	Schema node type	View	Behaviour
<hidden>	true	table node	Tree view	hidden for meta and content
			Default tabular view	not found
			CustomView (tabular or hierarchical)	forbidden
			Default form record	not found
			Default form record field	
			Custom form record	
		field node	Default tabular view	hidden for content and not sortable
			Default form record	hidden for content
			Default form record field	not found
<hiddenInView>	true	field node	CustomView (tabular or hierarchical)	hidden for meta, content ,filter and sort
			Custom form record	hidden for meta and content
<hiddenInSearch>	true	field node	Default tabular view	not filterable
			CustomView (tabular or hierarchical)	
			Default form record	
			Default form record field	
			Custom form record	
	textSearchOnly		Default tabular view	not filterable except for text search
			CustomView (tabular or hierarchical)	

XML element	Value	Schema node type	View	Behaviour
			Default form record	
			Default form record field	
			Custom form record	

Note

The above field nodes can only be under table nodes.

Procedure context

Depending on the data services operation being called, it is possible to overwrite the default procedure context configuration. They are defined in the request body and are applied within the built-in operation.

Procedure context can be applied to custom REST Toolkit services.

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

```
{
  "procedureContext":          // JSON Object (optional)
  {
    "commitThreshold": Integer // JSON Number (optional)
  },
  ...
}
```

For more information, see `ProcedureContext.setCommitThresholdAPI`, `SessionContext.getProcedureUtilityAPI` and `ProcedureUtility.executeAPI` in the Java API.

Exception handling

When an error occurs, a JSON exception response is returned to the caller. For example:

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

The response contains an HTTP status code and a table of errors. The severity level of each error is specified by a character, with one of the possible values (fatal, error, warning, info).

The HTTP error 422 (*Unprocessable entity*) corresponds to a functional error. It contains a user code under the `userCode` key and is a JSON String type.

Voir aussi [HTTP codes](#) [p 763]

Voir aussi `Severity.getLabelAPI`

104.4 Security

Authentication

Authentication is mandatory to access built-in services. Several authentication methods are available and described below. The descriptions are ordered by priority (EBX® applies the highest priority authentication method first).

- 'Token Authentication Scheme' method is based on the HTTP-Header Authorization, as described in [RFC 2617](#).

```
> Authorization: <tokenType> <accessToken>
```

For more information on this authentication scheme, see [Token authentication operations](#) [p 766].

Voir aussi [HTTP Authorization header policy](#) [p 380]

- 'Basic Authentication Scheme' method is based on the HTTP-Header Authorization in base64 encoding, as described in [RFC 2617 \(Basic Authentication Scheme\)](#).

```
If the user agent wishes to send the userid "Alibaba" and password "open sesame",
it will use the following header field:
> Authorization: Basic QWxpYmFiYTpwGVuIHNLc2FtZQ==
```

Note

The [WWW-Authenticate](#) [p 762] header can be valued with this method.

Voir aussi [HTTP Authorization header policy](#) [p 380]

- 'Standard Authentication Scheme' is based on the HTTP Request. User and password are extracted from request parameters. For more information on request parameters, see [Parameters](#) [p 708] section.

For more information on this authentication scheme, see `Directory.authenticateUserFromLoginPasswordAPI`.

- The 'REST Forward Authentication Scheme' is used only when calling a REST service from a [user service](#) [p 658], that reuses the current authenticated session.

For more information, see [Implementing a user service](#) [p 669] making a call to [REST data services](#) [p 676].

- 'Specific authentication Scheme' is based on the HTTP Request. For example, an implementation can extract a password-digest or a ticket from the HTTP Request. See `Directory.authenticateUserFromHttpRequestAPI` for more information.
- 'Anonymous authentication Scheme' is used only to access the REST services handling the authentication operations. The credentials acquisition, password changes, etc. implies that the user cannot be known yet.

Global permissions

Global access permissions can be independently defined for the REST built-in and REST OpenAPI services. See [Global permissions](#) [p 423] for more information.

Lookup mechanism

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

Operation / Protocol	Authentication methods and application conditions
REST / HTTP	Token [p 753] • The HTTP request must hold an Authorization header. • Authorization header value must start with the word EBX. • No login is provided in the URL parameters. Basic [p 753] • The HTTP request must hold an Authorization header. • Authorization header value must start with the word Basic. • No login is provided in the URL parameters. Standard [p 753] • The HTTP request must not hold an Authorization header. • A login and a password are provided in the URL parameters. Rest forward [p 753] • The HTTP request must not contain an Authorization header. • No login is provided in the URL parameters. Specific [p 753] • The HTTP request must not satisfy the conditions of the previous authentication methods. Anonymous [p 753] • None of the previous authentication methods can be applied. • The requested REST service is handling an authentication operation.

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

104.5 Monitoring

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

Voir aussi

[Configuring the EBX® logs](#) [p 376]

[TIBCO EBX® main configuration file](#) [p 371]

104.6 SOAP and REST comparative

Operations	SOAP	REST
Data		
Select metadata		X
Select or count records (with filter and/or view publication)	X	X
Selector for possible enumeration values (with filter)		X
Prepare for creation or duplication		X
Insert, update or delete records	X	X
Select or count history records (with filter and/or view publication)		X
Select node values from dataset	X	X
Update node value from dataset		X
Get table or dataset changes between dataspace or snapshots	X	
Refresh a replication unit	X	
Get credentials for records	X	
Generate service contract	WSDL	OpenAPI
Views		
Look up published table views		X
Dataspaces		
Select dataspace or snapshot information		X
Select root or children dataspace, or select snapshots		X
Create, close, merge a dataspace	X	X
Create, close a snapshot	X	X
Validate a dataspace or a snapshot	X	

Operations	SOAP	REST
Validate a dataset	X	
Locking, unlocking a dataspace	X	X
Workflow		
Start, resume or end a workflow	X	
Administration		
Manage the default directory content 'Users', 'Roles'... tables.	X	X
Open, close the user interface	X	X
Select, insert, update, delete operations for administration dataset		X
Select the system information	X	X
Other		
Develop web services from the Java API		X (*)

(*) See [REST Toolkit](#) [p 897] for more information.

104.7 Limitations

Date, time & dateTime format

Data services only support the following date and time formats:

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

JMS

- JMS protocol is not supported.

CHAPITRE 105

Built-in RESTful services

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Request](#)
3. [Response](#)
4. [Administration operations](#)
5. [Token authentication operations](#)
6. [Data operations](#)
7. [Form data operations](#)
8. [Beta feature: Dataspace operations](#)
9. [OpenAPI operations](#)
10. [Limitations](#)

105.1 Introduction

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

The HATEOAS approach of the built-in RESTful services also allows for an intuitive and straightforward navigation, which implies that the data details could be obtained through a link.

Note

All operations are stateless.

105.2 Request

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

Voir aussi

[Interactions](#) [p 749]

[Security](#) [p 753]

HTTP method

Considered HTTP methods for built-in RESTful services, are:

- GET: used to select master data defined in the URL (the size limit of the URL depends on the application server and on the browser; it must be lower than or equal to 2kB).
- POST: used to insert records in a table, or to select the master data defined in the URL (the size limit is 2MB or more, depending on the application server. Each parameter is limited to a value containing 1024 characters).
- PUT: used to update the master data defined in the URL.
- DELETE: used to delete either the record defined in the URL, or multiple records defined by the table URL and the record keys in the message body.

URL

REST URL contains:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/{categoryVersion}/
{specificPath}[:{extendedAction}]?{queryParameters}
```

Where:

- <ebx-dataservices> corresponds to the 'ebx-dataservices.war' web application's path. The path is composed by multiple, or none, URI segments followed by the web application's name.
- {category} corresponds to the [operation category](#) [p 759].
- {categoryVersion} corresponds to the category version: current value is v1.
- {specificPath} corresponds to a specific path inside the category.
- {extendedAction} corresponds to the extended action name (optional).
- {queryParameters} corresponds to [common](#) [p 761] or dedicated operation parameters passed in the URL.

Operation category

Specializes the operation; it is added in the path of the URL in {category}, and takes one of the following values:

admin	Administration operations reserved to administrators. See Administration operations [p 764] for more information.
auth	Manages token authentication method. See Token authentication operations [p 766] and Token Authentication Scheme [p 753] for more information.
data or data-compact	Lists dataset content; requests a table, a record or a field record content, including modified operations on dataset node, table, record and record field.(A compact format is available, to ease interaction in simple use cases.) Manages dataspace and snapshot life cycle. See Data operations [p 769], Beta feature: Dataspace operations [p 800] and Compact format limitations [p 811] for more information.
form-data or form-data-compact	Validates incoming data, and returns a report before inserting or updating a dataset node, record or record field. (A compact format is available to ease interaction for simple use cases.) See Form data operations [p 796] and Compact format limitations [p 811] for more information.
history	Lists the content of a history dataset; requests a history table, a history of a record or a history record. See Data operations [p 769] for more information. <i>Voir aussi History [p 269]</i>
open-api	Generates the OpenAPI document of the selected resource. See OpenAPI operations [p 809] for more information.

Header fields

These header field definitions are used by TIBCO EBX®.

Accept	Used to specify (by order of preference) content types to be used in the response: the first supported one will be chosen and specified in the response header Content-Type. Currently, the only supported one is application/json. If none is supported, the result depends on property <code>ebx.dataservices.rest.request.checkAccept</code>: • If true, an HTTP error response is returned with code 406. • If false, the response is returned with the default content type, that is application/json. Voir aussi Configuring data services [p 379]
Accept-Language	Used to specify the preferred locale for the response. The supported locales are defined in the schema model. If none of the preferred locales is supported, the default locale for the current model is used.
Authorization	Supported authentication schemes include 'Basic Authentication Scheme' and 'Token Authentication Scheme'; if another scheme is used, the request is rejected. Voir aussi Authentication [p 753]
Content-Type	Used to specify the request body media type. The supported types are application/json and application/x-www-form-urlencoded. If the request value is not supported, then an HTTP error message is returned with the code 415 (<i>Unsupported media type</i>). Voir aussi Configuring data services [p 379]
X-Requested-With	If present and in case of authentication failure, prevents from adding the <code>www-Authenticate</code> header in the response. Voir aussi Response header www-Authenticate [p 762]

See [RFC2616](#) for more information about HTTP Header Field Definitions.

Common parameters

These optional parameters are available for all data service operations.

Parameter	Description
<code>disableRedirectionToLastBroadcast</code>	This parameter only has impact in a D3 architecture. If <code>true</code>, access to a delivery dataspace on a D3 primary node is not redirected to the last broadcast snapshot. Otherwise, access to such a dataspace is always redirected to the last broadcast snapshot. If the specified dataspace is not a delivery dataspace on a D3 primary node, this parameter is ignored. Boolean type value. If this parameter is not present, the default is <code>false</code> (redirection to a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default [p 379] has been set. Voir aussi Primary node [p 481]
<code>ebx-indent</code> <code>indent</code> (deprecated since 6.0.0)	Specifies whether the response should be indented, to be easier to read for a human. Boolean type, default value is <code>false</code>.
<code>ebx-channel</code>	Specifies the session channel [p 750]. String type, possible values are: <code>dataServices</code> <code>ui</code> The default value is <code>dataServices</code>. Voir aussi <code>Constants.Data.PARAM_EBX_CHANNEL^{API}</code>

Message body

It contains the request data using the JSON format, see [Extended JSON request body](#) [p 815] and [Compact JSON request body](#) [p 841] .

Note

Requests may define a message body only when using POST or PUT HTTP methods.

105.3 Response

This chapter describes the responses returned by built-in RESTful services.

- See [Exception handling](#) [p 752] for details on standard error handling (where the HTTP code is greater than or equal to 300).

Header fields

These header field definitions are used by EBX®.

Content-Language	Indicates the locale used in the response for labels and descriptions.
Content-Type	Indicates the response body content type.
Location	If a new record has been successfully inserted, the query URL for this record is returned by this field.
WWW-Authenticate	This header field is added to the HTTP response when authentication fails with the 401 (<i>Unauthorized</i>) HTTP code. Its value consists of a list with at least one authentication method applicable to the request URI. It is present if and only if the following conditions are verified: - the 'Basic Authentication Scheme' method is enabled and - the X-Requested-With HTTP header is not present. If the client is able to interpret the authentication method, it is possible to resubmit the request providing the appropriate credentials. The <code>administration</code> property <code>ebx.dataservices.rest.auth.tryBasicAuthentication</code> [p 379] must be set to <code>true</code>. Voir aussi Request header X-Requested-With [p 760] Authentication [p 753]

HTTP codes

HTTP code	Description
200 (<i>OK</i>)	The request has been successfully handled.
201 (<i>Created</i>)	A new record has been created, in this case, the header field <code>Location</code> is returned with its resource URL.
204 (<i>No content</i>)	The request has been successfully handled but no response body is returned.
400 (<i>Bad request</i>)	The request URL or body is not well-formed or contains invalid content.
401 (<i>Unauthorized</i>)	Authentication has failed.
403 (<i>Forbidden</i>)	Permission was denied to read or modify the specified resource for the authenticated user. This error is also returned when the user: - is not allowed to modify a field mentioned in the request message body. - is not allowed to access the REST connector. For more details, see Global permissions [p 753].
404 (<i>Not found</i>)	The resource specified in the URL cannot be found.
406 (<i>Not acceptable</i>)	Content type defined in the request's <code>Accept</code> parameter is not supported. This error can be returned only if the EBX® property <code>ebx.rest.request.checkAccept</code> is set to <code>true</code> .
409 (<i>Conflict</i>)	A concurrent modification has occurred. Voir aussi Optimistic locking [p 792]
415 (<i>Unsupported media type</i>)	The request content is not supported, the request header value <code>Content-Type</code> is not supported by the operation.
422 (<i>Unprocessable entity</i>)	The new resource's content cannot be accepted for semantic reasons.
500 (<i>Internal error</i>)	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Message body

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

- HTTP codes from 200 included to 300 excluded: the content format depends on the associated request ([Extended JSON](#) [p 818] and [Compact JSON](#) [p 844] samples).

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

- HTTP codes greater than or equal to 300: the content describes the error. See [JSON](#) [p 752] for details on the format.

105.4 Administration operations

Administration operations are related to:

- the administration category.
- the administration dataspace accessible through the data category.

Note

administration category and administration dataspace can only be used by administrators.

Directory operations

The EBX® default directory configuration is manageable with built-in RESTful services. The users and roles tables, the mailing lists and other objects are concerned. For more information, see [Users and roles directory](#) [p 451].

Note

Triggers are present on the directory's tables, ensuring the data consistency.

The URL format is:

`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-directory/ebx-directory`

Directory configuration operations

The EBX® default directory configuration is manageable like dataset nodes. It can be accessed and modified through the data category operations. Each field is self-described when metadata is requested.

See [select](#) [p 769] and [update](#) [p 785] operations for more information.

Mailing lists operations

There are two default mailing lists that can be configured in the EBX® directory: one for everybody and one for administrators. These lists can be handled like dataset nodes through the data category operations.

See [select](#) [p 769] and [update](#) [p 785] operations for more information.

Directory users operations

Users can be manipulated like records of the data category using the operations of the latter. For security purposes, an administrator cannot delete himself. The user's salutation must be chosen among the ones available in the 'salutations' table.

See [select](#) [p 769], [update](#) [p 785], [insert](#) [p 782] and [delete](#) [p 787] operations for more information.

Directory roles operations

Roles are records of the data category and can be managed with its operations. EBX® roles are assigned to users through the 'usersRoles' association table. 'usersRoles' is automatically fed when the directory is administered through the user interface. However, it is not the case through data services and role assignments require manual operations. Roles inclusions are specified in the 'rolesInclusions'

association table. As for the 'usersRoles' table, the management of roles inclusions requires manual operations. Each table is self-descriptive when metadata is requested.

See [select](#) [p 769], [update](#) [p 785], [insert](#) [p 782] and [delete](#) [p 787] operations for more information.

User interface operations

The EBX® user interface can be opened or closed to users for maintenance needs. Handled information is similar to what is contained in the UI tab 'Administration' > 'User interface configuration' > 'Advanced perspective' > 'Graphical interface configuration' > 'Application locking'.

URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/Bebx-manager/ebx-manager/
domain/toolStatus
```

Voir aussi [Application locking](#) [p 428]

Retrieve user interface state

User interface status and the unavailability message are accessible like dataset nodes.

See [Select operation](#) [p 769] and the [Extended JSON](#) [p 823] or [Compact JSON](#) [p 845] example, for more information.

Open or close user interface

User interface status and the unavailability message can be modified like dataset nodes using the update operation. To open the user interface set the content of toolStatus to true, or to false to close it.

See [Update operation](#) [p 785] and the [Extended JSON](#) [p 818] or [Compact JSON](#) [p 843] examples, for more information.

System information operation

This operation returns system information on the EBX® server. This is accepted for GET and POST HTTP methods. Warning: no update will be possible in the POST HTTP method because the request body is ignored. The information returned is the same as the information contained in the log header kernel.log or in the UI tab 'Administration' > 'System Information'. The response contains several keys, labels, and values representing the configuration and status of EBX®. The mode of representation of the response may be flat or hierarchical.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/admin/v1/systemInformation
```

Voir aussi

[TIBCO EBX® main configuration file](#) [p 371]

[Repository administration](#) [p 412]

Parameters

The following parameter is applicable.

Parameter	Description
systemInformationMode	Specifies the returned mode: - flat: A flat representation under the following information groups: bootInfoEBX, repositoryInfo and bootInfoVM. - hierarchical: A hierarchical representation. String type, default value is flat.

HTTP codes

HTTP code	Description
200 (OK)	The system information was successfully returned.
400 (Bad request)	The request is not correct, it contains one of the following errors: - the HTTP method is not GET nor POST, - the HTTP parameter systemInformationMode is not correct, - the operation is not supported. - the request path is invalid.
403 (Forbidden)	The user is not an administrator.

Response body

It is returned, if and only if, the HTTP code is 200 (OK). The content structure depends on the provided parameter systemInformationMode or its default value.

See the [JSON](#) [p 858] example of the flat representation.

See the [JSON](#) [p 858] example of the hierarchical representation.

105.5 Token authentication operations

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

Note

The token timeout is modifiable through the administration property [ebx.dataservices.rest.auth.token.timeout](#) [p 379] (the default value is 30 minutes).

Create token operation

This operation requires using the POST HTTP method with a request containing the user's credentials and, optionally, [session parameters](#) [p 749].

URL format is:

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

Message body

A message body must be defined in the HTTP request. It necessarily contains one of the following set of data:

- A login and a password value. Both JSON attributes are mandatory and of String types.
See `Directory.authenticateUserFromLoginPasswordAPI` for more information.
- The specific JSON attribute set to true. When activated, this flag allows to performed a user authentication against the whole HTTP request. Warning, even if login and password attributes are defined in the JSON request's body, setting specific to true lead to a specific user authentication.

See `Directory.authenticateUserFromHttpRequestAPI` for more information.

See the [JSON](#) [p 857] examples of a token creation request.

HTTP codes

HTTP code	Description
200 (OK)	The token was successfully created.
400 (Bad request)	For one of the following reasons: • the syntax is not correct, • the HTTP method is not POST, • the operation is not supported.
401 (Unauthorized)	For one of the following reasons: • The login and/or password is/are incorrect. • Authentication data for other methods is defined.
422 (Unprocessable entity)	For one of the following reasons: • PasswordMustChange: The password must be changed (only available with the default directory). See Change password operation [p 768]. • RestrictedAccess: User access is closed for maintenance or other actions (reserved to administrators).

Response body

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

See the [JSON](#) [p 859] example of a token creation response.

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

Voir aussi ['Token authentication Scheme' method](#) [p 753]

Change password operation

This operation modifies the password of an existing user account. It can be used in an authenticated context: `login` parameter, if present, is checked against the current session or taken from it, if absent. It could also be used in an unauthenticated context, for example when the [Create token operation](#) [p 766] aborts with the HTTP code 422 (*Unprocessable entity*) with reason: `PasswordMustChange`.

It requires the use of:

- the EBX® default directory
- the POST HTTP method
- the message body containing the structure specified below

URL format is:

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

Message body

The message body must be defined in the request. It necessarily contains a `password` and a `passwordNew`, the `login` is optional (all are `String`).

See the [JSON](#) [p 857] example of a password change and token creation request.

HTTP codes

HTTP code	Description
204 (<i>No content</i>)	The password has been changed.
400 (<i>Bad request</i>)	For one of the following reasons: • the EBX® default directory is required, • the syntax is not correct, • the HTTP method is not POST, • the provided <code>login</code> and the user's session one mismatch • the operation is not supported.
401 (<i>Unauthorized</i>)	For the following reason: • The <code>login</code> and/or <code>password</code> is/are incorrect.
422 (<i>Unprocessable entity</i>)	For one of the following reasons: • <code>PasswordChangeAbort: passwordNew</code> is empty. • <code>PasswordChangeAbort: passwordNew</code> is equal to <code>password</code>. • <code>PasswordChangeAbort: password</code> is incorrect. • <code>RestrictedAccess</code>: User access is closed for maintenance or other actions (reserved to administrators).

Response body

If HTTP code 204 (*No content*) is returned, then the password has been modified.

Revoke token operation

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

URL format is:

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

Header fields

Authorization

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

```
> Authorization: <tokenType> <accessToken>
```

HTTP codes

HTTP code	Description
204 (<i>No content</i>)	The token has been revoked successfully.
400 (<i>Bad request</i>)	For one of the following reasons: - the configuration is not activated, - the syntax is incorrect, - the HTTP method is not POST, - the operation is not supported.
401 (<i>Unauthorized</i>)	Authentication has failed.

105.6 Data operations

The data category operations concern the datasets, the dataset fields, tables, records or record fields.

The data-compact category operations concern the dataset fields, tables, records or record fields.

The history category operations concern historized content from datasets, tables, records or record fields.

The form-data category operations concern the dataset fields, records or record fields when constraints must remain valid on content creation or update.

The form-data-compact category operations concern the dataset fields, records or record fields when constraints must remain valid on content creation or update.

See [Form data operations](#) [p 796] for more information.

Select operation

Select operation returns hierarchical content. This operation may use one of the following methods:

- GET HTTP method,
- POST HTTP method without message body or

- POST HTTP method with a message body, with `:select` URL extended action and optionally [session parameters](#) [p 749].

URL formats are:

- **Dataset tree**, depending on [operation category](#) [p 759]:

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

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

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}[:select]
```

Note

Terminal nodes and sub-nodes are not included.

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

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}[:select]
```

Note

Not applicable with the history category.

- **Table**, depending on [operation category](#) [p 759]:

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

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

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}[:select]
```

Voir aussi

[Count operation](#) [p 789]

[Look up table views operation](#) [p 795]

- **Record**, depending on [operation category](#) [p 759]:

the data or data-compact category returns the record content and/or metadata.

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

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}[:select]
```

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}[:select]?primaryKey={xpathExpression}
```

Note

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

- **Field**, depending on [operation category](#) [p 759]:

the data or data-compact category returns the field record content where structure depends on its type.

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

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}[:select]
```

Note

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

Where:

- {category} corresponds to the [operation category](#) [p 759] (possible values are: data or data-compact).
- {dataspace} corresponds to B followed by the dataspace identifier or to v followed by the snapshot identifier.
- {dataset} corresponds to the dataset identifier.
- {pathInDataset} corresponds to the path of the dataset node, that can be a group node or a table node.
- {encodedPrimaryKey} corresponds to the encoded representation of the primary key.

Voir aussi *RESEncodingHelper*^{API}

- {xpathExpression} corresponds to the record primary key, using the XPath expression.
- {pathInRecord} corresponds to the path starting from the table node.
- :select extended action is required when POST HTTP method is used with a body message.

Parameters

The following parameters are applicable to the select operation.

Parameter	Description
<code>includeContent</code>	Includes the content field with the content corresponding to the selection. Boolean type, default value is <code>true</code> .
<code>includeDetails</code>	Includes the details field in the metadata and in the content, for each indirect reachable resource. The returned value corresponds to its URL resource. Type Boolean, default value is <code>true</code> . Voir aussi includeMeta [p 772]
<code>includeHistory</code>	Includes those fields for a historized content: - The history property in the metadata. The returned value corresponds to a boolean value. - The historyDetails property in the content and for each indirectly reachable resource. This point is coupled to the includeDetails [p 772] parameter. The returned value corresponds to its URL resource. Boolean type, default value is <code>false</code> . Note The <code>includeHistory</code> parameter is ignored in the history category, the default value is <code>true</code>. Voir aussi includeMeta [p 772] includeDetails [p 772]
<code>includeLabel</code>	Includes the label field associated with each simple type content. Possible values are: <code>yes</code>: the label is included for the foreign key, enumeration, record and selector values. <code>all</code>: the label field is included, as for the <code>yes</code> value and also for the Content of simple type [p 837]. Voir aussi <code>SchemaNode.displayOccurrence^{API}</code> <code>no</code>: the label field is not included (integration use case). String type, default value is <code>yes</code> . Note The label field is not included if it is equal to the content field.
<code>includeMeta</code>	Includes the meta field corresponding to the description of the structure returned in the content field. Boolean type, default value is <code>false</code> . Voir aussi

Parameter	Description
	Metadata [p 824] includeHistory [p 772] includeDetails [p 772]
includeOpenApiDetails	Includes the OpenAPI specification URL for each describable node. Type Boolean, default value is false. Note This query parameter is ignored if the includeDetails [p 772] one is set to false. Voir aussi OpenAPI operations [p 809] includeDetails [p 772]
includeSelector	Includes the selector field in the response, for each indirect reachable resource. The returned value corresponds to its URL resource. Type Boolean, default value is true. Voir aussi selector [p 778]
includeSortCriteria	Includes the sortCriteria field corresponding to the list of sort criteria applied. The sort criteria parameters are added by using: • sort [p 775] • sortOnLabel [p 776] • viewPublication [p 777] Boolean type, default value is false. Example JSON [p 831]
includeTechnicals	Includes the internal technical data. Boolean type, default value is false. Note This parameter is ignored with the history category. Voir aussi Technical data [p 839]
includeValidation	Includes the validation report corresponding to the selection. Boolean type, default value is false. Note This parameter is ignored with the history category. Voir aussi includeDetails [p 772]

Table parameters

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

Parameter	Description
filter	Quick search predicate or complete XPath predicate [p 251] expression defines the field values to which the request is applied. If empty, all records will be retrieved. String type value. Note The history code operation value is usable with ebx-operationCode path field from the meta section associated with this field. Voir aussi sort by relevancy [p 776] Voir aussi Search predicate [p 252]
historyMode	Specifies the filter context applied on table. String type, possible values are: • CurrentDataSpaceOnly: history in current dataspace • CurrentDataSpaceAndAncestors: history in current dataspace and ancestors • CurrentDataSpaceAndMergedChildren: history in current dataspace and merged children • AllDataSpaces: history in all dataspace The default value is CurrentDataSpaceOnly. Voir aussi history [p 759] Note This parameter is ignored with the data category.
includeOcculting	Includes the records in occulting mode. Boolean type, default value is false. Voir aussi Inheritance [p 793]
primaryKey	Search a record by a primary key, using the XPath expression. The XPath predicate [p 251] expression should only contain field(s) of the primary key and all of them. Fields are separated by the operator and. A field is represented by one of the following possibilities according to its simple type: • For the date, time or dateTime types: use the date-equal(path, value) • For other types: indicate the path, the = operator and the value. Example with a composed primary key: ./pk1i=1 and date-equal(./pk2d, '2015-11-13') The response will only contain the corresponding record, otherwise an error is returned. Consequently, the other table parameters are ignored (as filter [p 774], viewPublication [p 777], sort [p 775], etc.) String type value.

Parameter	Description
pageFirstRecordFilter	<i>Deprecated since version 5.9.0, replaced by pageRecordFilter</i>
pageRecordFilter	Specifies a server side built filter, for pagination, pointing to a record. This filter is strongly linked to the pageAction value and should not be modified on the client side. The filter takes the form of a record XPath predicate [p 251] expression used to figure out the pagination contexts. String type. Voir aussi Pagination [p 850]
pageAction	Specifies the pagination action to perform from the identifier held by pageRecordFilter. String type, default value is first. The possible values are: • first • previous • next • last Voir aussi <i>RequestPagination</i>^{API}
pageSize	Specifies the number of records per page. Integer type, default value is based on the user preferences. Voir aussi ebx.dataservices.pagination.maxSize.default [p 379]
sort	Specifies that the operation result will be sorted according to the specified criteria. The criteria are composed of one or more criterion, the result will be sorted by priority from the left. A criterion is composed of the field path and, optionally, the sorting order (ascending or descending, on value or on label). This parameter can be combined with: 1. the sortOnLabel [p 776] parameter as a new criteria added after the sort. 2. the viewPublication [p 777] parameter as a new criteria added after the sort. The value structure is as follows: <path1>:<order>;...;<pathN>:<order> Where: • <path1> corresponds to the field path in priority 1. • <order> corresponds to the sorting order, with one of the following values: • asc: ascending order on value (default), • desc: descending order on value, • lasc: ascending order on label, • ldesc: descending order on label. String type, the default value orders according to the primary key fields (ascending order on value). Note

Parameter	Description
	The history code operation value is usable with the ebx-operationCode path field from the meta section associated with this field. Note This parameter is ignored when the sort by relevancy [p 776] is activated. Voir aussi <i>Request.setSortCriteria</i>^{API}
sortByRelevancy	Specifies that the operation result will be sorted by relevancy, only if this condition is fulfill: - The Recherche rapide [p 128] is activated by specifying an osd:search predicate directly in the filter [p 774] parameter. String type. The possible values are: 1asc: ascending order on label, 1desc: descending order on label. If the sort by relevancy is activated, the following parameters are ignored: sort [p 775], sortOnLabel [p 776], sortPriority [p 776] and sort criteria defined through the viewPublication [p 777]. Voir aussi Search predicate [p 252]
sortOnLabel	Specifies that the operation result will be sorted according to the record label. This parameter can be combined with: - the sort [p 775] parameter as a new criteria added before the sortOnLabel. - the viewPublication [p 777] parameter as a new criteria added after the sortOnLabel. The value structure is as follows: <order> Where: <order> corresponds to the sorting order, with one of the following values: 1asc: ascending order on label, 1desc: descending order on label. The behavior of this parameter is described in the section defaultLabel [p 551]. String type value. Voir aussi Limitation [p 811] Note This parameter is ignored when the sort by relevancy [p 776] is activated.
sortPriority	Overrides the default priority of sort groups: - sort - sortOnLabel - sortFromView Comma separated String type, default value is: sort,sortOnLabel,sortFromView.

Parameter	Description
	Note This parameter is ignored when the sort by relevancy [p 776] is activated.
viewPublication	Specifies the name of the published view. This parameter can be combined with: 1. the filter [p 774] parameter as the logical and operation. 2. the sort [p 775] parameter as a new criteria added before the viewPublication. 3. the sortOnLabel [p 776] parameter as new criteria added before the viewPublication. The behavior of this parameter is described in the section EBX® as a Web Component [p 232]. String type value. Note The sort criteria, defined through this parameter, is ignored when the sort by relevancy [p 776] is activated. Voir aussi Look up table views operation [p 795]

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or osd:resource (Example [JSON](#) [p 838]). By default, a pagination mechanism is always enabled. Some

selector's select operations require input values. Thus, the uses of POST HTTP method [message body](#) [p 816] allows to provide a record content.

Parameter	Description
selector	Specifies whether: • <code>true</code>: returns all possible values (includes their labels) • <code>false</code>: returns the current values for the current field. Boolean type, default value is <code>false</code>. Note This parameter is ignored with the history category.
firstElementIndex	Specifies the index of the first element returned by the selector. Must be an integer higher than or equal to 0. Integer type, default value is 0.
pageSize	Specifies the number of elements per page. Integer type, default value is based on the user preferences. Voir aussi ebx.dataservices.pagination.maxSize.default [p 379]
selectorFilter	Specifies the filter of the selector. String type value, the syntax complies with the Recherche rapide [p 128].

HTTP codes

HTTP code	Description
200 (<i>OK</i>)	The selected resource is successfully retrieved.
400 (<i>Bad request</i>)	The request is incorrect. This occurs when: • The selected field in a record or a dataset is sub-terminal, • The XPath predicate of the <code>filter</code> parameter is malformed or contains unfilterable nodes. • The XPath predicate of the <code>primaryKey</code> parameter is malformed or is not a record primary key. • The sort criteria of the <code>sort</code> parameter have an invalid syntax or contain unsortable nodes. • <code>pageAction</code> parameter value is not included in allowed values, or <code>pageRecordFilter</code> is malformed or non-existent when selecting next or previous page. • <code>pageSize</code> parameter value is below 2. • The table view for the <code>viewPublication</code> parameter is either hierarchical, non-existent or non-published. • The <code>selector</code> parameter is used for a non-enumerated node, or the <code>firstElementIndex</code> is negative, higher than or equal to the number of values.
403 (<i>Forbidden</i>)	The selected resource is hidden for the authenticated user.
404 (<i>Not found</i>)	The selected resource is not found.

Response body

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

Examples: [Extended JSON](#) [p 819], [Compact JSON](#) [p 844].

Prepare operations

Prepare for creation or duplication operations are used to create a new transient record with an initial content or with the content of a record to duplicate. A transient record corresponds to a content that is not persisted yet. Default values and fields, initialized by table triggers, are considered. Only field values with, at least, read-only access permissions are returned. These operations can optionally return metadata to improve and assist data capture on the client side. Auto-incremented fields are only returned in the metadata. By enabling the `selector` parameter, a transient record's field can be queried to the retrieve possible values of an enumerated field, a foreign key, etc. Furthermore, selector's select operations allows to provide input [record data](#) [p 816] to manage use cases like getting metadata on a custom programmatic enumeration, etc.

Voir aussi `TableTrigger.handleNewContext`^{API}

Voir aussi [Default value](#) [p 56]

After being modified on the client side, the record can be submitted to be persisted using the built-in [Form insert operation](#) [p 796] or [Insert operation](#) [p 782] operations.

These prepare operations may use one of the following methods:

- GET HTTP method,
- POST HTTP method without message body or
- POST HTTP method with a message body and optionally [session parameters](#) [p 749].

Available URL formats are:

- **Prepare for creation**

- **Record:**

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{tablePath}:prepareForCreation
```

- **Record field:**

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{tablePath}:prepareForCreation/{pathInRecord}
```

- **Prepare for duplication**

- **Record:**

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{tablePath}/{encodedPrimaryKey}:prepareForDuplication
```

- **Record field:**

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{tablePath}/{encodedPrimaryKey}:prepareForDuplication/{pathInRecord}
```

Where:

- {category} corresponds to the [operation category](#) [p 759] (possible values are: data or data-compact).
- {dataspace} corresponds to B followed by the dataspace identifier.
- {dataset} corresponds to the dataset identifier.
- {tablePath} corresponds to the table path node.
- {encodedPrimaryKey} corresponds to the encoded representation of the primary key.

Voir aussi *RESTEncodingHelper*^{API}

- {pathInRecord} corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the prepare operations:

Parameter	Description
<code>includeDetails</code>	Includes the <code>details</code> field in the metadata and the content, for each indirect reachable resource. The returned value corresponds to its URL resource. Type Boolean, default value is <code>true</code> . Voir aussi includeMeta [p 781]
<code>includeMeta</code>	Includes the <code>meta</code> field holding the description of the structure returned in the content field. Boolean type, default value is <code>false</code> . Voir aussi Metadata [p 824]
<code>includeSelector</code>	Includes the <code>selector</code> field in the content, for each indirect reachable resource. The returned value corresponds to its URL resource. Type Boolean, default value is <code>true</code> . Voir aussi selector [p 778]

Selector parameters

The available parameters are similar to the selector operation ones.

Voir aussi [Selector parameters](#) [p 777]

HTTP codes

HTTP code	Description
200 (OK)	The selected resource has been successfully retrieved.
400 (Bad request)	The request is incorrect. This occurs when: - The selected field in a record is sub-terminal, <code>pageSize</code> parameter value is below 2, or is a string different from unbounded. - The selector parameter is used for a non-enumerated node, or the <code>firstElementIndex</code> is negative, higher than or equal to the number of values.
401 (Unauthorized)	Authentication has failed.
403 (Forbidden)	The selected resource is hidden from the authenticated user.
404 (Not found)	The selected resource could not be found.

Response body

After a successful prepare for a creation or duplication request, the transient record is returned in the response body. The content depends on the provided parameters and selected data. However, it takes a format similar to the select operation record.

Examples: [Extended JSON](#) [p 821], [Compact JSON](#) [p 844].

Insert operation

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

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

`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}`

Where:

- {category} corresponds to the [operation category](#) [p 759] (possible values are: data or data-compact).
- {dataspace} corresponds to B followed by the dataspace identifier or to v followed by the snapshot identifier.
- {dataset} corresponds to the dataset identifier.
- {pathInDataset} corresponds to the path of the table node.

Parameters

The following parameters are applicable with the insert operation.

Parameter	Description
<code>includeDetails</code>	Includes the <code>details</code> field in the content to access to the details of the data. The returned value corresponds to its URL resources. Type <code>Boolean</code>, the default value is <code>false</code>. Note Only applicable on the record table.
<code>includeForeignKey</code>	Includes the <code>foreignKey</code> field in the answer for each record. The returned value corresponds to the value of a foreign key field that was referencing this record. <code>Boolean</code> type, the default value is <code>false</code>. Note Only applicable on the record table.
<code>includeLabel</code>	Includes the <code>label</code> field in the answer for each record. Possible values are: • <code>yes</code>: the <code>label</code> field is included. • <code>no</code>: the <code>label</code> field is not included (use case: integration). <code>String</code> type, the default value is <code>no</code>. Note Only applicable on the record table.
<code>updateOrInsert</code>	Specifies the behavior when the record to insert already exists: • If <code>true</code>: the existing record is updated with new data. For a request on a record table, the <code>code</code> field is added to the report in order to specify if this is an insert 201 or an update 204. • If <code>false</code> (default value): a client error is returned and the operation is aborted. <code>Boolean</code> type value.
<code>blockingConstraintsDisabled</code>	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. <code>Boolean</code> type, default value is <code>false</code>. See Blocking and non-blocking constraints [p 581] for more information.

Message body

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

- **Record**: similar to the select operation of a record but without the record's header (example [Extended JSON](#) [p 816], [Compact JSON](#) [p 842]).

- **Record table:** Similar to the select operation on a table but without the pagination information (example [Extended JSON](#) [p 817], [Compact JSON](#) [p 842]).

Voir aussi [Inheritance](#) [p 793]

HTTP codes

HTTP code	Description
200 (OK)	If the request relates to a record table . The insert request was applied successfully, an optional report is returned in the response body.
201 (Created)	If the request relates to a record . A new record has been created, in this case, the header field <code>Location</code> is returned with its resource URL.
204 (No content)	If the request relates to a record . Only available if <code>updateOrInsert</code> is true, an existing record has been successfully updated, in this case, the header field <code>Location</code> is returned with its resource URL.
400 (Bad request)	The request is incorrect. This occurs when the body message structure does not comply with what was mentioned in Message body [p 783].
403 (Forbidden)	Authenticated user is not allowed to create a record or the request body contains a read-only field.
404 (Not found)	The selected resource is not found.
409 (Conflict)	Concurrent modification, only available if <code>updateOrInsert</code> is true, the Optimistic locking [p 792] is activated and the content has changed in the meantime, it must be reloaded before update.
422 (Unprocessable entity)	The request cannot be processed. This occurs when: • A blocking validation error occurs (only available if <code>blockingConstraintsDisabled</code> is false). • The record cannot be inserted because a record with the same primary key already exists (only available if <code>updateOrInsert</code> is false). • The record cannot be inserted because the definition of the primary key is either non-existent or incomplete. • The record cannot be updated because the value of the primary key cannot be modified.

Response body

The response body format depends on the inserted object type:

- **Record:** is empty if the operation was executed successfully. The header field `Location` is returned with its URL resource.
- **Record table:** (optional) contains a table of element(s), corresponding to the insert operation report (example [JSON](#) [p 853]). This report is automatically included in the response body, if at least one of the following options is set:

- `includeForeignKey`
- `includeLabel`
- `includeDetails`

Voir aussi [Inheritance](#) [p 793]

Update operation

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

- **Dataset node:** modifies the values of terminal nodes contained in the selected node.
`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}`
- **Record:** modifies the content of selected record.
`http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}`

Note

Also available for POST HTTP method. In this case, the URL must point to the table and the parameter `updateOrInsert` must be set to `true`.

- **Field:** update of a single field of the selected record.
`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}`

Note

The field must be either a terminal node or above.

Where:

- `{category}` corresponds to the [operation category](#) [p 759] (possible values are: `data` or `data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `v` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the dataset node:
 - For dataset node operations, this must be any terminal node or above except table node,
 - For record and field operations, this corresponds to the table node.
- `{encodedPrimaryKey}` corresponds to the encoded representation of the primary key.

Voir aussi `RESEncodingHelper`^{API}

- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

Here are the parameters applicable with the update operation.

Parameter	Description
blockingConstraintsDisabled	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. Boolean type, default value is <code>false</code> . See Blocking and non-blocking constraints [p 581] for more information.
byDelta	Specifies the behavior for setting value of nodes that are not defined in the request body. This is described in the Update modes [p 854] section. Boolean type, the default value is <code>true</code> .
checkNotChangedSinceLastUpdateDate	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 792] section. DateTime type value.

Message body

The request must define a message body.

The structure is the same as for:

- the dataset node (sample [Extended JSON](#) [p 815], [Compact JSON](#) [p 841]),
- the record (sample [Extended JSON](#) [p 816], [Compact JSON](#) [p 842]),
- the record fields (sample [Extended JSON](#) [p 817], [Compact JSON](#) [p 842]),

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

Voir aussi [Inheritance](#) [p 793]

HTTP codes

HTTP code	Description
204 (<i>No content</i>)	The record, field or dataset node has been successfully updated.
400 (<i>Bad request</i>)	The request is incorrect. This occurs when the body request structure does not comply.
403 (<i>Forbidden</i>)	Authenticated user is not allowed to update the specified resource or the request body contains a read-only field.
404 (<i>Not found</i>)	The selected resource is not found.
409 (<i>Conflict</i>)	Concurrent modification, the Optimistic locking [p 792] is activated and the content has changed in the meantime, it must be reloaded before the update.
422 (<i>Unprocessable entity</i>)	The request cannot be processed. This occurs when: - A blocking validation error occurs (only available if <code>blockingConstraintsDisabled</code> is <code>false</code>). - The record cannot be updated because the value of the primary key cannot be modified.

Delete operation

The operation uses the DELETE HTTP method.

Two URL formats are available:

- **Record:** delete a record specified in the URL.
`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}`
- **Record table:** deletes several records in the specified table, while providing a consistent answer. This mode requires a body message containing a record table. The deletions are executed sequentially, according to the order defined in the table. When an error occurs during a table operation, all deletions are cancelled and an error message is displayed with detailed information.
`http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/{pathInDataset}`

Where:

- {dataspace} corresponds to B followed by the dataspace identifier or to v followed by the snapshot identifier.
- {dataset} corresponds to the dataset identifier.
- {pathInDataset} corresponds to the path of the table node.
- {encodedPrimaryKey} corresponds to the encoded representation of the primary key.

Voir aussi `RESEncodingHelper`^{API}

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

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

Voir aussi [Inheritance](#) [p 793]

Parameters

Here are the following parameters applicable with delete operation.

Parameter	Description
<code>includeOcculting</code>	Includes occulted records. Boolean type, the default value is <code>false</code> .
<code>inheritIfInOccultingMode</code>	<i>Deprecated since version 5.8.1</i> While it remains available for backward compatibility reasons, it will eventually be removed in a future version. Inherits the record if it is in occulting mode. Boolean type, the default value is <code>true</code> .
<code>checkNotChangedSinceLastUpdateDate</code>	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 792] section. DateTime type value.
<code>blockingConstraintsDisabled</code>	Specifies whether blocking constraints are ignored, if so, the operation is committed regardless of the validation error created, otherwise, the operation would be aborted. Boolean type, default value is <code>false</code> . See Blocking and non-blocking constraints [p 581] for more information.

Message body

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

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

Voir aussi *PrimaryKey^{API}*

Examples: [Extended JSON](#) [p 818], [Compact JSON](#) [p 843].

HTTP codes

HTTP code	Description
200 (<i>OK</i>)	The operation has been executed successfully. A report is returned in the response body.
400 (<i>Bad request</i>)	The request is incorrect. This occurs when: - the structure of the message body does not comply with Message body [p 788]. - the message body contains a record table while the URL specifies a record.
403 (<i>Forbidden</i>)	Authenticated user is not allowed to delete or occult the specified record.
404 (<i>Not found</i>)	The selected record is not found. In the child dataset context, it should be necessary to use the <code>includeOcculting</code> parameter.
409 (<i>Conflict</i>)	Concurrent modification, The Optimistic locking [p 792] is activated and the content has changed in the meantime, it must be reloaded before deleting the record. The parameter value <code>checkNotChangedSinceLastUpdateDate</code> exists but does not correspond to the actual last update date of the record.
422 (<i>Unprocessable entity</i>)	Only available if <code>blockingConstraintsDisabled</code> is <code>false</code> , the operation fails because of a blocking validation error.

Response body

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

Example [JSON](#) [p 854].

Count operation

Count operation may use one of the following methods:

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

The URL formats are:

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

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}:count
```

Note

Not applicable with the history category.

- Table** depending on the [operation category](#) [p 759]:
the data category returns the number of table records.

The history category returns the number of table history records.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}:count
```

- **Field** depending on the [operation category](#) [p 759]:

the data category counts the record fields.

The history category counts the history record field.

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}:count
```

Note

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

Where:

- {category} corresponds to the [operation category](#) [p 759] (possible values are: data or data-compact).
- {dataspace} corresponds to B followed by the dataspace identifier or to v followed by the snapshot identifier.
- {dataset} corresponds to the dataset identifier.
- {pathInDataset} corresponds to the path of the dataset node, that can be a group node or a table node.
- {encodedPrimaryKey} corresponds to the encoded representation of the primary key.

Voir aussi *RESEncodingHelper^{API}*

- {pathInRecord} corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the count operation.

Parameter	Description
count	This parameter is deprecated since 6.0.0 and has been replaced by the extended action from the URL. It is used to specify whether this is a count operation or a selection operation. Boolean type, default value is false.

Table parameters

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

Parameter	Description
filter	XPath predicate [p 251] expression defines the field values to which the request is applied. If empty, all records will be considered. String type value. Note The history code operation value is usable with the ebx-operationCode path field from the meta section associated with this field.
historyMode	Specifies the filter context applied on table. String type, possible values are: - CurrentDataSpaceOnly: history in current dataspace - CurrentDataSpaceAndAncestors: history in current dataspace and ancestors - CurrentDataSpaceAndMergedChildren: history in current dataspace and merged children - AllDataSpaces: history in all dataspace The default value is CurrentDataSpaceOnly. Voir aussi history [p 759] Note This parameter is ignored with the data category.
includeOcculting	Includes the records in occulting mode. Boolean type, default value is false.
viewPublication	Specifies the name of the published view to be considered during the count execution. This parameter can be combined with: the filter [p 791] parameter as the logical and operation. The behavior of this parameter is described in the section EBX® as a Web Component [p 232].

Selector parameters

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

Parameter	Description
selector	Specifies whether: • <code>true</code>: returns the number of all possible values • <code>false</code>: returns the number of possible values for the current field. Boolean type, default value is <code>false</code>. Note This parameter is ignored with the history category.
selectorFilter	Specifies the filter of the selector. String type value, the syntax complies with the Recherche rapide [p 128].

HTTP codes

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

Optimistic locking

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

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

See [Technical data](#) [p 839] for more information.

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

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

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

See the [JSON](#) [p 816] example of a record.

- **Field:** update of a single field of the selected record.

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

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

Note

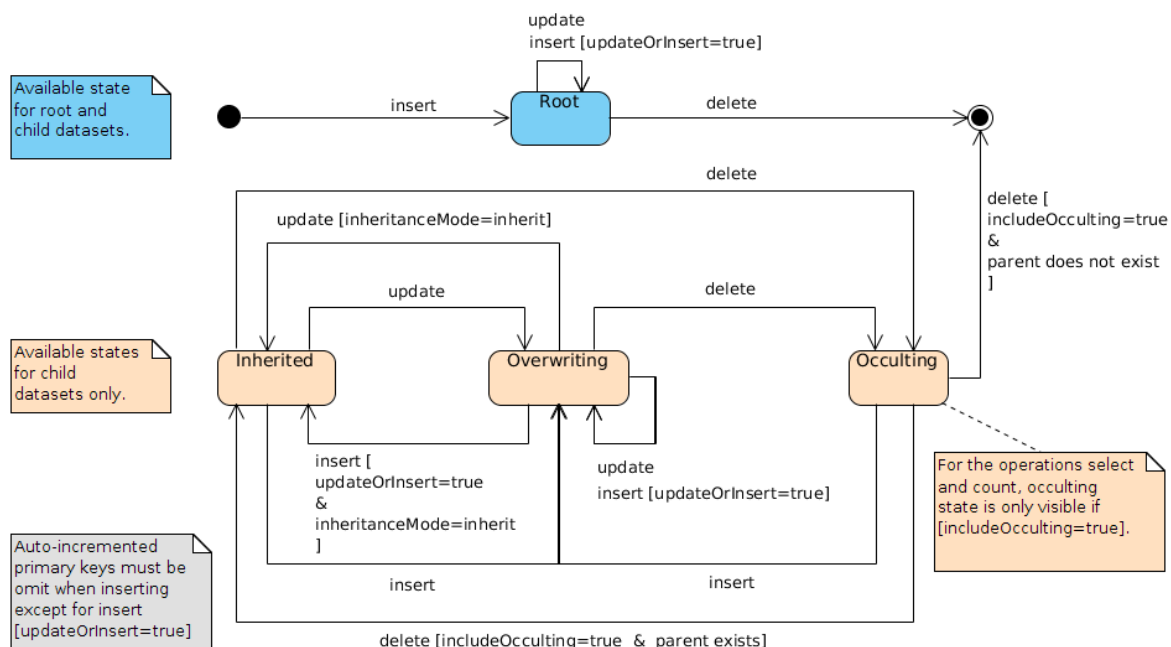
The `checkNotChangedSinceLastUpdateDate` parameter may be used more than once but only on the same record. This implies that if the request URL returns more than one record, the request will fail.

Inheritance

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

Voir aussi [Inheritance and value resolution](#) [p 288]

Record inheritance life cycle in built-in RESTful services



Inheritance properties

The following table describes properties related to the EBX® inheritance features.

Property	Location	Description
inheritance	record or table metadata	Specifies if dataset inheritance is activated for the table. The value is computed from the data model and cannot be modified through built-in RESTful services. Voir aussi inheritance property in metadata. [p 824]
inheritedField	field metadata	Specifies the field's value source. The source data are directly taken from the data model and cannot be modified through built-in RESTful services. Voir aussi inheritedField property in metadata. [p 825]
inheritanceMode	record in child dataset	Specifies the record's inheritance state. To set a record's inheritance from overwrite to inherit, its inheritanceMode value must be explicitly provided in the request. In this specific case, the content property will be ignored if present. occult and root explicit values are always ignored. An overwrite explicit value without a content property is ignored. Note Inherited record's fields are necessarily inherit. Note Root records in child dataset will always be root. Possible values are: root, inherit, overwrite, occult. For more information, see Record lookup mechanism [p 290].
	field in overwrite record	Specifies the field's inheritance state. To set a field's inheritance to inherit, its inheritanceMode value must be explicitly provided in the request. The content property will be ignored in this case. overwrite explicit value without a content property is ignored. Note inheritanceMode at field level does not appear for root, inherit and occult records. Note inheritedFieldMode and inheritanceMode properties cannot be both set on the same field. Possible values are: inherit, overwrite. For more information, see Inheritance and value resolution [p 288].
inheritedFieldMode	inherited field	Specifies the inherited field's inheritance state. To set a field's inheritance to inherit, its inheritedFieldMode value must be

Property	Location	Description
		explicitly provided in the request. The content property will be ignored in this case. overwrite explicit values without a content property are ignored. Note inheritedFieldMode and inheritanceMode properties cannot be both set on the same field. Note inheritedFieldMode has priority over the inheritanceMode property. Possible values are: inherit, overwrite. For more information, see Value lookup mechanism [p 291].

Look up table views operation

The "look up published views" operation may use one of the following methods:

- GET HTTP method or
- POST HTTP method without message body.

The URL format is:

```
http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
{tablePath: [^:]*}:publishedViews
```

Where:

- {dataspace} corresponds to B followed by the dataspace identifier or to v followed by the image identifier.
- {dataset} corresponds to the dataset identifier.
- {tablePath: [^:]*} corresponds to the table path.

Parameters

No specific parameter for this operation.

Codes HTTP

HTTP code	Description
200 (OK)	Information successfully returned.
400 (Bad request)	Incorrect request, contains an error.
401 (Unauthorized)	Authentication has failed.
403 (Forbidden)	The specified resource read permission has been denied to the authenticated user.
404 (Not found)	The selected resource cannot be found.

Response body

It contains a collection of authorized view information (including the access URI).

Example: [JSON](#) [p 860].

105.7 Form data operations

The form-data category operations concern the dataset fields, records or record fields when creating or modifying content must comply with constraints on user form submit. They are intended to be used in a user form management context.

The request body of an operation is very similar to the equivalent operation of the data category in the sense that an incoming data validation feature and a result report has been added to the response. The data validation can not be deactivated, a parameter `blockingConstraintsDisabled` from the data category operations is not applicable. The validation phase fails when, at least, one constraint with a `blocksCommit` level set to `onInsertUpdateOrDelete` or `onUserSubmit-checkModifiedValues` is violated.

See [Blocking and non-blocking constraints](#) [p 581] for more information.

Form insert operation

Insert form uses the `POST` HTTP method and requires a message body holding the data. This operation supports one or multiple records in a single transaction. Moreover, it is also possible to update record(s).

- **Record:** validate and insert a new record or modify an existing one in the selected table.
- **Record table:** validate and insert or modify one or more records in the selected table, while securing a consistent response. Operations are executed sequentially, in the order defined on the client side. When an error occurs during a table operation, all updates are cancelled and the client receives an error message with detailed information. The maximum number of records is limited to 100.

`http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}`

Where:

- `{category}` corresponds to the [operation category](#) [p 759] (possible values are: `form-data` or `form-data-compact`).
- `{dataspace}` corresponds to `B` followed by the dataspace identifier or to `V` followed by the snapshot identifier.
- `{dataset}` corresponds to the dataset identifier.
- `{pathInDataset}` corresponds to the path of the table node.

Parameters

The following parameters are applicable with this insert operation.

Parameter	Description
<code>includeDetails</code>	Includes the <code>details</code> field in the answer to access the data details. The returned value corresponds to its URL resources. Type <code>Boolean</code>, the default value is <code>false</code>. Note Only applicable for a multiple records insertion.
<code>includeForeignKey</code>	Includes the <code>foreignKey</code> field in the answer for each record. The returned value corresponds to the value of a foreign key field that was referencing this record. <code>Boolean</code> type, the default value is <code>false</code>. Note Only applicable for a multiple records insertion.
<code>includeLabel</code>	Includes the <code>label</code> field in the answer for each record. Possible values are: • <code>yes</code>: the <code>label</code> field is included. • <code>no</code>: the <code>label</code> field is not included (use case: integration). <code>String</code> type, the default value is <code>no</code>. Note Only applicable for a multiple records insertion.
<code>updateOrInsert</code>	Specifies the behavior when the record to insert already exists: • If <code>true</code>: the existing record is updated with new data. • If <code>false</code>: a client error is returned and the operation is aborted. <code>Boolean</code> type, default value is <code>false</code>.

Message body

The request must define a message body. The format is similar to the data category insert operation's [message body](#) [p 783].

HTTP codes

HTTP code	Description
200 (<i>OK</i>)	- The request body holds multiple records: the insert/update request was applied successfully, a report is returned in the response body. - The request body holds only one record and <code>updateOrInsert</code> is <code>true</code>: the update request was applied successfully, a report is returned in the response body.
201 (<i>Created</i>)	The request body holds only one non-existing record : a new record has been created and the header field <code>Location</code> is returned with its resource URL. Moreover, a report is returned in the response body.
400 (<i>Bad request</i>)	The request is incorrect. This occurs when the body message structure: - does not comply with what was mentioned in Message body [p 797]. - complies with what was mentioned in Message body [p 797], but number of record insert/update more than 100.
403 (<i>Forbidden</i>)	Authenticated user is not allowed to create a record or the request body contains a read-only field.
404 (<i>Not found</i>)	The selected resource is not found.
409 (<i>Conflict</i>)	Concurrent modification, only available if <code>updateOrInsert</code> is <code>true</code> , the Optimistic locking [p 792] is activated and the content has changed in the meantime, it must be reloaded before update.
422 (<i>Unprocessable entity</i>)	The request cannot be processed. This occurs when: - A blocking constraint has been violated. In this case an appropriate report is returned in the response body. - The record cannot be inserted because another one with the same primary key already exists (only available if <code>updateOrInsert</code> is <code>false</code>). - The record cannot be inserted because the definition of the primary key is either non-existent or incomplete. - The record cannot be updated because the value of the primary key cannot be modified.

Response body

The response body will always hold a validation report. However in case of failure, the response body corresponds to a `JSON Exception handling response`.

- Record:** The header field `Location` is returned with its URL resource. The validation report will be include in response body.
- Record table:** (optional) Contains a list of validations report for each element, corresponding to the multiple validation report.

Voir aussi [Form data category](#) [p 847]

Form update operation

As for the data or data-compact category update operation, it allows the modification of a single dataset or record and the PUT HTTP method must be used. Available URL formats are:

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

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}
```

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

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}
```

Note

Also available for POST HTTP method. In this case, the URL must point to the table and the parameter `updateOrInsert` must be set to `true`.

- **Field:** validates and update of a single field of the selected record.

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/{category}/v1/{dataspace}/{dataset}/{pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
```

Note

The field must be either a terminal node or above.

Where:

- {category} corresponds to the [operation category](#) [p 759] (possible values are: form-data or form-data-compact).
- {dataspace} corresponds to B followed by the dataspace identifier or to v followed by the snapshot identifier.
- {dataset} corresponds to the dataset identifier.
- {pathInDataset} corresponds to the path of the dataset node:
 - for dataset node operations, this must be any terminal node or above except table node,
 - for record and field operations, this corresponds to the table node.
- {encodedPrimaryKey} corresponds to the encoded representation of the record primary key.

Voir aussi *RESTEncodingHelper*^{API}

- {pathInRecord} corresponds to the path starting from the table node.

Parameters

Here are the parameters applicable with the update operation.

Parameter	Description
byDelta	Specifies the behavior for setting value of nodes that are not defined in the request body. This is described in the Update modes [p 854] section. Boolean type, the default value is true.
checkNotChangedSinceLastUpdateDate	Timestamp in datetime format used to ensure that the record has not been modified since the last read. Also see the Optimistic locking [p 792] section. DateTime type value.

Message body

The request must define a message body. The format is the same as for the data category update operation [message body](#) [p 786].

HTTP codes

HTTP code	Description
200 (<i>Ok</i>)	The update request was applied successful and a report is returned in the response body.
400 (<i>Bad request</i>)	The request is incorrect. This occurs when the body request structure does not comply.
403 (<i>Forbidden</i>)	Authenticated user is not allowed to update the specified resource or the request body contains a read-only field.
404 (<i>Not found</i>)	The selected resource is not found.
409 (<i>Conflict</i>)	Concurrent modification, the Optimistic locking [p 792] is activated and the content has changed in the meantime, it must be reloaded before the update.
422 (<i>Unprocessable entity</i>)	The request cannot be processed. This occurs when: - A blocking constraint has been violated. In this case an appropriate report is returned in the response body. - The record cannot be updated because the value of the primary key cannot be modified.

Response body

The response body has the same format and behavior as the [Form Insert operation](#) [p 798].

105.8 Beta feature: Dataspace operations

These operations perform a selection or life cycle management over dataspace.

Voir aussi [Dataspaces](#) [p 100]

Beta feature: Select dataspaces or snapshots

Select operation may use one of the following methods: GET or POST. A pagination mechanism is always enabled.

URL formats are:

- **Root:** returns root dataspaces
`http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/`
- **Children:** returns children dataspaces of a given dataspace
`http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/{dataspace}:children`
- **Snapshots:** returns snapshots of a given dataspace
`http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/{dataspace}:snapshots`
- **Information:** returns a dataspace, or a snapshot, information
`http[s]://<host>[:<port>]/ebx-dataservices/rest/data/v1/{dataspace}:information`

Where:

- {dataspace} corresponds to B followed by the dataspace's identifier or to v followed by the snapshot's identifier.

Parameters

The following query parameters are applicable to **Root**, **Children** and **Snapshots** operations.

Parameter	Description
includeClosed	Includes the closed dataspace to the selection. Boolean type, default value is false.
includeAdministration	Includes the administration dataspace to the selection. Boolean type, default value is false.
pageRecordFilter	Specifies a server side built filter, for pagination, pointing to a record. This filter is strongly linked to the pageAction value and should not be modified on the client side. The filter takes the form of a record XPath predicate [p 251] expression used to figure out the pagination contexts. String type. <i>Voir aussi Pagination [p 850]</i>
pageAction	Specifies the pagination action to perform from the identifier held by pageRecordFilter. String type, default value is first. The possible values are: • first • previous • next • last <i>Voir aussi RequestPagination^{API}</i>
pageSize	Specifies the maximum number of records per page. Integer type, default value is based on the user preferences. The value should be between 2 and 100.

HTTP codes

HTTP code	Description
200 (<i>OK</i>)	The request has been successfully handled.
400 (<i>Bad request</i>)	The request is incorrect. This occurs when: - pageAction parameter's value is not included in allowed values. - pageRecordFilter is malformed or non-existent when selecting next or previous page. - pageSize parameter's value is out of bound.
401 (<i>Unauthorized</i>)	Authentication has failed.
403 (<i>Forbidden</i>)	The selected resource is hidden from the authenticated user.
404 (<i>Not found</i>)	The selected resource could not be found.

Response body

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

The format is linked to the selected object type:

- For **Root**, **Children**, or **Snapshots**, see the [JSON](#) [p 861] example.
- For **Information**, see the [JSON](#) [p 863] example.

Beta feature: Create a child dataspace or a snapshot

Creates the dataspace or snapshot as specified. This operation use the POST method with a body request (with no specific query parameter).

Voir aussi `Repository.createHome`^{API}

URL format are:

- Dataspace:**
`http[s]://<host>[:<port>]/.../data/v1/{dataspace}:createDataspace`
- Snapshot:**
`http[s]://<host>[:<port>]/.../data/v1/{dataspace}:createSnapshot`

Where:

- {dataspace} corresponds to B followed by the dataspace identifier.

Request body

The body specifies the features of the dataspace or the snapshot to create.

Voir aussi `HomeCreationSpec`^{API}

See the [JSON](#) [p 864] example.

HTTP codes

HTTP code	Description
201 (<i>Created</i>)	A new dataspace or snapshot has been created, in this case, the header field <code>Location</code> is returned with its resource URL.
400 (<i>Bad request</i>)	The request body is not well-formed or contains invalid content.
401 (<i>Unauthorized</i>)	Authentication has failed.
403 (<i>Forbidden</i>)	Permission was denied to create the specified resource for the authenticated user.
404 (<i>Not found</i>)	The resource specified in the request body cannot be found.
500 (<i>Internal error</i>)	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Beta feature: Locking a dataspace

Locks the specified dataspace. If the dataspace is already locked:

- by the same user, then the lock is kept,
- by another user and during the wait duration, then the lock is acquired,
- otherwise, the lock is rejected.

This operation uses the `POST` method and consumes `Content-Type` header set to:

- `application/x-www-form-urlencoded`: with HTTP parameters in the body or
- `application/json`: with HTTP parameters in the URL and [Session parameters](#) [p 749] into JSON body.

When it succeeds, no response body is returned.

Voir aussi

[Permissions for locking or unlocking a dataspace](#) [p 295]

`LockSpec.lockAPI`

URL format is:

`http[s]://<host>[:<port>]/.../data/v1/{dataspace}:lock`

Where:

- `{dataspace}` corresponds to `B` followed by the dataspace identifier.

Parameters

The following query parameters are applicable to the operation.

Parameter	Description
<code>durationToWaitForLock</code>	When the dataspace is locked by another user, specifies the maximum duration to wait for acquiring the lock. The duration is specified in seconds. If the value is set to 0, then the locking attempt is performed immediately. Due to several reasons, the wait duration can not exceed 60 seconds. Otherwise, the value is overwritten with its maximum value. Integer type, default value is 0.

HTTP codes

HTTP code	Description
204 (<i>No content</i>)	The request has been successfully handled but no response body is returned.
400 (<i>Bad request</i>)	The request URL or body is not well-formed or contains invalid content.
401 (<i>Unauthorized</i>)	Authentication has failed.
403 (<i>Forbidden</i>)	Permission was denied to lock the specified resource for the authenticated user.
404 (<i>Not found</i>)	The resource specified in the URL cannot be found.
409 (<i>Conflict</i>)	The resource is already locked by another user, the lock is rejected after <code>durationToWaitForLock</code> parameter's value.
500 (<i>Internal error</i>)	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Beta feature: Unlocking a dataspace

Unlocks the specified dataspace. If the dataspace is:

- locked by the same user, then the lock is released,
- locked by another user, the current user is an administrator, and the `forceByAdministrator` query parameter is set to `true`, then the lock is released,
- not locked, the lock status is unchanged,
- otherwise, the unlock is rejected.

This operation uses the POST method and consumes Content-Type header set to:

- `application/x-www-form-urlencoded`: with HTTP parameters in the body or
- `application/json`: with HTTP parameters in the URL and [Session parameters](#) [p 749] into JSON body.

When it succeeds, no response body is returned.

Voir aussi

[Permissions for locking or unlocking a dataspace](#) [p 295]

`LockSpec.unlockAPI`

URL format is:

`http[s]://<host>[:<port>]/.../data/v1/{dataspace}:unlock`

Where:

- {dataspace} corresponds to B followed by the dataspace identifier.

Parameters

The following query parameters are applicable to the operation.

Parameter	Description
<code>forceByAdministrator</code>	When the dataspace is locked by another user, an administrator can force the unlocking. Boolean type, default value is <code>false</code> .

HTTP codes

HTTP code	Description
204 (<i>No content</i>)	The request has been successfully handled but no response body is returned.
400 (<i>Bad request</i>)	The request URL or body is not well-formed or contains invalid content.
401 (<i>Unauthorized</i>)	Authentication has failed.
403 (<i>Forbidden</i>)	Permission was denied to unlock the specified resource for the authenticated user.
404 (<i>Not found</i>)	The resource specified in the URL cannot be found.
409 (<i>Conflict</i>)	The resource is already locked by another user, or the current user is administrator but <code>forceByAdministrator</code> parameter is <code>false</code> , the unlock is rejected.
500 (<i>Internal error</i>)	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Beta feature: Merge a dataspace

Merges the specified dataspace to its parent. It is possible to perform deletion, after the merge, on history and / or on data.

This operation uses the POST method and consumes Content-Type header set to:

- `application/x-www-form-urlencoded`: with HTTP parameters in the body or
- `application/json`: with HTTP parameters in the URL and [Session parameters](#) [p 749] into JSON body.

When it succeeds, no response body is returned.

Note

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

Voir aussi

ProcedureContext.doMergeToParent^{API}

[Deleting data and history](#) [p 418]

URL format is:

`http[s]://<host>[:<port>]/.../data/v1/{dataspace}:merge`

Where:

- {dataspace} corresponds to B followed by the dataspace identifier.

Parameters

The following query parameters are applicable to the operation.

Parameter	Description
deleteHistoryOnMerge	Sets whether the history associated with the specified dataspace will be deleted upon merge. Boolean type, default value is false. When this parameter is not specified in the request, the default value is false. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> in the EBX® main configuration file [p 379].
deleteDataOnMerge	Sets whether the specified dataspace and its associated snapshots will be deleted upon merge. Boolean type, default value is false. When this parameter is not specified in the request, the default value is false. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> in the EBX® main configuration file [p 379].

HTTP codes

HTTP code	Description
204 (<i>No content</i>)	The request has been successfully handled but no response body is returned.
400 (<i>Bad request</i>)	The request URL contains invalid content.
401 (<i>Unauthorized</i>)	Authentication has failed.
403 (<i>Forbidden</i>)	Permission was denied to merge the specified resource for the authenticated user.
404 (<i>Not found</i>)	The resource specified in the URL cannot be found.
500 (<i>Internal error</i>)	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

Beta feature: Close a dataspace or a snapshot

Closes the specified dataspace or snapshot. It is possible to perform deletion, after close, on history and / or on data.

This operation uses the POST method and consumes Content-Type header set to:

- application/x-www-form-urlencoded: with HTTP parameters in the body or
- application/json: with HTTP parameters in the URL and [Session parameters](#) [p 749] into JSON body.

When it succeeds, no response body is returned.

Voir aussi

Repository.closeHome^{API}

[Deleting data and history](#) [p 418]

URL format is:

http[s]://<host>[:<port>]/.../data/v1/{dataspace}:close

Where:

- {dataspace} corresponds to B followed by the dataspace identifier or to v followed by the snapshot identifier.

Parameters

The following query parameters are applicable to the operation.

Parameter	Description
<code>deleteHistoryOnClose</code>	Sets whether the history associated with the specified dataspace will be deleted upon close. Boolean type, default value is <code>false</code>. When this parameter is not specified in the request, the default value is <code>false</code>. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.historyDeletionOnCloseOrMerge.default</code> in the EBX® main configuration file [p 379].
<code>deleteDataOnClose</code>	Sets whether the specified dataspace and its associated snapshots will be deleted upon close. Boolean type, default value is <code>false</code>. When this parameter is not specified in the request, the default value is <code>false</code>. It is possible to redefine the default value by specifying the property <code>ebx.dataservices.dataDeletionOnCloseOrMerge.default</code> in the EBX® main configuration file [p 379].

HTTP codes

HTTP code	Description
204 (<i>No content</i>)	The request has been successfully handled but no response body is returned.
400 (<i>Bad request</i>)	The request URL or body is not well-formed or contains invalid content.
401 (<i>Unauthorized</i>)	Authentication has failed.
403 (<i>Forbidden</i>)	Permission was denied to close the specified resource for the authenticated user.
404 (<i>Not found</i>)	The resource specified in the URL cannot be found.
422 (<i>Unprocessable entity</i>)	A blocking constraint has been violated.
500 (<i>Internal error</i>)	Unexpected error thrown by the application. Error details can usually be found in EBX® logs.

105.9 OpenAPI operations

Overview

The [open-api](#) [p 759] category operations comply with the OpenAPI specification 3.0.X to generate JSON documents. These documents facilitate development and consumption by structuring and describing the available REST built-in resources and the operations associated with them.

OpenAPI document HATEOAS links are available through the select data operations when using the [includeOpenApiDetails](#) [p 773] query parameter.

The REST OpenAPI services permissions are globally defined in [Global permissions](#) [p 753].

Making the proper types mapping for generating a client, in a specific language, is essential. The [OpenAPI format](#) can be used to validate the input or to map the value to a specific type, in the chosen programming language. See [Content of simple type](#) [p 837] for more information.

Note

Tools that do not support a specific format may default back to the type alone.

Generate OpenAPI document

The operation uses the GET or POST HTTP method to generate the OpenAPI document of a dataset table or a schema node.

The URL format is:

```
http[s]://<host>[:<port>]/ebx-dataservices/rest/open-api/v1/{category}/v1/{dataspace}/{dataset}/{pathInDataset}
```

Where:

- {category} corresponds to an [operation category](#) [p 759] among [data](#) [p 769] or [form-data](#) [p 796].
- {dataspace} corresponds to B followed by the dataspace identifier or to v followed by the snapshot identifier.
- {dataset} corresponds to the dataset identifier.
- {pathInDataset} corresponds to the path of:
 - a table node,
 - a dataset terminal node or above.

Note

The generated document will not depend on user permissions. The whole fields will be presented.

HTTP codes

HTTP code	Description
200(OK)	The request has been successfully handled.
401(Unauthorized)	Authentication has failed.
403(Forbidden)	Permission was denied to read the specified resource.
404(Not found)	The resource, specified in the URL, cannot be found.

105.10 Limitations

General limitations

- Indexes, in the request URL {pathInDataset} or {pathInRecord}, are not supported.
- Nested aggregated lists are not supported.
- Dataset nodes and field operations applied to nodes that are sub-terminal are not supported.
See [Access properties](#) [p 597] for more information about terminal nodes.

Compact format limitations

- [Inheritance](#) [p 793], [Tracking information](#) [p 749], [Session parameters](#) [p 749] and [Procedure context](#) [p 752] are not handled in the [compact format](#) [p 841]. Use the [extended](#) [p 815] one instead.
- [History category](#) [p 759] is not supported in the [compact format](#) [p 841].

Read operations

- Within the selector, the pagination context is limited to the nextPage property.
- When the [sortByRelevancy](#) [p 776] parameter is activated, the ones below are ignored: [sort](#) [p 775], [sortOnLabel](#) [p 776], [sortPriority](#) [p 776] and sort criteria defined through the [viewPublication](#) [p 777].
- Within the viewPublication parameter, the hierarchical view is not supported.
- The sortOnLabel parameter ignores programmatic labels.
- The system information response's properties cannot be browsed through the REST URL with the hierarchical representation.
See [System information operation](#) [p 765] for more information.
- The prepare for creation operation does not support a dataset node.
- The [select dataspace operation](#) [p 801] does not sort them by label.
- The [select snapshots operation](#) [p 801] does not sort them by creation date.
- The [select snapshots operation](#) [p 801] can not include the initial snapshots.

Write operations

- Association fields cannot be updated, therefore, the list of associated records cannot be modified directly.
- Control policy onUserSubmit-checkModifiedValues of the user interface is not supported. To retrieve validation errors, invoke the select operation on the resource by including the includeValidation parameter.
See [Blocking and non-blocking constraints](#) [p 581] for more information.

Directory operations

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

OpenAPI operations

- The document generation is not available through the user interface.
- The document generation REST services does not support the YAML format.
- Supports only [Data operations](#) [p 769] and [Form data operations](#) [p 796] except:
 - Select **Dataset tree**, **Dataset node** and **Field** operations.
 - Insert or delete multiple records in a single request.
 - Use of the HTTP header content-type: `x-www-form-urlencoded` to send query parameters in the body request.
- The '[Basic Authentication Scheme](#)' [p 753] is the only described method.
- The document generation is not supported for custom REST toolkit services.

CHAPITRE 106

Introduction

Ce chapitre contient les sections suivantes :

1. [Overview](#)

106.1 Overview

The JSON (JavaScript Object Notation) is the data-interchange format used by TIBCO EBX®'s [RESTful operations](#) [p 757].

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

Two JSON formats are available:

- Using the [extended format](#) [p 815], the data context is exhaustive and contains features to retrieve technical information and metadata, except for association fields and selection nodes. However, these fields are reachable from the response through URL links named details included by default.
- Using the [compact format](#) [p 841], the data context is limited to master data without any technical information nor metadata except links for the enumerated fields.

The amount of retrieved data is limited by a [pagination](#) [p 850] mechanism which can be configured.

URL links allow reaching:

- Tables, records, dataset non-terminal nodes, foreign keys, resource fields through the `details` property. See [includeDetails](#) [p 772] for more information.
- Possible values for foreign keys or enumerations, by activating the `selector` parameter. See [includeSelector](#) [p 773] for more information.

Voir aussi [Activation and configuration](#) [p 749]

Note

JSON data are always encoded with the UTF-8 charset.

CHAPITRE 107

Extended

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Global structure](#)
3. [Metadata](#)
4. [Sort criteria](#)
5. [Validation report](#)
6. [Constraints](#)
7. [Content](#)

107.1 Introduction

The JSON extended format is used to retrieve master data, technical information and metadata. It is designed in an expanded way that allows to include several features such as validation, sorting and so on. To activate the extended format, the unsuffixed REST category, like data or form-data, must be used in the URL.

107.2 Global structure

JSON Request body

The request body is represented by a JSON object whose content varies according to the operation and the category.

Data category

The request body contains at least a content property which hold master data values.

- **Dataset node**

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

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

```

    },
    "nodeName3": {
      "content": "Hello"
    }
  }
}

```

Voir aussi [Update operation](#) [p 785]

- **Record**

Specifies the target record content by setting the value of each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is used for table record insert, record update or selector's select operation when local enumeration dependency field values are declared.

Voir aussi [Inheritance](#) [p 793]

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

Voir aussi [Optimistic locking](#) [p 792]

```

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

```

Voir aussi

[Insert operation](#) [p 782]

[Update operation](#) [p 785]

- **Record fields**

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

Voir aussi [Inheritance](#) [p 793]

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

Voir aussi [Update operation](#) [p 785]

- **Record table**

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

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

Voir aussi

[Insert operation](#) [p 782]

[Update operation](#) [p 785]

- **Record table to deleted**

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

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

Voir aussi [Delete operation](#) [p 787]

- **Field**

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

The request has the same structure as defined in [node value](#) [p 836] by only keeping the content entry. Additional entries are simply ignored.

Voir aussi [Update operation](#) [p 785]

- **Open or close user interface**

Specifies whether the user interface is open or close and the unavailability message.

```
{
  "content": {
    "toolStatus": {
      "content": true    // or false
    },
    "toolStatusCloseMessage": {
      "content": "Access is temporarily forbidden for maintenance."
    }
  }
}
```

Voir aussi [User interface operations](#) [p 765]

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

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

JSON Response body

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

Data category

The selection operation contains two different parts.

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

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

- **Dataset tree**

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

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

```

    "details": "http://.../rootName/settings",
    "openApiDetails": "http://.../open-api/.../rootName/settings",
    "content": {
      "weekTimeSheet": {
        "details": "http://.../rootName/settings/settingA",
        "openApiDetails": "http://.../open-api/.../rootName/settings/settingA"
      },
      "vacationRequest": {
        "details": "http://.../rootName/settings/settingB",
        "openApiDetails": "http://.../open-api/.../rootName/settings/settingB"
      }
    },
    "table1": {
      "details": "http://.../rootName/table1",
      "openApiDetails": "http://.../open-api/.../rootName/table1"
    },
    "table2": {
      "details": "http://.../rootName/table2",
      "openApiDetails": "http://.../open-api/.../rootName/table2"
    }
  }
}

```

The meta and validation properties are optional.

Voir aussi

[Metadata](#) [p 824]

[Validation report](#) [p 832]

[Select operation](#) [p 769]

- **Dataset node**

Contains the list of terminal nodes under the specified node.

```

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

```

Voir aussi [Select operation](#) [p 769]

- **Table**

JSON object containing the following properties:

- (Optional) The [table meta data](#) [p 824],
- (Optional) The sort criteria applied,
- (Optional) The table validation report,
- The rows property corresponding to a JSON Array of the selected records. Each record is represented by an JSON object. If no record is selected then the JSON Array is empty.
- (Optional) the pagination property, containing [pagination](#) [p 850] data.

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

Voir aussi [Select operation](#) [p 769]

- **Record**

JSON object containing:

- The label,
- (Optional) The record URL,
- (Optional) The [technical data](#) [p 839],
- (Optional) The [table metadata](#) [p 824],

- (Optional) The record validation report,
- (Optional) The inheritance mode of the record, which can be: root, inherit, overwrite or occult.

Voir aussi

[*Record lookup mechanism*](#) [p 290]

[*Inheritance*](#) [p 793]

- The record content.

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

```

    ]
  },
  "content": {
    "pk": {
      "content": "pk1"
    },
    "name": {
      "content": "Name1"
    },
    "name-fr": {
      "content": "Name1",
      "inheritedFieldMode": "inherit"
    },
    "parent": {
      "content": null,
      "selector": "http://.../rootName/table1?selector=true",
      "validation": [
        {
          "level": "error",
          "message": "Field 'Parent' is mandatory."
        }
      ]
    }
  ]
},
"validation": {
  ...
}
}

```

Voir aussi

[Select operation](#) [p 769]

[Prepare operations](#) [p 779]

- **Fields**

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

For other kinds of nodes, contains the current [node value](#) [p 836].

Voir aussi [Select operation](#) [p 769]

- **Retrieve the user interface state**

Contains the user interface status and the unavailability message.

```

{
  "content": {
    "toolStatus": {
      "content": true,
      "label": "Open",
      "selector": "http://.../domain/toolStatus/toolStatus?selector=true"
    },
    "toolStatusCloseMessage": {
      "content": "Access is temporarily forbidden for maintenance."
    }
  }
}

```

Voir aussi [User interface operations](#) [p 765]

Note

Nodes, records and fields, property and values may be hidden depending on their resolved permissions (see [permissions](#) [p 293]) .

107.3 Metadata

This section can be activated on demand with the [includeMeta](#) [p 772] parameter. It describes the structure and the JSON typing of the content section.

This section is deactivated by default for selection operations.

Voir aussi

[Select operation](#) [p 769]

[Prepare operations](#) [p 779]

Structure for table

Table metadata is represented by a JSON object with the following properties:

JSON property	JSON format	Description	Required
name	String	Specifies the name of the authorized table defined in the model.	Yes
label	String	Specifies the table's label. If undefined, the name of the schema node is returned.	Yes
description	String	Specifies the table's description.	No
type	String	Specifies the node's type, the value is always equal to: table.	Yes
minOccurs	Number	Specifies the number of minimum authorized record(s).	Yes
maxOccurs	Number or String	Specifies the number of maximum authorized record(s) or unbounded.	Yes
history	Boolean	Specifies if the table content is historized. Its value is true if history is activated, false otherwise. <i>Voir aussi History [p 269]</i>	No
primaryKeyFields	Array	Specifies the primary key composition which is represented as an array of the paths.	Yes
inheritance	Boolean	Specifies whether the dataset inheritance is activated for the table. Its value is true if inheritance is activated, false otherwise. <i>Voir aussi Inheritance and value resolution [p 288]</i>	No
fields	Array	Specifies the direct children of the record node which are represented as an array of fields. Each field may also recursively contain sub-fields.	Yes

Structure for field

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

JSON property	JSON format	Description	Required
name	String	Specifies the name of the authorized field defined in the model.	Yes
label	String	Specifies the node's label. If undefined, the name of the schema node is returned.	Yes
description	String	Specifies the field's description.	No
type	String	Specifies the field's type, which can be a: simple type [p 837], group, table, foreignKey [p 825], association [p 826], etc.	Yes
minOccurs	Number	Specifies the number of minimum authorized occurrence(s).	Yes
maxOccurs	Number or String	Specifies the number of maximum authorized occurrence(s) or unbounded.	Yes
readOnly	Boolean	Specifies whether the field access permission is defined as read-only from the data model access properties. That is, if its data can be read or written. Its value is true if the field is read only, false otherwise. Voir aussi Access properties [p 597] <i>AccessPermission^{API}</i>	Yes
autoIncrement	Boolean	Specifies whether the field is auto-incremented. This property is only available for integer field type. Its value is true if the field is auto-incremented, false otherwise. Voir aussi Auto-incremented values [p 589]	No
checkNullInput	Boolean	Specifies whether the check of the constraints on null fields, at user input, is activated or not. Its value is true when activated, false otherwise. The constraint check is activated when the field has the property osd:checkNullInput [p 584] set to true.	Yes
inheritedField	Object	Holds information related to the inherited field's value source. <pre>{ "inheritedField": { "sourceRecord": "/path/to/record", // (optional) "sourceNode": "../path/to/Node" } }</pre> Voir aussi Inheritance and value resolution [p 288]	No
foreignKey	Object	Holds information related to the target table.	No (*)

JSON property	JSON format	Description	Required
		Voir aussi Structure for foreign key field [p 827]	
association	Object	Holds information related to the association table. Voir aussi Structure for the association field [p 828]	No (*)
enumeration	String	Specifies if the field is an enumeration value. Possible values are: - dynamic - foreignKey - nomenclature - programmatic - resource - static Voir aussi SchemaFacetEnumeration ^{API} According to its value, indicates if the field can be used for retrieving possible values by using the selector [p 837] request parameter.	No
enumerationDependencies	Array of JSON Object	Specifies the metadata enumeration dependencies list. Only available for enumeration types among: foreignKey, dynamic or programmatic. Voir aussi Structure for the enumeration dependencies [p 830]	No
valueFunction	Boolean	Specifies if the field is a computed value. Voir aussi Computed values [p 587]	No
linkedField	Object	Holds information related to the linked field. Voir aussi <i>SchemaNode.getSchemaLinkedField()</i> <i>SchemaNode.getSchemaLinkedField</i> ^{API} Structure for the linked field [p 829]	No
pathInDataset	String	Specifies the relative field's path starting from the schema node.	No (**)
pathInRecord	String	Specifies the relative field's path starting from the table node.	No (*)
filterable	Boolean	Specifies whether the field can be used to filter records using the filter [p 774] request parameter.	No (*)
hiddenFilterPolicy	String	Specifies the hidden filter policy. The possible values are: textSearchOnly, true or false.	No (*)

JSON property	JSON format	Description	Required
		Voir aussi Hiding a field in structured search tools [p 600] <code>SchemaNodeDefaultView.getHiddenInSearch^{APT}</code>	
sortable	Boolean	Specifies whether the field can be used in sort criteria using the sort [p 775] request parameter.	No (*)
constraints	Array of JSON Object	Specifies the field's non-programmatic constraints. Voir aussi Constraints [p 832]	No
fields	Array of JSON Object	Holds the structure and typing of each field group.	No

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

(**) Only available for dataset tree operations.

Structure for foreign key field

The foreign key field metadata is represented by a JSON object.

```
{
  "dataspace": "BAuthors",
  "dataset": "Authors",
  "tablePath": "/root/Authors",
  "details": "http://.../BAuthors/Authors/root/Authors"
}
```

It holds the following properties:

JSON property	JSON format	Description	Required
dataspace	String	Specifies the target dataspace or snapshot identifier. If not specifies then it corresponds to the same dataspace as the table's one.	No
dataset	String	Specifies the target dataset's identifier. If not specifies then it corresponds to the same dataset as the table's one.	No
tablePath	String	Specifies the target table's path.	Yes
details	String	Specifies the target table's REST resource URL. Voir aussi includeDetails parameter [p 772]	No
historyDetails	String	Specifies the history target table's REST resource URL. Voir aussi includeDetails parameter [p 772]	No

Structure for the association field

The association field metadata is represented by a JSON object. This object holds properties which depend on the association type:

- tableRefInverse

```
{
  "type": "tableRefInverse",
  "dataspace": "BTitles",
  "dataset": "Titles",
  "tablePath": "/root/Titles",
  "details": "http://.../BTitles/Titles/root/Titles",
  "fieldToSource": "/root/Titles/au_id",
  "filter": "./unit_price > 10"
}
```

- linkTable

```
{
  "type": "linkTable",
  "tablePath": "/root/Inventory",
  "details": "http://.../BInventory/Inventory/root/Inventory",
  "linkTablePath": "/root/Inventory",
  "fieldToSource": "/root/Inventory/store",
  "fieldToTarget": "/root/Inventory/item",
  "filter": "./price > 10"
}
```

- xpathLink

```
{
  "type": "xpathLink",
  "tablePath": "/root/Inventory",
  "details": "http://.../BAuthors/Authors/root/Inventory",
  "predicate": "/root/Inventory[./price < 100]",
  "filter": "./price > 10"
}
```

It holds the following properties:

JSON property	JSON format	Description	Required
type	String	Specifies the association type. The possible values are: <code>tableRefInverse</code> , <code>linkTable</code> , <code>xpathLink</code> Voir aussi Associations [p 558]	Yes
dataspace	String	Specifies the target dataspace or snapshot identifier. If not specifies, then it corresponds to the same dataspace as the table's one.	No
dataset	String	Specifies the target dataset's identifier. If not specifies, then it corresponds to the same dataset as the table's one.	No
tablePath	String	Specifies the path of the target table.	Yes
details	String	Specifies the target table's REST resource URL. Voir aussi includeDetails parameter [p 772]	No
fieldToSource	String	Specifies the field which refers to the source table of the association. Defined if the association type is <code>tableRefInverse</code> or <code>linkTable</code> .	No
fieldToTarget	String	Specifies the path of the field to the target table. Defined if the association type is <code>linkTable</code> .	No
linkTablePath	String	Specifies the path of the link table. Defined if the association type is <code>linkTable</code> .	No
predicate	String	Specifies the criteria of the association, relative to the current node. Defined if the association type is <code>xpathLink</code> .	No
filter	String	Specifies the XPath predicate expression to filter the associated objects.	No

Structure for the linked field

The linked field metadata is represented by a JSON object.

```
{
  "relationshipField": "/au_id",
  "tablePath": "/root/Authors",
  "linkedFieldPath": "/birth_date",
  "details": "http://.../BAuthors/Authors/root/Authors"
}
```

It holds the following properties:

JSON property	JSON format	Description	Required
relationshipField	String	Specifies the field's path holding the relationship used by this linked field. <i>Voir aussi <code>SchemaLinkedField.getRelationshipNode</code>^{API}</i>	Yes
tablePath	String	Specifies the table's path referred by this linked field.	Yes
linkedFieldPath	String	Specifies the path referred by this linked field. <i>Voir aussi <code>SchemaLinkedField.getLinkedField</code>^{API}</i>	Yes
details	URI	Specifies the target table's REST resource URL. <i>Voir aussi includeDetails parameter [p 772]</i>	No

Voir aussi

[Linked fields](#) [p 568]

`SchemaLinkedField`^{API}

Structure for the enumeration dependencies

The metadata enumeration dependency object is represented by a JSON object.

```
{
  "localModify": false,
  "dataspace": "BAuthors",
  "dataset": "Authors",
  "targetPath": "/root/Authors",
  "details": "http://.../BAuthors/Authors/root/Authors"
}
```

It holds the following properties:

JSON property	JSON format	Description	Required
localModify	Boolean	Specifies whether the enumeration dependency field is impacted by modifications performed on a field in the same record.	Yes
dataspace	String	Specifies the target dataspace or snapshot identifier of the enumeration dependency. If not specified then it corresponds to the same dataspace as the record's one.	No
dataset	String	Specifies the target dataset's identifier of the enumeration dependency. If not specifies then it corresponds to the same dataset as the record's one.	No
targetPath	String	Specifies the target path of the enumeration dependency.	Yes
details	String	Specifies the target REST resource table URL of the enumeration dependency. Voir aussi includeDetails parameter [p 772]	No

107.4 Sort criteria

The sort criteria, applied to the request, can be returned on demand, by using the [includeSortCriteria](#) [p 773] parameter (deactivated by default). If it is activated, a sortCriteria property is directly added to the response root node.

A sortCriteria property is represented as a JSON Array that contains ordered sort criterion. Each sort criterion corresponds to a JSON object with the following properties:

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

The sortByRelevancy property is represented by a JSON Array, itself holding a JSON object with the following properties:

JSON property	JSON format	Description	Required
order	String	Defines whether the sort is done in ascending or descending order. Possible values are: lasc or ldesc. See sortByRelevancy [p 776] for more information.	Yes

107.5 Validation report

The validation can be activated on demand with the [includeValidation](#) [p 773] parameter (deactivated by default). If it is activated, validation properties are directly added on target nodes with one or several messages. For messages without target node's path, a validation property is added on the root node.

A validation property is represented by a JSON Array, holding a JSON object per message, corresponding to a validation item, with the following properties:

JSON property	JSON format	Description	Required
level	String	Severity level of the validation item. The possible values are: info, warn, error, fatal.	Yes
message	String	Description of the validation item.	Yes
details	String corresponding to an absolute URL.	URL of the resource associated with the validation item. Only available on table and dataset scopes, if the associated resource exist and if it is included. Voir aussi includeDetails parameter [p 772]	No

107.6 Constraints

This section is automatically activated when the node has a read-write permission. It provides the non-programmatic constraints declared on the data model.

The constraints property is represented by a JSON Array. Every constraint is taking the form of a JSON object. They have the following properties:

JSON property	JSON format	Description	Required
type	String	Specifies the type of the constraint Voir aussi Constraint Types [p 834]	Yes
level	String	Specifies the severity level of the constraint. The possible values are: fatal, error, warning or info.	Yes
content	Object	Specifies the value of the non-programmatic constraint. This value can be a simple type, like a String, or a JSON Object.	No
min	Integer	Specifies the minimum value for excludeSegment [p 834] constraint type.	No
max	Integer	Specifies the maximum value for excludeSegment [p 834] constraint type.	No
message	String	Specifies the validation message returned when the value of the field does not comply with the constraint.	Yes
blocksCommit	String	Specifies the control policy management for blocking or non-blocking constraint. See Blocking and non-blocking constraints [p 581] for more information.	No

Constraint Types

This section lists the non-programmatic constraints types.

Type	Description
mandatory	Defines the field is mandatory
fixedLengthStatic	Defines the exact number of characters required for this field.
fixedLengthDynamic	Points out a field which provides the exact number of characters required for this one.
minLengthStatic	Defines the minimum number of characters allowed for this field.
minLengthDynamic	Points out a field which provides the minimum number of characters allowed for this one.
maxLengthStatic	Defines the maximum number of characters allowed for this field.
maxLengthDynamic	Points out a field which provides the maximum number of characters allowed for this field.
excludeSegment	Defines an inclusive range of not allowed values for this field.
excludeValue	Defines a list of not allowed values for this field.
fractionDigits	Defines the maximum number of decimal allowed for this field.
pattern	Defines the regular expression pattern that must be match by the field's value.
totalDigits	Defines the maximum number of digits allowed for this integer or decimal field.
boundaryMinInclusiveStatic	Defines the minimum value allowed (including the minimum value) for this field.
boundaryMinExclusiveStatic	Defines the minimum value allowed (excluding the minimum value) for this field.
boundaryMinInclusiveDynamic	Defines a field that provide the minimum value allowed (including the minimum value) for this field.
boundaryMinExclusiveDynamic	Defines a field that provide the minimum value allowed (excluding the minimum value) for this field.
boundaryMaxInclusiveStatic	Defines the maximum value allowed (including maximum value) for this field.
boundaryMaxExclusiveStatic	Defines the maximum value allowed (excluding the maximum value) for this field.
boundaryMaxInclusiveDynamic	Defines a field that provide the maximum value allowed (including the maximum value) for this field.

Type	Description
boundaryMaxExclusiveDynamic	Defines a field that provide the maximum value allowed (excluding the maximum value) for this field.

Voir aussi

[XML Schema supported facets](#) [p 571]

[Extended facets](#) [p 575]

107.7 Content

This section can be deactivated on demand with the [includeContent](#) [p 772] parameter (activated by default). It provides the content of the record values, dataset, or field of one of the content fields for an authorized user. It also has additional information, including labels, technical information, URLs...

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

Node value

JSON property	JSON format	Description	Required
content	Content of simple type [p 837] Content of group and list [p 838]	Contains the node value. Available for all nodes except for association and selection. However, their content can be retrieved by invoking the URL provided in their details property.	No
details	String corresponding to an absolute REST resource URL.	Returns the node details when invoked. The response type depends on the meta type. - foreignKey: target record (available on table, record and field operations). - resource: target resource [p 576] (available on dataset node, table, record and field operations). - association: target table containing the associated records (available on table and record operations). - selection: target table containing the associated records (available on table and record operations). - group: target dataset group node (available on dataset tree operation). - table: target table (available on dataset tree operation). Example: http://.../BReference/dataset/root/table/pk/associationField	No
historyDetails	String corresponding to an absolute REST resource URL.	Returns the node history details when invoked. Voir aussi includeHistory [p 772]	No
label	String	Contains the foreign key or enumeration label in the current locale. The default label is returned if the current locale is not supported.	No
inheritanceMode	String	Contains the node's inheritance state, considering only dataset inheritance. <code>inheritedFieldMode</code> and <code>inheritanceMode</code> properties cannot be both defined on the same node. Voir aussi inheritedFieldMode [p 836] Record lookup mechanism [p 290]	No
inheritedFieldMode	String	Contains the node's field inheritance state, considering dataset and field inheritance. When both inheritances are used, field inheritance has priority over the dataset one. <code>inheritedFieldMode</code> and <code>inheritanceMode</code> properties cannot be both defined on the same node. Voir aussi	No

JSON property	JSON format	Description	Required
		inheritanceMode [p 836] Value lookup mechanism [p 291]	
selector	String corresponding to an absolute URL.	Contains the appropriate URL for selector [p 837] operation. Example: http://.../BReference/dataset/root/table/pk/enumField?selector=true	No
validation	Array	Contains the validation report that concerns the current node context. Voir aussi Validation report [p 832]	No

Content of simple type

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

Voir aussi [Simple types formats](#) [p 852]

Content of group and list

XML Schema	JSON format	Examples	Meta type
Group <code>xs:complexType</code>	object Contains a property per sub-node.	Example for a simple-occurrence group. <pre>{ "road": { "content": "11 rue scribe" }, "zipcode": { "content": "75009" }, "country": { "content": "France" } }</pre>	group
List <code>maxOccurs > 1</code>	Array Contains an array of all field occurrences represented by a JSON Object. Each object is represented as a node value [p 836].	Example for a multi-occurrence field of <code>xs:int</code> type. <pre>[{ "content": 0 }, { "content": 1 }, { "content": 2 }, { "content": 3 }]</pre> Example for a multi-occurrence group. <pre>[{ "content": { "road": { "content": "11 rue scribe" }, "zipcode": { "content": "75009" }, "country": { "content": "France" } } }, { "content": { "road": { "content": "711 Atlantic Ave" }, "zipcode": { "content": "MA 02111" }, "country": { "content": "United States" } } }]</pre>	Meta of simple type [p 852], or group

Selector

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

- rows corresponding to an Array of JSON object where each one contains two entries: the returned content that can be persisted and the corresponding label. The list of established possible items depends on the current context.
- (Optional) pagination containing [pagination](#) [p 850] data (activated by default).

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

Voir aussi [includeSelector](#) [p 773]

Technical data

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

JSON property	JSON format	Description	Required
creationDate	String	Creation date	Yes
creationUser	String	Creation user's identifier.	Yes
lastUpdateDate	String	Last update date.	Yes
lastUpdateUser	String	Last update user's identifier.	Yes

```
{
  ...
  "creationDate": "2015-12-24T19:00:53.158",
  "creationUser": "admin",
  "lastUpdateDate": "2015-12-25T00:00:00.001",
  "lastUpdateUser": "admin",
  ...
}
```


CHAPITRE 108

Compact

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Global structure](#)
3. [Content](#)

108.1 Introduction

The JSON compact format purpose is to retrieve the master data using a lightweight structure. It follows the key-value design to display data in the simplest way and without any technical information. To activate the compact format, the compact suffixed REST category, like data-compact or form-data-compact, must be used in URL.

108.2 Global structure

JSON Request body

The request body is represented by a JSON object whose content varies according to the operation and the category.

Data category

- **Dataset node**

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

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

To ensure a JSON symmetric structure between the HTTP request and the HTTP response, enumeration node request format is like the following:

```
{
  "enumerationNode": {
    "key": "a key value"
  }
}
```

Voir aussi [Update operation](#) [p 785]

- **Record**

Specifies the target record content by setting the value of each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is used for table record insert, record update or selector's select operation when local enumeration dependency field values are declared.

```
{
  "gender": {
    "key": "M"
  },
  "lastName": "Chopin",
  "lastName-en": "Chopin",
  "firstName": "Fryderyk",
  "firstName-en": "Frédéric",
  "birthDate": "1810-03-01",
  "deathDate": "1849-10-17",
  "jobs": [
    {
      "key": "CM"
    },
    {
      "key": "PI"
    }
  ],
  "infos": [
    "https://en.wikipedia.org/wiki/Chopin"
  ]
}
```

Voir aussi

[Insert operation](#) [p 782]

[Update operation](#) [p 785]

- **Record fields**

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

```
{
  "jobs": [
    {
      "key": "CM"
    },
    {
      "key": "PI"
    }
  ]
}
```

Voir aussi [Update operation](#) [p 785]

- **Record table**

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

```
[
  {
    "gender": {
      "key": "M"
    },
    "lastName": "Saint-Saëns",
    "firstName": "Camille",
    "birthDate": "1835-10-09",
  }
]
```

```

    ...
  },
  {
    "gender": {
      "key": "M"
    },
    "lastName": "Debussy",
    "firstName": "Claude",
    "birthDate": "1862-10-22",
    ...
  }
]

```

Voir aussi

[Insert operation](#) [p 782]

[Update operation](#) [p 785]

- **Record table to delete**

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

```

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

```

Voir aussi [Delete operation](#) [p 787]

- **Field**

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

The request has the same structure as defined in [node value](#) [p 845].

Voir aussi [Update operation](#) [p 785]

- **Open or close user interface**

Specifies whether the user interface is open or close and the unavailability message.

```

{
  "toolStatus": {
    "key": true,      // or false
    "label": "Open"
  },
  "toolStatusCloseMessage": "Access is temporarily forbidden for maintenance."
}

```

Voir aussi [User interface operations](#) [p 765]

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

- Association node,
- Selection node,
- Value function,

- JavaBean field that does not have a setter,
- Unwritable permission on node for the authenticated user.

JSON Response body

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

Data category

- **Dataset node**

Contains the list of terminal nodes under the specified node.

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

Voir aussi [Select operation](#) [p 769]

- **Table**

JSON object containing the following properties:

- rows corresponding to JSON Array of selected records. Each record is represented by a JSON object. If no record is selected, then the JSON Array is empty.
- (Optional) pagination containing [pagination](#) [p 850] data.

```
{
  "rows": [
    {
      "id": 1,
      "firstName": "Claude",
      "lastName": "Levi-Strauss"
    },
    {
      "id": 2,
      "firstName": "Sigmoud",
      "lastName": "Freud"
    },
    {
      "id": 3,
      "firstName": "Alfred",
      "lastName": "Dreyfus"
    }
  ],
  "pagination": {
    "firstPage": null,
    "previousPage": null,
    "nextPage": "http://.../root/individu?pageRecordFilter=./id=9&pageSize=9&pageAction=next",
    "lastPage": "http://.../root/individu?pageSize=9&pageAction=last"
  }
}
```

Voir aussi [Select operation](#) [p 769]

- **Record**

JSON object containing the record content.

```
{
  "pk": "pk1",
  "name": "Name1",
  "name-fr": "Name1",
  "parent": null
}
```

Voir aussi[Select operation](#) [p 769][Prepare operations](#) [p 779]

- **Fields**

The compact json format does not support the association and selection nodes.

For other kinds of nodes, they contain the current [node value](#) [p 845].

Voir aussi [Select operation](#) [p 769]

- **Retrieve the user interface state**

Contains the user interface status and the unavailability message.

```
{
  "toolStatus": {
    "key": true,      // or false
    "label": "Open"
  },
  "toolStatusCloseMessage": "Access is temporarily forbidden for maintenance."
}
```

Voir aussi [User interface operations](#) [p 765]

Note

Nodes, records and fields property and value may be hidden depending on their resolved permissions (see [permissions](#) [p 293]) .

108.3 Content

This section is always included and contains master data without any additional fields.

Node value

The node value contains only the data or the label and the details link in case of enumeration. It is available for all nodes except association and selection.

Content of simple types

Corresponds to a key-value JSON entry which describes the node content.

Voir aussi [Simple types formats](#) [p 852]

Content of group, list and enumeration

XML Schema	JSON format	Examples
Group <code>xs:complexType</code>	Object Contains a property per sub-node.	Example for a simple-occurrence group. <pre>{ "road": "11 rue scribe", "zipcode": "75009", "country": "France" }</pre>
List <code>maxOccurs > 1</code>	Array Contains an array of all field occurrences represented by a JSON Object or a simple type. Each JSON Object is composed of node values [p 845].	Example for a multi-occurrence field of <code>xs:int</code> type. <pre>[0, 1, 2, 3, 4]</pre> Example for a multi-occurrence group. <pre>[{ "road": "11 rue scribe", "zipcode": "75009", "country": "France" }, { "road": "711 Atlantic Ave", "zipcode": "MA 02111", "country": "United States" }]</pre>
Enumeration <code>xs:string</code>	Object Contains key, link and label properties.	Example for a foreign key. <pre>{ "key": "1", "details": "http://.../Bdataspace/dataset/root/nationality/FRA", "label": "Française" }</pre>

CHAPITRE 109

Common

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Global structure](#)
3. [Form validation report](#)
4. [Content](#)
5. [Update modes](#)
6. [Known limitations](#)

109.1 Introduction

The common specifications are used in both [compact](#) [p 841] and [Extended](#) [p 815] formats.

109.2 Global structure

JSON Response body

Form data category

The response body contains a JSON object per handled resources. That is, the JSON root object for a single handled resource is a JSON object as well as for multiple handled resources. The several properties are directly placed into the JSON object.

- If the insert/update of a single record request or update field in table/dataset request was successful, the code JSON property may be absent and if not, it corresponds to the HTTP response code associated to the handled resource. The validation report is always present. Some properties are optional and added when requested using request parameters. See [Form data operations](#) [p 796] for more information.

```
{
  "validation": [
    {
      "level": "error",
      "message": "Field 'Value' is mandatory.",
      "blocksCommit": "never",
      "pathInRecord": "/value"
    },
    {
      "level": "error",
      "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
      "blocksCommit": "onUserSubmit-checkModifiedValues",
    }
  ]
}
```

```

    "pathInRecord": "/code"
  },
  {
    "level": "warning",
    "message": "Value 'AD150' is prohibited.",
    "blocksCommit": "never",
    "pathInRecord": "/code"
  }
]
}

```

- If the insert multiple record request was successful, the code JSON property may be absent and if not, it corresponds to the HTTP response code associated to the handled resource. The validation report is always present. Some properties are optional and added when requested using request parameters.

```

{
  "rows": [
    {
      "label": "My Lable1",
      "details": "http://.../root/table/1",
      "foreignKey": "1",
      "validation": [
        {
          "level": "error",
          "message": "Field 'Value' is mandatory.",
          "blocksCommit": "never",
          "pathInRecord": "/value"
        },
        {
          "level": "error",
          "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
          "blocksCommit": "onUserSubmit-checkModifiedValues",
          "pathInRecord": "/code"
        },
        {
          "level": "warning",
          "message": "Value 'AD150' is prohibited.",
          "blocksCommit": "never",
          "pathInRecord": "/code"
        }
      ]
    },
    {
      "label": "My Lable2",
      "details": "http://.../root/table/2",
      "foreignKey": "2",
      "validation": [
        {
          "level": "error",
          "message": "Field 'Value' is mandatory.",
          "blocksCommit": "never",
          "pathInRecord": "/value"
        },
        {
          "level": "error",
          "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
          "blocksCommit": "onUserSubmit-checkModifiedValues",
          "pathInRecord": "/code"
        },
        {
          "level": "warning",
          "message": "Value 'AD150' is prohibited.",
          "blocksCommit": "never",
          "pathInRecord": "/code"
        }
      ]
    }
  ]
}

```

- If the insert/update of a single record request or update field in table/dataset failed, the response body corresponds to a JSON Exception handling response.

```

{
  "code": 422,
  "errors": [
    {
      "level": "error",
      "userCode": "Validation",
      "message": "Field 'Category' is mandatory.",
    }
  ]
}

```



```

    "blocksCommit": "never",
    "pathInRecord": "/category"
  },
  {
    "level": "error",
    "userCode": "Validation",
    "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
    "blocksCommit": "onUserSubmit-checkModifiedValues",
    "pathInRecord": "/code"
  }
]
}

```

- If the insert multiple record request failed, the response body corresponds to a JSON Exception handling response.

```

{
  "code": 422,
  "errors": [
    {
      "level": "error",
      "rowIndex": 0,
      "userCode": "Validation",
      "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
      "blocksCommit": "onUserSubmit-checkModifiedValues",
      "pathInRecord": "/code"
    },
    {
      "level": "error",
      "rowIndex": 0,
      "userCode": "Validation",
      "message": "Field 'Value Less Than' is mandatory.",
      "blocksCommit": "never",
      "pathInRecord": "/less_than_value"
    },
    {
      "level": "error",
      "rowIndex": 1,
      "userCode": "Validation",
      "message": "Values between 'AD100' and 'AD200' (included) are prohibited.",
      "blocksCommit": "onUserSubmit-checkModifiedValues",
      "pathInRecord": "/code"
    },
    {
      "level": "error",
      "rowIndex": 1,
      "userCode": "Validation",
      "message": "Field 'Value Less Than' is mandatory.",
      "blocksCommit": "never",
      "pathInRecord": "/less_than_value"
    }
  ]
}

```

Voir aussi

[Form validation report](#) [p 849]

[Exception handling](#) [p 752]

109.3 Form validation report

This report can be retrieve by using the [Form data operations](#) [p 796]. It provides every validation constraint on the requested resource.

Those constraints have the following properties:

JSON property	JSON format	Description	Required
message	String	Description of the constraint.	Yes
blocksCommit	String	Control policy of the constraint. The possible values are: onInsertUpdateOrDelete, onUserSubmit-checkModifiedValues, never. See Blocking and non-blocking constraints [p 581] for more information.	Yes
level	String	Severity level of the constraint. The possible values are: info, warning, error or fatal.	Yes
pathInDataset	String	Relative field path starting from the schema node.	No(**)
pathInRecord	String	Relative field path starting from the table node.	No (*)

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

(**) Only available for dataset operations.

Voir aussi [Form data category](#) [p 847]

109.4 Content

Pagination

This feature allows to return a limited and parameterizable number of data. Pagination can be applied to data of the following types: records, association values, selection node values, selectors and dataspace. A context named pagination is always returned. This context allows browsing data similarly to the UI.

The pagination is always enabled.

Voir aussi

[Select operation](#) [p 769]

[Beta feature: Select dataspace or snapshots](#) [p 801]

Detailed information related to this context can be found hereafter:

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

Note

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

Note

(**) Not present on selector.

Content of simple type

XML Schema	JSON format	Examples	Meta type	OpenAPI
xs:string xs:Name osd:text	String (Unicode characters, cf. RFC4627)	"A text" "The escape of \"special character\" is preceded by a backslash." null	string name text	type: string format: n/a
osd:html	String (Unicode characters, cf. RFC4627)	" An HTML tag can thus be written without trouble "	html	type: string format: html
osd:email	String (Unicode characters, cf. RFC4627)	"employee@mycompany.com"	email	type: string format: email
osd:locale	String (Language tag, cf. RFC1766)	"en-US"	locale	type: string format: locale
xs:string (Foreign key)	String contains the value of the formatted foreign key.	"0" "true 99"	foreignKey	type: string format: n/a
xs:boolean	Boolean	true false null	boolean	type: boolean format: n/a
xs:decimal	Number or null	-10.5 20.001 15 -1e-13	decimal	type: number format: double
xs:date	String with format: "yyyy-MM-dd"	"2015-04-13"	date	type: string format: date
xs:time	String with format: "HH:mm:ss" "HH:mm:ss.SSS"	"11:55:00" "11:55:00.000"	time	type: string format: date-time
xs:dateTime	String with format: "yyyy-MM-ddTHH:mm:ss" "yyyy-MM-ddTHH:mm:ss.SSS"	"2015-04-13T11:55:00" "2015-04-13T11:55:00.000"	dateTime	type: string format: date-time
xs:anyURI	String (Uniform Resource Identifier, cf. RFC3986)	"https://fr.wikipedia.org/wiki/René_Descartes"	anyURI	type: string format: uri

XML Schema	JSON format	Examples	Meta type	OpenAPI
xs:int xs:integer	Number or null	1596	int	type: integer format: int32
osd:resource	String contains the resource formatted value.	"ebx-tutorial:ext-images:frontpages/Bach.jpg"	resource	type: string format: n/a
osd:color	String with format: "#[A-Fa-f0-9]{6}" contains the formatted value for the color.	"#F6E0E0"	color	type: string format: n/a
osd:datasetName	String with format: "[a-zA-Z_-][a-zA-Z0-9_-]*" and 64 characters max. contains the formatted value of the dataset name.	"ebx-tutorial"	dataset	type: string format: n/a
osd:dataspaceKey	String with format: "[BV][a-zA-Z0-9_:\-\\]+ and 33 characters max. contains the formatted key value of the dataspace.	"Bebx-tutorial"	dataspace	type: string format: n/a

Insert operation report

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

- `rows` contains a JSON Array, where each element corresponds to the result of a request element.
- `code` contains an int of JSON Number type, and allows to know whether the record has been inserted or updated. This property is included if, and only if, the `updateOrInsert` parameter is set to `true`.
- `foreignKey` contains a string of JSON String type, corresponding to the content to be used as a foreign key for this record. This property is included if, and only if, the parameter `includeForeignKey` is set to `true`.
- `label` contains a string of JSON String type, and allows to retrieve the record label. This property is included if, and only if, the parameter `includeLabel` is set to `yes`.
- `details` containing a string of JSON String type, corresponding to the resource URL. This property is included if, and only if, the parameter `includeDetails` is set to `true`.

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

```
}
  ]
}
```

Voir aussi [Insert operation](#) [p 782]

Delete operation report

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

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

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

Voir aussi [Delete operation](#) [p 787]

109.5 Update modes

The byDelta mode allows to ignore data model elements that are missing from the JSON source document. This mode is enabled (by default) through RESTful operations. The following table summarizes the behavior of insert and update operations when elements are not included in the source document.

See the RESTful data services operations [update](#) [p 785] and [insert](#) [p 782], as well as `ImportSpec.setByDeltaAPI` in the Java API for more information.

State in the JSON source document	Behavior
The property does not exist in the source document	If the byDelta mode is activated (default): - For the update operation, the field value remains unchanged. - For the insert operation, the behavior is the same as when the byDelta mode is disabled. If the byDelta mode is disabled through the RESTful operation parameter: The target field is set to one of the following values: - If the element defines a default value, the target field is set to that default value. - If the element is of a type other than a string or list, the target field value is set to <code>null</code>. - If the element is an aggregated list, the target field value is set to an empty list value. - If the element is a string that differentiates <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 differentiate the two, an empty string. - If the element (simple or complex) is hidden in the data services, the target value remains unchanged. Voir aussi Hiding a field in Data Services [p 600] Note The user performing the import must have the required permissions to create or change the target field value. Otherwise, the operation will be aborted.
The element is present and its value is <code>null</code> (for example, "content": null)	The target field is always set to <code>null</code> except for lists, in which case it is not supported.

109.6 Known limitations

Field values

The value of fields `xs:date`, `xs:time` and `xs:dateTime` does not contain a time zone associated with the JSON-primitive type.

Indexing and search strategy

[Filterable](#) [p 826] and [sortable](#) [p 827] values from the metadata are limited to the default search strategy.

Voir aussi [Search](#) [p 315]

CHAPITRE 110

Others

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Global structure](#)

110.1 Introduction

The [Extended](#) [p 815] and [Compact](#) [p 841] JSON formats can handle most of the use cases, however some operations and REST categories require specific formats like the followings.

110.2 Global structure

JSON Request body

The Request body is represented by a JSON object whose content varies according to the operation and the category.

Auth category

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

- **Token creation**

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

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

Specifies the `specific` attribute, to activate the user authentication against the HTTP request, for an authentication token creation attempt.

```
{
  "specific": true          // JSON Boolean
}
```

Voir aussi [Create token operation](#) [p 766]

- **Password change**

Specifies the login, password and passwordNew to use for the password change.

```
{
```

```

"login": "...",           // JSON String
"password": "...",        // JSON String
"passwordNew": "..."     // JSON String
}

```

Voir aussi [Change password operation](#) [p 768]

JSON Response body

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

Admin category

The selection operation for this category only provides the requested values under a content property.

- **System information**

Contains EBX® instance's system information. The representation of these data can be flat or hierarchical.

Flat representation:

```

{
  "content": {
    "bootInfoEBX": {
      "label": "EBX® configuration",
      "content": {
        "product.version": {
          "label": "EBX® product version",
          "content": "5.8.1 [...] Enterprise Edition"
        },
        "product.configuration.file": {
          "label": "EBX® main configuration file",
          "content": "System property [ebx.properties=../ebx.properties]"
        },
        // others keys
      }
    },
    "repositoryInfo": {
      "label": "Repository information",
      "content": {
        "repository.identity": {
          "label": "Repository identity",
          "content": "00905A5753FD"
        },
        "repository.label": {
          "label": "Repository label",
          "content": "My repository"
        },
        // others keys
      }
    },
    "bootInfoVM": {
      "label": "System information",
      "content": {
        "java.home": {
          "label": "Java installation directory",
          "content": "C:\\JTools\\jdk1.8.0\\jre"
        },
        "java.vendor": {
          "label": "Java vendor",
          "content": "Oracle Corporation"
        },
        // others keys
      }
    }
  }
}

```

Hierarchical representation:

```

{
  "content": {
    "bootInfoEBX": {
      "label": "EBX® configuration",

```

```

"content": {
  "product": {
    "content": {
      "version": {
        "label": "EBX® product version",
        "content": "5.8.1 [...] Enterprise Edition"
      },
      "configuration": {
        "content": {
          "file": {
            "label": "EBX® main configuration file",
            "content": "System property [ebx.properties=./ebx.properties]"
          }
        }
      }
    }
  },
  "vm": {
    "content": {
      "startTime": {
        "label": "VM start time",
        "content": "2017/09/11-10:04:17-0729 CEST"
      },
      "identifier": {
        "label": "VM identifier",
        "content": "1"
      }
    }
  },
  // other hierarchical keys
}
}
}

```

Voir aussi [System information operation](#) [p 765]

Auth category

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

- **Token creation**

Contains the token value and its type.

```

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

```

Voir aussi [Create token operation](#) [p 766]

Data category

Look up table views

The response body contains a content property holding a JSON Array, itself composed by JSON objects with the following properties:

JSON property	JSON format	Description	Required
details	String	Corresponds to the view access URL.	Yes
label	String	View's label.	No
viewPublication	String	Published view's name. Voir aussi Table viewPublication parameter [p 777]	Yes
viewType	String	Enumeration whose value corresponds to one of the following: - SimpleTabular: Simple tabular view. - Hierarchy: Hierarchical view.	Yes

```
{
  "content": [
    {
      "details": "http://.../data/v1/Bebx-directory/ebx-directory/directory/users?viewPublication=custom-
directory",
      "label": "My custom directory view",
      "viewPublication": "custom-directory",
      "viewType": "SimpleTabular"
    },
    {
      ...
    }
  ]
}
```

Voir aussi [Look up table views operation](#) [p 795]

Beta feature: Dataspaces selection

The returned response body contains dataspace in a rows JSON Array property, where each inner JSON object corresponds to a dataspace with the following properties:

JSON property	JSON format	Description	Required
label	String	Documentation label in the current locale.	No
description	String	Documentation description in the current locale.	No
details	String	Specifies the dataspace's REST resource URL.	Yes
information	String	Specifies the dataspace's information REST resource URL.	Yes
key	String	Specifies the dataspace or snapshot formatted key. Format: [BV][a-zA-Z0-9_:. \-\\]{1,33} and percentage encoded afterward.	Yes
isSelectAllowed	Boolean	Specifies if the dataspace can be selected, according to the user's permissions.	Yes
hasChildren	Boolean	Specifies if the dataspace has children, according to the user's permissions. Note Not applicable for snapshots.	Yes
children	String	Specifies the dataspace's children REST resource URL. If hasChildren property key value is false then the returned value is null. Note Not applicable for snapshots.	No
snapshots	String	Specifies the dataspace's snapshots REST resource URL. Note Not applicable for snapshots.	Yes

```
{
  "rows": [
    {
      "label": "Master Data - Reference",
      "description": "Reference dataspace in EBX.",
      "details": "http://.../data/v1/BReference",
      "information": "http://.../data/v1/BReference:information",
      "key": "BReference",
      "closed": false,
      "isSelectAllowed": true,
      "hasChildren": true,
      "children": "http://.../data/v1/BReference:children",
      "snapshots": "http://.../data/v1/BReference:snapshots"
    }
  ],
  {
```

```
    // An other dataspace
  }
],
"pagination": {
  "firstPage": null,
  "previousPage": null,
  "nextPage": null,
  "lastPage": null
}
}
```

Voir aussi

[Pagination](#) [p 850]

[Beta feature: Select dataspace or snapshots](#) [p 801]

Beta feature: Dataspace information

The response body contains a content JSON object property, holding the following properties:

JSON property	JSON format	Description	Required
content	Object	Corresponds to the dataspace or the snapshot information.	Yes
key	String	Specifies the dataspace or snapshot formatted key value. Format: [BV][a-zA-Z0-9_:.\\-\\]+ limited to 33 characters maximum.	Yes
documentation	Array of JSON Object	Corresponds to the localized documentation with a JSON Object by locale.	No
locale	String	Documentation locale (nested under the documentation property).	Yes
label	String	Documentation label (nested under the documentation property).	No
description	String	Documentation description (nested under the documentation property).	No
closed	Boolean	Specifies if the dataspace is closed.	Yes
locked	Boolean	Specifies if the dataspace is locked. Note Not applicable for snapshots.	Yes
parent	String	Specifies the parent dataspace or snapshot formatted key value.	No
administration	Boolean	Specifies if the dataspace or the snapshot is an administration one.	Yes
owner	String	Specifies the owner profile.	No
creator	String	Specifies the creator profile.	Yes
creationDate	Date	Specifies the creation date.	Yes

```
{
  "content": {
    "key": "BReference",
    "documentation": [
      {
        "locale": "en-US",
        "label": "Master Data - Reference",
        "description": "Reference dataspace in EBX."
      },
      {
        "locale": "fr-FR",
        "label": "Données - Référence",
        "description": "Espace de données référence de EBX."
      }
    ]
  }
}
```

```

    ],
    "closed": false,
    "locked": false,
    "parent": null,
    "administration": false,
    "relationalMode": false,
    "owner": "Badministrato",
    "creator": "Badministrato",
    "creationDate": "2019-04-28T19:49:04.838"
  }
}

```

Voir aussi [Beta feature: Select dataspace or snapshots](#) [p 801]

Beta feature: Dataspace child or snapshot creation

The response body contains a content JSON object property, holding the following properties:

JSON property	JSON format	Description	Required
key	String	Specifies the dataspace or snapshot formatted key value. Format: [BV][a-zA-Z0-9_:\-\]+ limited to 33 characters maximum. The default value is a timestamp.	No
owner	String	Specifies the owner profile. Default value is null.	No
documentation	Array of JSON object	Corresponds to the localized documentation with a JSON object by locale. Default value is null.	No
locale	String	Documentation locale (nested under the documentation property).	Yes
label	String	Documentation label (nested under the documentation property). Default value is null.	No
description	String	Documentation description (nested under the documentation property). Default value is null.	No
dataspaceKeyToCopyPermissionsFrom	String	Specifies the dataspace's formatted key value from which to copy permissions. Note Not applicable for snapshots creation.	No

```

{
  "content": {
    "key": "BMyData",
    "owner": "Beveryone",
    "documentation": [
      {
        "locale": "en-US",
        "label": "My dataspace",
        "description": "This space contains my data"
      }
    ]
  }
}

```



```
    }  
  ],  
  "dataspaceKeyToCopyPermissionsFrom": "BReference"  
}
```

Voir aussi [Beta feature: Create a child dataspace or a snapshot](#) [p 803]

SQL in EBX®

CHAPITRE 111

Introduction

Ce chapitre contient les sections suivantes :

1. [Overview](#)
2. [Mapping data model entities](#)
3. [Mapping data types](#)
4. [SQL syntax](#)
5. [Limitations and performance guidelines](#)

111.1 Overview

This documentation covers Structured Query Language (SQL) queries and expressions in EBX®. EBX® supports standard SQL queries to retrieve rows selected from one or more tables. Some EBX® SQL language features are extensions to the standard. Supported EBX® SQL syntax includes: table expressions (SELECT, FROM, WHERE, GROUP BY and HAVING clauses), DISTINCT, ORDER BY, LIMIT and OFFSET, combining queries (UNION [ALL]), and WITH (except RECURSIVE modifier).

The goal of this API is to provide to developers the ability to retrieve data from EBX® using a well-known standard.

EBX® SQL is accessible through Java APIs, especially from the class `Query`^{API}. The queries also support parameters. See `Query.setParameter`^{API}.

111.2 Mapping data model entities

The following section provides a detailed explanation about the mapping of the EBX® concepts into SQL.

Table (in data model)

EBX® tables are mapped naturally to SQL tables. In the data model, there can be more than one EBX® table with the same name. This ambiguity can occur when tables are in groups. To remove the ambiguity, use the full path of the table surrounded by double quotes (for example, "my_group/my_table" no longer conflicts with "other_group/my_table"). You can also use the entity name of the table, which is unique inside the data model. You can use the table name only if it does not collide with an entity name or another table name.

Fields

In SQL Standard, the structure of a table consists of one or more columns. Every element (including fields) whose parent is an EBX® table, is mapped to a column.

Groups

In SQL Standard, the structure of a table consists of one or more columns. Every element whose parent is an EBX® table is mapped to a column. This includes groups that are mapped to SQL columns as *SQL structure types*.

Associations

In SQL Standard, querying data among multiple tables is based on foreign keys and primary keys. These concepts in EBX® are similar to those in SQL. Therefore, joins between tables in SQL can also be done using EBX® foreign and primary keys.

111.3 Mapping data types

Handling data through SQL is highly dependent on its data type. For example, in predicates, columns can be compared only if they have the same SQL data type. The SQL data types are types according to the type in the data model.

Simple data types

Supported standard SQL data types

This table lists all of the simple types defined in the XML Schema that are supported by EBX®, along with their corresponding standard SQL types.

XML Schema type	SQL type	Java type	Notes
xs:string	VARCHAR	<code>java.lang.String</code>	
xs:boolean	BOOLEAN	<code>java.lang.Boolean</code>	Values: TRUE, FALSE, UNKNOWN
xs:decimal	DECIMAL	<code>java.math.BigDecimal</code>	
xs:dateTime	TIMESTAMP	<code>java.lang.Date</code>	
xs:time	TIME	<code>java.lang.Date</code>	The date portion of the returned Date is always set to '1970/01/01'.
xs:date	DATE	<code>java.lang.Date</code>	The time portion of the returned Date is always set to the beginning of the day (that is, '00:00:00').
xs:anyURI	VARCHAR	<code>java.net.URI</code>	
xs:Name (xs:string restriction)	VARCHAR	<code>java.lang.String</code>	
xs:int	INT	<code>java.lang.Integer</code>	

Extended simple types defined by EBX®

EBX® provides pre-defined simple data types. These types are defined by the internal schema `common-1.0.xsd`. Their definition is detailed in the section [Extended simple types defined by EBX®](#) [p 536]

XML Schema type	SQL type	Java class
<code>osd:text</code> (xs:string restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:html</code> (xs:string restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:email</code> (xs:string restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:password</code> (xs:string restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:color</code> (xs:string restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:resource</code> (xs:anyURI restriction)	internal type	internal class
<code>osd:locale</code> (xs:string restriction)	VARCHAR	<code>java.util.Locale</code>
<code>osd:dataspaceKey</code> (xs:string restriction)	VARCHAR	<code>java.lang.String</code>
<code>osd:datasetName</code> (xs:string restriction)	VARCHAR	<code>java.lang.String</code>

List (multi-valued) types

Lists are handled as SQL Arrays. Their corresponding Java class is `java.util.List`.

Complex types

Complex types are handled as [SQL Structured types](#). Their corresponding Java class is `Object[]`. This applies to foreign keys (see below) and [groupe](#) [p 27]s, because they are defined through complex types. Use the *dot operator* to access fields inside the SQL Structure types. For example, use `address.street` to access the field `street` of the field `address`, if it is a complex type. When you reference a sub-field of a complex type in a query, you must always use the table name or alias:

- `SELECT customer.address.street FROM customer`
- `SELECT c.address.street FROM customer c`

TableRef types

In EBX®, a table can have a primary key composed of multiple fields. Foreign keys are defined by a single field with the [osd:tableRef](#) [p 554] declaration. The standard SQL syntax has been extended to extract the value of any targeted primary key field. In the [Extraction of foreign keys example](#) [p 256], the following SQL expressions are valid:

- `tableA.fkb.id = 123`

- `YEAR(tableA.fkb.date) > 2018`

Note

Even if the primary key is composed of only one field, the name of the field must be specified to access the value. For example, if the primary key is composed of a single id, fkb.id must be used to access the value, as in `tableA.fkb.id = 123`

A typical example of tableRef usage would be SQL joins:

```
SELECT * FROM employee JOIN department ON employee.fkDept.id = department.id
```

Or in case the referenced table has a composite primary key:

```
SELECT * FROM tableA JOIN tableB ON tableA.fkB.id1 = tableB.id1 AND tableA.fkB.id2 = tableB.id2
```

System columns

Apart from the fields present in a table, EBX® SQL provides some extra system columns. These columns are not returned by a SQL statement, unless they are explicitly referenced. For example, "SELECT * FROM ..." does not return systems columns, but "SELECT systemColumnName FROM ..." does.

Name	Description	SQL type	Java class	Examples
\$adaptation	The Adaptation ^{API} representing the table record.	internal type	com.onwbp.adaptation.Adaptation	SELECT t."\$adaptation" FROM myTable t WHERE t.value>100
\$pk	String representation of the primary key of the record (see PrimaryKey. syntax ^{API}).	VARCHAR	java.lang.String	SELECT t.* FROM myTable1 t WHERE t."\$pk"='123' SELECT t.* FROM myTable2 t WHERE t."\$pk"='123 abc' SELECT t."\$pk" FROM myTable3 t WHERE t.value>100 SELECT t.value FROM myTable3 t ORDER BY t."\$pk"

111.4 SQL syntax

Supported standard operators and functions

An operator is a reserved word or a character used primarily in a SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in a SQL statement, and to serve as conjunctions for multiple conditions in a statement. EBX® supports most of the SQL standard operators and functions. Some functions and operators can have optional parameters: they are surrounded by square brackets in the documentation. Generally, there are five types of operators and functions:

- [Comparison operators and functions](#) [p 876]
- [Arithmetic operators and functions](#) [p 880]

- [Logical operators](#) [p 884]
- [String operators and functions](#) [p 888]
- [Date and time functions](#) [p 892]

The following table lists all of the operators' associativity and precedence, highest to lowest.

Operator	Associativity
.	left
[] (array element)	left
+ - (unary plus, minus)	right
* / %	left
+ -	left
BETWEEN, IN, LIKE, CONTAINS, and so on	-
< > = <= >= <> !=	left
IS NULL, IS FALSE, IS NOT TRUE, and so on	-
NOT	right
AND	left
OR	left

Escaping identifiers

In the following cases, the identifier must be escaped by using double quotes:

- when using the absolute path to identify a table (for example, `"/root/myTable"`).
- when the field to identify is a reserved word (for example, `"user"`, `"order"`).
- when referring to a system column with a table alias (for example, `t."$adaptation"`, `t."$pk"`).

The following example shows a query to illustrate all cases:

```
SELECT t."user", t."$pk" FROM "/root/myTable" t WHERE t."order" = 1
```

Explain plan

EBX® SQL supports `EXPLAIN PLAN FOR ...` syntax to get the plan information of a query. The result is similar to `Query.explainAPI`.

Example: `EXPLAIN PLAN FOR SELECT id FROM myTable`

111.5 Limitations and performance guidelines

- Certain internal join optimizations do not support RIGHT and FULL joins, so avoid these join types if possible.
- The maximum precision and scale for numeric or decimal values is 1000.
- Queries using GROUP BY and/or aggregate functions (MIN, MAX, and so on) are not optimized, except for COUNT, which can be optimized in some circumstances.
- Currently, MIN and MAX operators do not exploit internal indices. Instead, use the following equivalent queries, which are probably more efficient:

```
SELECT val FROM myTable ORDER BY val DESC NULLS LAST LIMIT 1 instead of SELECT MAX(val)
FROM myTable
```

```
SELECT val FROM myTable ORDER BY val LIMIT 1 instead of SELECT MIN(val) FROM myTable
```


CHAPITRE **112**

Comparison operators

The table below lists all the SQL comparison operators supported by EBX®, along with their standard SQL syntax. Some operators may have optional parameters: they are surrounded by square brackets.

Operator syntax	Description and example(s)
value1 = value2	Equals SELECT 1 = 0 : false SELECT 4 = 4 : true
value1 <> value2	Not equals SELECT 1 <> 0 : true SELECT 4 <> 4 : false
value1 > value2	Greater than SELECT 1 > 0 : true SELECT 4 > 4 : false
value1 >= value2	Greater than or equal SELECT 1 >= 0 : true SELECT 4 >= 4 : true
value1 < value2	Lower than SELECT 1 < 0 : false SELECT 4 < 4 : false
value1 <= value2	Less than or equal SELECT 1 <= 0 : false SELECT 4 <= 4 : true
value IS NULL	Whether value is null SELECT 1 IS NULL : false SELECT NULL IS NULL : true
value IS NOT NULL	Whether value is not null SELECT 1 IS NOT NULL : true

Operator syntax	Description and example(s)
	SELECT NULL IS NOT NULL : false
value1 IS DISTINCT FROM value2	Whether two values are not equal, treating null values as the same SELECT 1 IS DISTINCT FROM 1 : false SELECT 1 IS DISTINCT FROM 4 : true SELECT 1 IS DISTINCT FROM NULL : true SELECT NULL IS DISTINCT FROM NULL : false
value1 IS NOT DISTINCT FROM value2	Whether two values are equal, treating null values as the same SELECT 1 IS NOT DISTINCT FROM 1 : true SELECT 1 IS NOT DISTINCT FROM 4 : false SELECT 1 IS NOT DISTINCT FROM NULL : false SELECT NULL IS NOT DISTINCT FROM NULL : true
value1 BETWEEN value2 AND value3	Whether value1 is greater than or equal to value2 and less than or equal to value3 SELECT 4 BETWEEN 3 AND 10 : true SELECT 1 BETWEEN 3 AND 10 : false
value1 NOT BETWEEN value2 AND value3	Whether value1 is greater than or equal to value2 and less than or equal to value3 SELECT 4 NOT BETWEEN 3 AND 10 : false SELECT 1 NOT BETWEEN 3 AND 10 : true
string1 LIKE string2	Whether string1 matches pattern string2. The wildcard '%' represent zero, one or multiple characters. To find if string1 starts with a sequence, use pattern 'sequence%'. The matching is case sensitive. To do a case insensitive matching, use UPPER (or LOWER) on string1 and pattern. SELECT firstname FROM employee WHERE name LIKE 'S%' : John, Jennifer SELECT firstname FROM employee WHERE UPPER(name) LIKE 'SM%' : John SELECT firstname FROM employee WHERE name LIKE '_m%' : John
string1 NOT LIKE string2	Whether string1 does not matches pattern string2. The wildcard '%' represent zero, one or multiple characters. To find if string1 does not start with a sequence, use pattern 'sequence%'. The matching is case sensitive. To do a case insensitive matching, use UPPER (por LOWER) on string1 and pattern. SELECT firstname FROM employee WHERE name NOT LIKE 'S%' : Maria SELECT firstname FROM employee WHERE UPPER(name) NOT LIKE 'SM%' : Maria, Jennifer SELECT firstname FROM employee WHERE name NOT LIKE '_m%' : Maria, Jennifer
value IN (value [, value]*)	Whether value is equal to a value in a list

Operator syntax	Description and example(s)
	<code>SELECT firstname FROM employee WHERE name IN ('Smith', 'Hamilton') : John, Maria</code>
value NOT IN (value [, value]*)	Whether value is not equal to every value in a list <code>SELECT firstname FROM employee WHERE name NOT IN ('Smith', 'Hamilton') : Jennifer</code>
value IN (sub-query)	Whether value is equal to a row returned by sub-query <code>SELECT e.firstname FROM employee e WHERE e.department.id IN (SELECT d.id FROM department d WHERE d.name='IT') : John</code>

CHAPITRE **113**

Arithmetic operators and functions

The table below lists all the SQL arithmetic operators and functions supported by EBX®, along with their standard SQL syntax. Some functions may have optional parameters: they are surrounded by square brackets.

Operator syntax	Description and example(s)
+ numeric	Returns numeric SELECT +4 : 4
- numeric	Returns numeric SELECT -4 : -4
numeric1 + numeric2	Returns numeric1 plus numeric2 SELECT 18 + 4 : 22
numeric1 - numeric2	Returns numeric1 minus numeric2 SELECT 18 - 4 : 14
numeric1 * numeric2	Returns numeric1 multiplied by numeric2 SELECT 18 * 4 : 72
numeric1 / numeric2	Returns numeric1 divided by numeric2 SELECT 18 / 4 : 4
ABS(numeric)	Returns the absolute value of numeric SELECT ABS(-243.5) : 243.5
ACOS(numeric)	Returns the arc cosine of numeric SELECT ACOS(-1) : 3.141592653589793
ASIN(numeric)	Returns the arc sine of numeric SELECT ASIN(0.25) : 0.25
ATAN(numeric)	Returns the arc tangent of numeric SELECT ATAN(2.5) : 1.107148717774778

Operator syntax	Description and example(s)
ATAN2(numeric, numeric)	Returns the arc tangent of the numeric coordinates SELECT ATAN2(0.50, 1) : 0.4636476090008061
CEIL(numeric)	Rounds numeric up, and returns the smallest number that is greater than or equal to numeric SELECT CEIL(25.75) : 26
CEILING(numeric)	Rounds numeric up, and returns the smallest number that is greater than or equal to numeric SELECT CEILING(25.75) : 26
COS(numeric)	Returns the cosine of numeric SELECT COS(2) : -0.4161468365471424
COT(numeric)	Returns the cotangent of numeric SELECT COT(6) : -3.436353004180128
DEGREES(numeric)	Converts numeric from radians to degrees SELECT DEGREES(1.5) : 85.94366926962348
EXP(numeric)	Returns e raised to the power of numeric SELECT EXP(1) : 2.718281828459045
FLOOR(numeric)	Rounds numeric down, and returns the largest number that is less than or equal to numeric SELECT FLOOR(25.75) : 25
LN(numeric)	Returns the natural logarithm (base e) of numeric SELECT LN(2) : 0.6931471805599453
LOG10(numeric)	Returns the base-10 logarithm of numeric SELECT LOG10(2) : 0.3010299956639812
MOD(numeric1, numeric2)	Returns the remainder (modulus) of numeric1 divided by numeric2. The result is negative only if numeric1 is negative SELECT MOD(18, 4) : 2
POWER(numeric1, numeric)	Returns numeric1 raised to the power of numeric2 SELECT POWER(4, 2) : 16.0
RADIANS(numeric)	Converts numeric from degrees to radians SELECT RADIANS(180) : 3.141592653589793
RAND([seed])	Returns a random double using numeric as the seed value if specified SELECT RAND(6) : 0.9891840064573959

Operator syntax	Description and example(s)
RAND_INTEGER([seed,] numeric)	Generates a random integer between 0 and numeric - 1 inclusive, initializing the random number generator with seed if specified SELECT RAND_INTEGER(100, 5) : 0 SELECT RAND_INTEGER(6, 100) : 11
ROUND(numeric1, numeric2)	Rounds numeric1 to numeric2 places right to the decimal point SELECT ROUND(135.375, 2) : 135.38
SIGN(numeric)	Returns the signum of numeric SELECT SIGN(255.5) : 1
SIN(numeric)	Returns the sine of numeric SELECT SIN(2) : 0.9092974268256817
SQRT(numeric)	Returns the square root of numeric SELECT SQRT(64) : 8.0
TAN(numeric)	Returns the tangent of numeric SELECT TAN(1.75) : -5.52037992250933
TRUNCATE(numeric1[, numeric2])	Truncates numeric1 to 0 (if no numeric2 specified) places right to the decimal point SELECT TRUNCATE(135.375) : 135 SELECT TRUNCATE(135.375, 2) : 135.37

CHAPITRE **114**

Logical operators

The table below lists all the SQL logical operators supported by EBX®, along with their standard SQL syntax.

Operator syntax	Description and example(s)
boolean1 OR boolean2	Whether boolean1 is TRUE or boolean2 is TRUE SELECT TRUE OR TRUE : true SELECT TRUE OR FALSE : true SELECT FALSE OR TRUE : true SELECT FALSE OR FALSE : false
boolean1 AND boolean2	Whether boolean1 is TRUE and boolean2 is TRUE SELECT TRUE AND TRUE : true SELECT TRUE AND FALSE : false SELECT FALSE AND TRUE : false SELECT FALSE AND FALSE : false
NOT boolean	Whether boolean is not TRUE; returns UNKNOWN if boolean is UNKNOWN SELECT NOT TRUE : false SELECT NOT FALSE : true SELECT NOT UNKNOWN : null
boolean IS FALSE	Whether boolean is FALSE; returns FALSE if boolean is UNKNOWN SELECT TRUE IS FALSE : false SELECT FALSE IS FALSE : true
boolean IS NOT FALSE	Whether boolean is not FALSE; returns TRUE if boolean is UNKNOWN SELECT TRUE IS NOT FALSE : true SELECT FALSE IS NOT FALSE : false
boolean IS TRUE	Whether boolean is TRUE; returns TRUE if boolean is UNKNOWN SELECT TRUE IS TRUE : true

Operator syntax	Description and example(s)
	SELECT FALSE IS TRUE : false
boolean IS NOT TRUE	Whether boolean is not TRUE; returns FALSE if boolean is UNKNOWN SELECT TRUE IS NOT TRUE : false SELECT FALSE IS NOT TRUE : true
boolean IS UNKNOWN	Whether boolean is UNKNOWN SELECT TRUE IS UNKNOWN : false SELECT FALSE IS UNKNOWN : false
boolean IS NOT UNKNOWN	Whether boolean is not UNKNOWN SELECT TRUE IS NOT UNKNOWN : true SELECT FALSE IS NOT UNKNOWN : true

CHAPITRE **115**

String operators and functions

The table below lists all the SQL string operators and functions supported by EBX®, along with their standard SQL syntax. Some functions may have optional parameters: they are surrounded by square brackets.

Operator syntax	Description and example(s)
string string	Concatenates two character strings SELECT 'Hello ' 'world !' : Hello world !
CHAR_LENGTH(string)	Returns the number of characters in a character string SELECT CHAR_LENGTH('Alfreds Futterkiste') : 19
CHARACTER_LENGTH(string)	As CHAR_LENGTH(string) SELECT CHARACTER_LENGTH('Alfreds Futterkiste') : 19
UPPER(string)	Returns a character string converted to upper case SELECT UPPER('SQL Tutorial is FUN!') : SQL TUTORIAL IS FUN!
POSITION(string1 IN string2 [FROM integer])	Returns the position of the first occurrence of string1 in string2 starting at a given point if specified SELECT POSITION('is fun' IN 'Tutorial is FUN!') : 0 SELECT POSITION('fun' IN 'Tutorial is FUN, very FUN!' FROM 17) : 0
TRIM({ BOTH LEADING TRAILING } character FROM string)	Removes the longest string containing only the character from the start/end/both ends of string SELECT TRIM(' ' FROM ' #SQL Tutorial! ') : #SQL Tutorial! SELECT TRIM(LEADING ' ' FROM ' #SQL Tutorial! ') : #SQL Tutorial! SELECT TRIM(TRAILING ' ' FROM ' #SQL Tutorial! ') : #SQL Tutorial! SELECT TRIM(BOTH ' ' FROM ' #SQL Tutorial! ') : #SQL Tutorial!

Operator syntax	Description and example(s)
OVERLAY(string1 PLACING string2 FROM integer [FOR integer2])	Replaces a substring of string1 with string2 <pre>SELECT OVERLAY('Tutorial is very FUN!' PLACING 'VERY' FROM 13) : Tutorial is VERY FUN!</pre> <pre>SELECT OVERLAY('Tutorial is very FUN!' PLACING 'VERY' FROM 13 FOR 4) : Tutorial is VERY FUN!</pre>
SUBSTRING(string FROM integer [FOR integer])	Returns a substring of a character string starting at a given point <pre>SELECT SUBSTRING('Tutorial is very FUN!' FROM 13) : very FUN!</pre> <pre>SELECT SUBSTRING('Tutorial is very FUN!' FROM 13 FOR 4) : very</pre>

CHAPITRE **116**

Date and time functions

The table below lists all the SQL date and time functions supported by EBX®, along with their standard SQL syntax. Some functions may have optional parameters: they are surrounded by square brackets.

Operator syntax	Description and example(s)
CURRENT_TIME	Returns the current time in the session time zone, in a value of datatype <code>TIMESTAMP WITH TIME ZONE</code>
CURRENT_DATE	Returns the current date in the session time zone, in a value of datatype <code>DATE</code>
CURRENT_TIMESTAMP	Returns the current date and time in the session time zone, in a value of datatype <code>TIMESTAMP WITH TIME ZONE</code>
YEAR(date)	Extracts and returns the value of the year from a datetime value expression. Returns an integer. <pre>SELECT YEAR(TIMESTAMP '1971-07-20 09:34:21') : 1971</pre> <pre>SELECT YEAR(DATE '1968-07-20') : 1968</pre> <pre>SELECT YEAR(hiringDate) FROM employee WHERE name='Smith' : 2015</pre> <pre>SELECT YEAR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 2017</pre>
QUARTER(date)	Extracts and returns the value of the quarter from a datetime value expression. Returns an integer between 1 and 4. <pre>SELECT QUARTER(TIMESTAMP '1971-07-20 09:34:21') : 3</pre> <pre>SELECT QUARTER(hiringDate) FROM employee WHERE name='Smith' : 4</pre> <pre>SELECT QUARTER(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 2</pre>
QUARTER(date)	Extracts and returns the value of the month from a datetime value expression. Returns an integer between 1 and 12. <pre>SELECT MONTH(TIMESTAMP '1971-07-20 09:34:21') : 7</pre> <pre>SELECT MONTH(hiringDate) FROM employee WHERE name='Smith' : 10</pre>

Operator syntax	Description and example(s)
	SELECT MONTH(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 6
WEEK(date)	Extracts and returns the value of the week from a datetime value expression. Returns an integer between 1 and 53. SELECT WEEK(TIMESTAMP '1971-07-20 09:34:21') : 29 SELECT WEEK(hiringDate) FROM employee WHERE name='Smith' : 42 SELECT WEEK(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 26
DAYOFYEAR(date)	Extracts and returns the value of the day of year from a datetime value expression. Returns an integer between 1 and 366. SELECT DAYOFYEAR(TIMESTAMP '1971-07-20 09:34:21') : 201 SELECT DAYOFYEAR(hiringDate) FROM employee WHERE name='Smith' : 287 SELECT DAYOFYEAR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 179
DAYOFMONTH(date)	Extracts and returns the value of the day of month from a datetime value expression. Returns an integer between 1 and 31. SELECT DAYOFMONTH(TIMESTAMP '1971-07-20 09:34:21') : 20 SELECT DAYOFMONTH(hiringDate) FROM employee WHERE name='Smith' : 14 SELECT DAYOFMONTH(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 28
DAYOFWEEK(date)	Extracts and returns the value of the day of week from a datetime value expression. Returns an integer between 1 and 7. SELECT DAYOFWEEK(TIMESTAMP '1971-07-20 09:34:21') : 3 SELECT DAYOFWEEK(hiringDate) FROM employee WHERE name='Smith' : 4 SELECT DAYOFWEEK(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 4
HOURL(date)	Extracts and returns the value of the hours from a datetime value expression. Returns an integer between 0 and 23. SELECT HOUR(TIMESTAMP '1971-07-20 09:34:21') : 9 SELECT HOUR(TIME '10:34:21') : 10 SELECT HOUR(workStart) FROM employee WHERE name='Smith' : 8 SELECT HOUR(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 16
MINUTE(date)	Extracts and returns the value of the minutes from a datetime value expression. Returns an integer between 0 and 59. SELECT MINUTE(TIMESTAMP '1971-07-20 09:34:21') : 34

Operator syntax	Description and example(s)
	<pre> SELECT MINUTE(TIME '10:35:21') : 35 SELECT MINUTE(workStart) FROM employee WHERE name='Smith' : 0 SELECT MINUTE(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 6 </pre>
SECOND(date)	Extracts and returns the value of the seconds from a datetime value expression. Returns an integer between 0 and 59. <pre> SELECT HOUR(TIMESTAMP '1969-07-20 09:34:21') : 9 SELECT SECOND(TIME '10:34:22') : 22 SELECT SECOND(workStart) FROM employee WHERE name='Smith' : 0 SELECT SECOND(lastProfileUpdateTime) FROM employee WHERE name='Smith' : 12 </pre>

CHAPITRE **117****EBX® SQL functions**

The table below lists all the EBX® built-in SQL functions, along with their syntax. Some functions may have optional parameters: they are surrounded by square brackets.

Operator syntax	Description and example(s)
<code>FK_AS_STRING('referencedDatasetName', 'referencedTableName', tableRefValue)</code>	Returns the string representation of a tableRef value. See QueryBuilder to know more about dataset registered names <code>SELECT e.name FROM employee e WHERE FK_AS_STRING('_public', '/root/department', e.department) = '1' : Smith</code> <code>SELECT FK_AS_STRING('_public', '/root/department', e.department) FROM employee e WHERE e.name = 'Smith' : 1</code>

CHAPITRE 118

REST Toolkit

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Application definitions](#)
3. [Service and operation definitions](#)
4. [Serialization of a table record](#)
5. [Authentication and lookup mechanism](#)
6. [REST authentication and permissions](#)
7. [URI builders](#)
8. [Exception handling](#)
9. [Monitoring](#)
10. [Packaging and registration](#)

118.1 Introduction

TIBCO EBX® offers the possibility to develop custom REST services using the REST Toolkit. The REST Toolkit supports JAX-RS 2.1 ([JSR-370](#)) and JSON-B ([JSR-367](#)).

A REST service is implemented by a Java class and its operations are implemented by Java methods. The response can be generated by serializing POJO objects. The request input can be unserialized to POJOs. Various input and output formats, including JSON, are supported. For more details on supported formats, see [media types](#) [p 899].

Rest Toolkit supports the following:

- Injectable objects

EBX® provides injectable objects useful to authenticate the request's user, to access the EBX® repository or to build URIs without worrying about the configuration (for example [reverse-proxy](#) [p 749] or [REST forward](#) [p 676] modes);

JAX-RS injectable objects are also supported.

- Annotations

EBX® provides annotations to describe resources, grant anonymous access or add restrictions to a method.

JAX-RS and JSON-B annotations are also supported.

- logging and debugging utilities.

Voir aussi [JAX-RS: Java™ API for RESTful Web Services 2.1](#)

118.2 Application definitions

An EBX® module, that includes custom REST services, must provide at least one REST Toolkit application class. A REST Toolkit application class extends the EBX® `RESTApplicationAbstract`^{API} class. The minimum requirement is to define the base URL, using the `@ApplicationPath` annotation and the set of packages to scan for REST service classes.

Note

Only packages accessible from the web application's classloader can be scanned.

Note

It is possible to register REST resource classes or singletons, packaged inside or outside the web application archive, through the **`ApplicationConfigurator.register(java.lang.Class)`** `ApplicationConfigurator.register`^{API} or **`ApplicationConfigurator.register(java.lang.Object)`** `ApplicationConfigurator.register`^{API} methods.

Note

If no packages scope is defined, then every class reachable from the web application's classloader will be scanned.

The application path cannot be "/" and must not collide with an existing resource from the module. It is recommended to use "/rest" (the value of the `RESTApplicationAbstract.REST_DEFAULT_APPLICATION_PATH` constant).

EBX® `Documentation`^{API} annotation is optional. It is displayed to administrators in 'Technical configuration' > 'Modules and data models' or when logging and debugging.

```
import javax.ws.rs.*;

import com.orchestranetworks.rest.*;
import com.orchestranetworks.rest.annotation.*;

@ApplicationPath(RESTApplicationAbstract.REST_DEFAULT_APPLICATION_PATH)
@Documentation("My REST sample application")
public final class RESTApplication extends RESTApplicationAbstract
{
    public RESTApplication()
    {
        // Adds one or more package names which will be used to scan for components.
        super((cfg) -> cfg.addPackages(RESTApplication.class.getPackage()));
    }
}
```

118.3 Service and operation definitions

A REST Toolkit service is implemented by a Java class and its operations are implemented by its methods.

Class and methods can be annotated by `@Path` to specify the access path. The `@Path` annotation value defined at the class level will prepend the ones defined on methods. Warning, only one `@Path` annotation is allowed per class or method.

Media types accepted and produced by a resource are respectively defined by the `@Consumes` and `@Produces` JAX-RS annotations. The supported media types are:

- `application/json` ([MediaType.APPLICATION_JSON_TYPE](#))
- `application/octet-stream` ([MediaType.APPLICATION_OCTET_STREAM_TYPE](#))
- `application/x-www-form-urlencoded` ([MediaType.APPLICATION_FORM_URLENCODED_TYPE](#))
- `multipart/form-data` ([MediaType.MULTIPART_FORM_DATA_TYPE](#))
- `text/css`
- `text/html` ([MediaType.TEXT_HTML_TYPE](#))
- `text/plain` ([MediaType.TEXT_PLAIN_TYPE](#))

Valid HTTP(S) methods are specified by JAX-RS annotations `@GET`, `@POST`, `@PUT`, etc. Only one of these annotations can be set on each Java method (this means that a Java method can support only one HTTP method).

Warning: URL parameters with a name prefixed with `ebx-` are reserved by REST Toolkit and should not be defined by custom REST services, unless explicitly authorized by the EBX® documentation.

URL and sample

The REST URL to access the description service for the sample is defined below:

`http[s]://<host>[:<port>]/<path to webapp>/rest/track/v1/description`

Where:

- `<path to webapp>` corresponds to the web application's path holding the REST Toolkit application, itself serving the sample service. The path is composed by multiple, or none, URI segments followed by the web application's name.

Note

Please note that `/rest/track/v1/description` corresponds to the concatenation of the application's `@ApplicationPath` and service's `@Path` annotations.

The following REST Toolkit service sample provides methods to query and manage track data:

```
import java.net.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;
import java.util.stream.*;

import javax.servlet.http.*;
import javax.ws.rs.*;
import javax.ws.rs.container.*;
import javax.ws.rs.core.*;

import com.orchestranetworks.rest.annotation.*;
import com.orchestranetworks.rest.inject.*;

/**
 * The REST Toolkit Track service v1.
 */
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@Path("/track/v1")
@Documentation("Track service")
public final class TrackService
{
    @Context
    private ResourceInfo resourceInfo;
```

```

@Context
private SessionContext sessionContext;

private static final Map<Integer, TrackDTO> TRACKS = new ConcurrentHashMap<>();

/**
 * Gets service description
 */
@GET
@Path("/description")
@Documentation("Gets service description")
@Produces({ MediaType.TEXT_PLAIN, MediaType.APPLICATION_JSON })
@AnonymousAccessEnabled
public String handleServiceDescription()
{
    return this.resourceInfo.getResourceMethod().getAnnotation(Documentation.class).value();
}

/**
 * Selects tracks.
 */
@GET
@Path("/tracks")
@Documentation("Selects tracks")
public Collection<TrackDTO> handleSelectTracks(
    @QueryParam("singerFilter") final String singerFilter, // a URL parameter holding a Java regular expression
    @QueryParam("titleFilter") final String titleFilter) // a URL parameter holding a Java regular expression
{
    final Pattern singerPattern = TrackService.compilePattern(singerFilter);
    final Pattern titlePattern = TrackService.compilePattern(titleFilter);

    return TRACKS.values()
        .stream()
        .filter(Objects::nonNull)
        .filter(track -> singerPattern == null || singerPattern.matcher(track.singer).matches())
        .filter(track -> titlePattern == null || titlePattern.matcher(track.title).matches())
        .collect(Collectors.toList());
}

private static Pattern compilePattern(final String aPattern)
{
    if (aPattern == null || aPattern.isEmpty())
        return null;

    try
    {
        return Pattern.compile(aPattern);
    }
    catch (final PatternSyntaxException ignore)
    {
        // ignore invalid pattern
        return null;
    }
}

/**
 * Counts all tracks.
 */
@GET
@Path("/tracks:count")
@Documentation("Counts all tracks")
public int handleCountTracks()
{
    return TRACKS.size();
}

/**
 * Selects a track by id.
 */
@GET
@Path("/tracks/{id}")
@Documentation("Selects a track by id")
public TrackDTO handleSelectTrackById(@PathParam("id") Integer id)
{
    final TrackDTO track = TRACKS.get(id);
    if (track == null)
        throw new NotFoundException("Track id [" + id + "] does not found.");
    return track;
}

/**
 * Deletes a track by id.
 */
@DELETE
@Path("/tracks/{id}")

```

```

@Documentation("Deletes a track by id")
public void handleDeleteTrackById(@PathParam("id") Integer id)
{
    if (!TRACKS.containsKey(id))
        throw new NotFoundException("Track id [" + id + "] does not found.");
    TRACKS.remove(id);
}

/**
 * Inserts or updates one or several tracks.
 * <p>
 * The complex response structure corresponds to one of:
 * <ul>
 * <li>An empty content with the <code>location</code> HTTP header defined
 *     to the access URI.</li>
 * <li>A JSON array of {@link ResultDetailsDTO} objects.</li>
 * </ul>
 */
@POST
@Path("/tracks")
@Documentation("Inserts or updates one or several tracks")
public Response handleInsertOrUpdateTracks(List<TrackDTO> tracks)
{
    int inserted = 0;
    int updated = 0;

    final ResultDetailsDTO[] resultDetails = new ResultDetailsDTO[tracks.size()];
    int resultIndex = 0;

    final URI base = this.sessionContext.getUriInfoUtility()
        .createBuilderForRESTApplication()
        .path(this.getClass())
        .segment("tracks")
        .build();

    for (final TrackDTO track : tracks)
    {
        final String id = String.valueOf(track.id);
        final URI uri = UriBuilder.fromUri(base).segment(id).build();

        final int code;
        if (TRACKS.containsKey(track.id))
        {
            code = HttpServletResponse.SC_NO_CONTENT;
            updated++;
        }
        else
        {
            code = HttpServletResponse.SC_CREATED;
            inserted++;
        }

        TRACKS.put(track.id, track);

        resultDetails[resultIndex++] = ResultDetailsDTO.create(
            code,
            null,
            String.valueOf(track.id),
            uri);
    }

    if (inserted == 1 && updated == 0)
        return Response.created(resultDetails[0].details).build();

    return Response.ok().entity(resultDetails).build();
}

/**
 * Updates one track.
 */
@PUT
@Path("/tracks/{id}")
@Documentation("Update one track")
public void handleUpdateOneTrack(@PathParam("id") Integer id, TrackDTO aTrack)
{
    final TrackDTO track = TRACKS.get(id);
    if (track == null)
        throw new NotFoundException("Track id [" + id + "] does not found.");

    if (aTrack.id != null && !aTrack.id.equals(track.id))
        throw new BadRequestException("Selected track id [" + id
            + "] is not equals to body track id.");

    TRACKS.put(aTrack.id, aTrack);
}

```

```
}
```

This REST service uses the following Java classes, which represent a Data Transfer Objects (DTO), to serialize and deserialize data:

```
/**
 * DTO for a track.
 */
public final class TrackDTO
{
    public Integer id;
    public String singer;
    public String title;
}

import java.net.*;

/**
 * DTO for result details.
 */
@JsonbPropertyOrder({ "code", "label", "foreignKey", "details" })
public final class ResultDetailsDTO
{
    public int code;
    public String label;
    public String foreignKey;
    public URI details;

    public static ResultDetailsDTO create(
        final int aCode,
        final String aForeignKey,
        final URI aDetails)
    {
        return ResultDetailsDTO.create(aCode, null, aForeignKey, aDetails);
    }

    public static ResultDetailsDTO create(
        final int aCode,
        final String aLabel,
        final String aForeignKey,
        final URI aDetails)
    {
        final ResultDetailsDTO result = new ResultDetailsDTO();
        result.code = aCode;
        result.label = aLabel;
        result.foreignKey = aForeignKey;
        result.details = aDetails;
        return result;
    }
}
```

118.4 Serialization of a table record

Built-in serializers

Default JSON serializers and deserializers are provided to handle table records when declared in DTOs as `ContentHolderAPI`s. Both [extended](#) [p 816] and [compact](#) [p 842] JSON formats of record are supported.

```
/**
 * DTO for a singer.
 */
public final class SingerDTO
{
    @Table(
        dataModel = "urn:ebx:module:tracks-module:/WEB-INF/ebx/schemas/tracks.xsd",
        tablePath = "/root/Singers")
    public ContentHolder content;
}
```

A same DTO can be used for serialization and deserialization. In case of serialization, a `ContentHolderForInputAPI` instance will be automatically created and filled with the proper data. Afterwards, this instance will be able to copy its data into a `ValueContextForUpdateAPI`. To deserialize a table records, a `ContentHolderForOutputAPI` must be created in the REST operation JAVA method

and returned. The provided Adaptation^{API} data will then be transformed into a valid piece of JSON and placed into the HTTP response body.

```
/**
 * The REST Toolkit Singer service v1.
 */
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@Path("/singer/v1")
@Documentation("Singer service")
public final class SingerService
{
    ...

    /**
     * Selects a singer by id.
     */
    @GET
    @Path("/singers/{id}")
    @Documentation("Selects a singer by id")
    public SingerDTO handleSelectSingerById(final @PathParam("id") Integer id)
    {
        // find the singer adaptation by id
        final Adaptation singerRecord = ... ;

        final SingerDTO singerDTO = new SingerDTO();
        singerDTO.content = ContentHolderForOutput.createForRecord(singerRecord);
        return singerDTO;
    }

    /**
     * Inserts one singer.
     */
    @POST
    @Path("/singers")
    @Documentation("Inserts one singer")
    public void handleInsertOneSinger(final SingerDTO aSingerDTO)
    {
        final ProgrammaticService svc = ... ;
        final AdaptationTable singersTable = ... ;
        final ProcedureResult procedureResult = svc.execute(
            (aContext) -> {
                final ValueContextForUpdate createContext = aContext.getContextForNewOccurrence(singersTable); ;
                aSingerDTO.content.asContentHolderForInput().copyTo(createContext);
                aContext.doCreateOccurrence(createContext, singersTable);
            });

        if (procedureResult.hasFailed())
            throw new UnprocessableEntityException(
                procedureResult.getException().getLocalizedMessage());
    }

    /**
     * updates one singer.
     */
    @PUT
    @Path("/singers/{id}")
    @Documentation("updates one singer")
    public void handleUpdateOneSinger(@PathParam("id") final Integer id, final SingerDTO aSingerDTO)
    {
        final ProgrammaticService svc = ... ;
        final AdaptationTable singersTable = ... ;
        final ProcedureResult procedureResult = svc.execute(
            (aContext) -> {
                // find the singer adaptation by id
                final Adaptation singerRecord = ... ;

                if (singerRecord == null)
                    throw new NotFoundException("Singer with the id [" + id + "] has not been found.");

                final ValueContextForUpdate createContext = aContext.getContext(singerRecord.getAdaptationName()); ;
                aSingerDTO.content.asContentHolderForInput().copyTo(createContext);
                aContext.doModifyContent(singerRecord, createContext);
            });

        if (procedureResult.hasFailed()){
            final Exception ex = procedureResult.getException();
            final Throwable cause = ex.getCause();
            if(cause instanceof NotFoundException)
                throw (NotFoundException) cause;

            throw new UnprocessableEntityException(ex.getLocalizedMessage());
        }
    }
}
```

```
}
}
```

The default JSON format of the responses, only composed of business data fields, is called *compact*.

```
{
  "singer":{
    "firstname":"Frank",
    "lastname":"Sinatra"
  }
}
```

To add technical fields or metadata, the `ExtendedOutputAPI` annotation must be placed over the `ContentHolderAPI` field. The annotation must declare every wished options or the *ALL* one.

```
/**
 * DTO for a singer with technical fields.
 */
public final class SingerWithTechnicalsDTO
{
    @Table(
        dataModel = "urn:ebx:module:tracks-module:/WEB-INF/ebx/schemas/tracks.xsd",
        tablePath = "/root/Singers")
    @ExtendedOutput({Include.LABEL, Include.TECHNICALS, Include.CONTENT})
    public ContentHolder content;
}

{
  "singer":{
    "label": "23",
    "creationDate": "2018-09-04T15:35:10.706",
    "creationUser": "user",
    "lastUpdateDate": "2018-10-02T17:05:47.090",
    "lastUpdateUser": "user",
    "content":{
      "firstname":{
        "content":"Frank",
        "label":"First Name"
      },
      "lastname":{
        "content":"Sinatra",
        "label":"Last Name"
      }
    }
  }
}
```

Custom serializers

Since TIBCO EBX® is based on JSON-B ([JSR-367](#)), custom serializers and deserializers can be defined through [JsonbTypeSerializer](#) and [JsonbTypeDeserializer](#) annotations.

```
/**
 * DTO for a vinyl.
 */
public final class VinylDTO
{
    @JsonbTypeSerializer(CustomTrackDtoSerializer.class)
    @JsonbTypeDeserializer(CustomTrackDtoDeserializer.class)
    public TrackDTO track;
}
```

118.5 Authentication and lookup mechanism

A custom REST service developed with REST Toolkit supports the same authentication methods and lookup mechanism as the built-in REST data services. However, there is a slight difference concerning the 'Anonymous authentication Scheme' since its scope can be wider by using the `AnonymousAccessEnabledAPI`. See [REST authentication and permissions](#) [p 905] for more information.

Voir aussi

[Authentication](#) [p 753]

[Lookup mechanism](#) [p 754]

118.6 REST authentication and permissions

By default, every REST resource Java method requires users to be authenticated.

However, some methods may need to be accessible anonymously. These methods must be annotated by `AnonymousAccessEnabled`^{API}.

Some methods may need to be restricted to given profiles. These methods may be annotated by `Authorization`^{API} to specify an authorization rule. An authorization rule is a Java class that implements the `AuthorizationRule`^{API} interface.

```
import javax.ws.rs.*;

import com.orchestranetworks.rest.annotation.*;

/**
 * The REST Toolkit service v1.
 */
@Path("/service/v1")
@Documentation("Service")
public final class Service
{
    ...

    /**
     * Gets service description
     */
    @GET
    @AnonymousAccessEnabled
    public String handleServiceDescription()
    {
        ...
    }

    /**
     * Gets restricted service
     */
    @GET
    @Authorization(IsUserAuthorized.class)
    public RestrictedServiceDTO handleRestrictedService()
    {
        ...
    }
}
```

118.7 URI builders

REST Toolkit provides an utility interface `URIInfoUtility`^{API} to generate URIs. An instance of this interface is accessible through the injectable built-in object `SessionContext`^{API}.

URI builders for built-in

Several URI builders interfaces have been designed to ease built-in RESTful services URI build. Each interface constitute an aggregation of methods related to the same functional concept. This division allows development of modular and generic algorithms. Some of these interfaces are themselves aggregation of other ones, leading to intuitive use of the builders since only consistent combination of method calls are allowed. For example, a URI builder configured for the REST hierarchy category will not allow calls to record URI build methods, since record access are part of the REST data category concept.

Moreover, these URI builders are preconfigured according to:

- the data from the incoming HTTP request which constitute the base configuration,
- the [Module public path prefix](#) [p 386] which prepends the EBX® module segment, if it exists,

- the [data services lineage setting](#) [p 381] which overrides the whole previous configuration, if it exists,
- The [REST forward](#) [p 676] mode which overrides partially the previous configuration since only a piece of the URI's path is kept, if it is activated.

See `URIBuilderForBuiltinAPI` and `CategoryURIBuilderAPI` for more information.

URI builders for resources

URI builders to generate resource access URI are also available. Since resources are seen as only one functional concept, every methods have been defined in the same interface.

Like the URI builder for built-in data services, these ones are already preconfigured according to:

- the data from the incoming HTTP request which constitute the base configuration,
- the [external resources settings](#) [p 381] which overrides all or part of the previous configuration, if they exist,
- the [Module public path prefix](#) [p 386] which prepends the EBX® module segment, if it exists.

URI builders for server and application

When URIs to services or resources not covered by the previous builders must be build, the default preconfigured URI builders should be used. There are two default URI builders which only differ in their ending path segment. The first one end its path at the server's segment (just before the EBX® module segment) and the second at the REST application's last segment (defined in the `@ApplicationPath` annotation).

These default URI builders are already preconfigured according to:

- the data from the incoming HTTP request which constitute the base configuration,
- the [Module public path prefix](#) [p 386] which prepends the EBX® module segment, if it exists.

118.8 Exception handling

A REST Toolkit Java method can produce a standard HTTP error response by throwing a Java exception that extends the JAX-RS class `javax.ws.rs.WebApplicationException`. JAX-RS defines exceptions for various HTTP status codes. EBX® defines `UnprocessableEntityExceptionAPI` that adds support for the HTTP 422(*Unprocessable entity*) code.

Plain Java exceptions are mapped to the HTTP status code 500 (*Internal server error*).

```
{
  "userMessage": "...", // Mandatory localized message
  "errorCode": "999", // EBX® error code (optional, used mainly for HTTP error 422)
  "errors": [
    // Internal messages useful when debugging (optional).
    // Usually not displayed to the user and not localized.
    "Message 1", "Message 2" ]
}
```

118.9 Monitoring

REST Toolkit events monitoring is similar to the data services log configuration. The difference is the property key which must be `ebx.log4j.category.log.restServices`.

Voir aussi[Monitoring](#) [p 754][Configuring the EBX® logs](#) [p 376]

Some additional properties are available to configure the log messages. See [Configuring REST toolkit services](#) [p 380] for further information.

118.10 Packaging and registration

All applications and components are required to be packaged into the module's Web Application (war file).

The JAX-RS libraries, except the JAX-RS client API, are already included in `ebx.jar` and must not be packaged in the war file.

See [Java EE deployment](#) [p 339] for more information.

The registration of a REST Toolkit application is integrated into the EBX® module registration process. The registration class must extend `ModuleRegistrationListenerAPI`, declare the Servlet 3.0 annotation `@WebListener` and override the `handleContextInitialized` method.

See [Module registration](#) [p 514] for more information.

```
import javax.servlet.annotation.*;

import com.orchestranetworks.module.*;

@WebListener
public final class RegistrationModule extends ModuleRegistrationListener
{
    @Override
    public void handleContextInitialized(final ModuleInitializedContext aContext)
    {
        // Registers dynamically a REST Toolkit application.
        aContext.registerRESTApplication(RESTApplication.class);
    }
}
```

EBX® Scripting

CHAPITRE **119**

Record permission

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Lexical structure](#)
3. [Identifiers](#)
4. [Types](#)
5. [Field access](#)
6. [Operators](#)
7. [Null value management](#)
8. [If statement](#)
9. [Return statement](#)
10. [Context](#)
11. [Functions](#)

119.1 Introduction

Use the record permission DSL (Domain Specific Language) to specify access rules on records of a given table.

The main goals of the record permission DSL are to not require Java coding, and to be easy to use by people without deep programming knowledge.

You can specify permission on any table using a script. The script consists of a sequence of **if then else** and **return** statements that indicate the permission for a record.

You can edit the script using the [Data Model Assistant \(DMA\)](#) [p 36].

119.2 Lexical structure

Introduction

The script has following structure:

```
begin
  <statement 1>
  <statement 2>
  ...
```

```
<last statement>
end
```

All statements except the last one must be an ["if then"](#) [p 917] or an ["if then else"](#) [p 918] statement.

The last statement can be an ["if then"](#) [p 917], ["if then else"](#) [p 918] or ["return"](#) [p 919] statement.

Example:

```
if isMember(administrator) then
  // Administrator has no restrictions.
  return readWrite;

if isMember('french-team') and record.Country='F' then
  //Members of 'french-team' can view and modify data for France.
  return readWrite;

if isMember('us-team') and record.Country='US' then
  //Members of 'us-team' can view and modify data for US.
  return readWrite;

// This statement is not actually needed as 'hidden' is the default permission.
return hidden;
```

Character set

The Unicode character set is supported.

Character case sensitivity

The DSL is case-sensitive.

Comments

A single line comment extends from // to the end of the current line:

```
// This is a comment
if record.LastName = 'Doe' then // This is another comment.
  return readOnly;
```

A multi-line comment extends from /* and ends with */:

```
/* This is a sample of a multi-line
   comment */
if record.isActive then
  return readWrite;
```

Keywords

There are two types of keyword:

- **Reserved keywords** are: **if**, **then**, **else**, **begin**, **end**, **return**, **null**, **and**, **or**, **not**, **true**, **false**.
- **Unreserved keywords** are all other keywords defined by the DSL.

The main difference between the two types of keywords is that unreserved ones, but not reserved ones, can be used as plain (unquoted) identifiers.

119.3 Identifiers

Unquoted identifier

An **unquoted identifier** is an unlimited-length sequence of letters, digits or underscore (_). The first character must be a letter or an underscore.

Valid letters are **a to z** and **A to Z**. Valid digits are **0 to 9**.

An unquoted identifier cannot be equal to a reserved keyword.

Quoted identifiers

A quoted identifier is an unlimited length of any Unicode character except a double quote (").

Quoted identifiers **must** be used surrounded by double quotes.

An unquoted identifier can be used surrounded by double quotes. This means that identifier "a_name" is equal to a_name.

Quoted identifiers can be reserved keywords.

119.4 Types

Supported types

The following types are supported:

Type	EBX corresponding types
Boolean	xs:boolean
Decimal	xs:decimal xs:int xs:integer
String	xs:string xs:anyURI xs:Name osd:text osd:html osd:email osd:password osd:color osd:resource osd:locale osd:dataspaceKey osd:datasetName
Timestamp	xs:dateTime
Date	xs:date
Time	xs:time

Literals

String literal

String literals can be any sequence of Unicode characters surrounded by single quotes. The following table displays characters that need to be replaced by an escape sequence:

Character	Escape sequence
Tab	\t
Backspace	\b
New line	\n
Carriage return	\r
Form feed	\f
Single quote	\'
Backslash	\\

A character can be specified by a Unicode escape sequence that has the format \uXXXX with XXXX the hexadecimal code of the Unicode character.

Examples

Value	Syntax
O'Harra	'O\Harra'
Noël	'No\u00EBI'
été	'\u00e9\u00E9'

Note

An invalid escape or Unicode sequence generates an error at compile time.

Decimal literal

The following decimal formats are supported:

Format	Examples
Integer	546 -67
Floating point	54.987 -433.876 0.00054 -0.0032
Exponent notation	34.654e-5 -45E+65 1.543e23

Timestamp literal

Timestamp literals have the format **dt(yyyy-MM-dd hh:mm:ss.sss)**.

Seconds are optional. When seconds are not specified, 0 is assumed. Seconds can have fractions up to millisecond precision.

The dates that are not valid in the Gregorian calendar generate an error at compile time.

Examples:

```
dt(2010-01-02 00:00:00.000)
dt(2019-2-3 12:56:7)
dt(2019-2-3 12:56:7.5)
dt(2019-5-7 1:6)
```

Date literal

Date literals have format **d(yyyy-MM-dd)**.

The dates that are not valid in the Gregorian calendar generate an error at compile time.

Examples:

```
d(2010-01-02)
d(2019-2-3)
dt(2019-5-7)
```

Time literal

Time literals have the format **t(hh:mm:ss.sss)**.

Seconds are optional. When seconds are not specified, 0 is assumed. Seconds can have fractions up to millisecond precision.

Invalid times generate an error at compile time.

Examples:

```
t(00:00:00)
t(12:56:7)
t(12:56:7.5)
t(1:6)
```

Boolean literal

Boolean literals are **true** and **false**.

119.5 Field access

Simple fields

Only access to optimized/indexed table field is supported. Dot notation is used to access tables fields. For example, a condition of current table fields whose path is **./OfficeAddress/City** would be:

```
if record.OfficeAddress.City = 'Paris' then
  return readWrite;
```

The alias **record** always refers to the current record. Depending on context, other aliases might be available.

Each step (parts separated by a dot) is an [identifier](#) [p 911]. This means that the following quoted notation can be used for any step:

```
if record."OfficeAddress".City = 'Paris' then
  return readWrite;
```

This is useful for steps equal to a reserved keyword or using characters, such as the minus character (-) or dot (.), which are not compatible with unquoted identifiers.

At runtime, any step can evaluate to **null**. In this case, the full field expression evaluates to **null**.

Foreign table fields

You can access foreign tables by "following" foreign keys using dot notation.

In the following example, the field **Supervisor** is a foreign key:

```
if record.Supervisor.Name = 'John Doe' then
  return readWrite;
```

There can be multiple levels of foreign keys, such as in the following example:

```
if record.Supervisor.Supervisor.Supervisor.Name = 'John Doe' then
  return readOnly;
```

List (multi-valued) field access

Multi-valued fields are not supported.

Associations

Aggregate functions can take as input expressions based on an association. In the following example, the field **ManagedUsers** is an association:

```
// All users that manage at least 2 persons have read write access.
if count(record.ManagedUsers[]) >= 2 then
  return readWrite;
```

You can apply a filter on an association. You must use an alias to access fields from the association. An association alias is declared using **::**. In the following example, the alias is **u1**:

```
// All users that manage at least one person whose office is in Briton has read only access.
if exists(record.ManagedUsers:u1[u1.OfficeAddress.City='Briton']) then
  return readOnly;
```

A filter on an association can reference a field of the current record:

```
// All users that manage at least one person whose office is in the same city as user has read only access.
if exists(record.ManagedUsers:t1[t1.OfficeAddress.City=record.OfficeAddress.City]) then
  return readOnly;
```

Note

Currently, it is not possible to:

- Use an aggregate function in between [].
- Select fields of an association to aggregate values.

119.6 Operators

By default, operation evaluation order is based on precedence and associativity. The order of evaluation can be indicated explicitly by using parentheses.

The following table shows all operators from highest to lowest precedence and their associativity:

Precedence Level	Operator	Operand type	Result type	Associativity
8	[] (access to an element of a list) .(access to fields) () (parenthesis)	List index must be a decimal.	Can be any type.	Left to right.
7	not	Boolean.	Boolean.	
6	* /	Decimal.	Decimal.	Left to right.
5	+ -	Decimal.	Decimal.	Left to right.
4	< <= > >=	String, Decimal, timestamp, date, time (3).	Boolean.	Not associative.
3	= <>	String, decimal, timestamp, date, time, boolean (3).	Boolean.	
2	and	Boolean.	Boolean.	Left to right.
1	or	Boolean.	Boolean.	Left to right.

119.7 Null value management

Arithmetic operators

An arithmetic operator (*, /, + and -) returns **null** if any operand is **null**.

Comparison operators

A Comparison operator (<, <=, >, >=, = and <>) returns **null** if any operand is **null**.

Boolean operators

Boolean operators use thread-value logic.

Truth table for **and** is:

And	true	false	null
true	true	false	null
false	false	false	false
null	null	false	null

Truth table for **or** is:

Or	true	false	null
true	true	true	true
false	true	false	null
null	true	null	null

Index expressions

An indexed expression with an index that evaluates **null** or is out of range returns **null**.

Functions

Functions usually return **null** if a parameter is **null**. An exception is the function **isNull(value)**, which never returns **null**.

119.8 If statement

"If then" statement

An "if then" statement has the following syntax:

```
if condition-expression then
```

```
then-body-statement
```

The condition expression must evaluate to a boolean type.

If the expression evaluates to **true**, the "then" statement is executed. If the expression evaluates to **false** or **null**, the "then" statement is ignored.

The 'then' statement is a [body statement](#) [p 918].

"If then else" statement

An "if then else" statement has the following syntax:

```
if condition-expression then
  then-body-statement
else
  else-body-statement
```

The condition expression must be of boolean type.

If the expression evaluates to **true**, then the "then" statement is executed. If the expression evaluates to **false** or **null**, then the "else" statement is executed.

A "then" or "else" statement is a [body statement](#) [p 918].

Note

The expression:

```
if condition-expression then
  statement-a;
else
  statement-b;
```

might not be equivalent to:

```
if not condition-expression then
  statement-b;
else
  statement-a;
```

Indeed, if the expression is null, then the "else" statement is executed in both cases.

"Then" or "else" body statement

A body statement can be:

- A [return statement](#) [p 919],
- An ["if then"](#) [p 917] or ["if then else"](#) [p 918] statement,
- A statement block.

A **statement block** is used to group multiple statements. It starts with the keyword **begin** and ends with the keyword **end**.

```
begin
  <statement 1>
  <statement 2>
  ...
  <last statement>
end
```

All statements except last one must be an ["if then"](#) [p 917] or an ["if then else"](#) [p 918] statement.

The last statement can be a ["if then"](#) [p 917], ["if then else"](#) [p 918] or a ["return"](#) [p 919] statement.

```
if isMember('sales-team') then
begin
  if record.Country='F' then
    return readWrite;
```

```

    if record.Country='UK' then
        return readOnly;
    end
    else
    begin
        if record.Country='D' then
            return readOnly;

            if record.Country='B' then
                return readWrite;

            return hidden;
        end
    end

```

119.9 Return statement

A return statement specifies access to records that meet given conditions.

The following table shows valid return statements.

Return statement	Description
return hidden;	The record is hidden (the user has no access).
return readOnly;	The record is read only for the current user.
return readWrite;	The record can be read and modified by the current user.

If no return statement applies to a given record, the value 'hidden' is assumed.

119.10 Context

Introduction

Record permissions can depend on the current dataspace, dataset, or session.

Dataspace

The predefined alias **dataspace** provides access to information on the current dataspace.

This alias gives access to the following fields:

Name	Type	Description
name	string	The name of the dataspace. Because the dataspace namespace and the snapshot namespace are independent, the returned string is only an identifier in the context of one of the namespaces. For a global dataspace or snapshot identifier, use the field id .
id	string	The persistent identifier of a dataspace or snapshot. Compared to name , this identifier additionally specifies whether this id is for a dataspace or a snapshot.
isSnapshot	boolean	Is true if dataspace is a snapshot and false if dataspace is a branch.

Example:

```
if dataspace.name = 'branch-R' then
  return readOnly;
```

Dataset

The predefined alias **dataset** provides access to information on the current dataset.

This alias has the following fields:

Name	Type	Description
name	string	The name of the dataset.

Example:

```
if dataset.name = 'TEST' then
  return readWrite;
```

Session

The predefined alias **session** provides access to information on the current user session.

This alias has the following fields:

Name	Type	Description
userId	string	The current user's id.
userEmail	string	The current user's email.

To check current user's roles, see [Roles](#) [p 921].

Example:

```
if session.userId = 'jdoe' then
  return readWrite;
```


119.11 Functions

Roles

The following table describes **isMember()** function:

Syntax	Description
isMember('rolea', 'roleb'...)	Returns true if the current user has at least one of the specified roles. The following built-in roles must be specified without quotes: administrator, readOnly, everyone. Custom roles are specified as string literals. Specifying a role that does not exist is not an error. (It is considered as a role with no members.) Note The roles administrator and 'administrator' can be two distinct roles. This means that isMember(administrator) might not return the same result as is_member('administrator'). The same rule applies for other built in roles.

Example:

```
if isMember('SALES', 'SUPPORT') then
    return readWrite;

// Role administrator and not surrounded by quotes:
if isMember(administrator, 'SUPPORT') then
    return readOnly;
```

String matching functions

A string matching function takes three parameters:

- A **stringExpression** that evaluates to a string.
- A **pattern** that must be a string literal.
- An optional boolean literal **isCaseSensitive** that indicates if matching is case-sensitive. If omitted, it is assumed that matching is case-insensitive.

Syntax	Description
matches(stringExpression, pattern, isCaseSensitive)	Returns true if stringExpression matches a java regular expression pattern .
startsWith(stringExpression, pattern, isCaseSensitive)	Returns true if stringExpression starts with string pattern .
endsWith(stringExpression, pattern, isCaseSensitive)	Returns true if stringExpression ends with string pattern .
contains(stringExpression, pattern, isCaseSensitive)	Returns true if stringExpression contains a string pattern .
containsWholeWord(stringExpression, pattern, isCaseSensitive)	Returns true if stringExpression contains a whole word pattern .

Examples:

```
// Give read write access if first name starts with 'a', 'b' or 'c' (case-sensitive).
if matches(record.FirstName, '[a-c].*', true) then
    return readWrite;

// Give read only access if first name starts with 'Lé' (case-insensitive).
if startsWith(record.FirstName, 'Lé') then
    return readOnly;

// Give read only access if first name ends with 'my' (case-insensitive).
if endsWith(record.FirstName, 'my') then
    return readOnly;

// Give read only access if email contains with 'BeauMont@' (case-sensitive).
if contains(record.Email, 'BeauMont@', true) then
    return readOnly;

// Give read write access if last name contains the whole word 'Michel' (case-insensitive).
if containsWholeWord(record.LastName, 'Michel', false) then
    return readWrite;
```

Aggregate functions

The following aggregate function can be used with associations:

Syntax	Description
count(expression)	Returns the number of rows of an expression.
exists(expression)	Returns true if the expression evaluates to at least one row.

Examples:

```
// Give read write access if managed users count is at least 2.
if count(record.ManagedUsers[]) >= 2 then
    return readWrite;

// Give read only access if count of managed users that are not in Paris is less than 4.
if count(record.ManagedUsers:m1[m1.OfficeAddress.City<>'Paris']) < 4 then
    return readOnly;

// Give read write access if managed users count is at least 1.
if exists(record.ManagedUsers[]) then
    return readWrite;

// Give read write access if count of managed users that are not in Briton is at least 1.
if exists(record.ManagedUsers:u1[u1.OfficeAddress.City<>'Briton']) then
    return readWrite;
```

Miscellaneous functions

The following table describes built-in functions that return a boolean.

Syntax	Description
isNull(value)	Returns true if value is null. Value can be an expression be of any type.

Examples:

```
// Give read write access if no supervisor.
if isNull(record.supervisor) then
    return readWrite;
```

CHAPITRE 120

Function field

Ce chapitre contient les sections suivantes :

1. [Introduction](#)
2. [Lexical structure](#)
3. [Identifiers](#)
4. [Types](#)
5. [Literals](#)
6. [Operators](#)
7. [Assignments](#)
8. [Statement blocks](#)
9. [Variables and constants](#)
10. [Predefined variables](#)
11. [Functions and procedures](#)
12. [If statement](#)
13. [Loops](#)
14. [Units](#)
15. [Logging and debugging.](#)
16. [Initial script](#)

120.1 Introduction

Use the function DSL (Domain Specific Language) to define a function fieldfunction field.

This DSL is a procedural and strongly statically typed language.

To create and modify a function field, use the [Data Model Assistant \(DMA\)](#) [p 36].

The Data Model Assistant (DMA) includes a script editor that provides contextual code completion.

120.2 Lexical structure

Introduction

A script has following structure:

```
<unit usage statement 1>
<unit usage statement 2>
...
<unit usage statement N>

<function or procedure definition 1>
<function or procedure definition 2>
...
<function or procedure definition M>
```

For more information, see [Unit](#) [p 941] and [function and procedure](#) [p 937].

Example:

```
// This field returns the full address for current record.
export function getValue(): string
begin
  var address := record.FirstName | ' ' | record.LastName;

  for street in record.OfficeAddress.Street do
  begin
    address |= '\n' | street;
  end;

  address |= '\n' | record.OfficeAddress.ZipCode | ' ' | record.OfficeAddress.City;
  address |= '\n' | record.OfficeAddress.Country;

  return address;
end
```

Character set

The Unicode character set is supported.

Character case sensitivity

The DSL is case-sensitive.

Comments

A single line comment extends from // to the end of the current line:

```
// This is a comment
if record.LastName = 'Doe' then // This is another comment.
  return true;
```

A multi-line comment extends from /* and ends with */:

```
/* This is an example of a multi-line
   comment */
if record.isActive then
  return false;
```

Keywords

The reserved keywords are: **and**, **or**, **not**, **uses**, **as**, **export**, **typeof**, **mutable**, **immutable**, **unmodifiable**, **function**, **procedure**, **const**, **var**, **if**, **then**, **else**, **for**, **while**, **in**, **do**, **begin**, **end**, **return**, **true**, **false**, **null**.

Reserved keywords cannot be used as plain (unquoted) identifiers.

120.3 Identifiers

Unquoted identifier

An **unquoted identifier** is an unlimited-length sequence of letters, digits, or underscore (_). The first character must be a letter or an underscore.

Valid letters are **a** to **z** and **A** to **Z**. Valid digits are **0** to **9**.

An unquoted identifier may not be equal to a reserved keyword.

Quoted identifiers

A quoted identifier is an unlimited length of any Unicode character except double quote (").

Quoted identifiers **must** be used surrounded by double quotes.

An unquoted identifier can be used surrounded by double quotes. This means that identifier "a_name" is equal to a_name.

Quoted identifiers can be reserved keywords.

120.4 Types

Simple types

The following simple types are supported:

Type	Keyword	Properties	EBX® corresponding types
Boolean	boolean		xs:boolean
Decimal (unlimited precision)	decimal		xs:decimal
Integer (32 bits)	int		xs:int xs:integer
String	string		xs:string xs:Name osd:text osd:html osd:email osd:password ¹ osd:color osd:dataspaceKey osd:datasetName
Timestamp (millisecond precision and without time-zone)	timestamp	year month (1 to 12) day (1 to 31) hour (0 to 23) minute (0 to 59) second (0.000 to 59.999)	xs:dateTime
Date (without time-zone)	date	year month (1 to 12) day (1 to 31)	xs:date
Time (millisecond precision)	time	hour (0 to 23) minute (0 to 59) second (0.000 to 59.999)	xs:time
Locale	locale		osd:locale
URI (Uniform Resource Identifier)	uri		xs:anyURI
Resource	resource		osd:resource

¹ It is not possible to define a function field for type **osd:password**.

Complex types

A complex type is the type of a group (complex) node defined in an EBX® schema.

For more information, see the chapter [complex variables](#) [p 935]

Reference to a schema node's type (for example, when you declare a function parameter or a variable) implies using [keyword typeof](#) [p 928].

List types

The DSL supports lists. Declare a list by using the following syntax:

```
list<item_type>
```

Declare an unmodifiable list by using following syntax:

```
unmodifiable list<item_type>
```

An item cannot be added, removed, or replaced if the list is unmodifiable. An item of the list might or might not be modifiable, depending on its type.

Use indexed notation to set or get a value. (The first index is 0):

```
// Set first value of cities.
cities[0] := 'Paris';

// Get the second item of cities.
return cities[1];
```

An index can be any expression of decimal type. Convert the index value to an integer by dropping the fractional part.

A get expression returns null if the index is null, negative, or out of range.

A set expression reports an error at runtime if the index is null, negative, or out of range.

The property **size** returns the size of the list:

```
var size := cities.size;
```

You can iterate a list. For more information, see [for loops](#) [p 940].

Multi-value schema fields

Schema fields that are multi-valued are considered of list type.

A multi-valued schema field is a field that has its maximum number of values (maxOccurs) greater than one or set to "unbounded".

Indexed notation is used to access or set a value. The first index is 0. In the following example, record field 'OfficeAddress/street' is multi-valued:

```
// Return the second line of the street part of the address.
return OfficeAddress.street[1].
```

The **typeof** keyword can be used with multi-valued fields (see chapter [keyword typeof](#) [p 928]).

Mutable and immutable types

You cannot modify an immutable complex or list object.

You can modify a mutable complex or list object, but other constraints may apply that can prevent modifying some attributes of an object.

A simple type is always immutable.

You can assign a mutable value to a variable or a return value of a compatible immutable type, but assigning an immutable value to a mutable variable or a return value generates an error at compile time.

Keyword `typeof`

Use the keyword **`typeof`** to specify a type depending on a field or variable. You can use this keyword preceded by the keyword **`mutable`** or **`immutable`**.

For example, to reference the type of record field **`OfficeAddress`**, use the following syntax:

```
typeof record.OfficeAddress
```

By default, the type of a record field is always immutable. You can specify a mutable type using following syntax:

```
mutable typeof record.OfficeAddress
```

If a field is multi-valued, it is considered a list. You can reference the type of an item of the list using following syntax:

```
typeof record.Addresses[*]
```

You can use the keyword 'mutable' on types that have a **`mutable`** form:

```
mutable typeof record.Addresses[*]
```

If the type does not have a mutable form, then the keyword is simply ignored. This is the case for simple types.

===

120.5 Literals

String literal

String literals can be any sequence of Unicode characters surrounded by single quotes. The following table displays characters that need to be replaced by an escape sequence:

Character	Escape sequence
Tab	\t
Backspace	\b
New line	\n
Carriage return	\r
Form feed	\f
Single quote	\'
Backlash	\\

Specify a character by using a Unicode escape sequence that has format **`\uXXXX`**, where **`XXXX`** is the hexadecimal code of the Unicode character.

Examples

Value	Syntax
O'Harra	'O\Harra'
Noël	'No\u00EBI'
été	'\u00e9\u00E9'

Note

An invalid escape or Unicode sequence generates an error at compile time.

Decimal or integer literal

The following decimal formats are supported:

Format	Examples
Integer	546 -67
Floating point	54.987 -433.876 0.00054 -0.0032
Exponent notation	34.654e-5 -45E+65 1.543e23

Timestamp literal

Timestamp literals have the format **dt(yyyy-MM-dd hh:mm:ss.sss)**.

Seconds are optional. When they are not specified, 0 is assumed. Seconds can have fractions up to millisecond precision.

Any dates that are not valid in the Gregorian calendar generate errors at compile time.

Examples:

```
dt(2010-01-02 00:00:00.000)
dt(2019-2-3 12:56:7)
dt(2019-2-3 12:56:7.5)
dt(2019-5-7 1:6)
```

Date literal

Date literals have the format **d(yyyy-MM-dd)**.

Any dates that are not valid in the Gregorian calendar generate errors at compile time.

Examples:

```
d(2010-01-02)
```

```
d(2019-2-3)
dt(2019-5-7)
```

Time literal

Time literals have the format **t(hh:mm:ss.sss)**.

Seconds are optional. When they are not specified, 0 is assumed. Seconds can have fractions up to millisecond precision.

Invalid times generate an error at compile time.

Examples:

```
t(00:00:00)
t(12:56:7)
t(12:56:7.5)
t(1:6)
```

Boolean literal

A Boolean literal is either the keyword **true** or the keyword **false**.

Null literal

A null literal is the keyword **null**.

Use this literal only in the following situations:

- To set a variable or field,
- As a return value.

Note

It is not possible to use this keyword to test if a variable or a field is **null**. Instead, use the function **isNull()**.

120.6 Operators

Precedence

By default, operation evaluation order is based on precedence and associativity. Use parentheses to explicitly indicate the order of evaluation.

The following table shows all operators, from highest to lowest precedence, and their associativity:

Precedence Level	Operator	Operand type	Result type	Associativity
9	[] (access to an element of a list) .(access to fields) () (parenthesis)	List index must be a decimal.	Can be any type.	Left to right.
8	not	Boolean	Boolean	
7	* /	Decimal	Decimal	Left to right.
6	+ -	Decimal	Decimal	Left to right.
5	(string concatenation)	String	String	Left to right.
4	< <= > >=	String, Decimal, timestamp, date, time (3).	Boolean	Not associative.
3	= <>	String, decimal, timestamp, date, time, boolean (3).	Boolean	
2	and	Boolean	Boolean	Left to right.
1	or	Boolean	Boolean	Left to right.

Arithmetic operators

All arithmetic operators (*, /, + and -) are evaluated in decimal with 34 digits precision (IEEE 754R Decimal128 format). Result is **null** if any operand is **null**.

A decimal value is automatically converted when assigned to an **int** type variable. An error occurs if the value cannot be converted because it is not an integer, is greater than 2147483647, or is less than -2147483648.

String concatenation operator

The string concatenation operator (|) operands can be of any type and are automatically converted to a string before concatenation. The **null** value is replaced by the empty string.

The string concatenation operator (|) replaces all null operands by the empty string before executing the concatenation.

Boolean operators

Boolean operators use thread-value logic.

The truth table for the **and** operator is as follows:

And	true	false	null
true	true	false	null
false	false	false	false
null	null	false	null

The truth table for the **or** operator is as follows:

Or	true	false	null
true	true	true	true
false	true	false	null
null	true	null	null

Comparison operators

A comparison operator (<, <=, >, >=, = and <>) compares two operands of same type. Variables of 'int' type are always converted to **decimal** before comparison.

The returned value of a comparator is of **boolean** type. This value is **true** or **false** only if all operands are not **null**. It is **null** if any of its operand is **null**.

Built-in and unit functions

Built-in and unit functions usually return **null** if a parameter is **null**.

There can be exceptions, so be sure to read the [API documentation](#) [p 943].

120.7 Assignments

The following table shows all assignment operators:

Operator	Operand type	Description
<code>:=</code>	Can be used with any type.	Standard assignment.
<code> =</code>	string	<code>a = b</code> is equivalent to <code>a := a b</code>
<code>+=</code>	decimal	<code>a += b</code> is equivalent to <code>a := a + b</code>
<code>-=</code>	decimal	<code>a -= b</code> is equivalent to <code>a := a - b</code>
<code>*=</code>	decimal	<code>a *= b</code> is equivalent to <code>a := a * b</code>
<code>/=</code>	decimal	<code>a /= b</code> is equivalent to <code>a := a / b</code>

120.8 Statement blocks

A statement block is used to group multiple statements. It starts by the keyword **begin** and ends with the keyword **end**. Most statements need to be terminated by a `;`:

```
begin
  statement_1;
  statement_2;
end
```

A statement block can contain one or more statement blocks that can also contain statement blocks:

```
begin
  statement_1;
  begin
    statement_2_1;
    statement_2_3;
    begin
      statement_3_1;
      statement_3_2;
    end
  end
  statement_3;
end
```

120.9 Variables and constants

Introduction

Any statement in a block can declare a variable or a constant.

A variable or a constant always has a type that is fixed when the variable is declared.

A value can be assigned to a variable using the assignment operator `:=`.

It is an error to assign a value of the wrong type to a variable, or to change the value of a constant after its declaration.

Assignment statements

Declaration with type detection

A variable or constant type can be detected automatically if initialized when declared. This feature is called type inference.

Syntax for a variable is:

```
var variable_name := a_value;
```

The syntax for a constant is:

```
const const_name := a_value;
```

Examples:

```
begin
  var firstName := 'John';           // Variable of type string.
  var age := 30;                     // Variable of type decimal.
  var address := record.address;     // Variable of an immutable complex.

  const MAX_AGE := 100;               // Decimal constant.
  const DEFAULT_COUNTRY := 'France'; // String constant.
end
```

Note

A constant is not necessarily immutable.

Explicit type definition

It is possible to explicitly specify the variable's type.

The syntax for a variable is:

```
// Following variable initial value is null.
var variable_1_name : variable_1_type;

// Value a_value must be compatible with type variable_1_type.
var variable_2_name : variable_1_type := a_value;
```

The syntax for a constant is:

```
const const_name : variable_1_type := a_value;
```

Examples:

```
begin
  var firstName : string;
  firstName := 'John';

  var age : decimal;
  age := 30;
  var address : typeof record.address;
  address := record.address;

  const MAX_AGE : decimal := 100;
  const DEFAULT_COUNTRY : string := 'France';
end
```

Variables that are not initialized have a null value:

```
begin
  var firstName : string;
  if isNull(firstName) then
    do_something(); // Statement will be executed.
  end
```

The following statements have errors:

```
begin
  // Following will not compile because variable is not initialized and no type
  // is defined.
  var firstName;
  // Following will not compile because variable type is decimal and value is
  // a string.
  var age : decimal := 'test';
end
```

Scope

A variable or constant scope spans from its declaration to the end of the statement block where it was declared:

```
begin
  var firstName := 'John';
  var lastName := 'Doe';

  begin
    // Following is OK because firstName and lastName are visible.
    var fullName := firstName | ' ' | lastName;
    ....
  end

  // Following will NOT compile because fullName is out of the scope.
  var message := 'Please contact ' | fullName;
end
```

A variable with same name cannot exist in same block or sub blocks:

```
begin
  var firstName := 'John';

  begin
    var firstName := 'Bob'; // Error! Variable already declared.
    ....
  end
end
```

The following is correct:

```
begin
  begin
    var firstName := 'Bob';
    ....
  end
  // The following is not an error, because block where previous variable was
  // declared is ended.
  var firstName := 'John';
end
```

Complex variables

Variable or constants of complex type can be declared using type detection or [typeof keyword](#) [p 928].

Dot notation is used to access fields of a variable of complex type.

Examples:

```
// Declaration using type detection.
var address1 := record.OfficeAddress;

// Declaration using typeof notation.
var address2 : typeof record.OfficeAddress;
address2 := address1;
```

Each step (parts separated by a dot) is an [identifier](#) [p 925]. This means that following quoted notation can be used for any step:

```
var city := record."OfficeAddress".City;
```

This is useful for steps equal to a reserved keyword or using characters, such as the minus character (-) or dot (.), that are not compatible with unquoted identifiers.

At runtime, any step can evaluate to **null**. In this case the full field expression evaluates to **null**.

Multi-valued fields

Multi-valued fields are treated as ordered lists. Index notation [**index**] is used to access an item of the list. Indexes are 0 based (first index is 0).

In the following example, the field "Address" is multivalued:

```
var city := record.Address[1].City;
```

If an index is out of bound, the indexed expression will return **null**. In this case the full field expression evaluates to **null**.

An index may be a decimal expression. Only the integer part of the expression will be used.

120.10 Predefined variables

Introduction

Predefined variables allow access to the context. These contextual variables are constant and of an immutable complex type.

Record

The predefined variable **record** is available only if the current script is for a function field of a table. It allows read-only access to the current record.

Its fields are defined by the current EBX® schema.

Example:

```
// Returns the full name.
export function getValue(): string
begin
  return record.FirstName | ' ' | record.LastName;
end
```

Root

The predefined variable **root** is available only if the current script is for a function instance field of a dataset. It provides read-only access to the instance fields of current datasets.

Its fields are defined by the current EBX® schema.

Example:

```
// Returns information on current dataset data.
export function getValue(): string
begin
  return root.City | ' ' | root.Region;
end
```

Dataspace

The predefined variable **dataspace** provides access to information on the current dataspace.

This variable has following fields:

Name	Type	Description
name	string	The name of the dataspace. Since the dataspace namespace and the snapshot namespace are independent, the returned string is only an identifier in the context of one of the namespaces. For a global dataspace or snapshot identifier, use field id .
id	string	The persistent identifier of a dataspace or snapshot. Compared to name , this identifier additionally specifies whether this id is for a dataspace or a snapshot.
isSnapshot	boolean	Is true if dataspace is a snapshot and false if dataspace is a branch.

Example:

```
// Returns details on the current dataspace.
export function getValue(): string
begin
    if dataspace.isSnapshot then
        return 'Snapshot: ' | dataspace.name;
    else
        return 'Branch: ' | dataspace.name;
    end
end
```

Dataset

The predefined variable **dataset** provides access to information on the current dataset.

This variable has the following fields:

Name	Type	Description
name	string	The name of the dataset

Example:

```
// Returns the current dataset name.
export function getValue(): string
begin
    return dataset.name;
end
```

120.11 Functions and procedures

Functions

Functions are methods that return a value.

A function with no parameters has the following syntax:

```
function function_name(): return_value_type
begin
    ....
    return a_value;
end
```

A function with parameters has the following syntax:

```
function function_name(
  parameter_1 : parameter_1_type,
  parameter_2 : parameter_2_type...): return_value_type
begin
  ....
  return a_value;
end
```

The last statement of a function must always be a return statement. The function can include as many return statement as necessary.

Example:

```
function getFullName(firstName: string, lastName: string): string
begin
  if isNull(firstName) then
    return lastName;

  return firstName | ' ' | lastName;
end
```

It is illegal to return a value different from the one declared:

```
function getValue(): string
begin
  return 0; // Error! Return type is decimal, not string.
end
```

Exported function

An exported function can be called directly by EBX®.

Only one exported function is allowed per script. Its signature (name, and return type) depends on the type of the function field.

For the field of a record, definition must be similar to:

```
export function getValue(): typeof record.<path>
begin
  ...
end
```

For the field of a dataset, definition must be similar to:

```
export function getValue(): typeof root.<path>
begin
  ...
end
```

When a function field is first edited, EBX® provides an [initial script](#) [p 941] with the correct definition of the exported function.

Procedures

Procedures are methods that do not return a value.

A procedure with no parameters has the following syntax:

```
procedure procedure_name()
begin
  ....
end
```

A procedure with parameters has the following syntax:

```
procedure procedure_name(
  parameter_1 : parameter_1_type,
  parameter_2 : parameter_2_type...)
begin
  ....
```

```
end
```

A procedure cannot have a return statement, but the function can include as many return statements as necessary. A procedure statement cannot return a value.

Parameters

Simple type parameters are passed by value. This means that a function or a procedure receives a copy of the original value.

Complex and list types are passed by reference. This means that a function or a procedure receives the original object. If the function or procedure modifies the object, the original one is modified.

120.12 If statement

"If then" statements

An "if then" statement has the following syntax:

```
if condition-expression then
  then-statement
```

The condition expression must evaluate to a boolean type.

The 'then' statement can be a [statement block](#) [p 933].

"If then else" statements

An "if then else" statement has following syntax:

```
if condition-expression then
  then-statement
else
  else-statement
```

The condition expression must be of boolean type.

A 'then' or 'else' statement can be a [statement block](#) [p 933].

Note

Expression:

```
if condition-expression then
  statements-a;
else
  statements-b;
```

Cannot be equivalent to:

```
if not condition-expression then
  statements-b;
else
  statements-a;
```

Indeed, if the expression is null, the **else** statement is executed in both cases.

120.13 Loops

"For in do" loops

A "for in do" loop statement is used to select each item of a list and execute a block statement. It has following syntax:

```
for item_variable_name in list do
begin
    statement_a;
    statement_b;
    ...
end
```

The block statement is executed once for each value of the list. At each iteration, the read-only item variable takes a value of the list in the order of the list.

The name of the item variable must be unique in the current scope or an error will be generated at compile time.

If a single statement must be executed for each item, the following simpler syntax can be used:

```
for item_variable_name in list do statement;
```

The following example iterates a list of complex:

```
// Concatenate all city addresses in a single string.
var cities := '';
for address in record.Addresses do
begin
    if cities <> '' then cities |= ', ';
    cities |= address.city;
end
```

"While do" loops

A "while do" loop statement is used to execute a statement block until a condition is **true**. It has following syntax:

```
while condition do
begin
    statement_a;
    statement_b;
    ...
end
```

If a single statement must be executed, the following simpler syntax can be used:

```
while condition do statement_a;
```

The following example calculates factorial of a value:

```
function factorial(value : decimal): decimal
begin
    var factorial := value;
    while(value > 1) do
    begin
        value -= 1;
        factorial *= value;
    end
    return factorial;
end
```

120.14 Units

EBX® provides an API that is packaged in "units".

A unit can define multiple function or procedures.

Except for the default one, a unit must be declared before usage. The declaration must be at the top of the script and must follow these types of syntaxes:

```
uses package_name.unit_name_a;
uses package_name.unit_name_b as alias_b;
```

Currently, **package_name** is always **core**. A unit alias must be unique in a script.

Methods can be referenced using following syntaxes:

```
value1 := package_name.unit_name_a.function_a();
value2 := package_name.unit_name_a.function_b(parameter1, parameter2);

value3 := alias_a.function_c();
value4 := alias_a.function_d(parameter1, parameter2);

package_name.unit_name_a.procedure_a();
package_name.unit_name_a.procedure_b(parameter1, parameter2);

alias_a.procedure_c();
alias_a.procedure_d(parameter1, parameter2);
```

The following example uses the **core.list** unit to create a list:

```
uses core.list as list;

export function getValue(): typeof record.Cities
begin
    return list.of('Paris', 'Bruxelles', 'Berlin');
end
```

For details on the provided units, see the [API Documentation](#) [p 943].

120.15 Logging and debugging.

The unit [unit.log](#) [p 943] provides a function that can be used to log a message.

A message logged by scripts can be viewed using **Administration>Repository management>Scripting** EBX® menu.

Another useful feature is that if a runtime encounters an error while executing a script, it is usually logged with the line in the script where the error occurred.

120.16 Initial script

When a new function field is created using the [DMA](#) [p 36], an initial script is created.

The following example is a script created for a field of type string:

```
export function getValue(): string
begin
    return 'A string value';
end
```

The exported function signature (name, and return type) depends on the field's type and should not be changed.

CHAPITRE **121****Unit summary**

Units	Description
default [p 945]	The default unit.
core.complex [p 947]	Script unit that provides methods to create and update complex values.
core.date [p 949]	Script unit that provides date functions.
core.datetime [p 954]	Script unit that provides datetime functions.
core.list [p 960]	Script unit that provides methods to create and update lists.
core.locale [p 967]	Script unit that provides functions for managing locales.
core.log [p 969]	Script unit that provides methods to log messages.
core.math [p 972]	Script unit that provides mathematical functions and constants.
core.resource [p 981]	This script unit methods that generates resource references compatible with the schema type <code>ods:resource</code> .
core.string [p 986]	Script unit that provides string manipulation functions.
core.time [p 991]	Script unit that provides time functions.
core.uri [p 995]	Script unit that provides functions for managing Uniform Resource Identifier (URI) references.

CHAPITRE 122

Unit default

The default unit. This unit is automatically included in a script.

Methods from this script can be called directly and do not require a *uses* statement.

Methods
function isNull(value: any_type): boolean [p 945] Returns true if value is null.
function isTrueOrNull(value : boolean): boolean [p 945] Returns true if value is true or null.
function isFalseOrNull(value: boolean): boolean [p 946] Returns true if value is false or null.

Ce chapitre contient les sections suivantes :

1. [function isNull\(value: any_type\): boolean](#)
2. [function isTrueOrNull\(value : boolean\): boolean](#)
3. [function isFalseOrNull\(value: boolean\): boolean](#)

122.1 function isNull(value: any_type): boolean

Returns true if value is null.

Parameters :

value a value of any type.

Return :

true if value is null or else false. Never returns null.

122.2 function isTrueOrNull(value : boolean): boolean

Returns true if value is true or null.

Parameters :

value: a boolean value.

Return :

true if value is null or true, or else false. Never returns null.

122.3 function isFalseOrNull(value: boolean): boolean

Returns true if value is false or null.

Parameters :

value: a boolean value.

Return :

true if value is null or false, or else false. Never returns null.

CHAPITRE 123

Unit core.complex

Script unit that provides methods to create and update complex values.

Methods
of<complexType>() : complexType [p 947] Create an instance of a complex type.
of<foreignKeyType>() : foreignKeyType [p 948] Create an instance of a foreign key type.

Ce chapitre contient les sections suivantes :

1. [of<complexType>\(\) : complexType](#)
2. [of<foreignKeyType>\(\) : foreignKeyType](#)

123.1 of<complexType>() : complexType

Create an instance of a complex type.

Example:

```
uses core.complex as complex;

export function getValue(): typeof record.OfficeAddress
begin
  var value := complex.of<typeof record.OfficeAddress>();
  value.Street := '4323 Broadway';
  value.City := 'New York';
  value.State := 'NY';
  value.Zip := '10019';
  value.Country := 'USA';
  return value;
end
```

Function Types :

complexType: the object type. Is mandatory. Should be an expression typeof identifier.

Return :

the new object. Is always the mutable version of the specified type.

123.2 of<foreignKeyType>() : foreignKeyType

Create an instance of a foreign key type.

Example:

```
uses core.complex as complex;

export function getValue(): typeof record.Supervisor
begin
  var foreignKey := complex.foreignKeyOf<typeof record.Supervisor>();
  foreignKey.Id := 435;
  return foreignKey;
end
```

Function Types :

foreignKeyType: the foreign key type. Is mandatory. Should be an expression typeof identifier.

Return :

the new foreign key object. Is always mutable.

CHAPITRE 124

Unit core.date

Script unit that provides date functions.

Methods
function addYears(value: date, years: decimal): date [p 949] Adds years to a date.
function addMonths(value: date, months: decimal): datetime [p 950] Adds months to a date.
function addDays(value: date, days: decimal): date [p 950] Adds days to a date.
function fromDatetime(value: datetime): date [p 950] Converts a date and time to a date.
function days(startValue: date, endValue: date): decimal [p 950] Returns the number of days between two dates.
function seconds(startValue: date, endValue: date): decimal [p 951] Returns the number of seconds between two dates.

Ce chapitre contient les sections suivantes :

1. [function addYears\(value: date, years: decimal\): date](#)
2. [function addMonths\(value: date, months: decimal\): datetime](#)
3. [function addDays\(value: date, days: decimal\): date](#)
4. [function fromDatetime\(value: datetime\): date](#)
5. [function days\(startValue: date, endValue: date\): decimal](#)
6. [function seconds\(startValue: date, endValue: date\): decimal](#)

124.1 function addYears(value: date, years: decimal): date

Adds years to a date.

Parameters :

value: the input date

years: the number of years to be added. Must be an integer.

Return :

the new date or null if any parameter is null.

124.2 function addMonths(value: date, months: decimal): datetime

Adds months to a date.

Parameters :

value: the input date

months: the number of months to be added. Must be an integer.

Return :

the new date or null if any parameter is null.

124.3 function addDays(value: date, days: decimal): date

Adds days to a date.

Parameters :

value: the input date

months: the number of days to be added. Must be an integer.

Return :

the new date or null if any parameter is null.

124.4 function fromDatetime(value: datetime): date

Converts a date and time to a date. The input time part is ignored.

Parameters :

value: the input date and time

Return :

the new date or null if parameter is null.

124.5 function days(startValue: date, endValue: date): decimal

Returns the number of days between two dates.

Parameters :

startValue: the input start date

endValue: the input end date

Return :

the number of days or null if any parameter is null.

124.6 function seconds(startValue: date, endValue: date): decimal

Returns the number of seconds between two dates.

Parameters :

startValue: the input start date inclusive

endValue: the input end date exclusive

Return :

the number of seconds or null if any parameter is null.

CHAPITRE 125

Unit core.datetime

Script unit that provides datetime functions.

Methods
function addYears(value: datetime, years: decimal): datetime [p 955] Adds years to a date and time.
function addMonths(value: datetime, months: decimal): datetime [p 955] Adds months to a date and time.
function addDays(value: datetime, days: decimal): datetime [p 955] Adds days to a date and time.
function addHours(value: datetime, hours: decimal): datetime [p 956] Adds hours to a date and time.
function addMinutes(value: datetime, minutes: decimal): datetime [p 956] Adds minutes to a date and time.
function addSeconds(value: datetime, seconds: decimal): datetime [p 956] Adds seconds to a date and time.
function of(year: decimal, month: decimal, day: decimal, hour: decimal, minute: decimal, second: decimal): datetime [p 957] Creates a date and time given year, month, day, hours, minutes and seconds.
function fromDate(value: date): datetime [p 957] Converts a date to date and time with time set to 00:00:00.
function fromDateAndTime(dateValue: date, timeValue: time): datetime [p 957] Creates a date and time given date and time values.
function days(startValue: datetime, endValue: datetime): decimal [p 957] Returns the number of days between two date and times.
function seconds(startValue: datetime, endValue: datetime): decimal [p 958] Returns the number of seconds between two date and times.

Methods

Ce chapitre contient les sections suivantes :

1. [function addYears\(value: datetime, years: decimal\): datetime](#)
2. [function addMonths\(value: datetime, months: decimal\): datetime](#)
3. [function addDays\(value: datetime, days: decimal\): datetime](#)
4. [function addHours\(value: datetime, hours: decimal\): datetime](#)
5. [function addMinutes\(value: datetime, minutes: decimal\): datetime](#)
6. [function addSeconds\(value: datetime, seconds: decimal\): datetime](#)
7. [function of\(year: decimal, month: decimal, day: decimal, hour: decimal, minute: decimal, second: decimal\): datetime](#)
8. [function fromDate\(value: date\): datetime](#)
9. [function fromDateAndTime\(dateValue: date, timeValue: time\): datetime](#)
10. [function days\(startValue: datetime, endValue: datetime\): decimal](#)
11. [function seconds\(startValue: datetime, endValue: datetime\): decimal](#)

125.1 function addYears(value: datetime, years: decimal): datetime

Adds years to a date and time.

Parameters :

value: the input date and time

months: the number of years to be added. Must be an integer.

Return :

the new date and time or null if any parameter is null.

125.2 function addMonths(value: datetime, months: decimal): datetime

Adds months to a date and time.

Parameters :

value: the input date and time

months: the number of months to be added. Must be an integer.

Return :

the new date and time or null if any parameter is null.

125.3 function addDays(value: datetime, days: decimal): datetime

Adds days to a date and time.

Parameters :

value: the input date and time

months: the number of days to be added. Must be an integer.

Return :

the new date and time or null if any parameter is null.

125.4 function addHours(value: datetime, hours: decimal): datetime

Adds hours to a date and time.

Parameters :

value: the input date and time

hours: the number of hours to be added. Must be an integer.

Return :

the new date and time or null if any parameter is null.

125.5 function addMinutes(value: datetime, minutes: decimal): datetime

Adds minutes to a date and time.

Parameters :

value: the input date and time

minutes: the number of minutes to be added. Must be an integer.

Return :

the new date and time or null if any parameter is null.

125.6 function addSeconds(value: datetime, seconds: decimal): datetime

Adds seconds to a date and time. Fractions of seconds are supported with millisecond precision.

Parameters :

value: the input date and time

seconds: the number of seconds to be added. Fractions are rounded to three decimal digits.

Return :

the new date and time or null if any parameter is null.

125.7 function of(year: decimal, month: decimal, day: decimal, hour: decimal, minute: decimal, second: decimal): datetime

Creates a date and time given year, month, day, hours, minutes and seconds. Fractions of seconds are supported with millisecond precision.

Parameters :

year: the input year

month: the input month

day: the input day

hour: the input hour

minute: the input minute

second: the input second. Fractions are rounded to three decimal digits.

Return :

the new date and time or null if any parameter is null.

125.8 function fromDate(value: date): datetime

Converts a date to date and time with time set to 00:00:00.

Parameters :

value: the input date

Return :

the new date and time or null if parameter is null.

125.9 function fromDateAndTime(dateValue: date, timeValue: time): datetime

Creates a date and time given date and time values.

Parameters :

dateValue: the input date

timeValue: the input time

Return :

the new date and time or null if any parameter is null.

125.10 function days(startValue: datetime, endValue: datetime): decimal

Returns the number of days between two date and times. This function is equivalent to equivalent to days(toDate(startValue), toDate(endValue)).

Parameters :

startValue: the input start date and time

endValue: the input end date and time

Return :

the number of days or null if any parameter is null.

125.11 function seconds(startValue: datetime, endValue: datetime): decimal

Returns the number of seconds between two date and times. Fractions of seconds are supported with millisecond precision.

Parameters :

startValue: the input start date and time inclusive

endValue: the input end date and time exclusive

Return :

the number of seconds or null if any parameter is null.

CHAPITRE 126

Unit core.list

Script unit that provides methods to create and update lists.

Methods
function of<ItemType>(value: ItemType [, value: ItemType]...): list<ItemType> [p 961] Creates a list with the elements passed as parameters.
function isEmpty(value: list<ItemType>): boolean [p 962] Returns true if value is null or a zero size list.
function contains(values: list<ItemType>, element: ItemType): boolean [p 962] Returns true if values list contains an element.
function containsAll(values: list<ItemType>, element: ItemType): boolean [p 962] Returns true if values list contains all elements.
procedure add(values: list<ItemType>, element: ItemType) [p 963] Add an element to values list.
function remove(values: list<ItemType>, element: ItemType) : boolean [p 963] Remove the first occurrence of an element from values list.
procedure addAll(values: list<ItemType>, elements: list<ItemType>) : boolean [p 963] Add all elements to values list.
function removeAll(values: list<ItemType>, elements: list<ItemType>) : boolean [p 963] Remove all elements from values list.
function retainAll(values: list<ItemType>, elements: list<ItemType>) : boolean [p 964] Retains only the elements in the values list that are contained in the elements.
function indexOf(values: list<ItemType>, element: ItemType) [p 964] Return the index of the first occurrence of element in values list.
function lastIndexOf(values: list<ItemType>, element: ItemType) [p 964] Return the index of the last occurrence of element in values list.

Methods
function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType> [p 964] Returns a new list that contains a subsequence of elements from values list.
procedure sort(values: list<ItemType>) [p 965] Sorts the specified values list into ascending order, according to the natural ordering of its elements.
procedure reverseSort(values: list<ItemType>) [p 965] Sorts the specified values list into descending order, according to the natural ordering of its elements.
procedure reverse(values: list<ItemType>) [p 965] Reverses the order of elements in values.

Ce chapitre contient les sections suivantes :

1. [function of<ItemType>\(value: ItemType \[, value: ItemType\]...\): list<ItemType>](#)
2. [function isEmpty\(value: list<ItemType>\): boolean](#)
3. [function contains\(values: list<ItemType>, element: ItemType\): boolean](#)
4. [function containsAll\(values: list<ItemType>, element: ItemType\): boolean](#)
5. [procedure add\(values: list<ItemType>, element: ItemType\)](#)
6. [function remove\(values: list<ItemType>, element: ItemType\) : boolean](#)
7. [procedure addAll\(values: list<ItemType>, elements: list<ItemType>\) : boolean](#)
8. [function removeAll\(values: list<ItemType>, elements: list<ItemType>\) : boolean](#)
9. [function retainAll\(values: list<ItemType>, elements: list<ItemType>\) : boolean](#)
10. [function indexOf\(values: list<ItemType>, element: ItemType\)](#)
11. [function lastIndexOf\(values: list<ItemType>, element: ItemType\)](#)
12. [function subList<ItemType>\(values: list<ItemType>, startIndex: decimal, endIndex: decimal\): list<ItemType>](#)
13. [procedure sort\(values: list<ItemType>\)](#)
14. [procedure reverseSort\(values: list<ItemType>\)](#)
15. [procedure reverse\(values: list<ItemType>\)](#)

126.1 function of<ItemType>(value: ItemType [, value: ItemType]...): list<ItemType>

Creates a list with the elements passed as parameters. All elements must be of same type.

Example:

```
uses core.list as list;

function getCities(): list<string>
begin
    return list.of('Paris', 'Bruxelles', 'Berlin');
```

```
end
```

Function Types :

ItemType: the type of an item of the returned list. Is optional if at least one value is specified.

Parameters :

value: an item to add to the list. All items must be of same type.

Return :

the new list. It is updatable.

126.2 function isEmpty(value: list<ItemType>): boolean

Returns true if value is null or a zero size list.

Parameters :

values: an input list.

Return :

true if value is null or a zero size list, or else false. Never returns null.

126.3 function contains(values: list<ItemType>, element: ItemType): boolean

Returns true if values list contains an element.

Parameters :

values: the input list. Contents must be of the same kind then element.

element: the input element to look for. Must be of the same kind than values> contents.

Return :

true if value contains the element else false.

Returns null if values is null.

126.4 function containsAll(values: list<ItemType>, element: ItemType): boolean

Returns true if values list contains all elements.

Parameters :

values: the input list. Contents must be of the same kind then element.

elements: the input elements to look for. Must be of the same kind than values>.

Return :

true if value contains all elements else false.

Returns null if any parameters is null.

126.5 procedure add(values: list<ItemType>, element: ItemType)

Add an element to values list.

Parameters :

values: the input list.

element: the input element to add. Its type must be compatible with the type of the list.

126.6 function remove(values: list<ItemType>, element: ItemType) : boolean

Remove the first occurrence of an element from values list.

Parameters :

values: the input list.

element: the input element to add. Its type must be compatible with the type of the list.

Return :

true if values list contents was modified by this call else false.

126.7 procedure addAll(values: list<ItemType>, elements: list<ItemType>) : boolean

Add all elements to values list.

Parameters :

values: the input list.

elements: the elements to add. The element type must be compatible with the type of the list.

Return :

true after values list contents was modified by this call else false.

126.8 function removeAll(values: list<ItemType>, elements: list<ItemType>) : boolean

Remove all elements from values list.

Parameters :

values: the input list.

elements: the elements to remove. The element type must be compatible with the type of the list.

Return :

true if values list contents was modified by this call else false.

126.9 function retainAll(values: list<ItemType>, elements: list<ItemType>) : boolean

Retains only the elements in the `values` list that are contained in the `elements`. In other words, removes from the `values` list all of its elements that are not contained in the `elements`.

Parameters :

`values`: the input list.

`elements`: the elements to retain. The element type must be compatible with the type of the list.

Return :

true if `values` list contents was modified by this call else false.

126.10 function indexOf(values: list<ItemType>, element: ItemType)

Return the index of the first occurrence of `element` in `values` list.

Parameters :

`values`: the input list.

`element`: the input element to look for. Its type must be compatible with the type of the list.

Return :

the index of the first occurrence of `element` in `values` list or null if not found.

126.11 function lastIndexOf(values: list<ItemType>, element: ItemType)

Return the index of the last occurrence of `element` in `values` list.

Parameters :

`values`: the input list.

`element`: the input element to look for.

Return :

the index of the last occurrence of `element` in `values` list or null if not found.

126.12 function subList<ItemType>(values: list<ItemType>, startIndex: decimal, endIndex: decimal): list<ItemType>

Returns a new list that contains a subsequence of elements from `values` list.

The `subList` start from the specified `startIndex` and extends to the `endIndex` of the input list.

Function Types :

`ItemType`: the type of an item of the returned list. It is optional.

Parameters :

values: the input list

startIndex: the input start index. Must be a positive integer or zero.

endIndex: the input end index. Must be a positive integer or zero.

Return :

the new list or null if any parameter is null.

126.13 procedure sort(values: list<ItemType>)

Sorts the specified values list into ascending order, according to the natural ordering of its elements. Does nothing if values is null or a zero size list.

Parameters :

values: the input list to sort

126.14 procedure reverseSort(values: list<ItemType>)

Sorts the specified values list into descending order, according to the natural ordering of its elements. Does nothing if values is null or a zero size list.

Parameters :

values: the input list to sort.

126.15 procedure reverse(values: list<ItemType>)

Reverses the order of elements in values. Does nothing if values is null or a zero size list.

Parameters :

values: the input list to reverse.

CHAPITRE 127

Unit core.locale

Script unit that provides functions for managing locales.

Methods
function of(language: string, country: string): locale [p 967] Creates a locale.
function fromString(value: string): locale [p 967] Creates a locale from a string.

Ce chapitre contient les sections suivantes :

1. [function of\(language: string, country: string\): locale](#)
2. [function fromString\(value: string\): locale](#)

127.1 function of(language: string, country: string): locale

Creates a locale.

Parameters :

language: An ISO 639 alpha-2 or alpha-3 language code.

country: An ISO 3166 alpha-2 country code or a UN M.49 numeric-3 area code.

Return :

the locale.

127.2 function fromString(value: string): locale

Creates a locale from a string.

Parameters :

value: A locale, for example 'en-US'.

Return :

the locale.

CHAPITRE 128

Unit core.log

Script unit that provides methods to log messages.\n\nThis message can be viewed using administration user interface provide by EBX.

Methods
procedure error(message: string) [p 969] Logs an error message.
procedure warn(message: string) [p 969] Logs a warning message.
procedure info(message: string) [p 970] Logs an information message.

Ce chapitre contient les sections suivantes :

1. [procedure error\(message: string\)](#)
2. [procedure warn\(message: string\)](#)
3. [procedure info\(message: string\)](#)

128.1 procedure error(message: string)

Logs an error message.

Parameters :

value: message: the message.

128.2 procedure warn(message: string)

Logs a warning message.

Parameters :

value: message: the message.

128.3 **procedure info(message: string)**

Logs an information message.

Parameters :

value: message: the message.

CHAPITRE 129

Unit core.math

Script unit that provides mathematical functions and constants.

Methods
function sin(value: decimal): decimal [p 974] Returns the trigonometric sine of an angle.
function cos(value: decimal): decimal [p 975] Returns the trigonometric cosine of an angle.
function tan(value: decimal): decimal [p 975] Returns the trigonometric tangent of an angle.
function asin(value: decimal): decimal [p 975] Returns the arc sine of a value.
function acos(value: decimal): decimal [p 975] Returns the arc cosine of a value.
function atan(value: decimal): decimal [p 976] Returns the arc tangent of a value.
function toDegrees(value: decimal): decimal [p 976] Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
function toRadians(value: decimal): decimal [p 976] Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
function exp(value: decimal): decimal [p 977] Returns Euler's number <i>e</i> raised to the power of a decimal value.
function log(value: decimal): decimal [p 977] Returns the natural logarithm (base <i>e</i>) of the input value.
function scaleByPowerOfTen(value: decimal, power: decimal): decimal [p 977] Returns the value scale by power of ten.

Methods
function sqrt(value: decimal): decimal [p 977] Returns the positive square root of a value.
function cbrt(value: decimal): decimal [p 978] Returns the cube root of a decimal value.
function abs(value: decimal): decimal [p 978] Returns the absolute value of a decimal value.
function round(value: decimal): decimal [p 978] Returns the integer round value of the input value.
function roundHalfEven(value: decimal): decimal [p 978] Returns the integer round half even value of the input value.
function roundHalfDown(value: decimal): decimal [p 978] Returns the integer round half down value of the input value.
function roundUp(value: decimal): decimal [p 979] Returns the integer round up value of the input value.
function roundDown(value: decimal): decimal [p 979] Returns the integer round down value of the input value.
function floor(value: decimal): decimal [p 979] Returns the integer floor value of the input value.
function ceil(value: decimal): decimal [p 979] Returns the integer ceil value of the input value.
function pi(): decimal [p 979] Returns pi value : 3.
function e(): decimal [p 979] Returns e value : 2.

Ce chapitre contient les sections suivantes :

1. [function sin\(value: decimal\): decimal](#)
2. [function cos\(value: decimal\): decimal](#)
3. [function tan\(value: decimal\): decimal](#)
4. [function asin\(value: decimal\): decimal](#)
5. [function acos\(value: decimal\): decimal](#)
6. [function atan\(value: decimal\): decimal](#)
7. [function toDegrees\(value: decimal\): decimal](#)
8. [function toRadians\(value: decimal\): decimal](#)
9. [function exp\(value: decimal\): decimal](#)
10. [function log\(value: decimal\): decimal](#)
11. [function scaleByPowerOfTen\(value: decimal, power: decimal\): decimal](#)
12. [function sqrt\(value: decimal\): decimal](#)
13. [function cbrt\(value: decimal\): decimal](#)
14. [function abs\(value: decimal\): decimal](#)
15. [function round\(value: decimal\): decimal](#)
16. [function roundHalfEven\(value: decimal\): decimal](#)
17. [function roundHalfDown\(value: decimal\): decimal](#)
18. [function roundUp\(value: decimal\): decimal](#)
19. [function roundDown\(value: decimal\): decimal](#)
20. [function floor\(value: decimal\): decimal](#)
21. [function ceil\(value: decimal\): decimal](#)
22. [function pi\(\): decimal](#)
23. [function e\(\): decimal](#)

129.1 function sin(value: decimal): decimal

Returns the trigonometric sine of an angle.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Special cases:

If the input value is zero, then the result is a zero.

Parameters :

value: the input value in radians.

Return :

the sine of the input value or null if the input value is null.

129.2 function cos(value: decimal): decimal

Returns the trigonometric cosine of an angle.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value in radians.

Return :

the cosine of the input value or null if the input value is null.

129.3 function tan(value: decimal): decimal

Returns the trigonometric tangent of an angle.

Special cases:

If the input value is zero, then the result is a zero.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value in radians.

Return :

the tangent of the input value or null if the input value is null.

129.4 function asin(value: decimal): decimal

Returns the arc sine of a value.

The returned angle is in the range $-pi/2$ through $pi/2$.

Special cases:

If the input value is zero, then the result is a zero.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value. Absolute value must not be greater than 1.

Return :

the arc sine of the input value or null if the input value is null.

129.5 function acos(value: decimal): decimal

Returns the arc cosine of a value.

The returned angle is in the range 0.0 through pi .

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value. Absolute value must not be greater than 1.

Return :

the arc cosine of the input value or null if the input value is null.

129.6 function atan(value: decimal): decimal

Returns the arc tangent of a value.

The returned angle is in the range $-pi/2$ through $pi/2$.

Special cases:

If the input value is zero, then the result is a zero.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value.

Return :

the arc tangent of the input value or null if the input value is null.

129.7 function toDegrees(value: decimal): decimal

Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

The conversion from degrees to radians is generally inexact.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value in radians

Return :

the measurement of the input value in degrees or null if the input value is null.

129.8 function toRadians(value: decimal): decimal

Converts an angle measured in radians to an approximately equivalent angle measured in degrees.

The conversion from radians to degrees is generally inexact;

users should *not* expect `cos(toRadians(90.0))` to exactly equal `0.0`.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value in degrees

Return :

the measurement of the input value in radians or null if the input value is null.

129.9 function exp(value: decimal): decimal

Returns Euler's number e raised to the power of a decimal value.

Calculations are executed after conversion in double and use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value.

Return :

the value e^{value} where e is the base of the natural logarithms.

If the input value is null returns null.

129.10 function log(value: decimal): decimal

Returns the natural logarithm (base e) of the input value.

Calculations use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value. Must be strictly greater than zero.

Return :

the natural logarithm of the input value a or null if input value is null.

129.11 function scaleByPowerOfTen(value: decimal, power: decimal): decimal

Returns the value scale by power of ten.

Parameters :

value: the input value.

power: the power of ten that the value will be scale to to. Must be an integer.

Return :

the input value scaled by the power of ten or null if input value or power is null.

129.12 function sqrt(value: decimal): decimal

Returns the positive square root of a value.

Calculations use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value. Must be positive or null.

Return :

the positive square root of the input value or null if input value is null.

129.13 function cbrt(value: decimal): decimal

Returns the cube root of a decimal value. Calculations use double-precision 64-bit IEEE 754 floating point.

Parameters :

value: the input value.

Return :

the cube root of the input value or null if input value is null.

129.14 function abs(value: decimal): decimal

Returns the absolute value of a decimal value.

Parameters :

value: the input value.

Return :

the absolute value of the input value or null if input value is null.

129.15 function round(value: decimal): decimal

Returns the integer round value of the input value. This method uses the standard round half up rounding mode.

Parameters :

value: the input value.

Return :

the round half up value of the input value or null if input value is null.

129.16 function roundHalfEven(value: decimal): decimal

Returns the integer round half even value of the input value.

Parameters :

value: the input value.

Return :

the round half even value of the input value or null if input value is null.

129.17 function roundHalfDown(value: decimal): decimal

Returns the integer round half down value of the input value.

Parameters :

value: the input value.

Return :

the round half down value of the input value or null if input value is null.

129.18 function roundUp(value: decimal): decimal

Returns the integer round up value of the input value.

Parameters :

value: the input value.

Return :

the round up value of the input value or null if input value is null.

129.19 function roundDown(value: decimal): decimal

Returns the integer round down value of the input value.

Parameters :

value: the input value.

Return :

the round down value of the input value or null if input value is null.

129.20 function floor(value: decimal): decimal

Returns the integer floor value of the input value.

Parameters :

value: the input value.

Return :

the floor value of the input value or null if input value is null.

129.21 function ceil(value: decimal): decimal

Returns the integer ceil value of the input value.

Parameters :

value: the input value.

Return :

the ceil value of the input value or null if input value is null..

129.22 function pi(): decimal

Returns pi value : 3.141592653589793238462643383279503

Return :

pi value.

129.23 function e(): decimal

Returns e value : 2.718281828459045235360287471352662

Return :

e value.

CHAPITRE 130

Unit core.resource

This script unit methods that generates resource references compatible with the schema type `ods:resource`.

Example:

```
uses core.resource as resource;

export function getValue(): typeof record.functions.ResourceValue
begin
  return resource.toImage('ebx-test', 'on_anim_wait.gif');
end
```

Methods
function toImage(moduleName: string, filePath: string): string [p 982] Returns a reference to an image resource.
function toIcon(moduleName: string, filePath: string): string [p 982] Returns a reference to an icon resource.
function toJavaScript(moduleName: string, filePath: string): string [p 982] Returns a reference to an JavaScript file resource.
function toStyleSheet(moduleName: string, filePath: string): string [p 982] Returns a reference to an style sheet file resource.
function toHtml(moduleName: string, filePath: string): string [p 983] Returns a reference to an image resource.

Ce chapitre contient les sections suivantes :

- 1. [function toImage\(moduleName: string, filePath: string\): string](#)
- 2. [function toIcon\(moduleName: string, filePath: string\): string](#)
- 3. [function toJavaScript\(moduleName: string, filePath: string\): string](#)
- 4. [function toStyleSheet\(moduleName: string, filePath: string\): string](#)
- 5. [function toHtml\(moduleName: string, filePath: string\): string](#)

130.1 function tolImage(moduleName: string, filePath: string): string

Returns a reference to an image resource.

Parameters :

moduleName: the module name.

filePath: the local file path for the resource.

Return :

the resource string compatible with the schema type ods:resource.

130.2 function tolcon(moduleName: string, filePath: string): string

Returns a reference to an icon resource.

Parameters :

moduleName: the module name.

filePath: the local file path for the resource.

Return :

the resource string compatible with the schema type ods:resource.

130.3 function toJavaScript(moduleName: string, filePath: string): string

Returns a reference to an JavaScript file resource.

Parameters :

moduleName: the module name.

filePath: the local file path for the resource.

Return :

the resource string compatible with the schema type ods:resource.

130.4 function toStyleSheet(moduleName: string, filePath: string): string

Returns a reference to an style sheet file resource.

Parameters :

moduleName: the module name.

filePath: the local file path for the resource.

Return :

the resource string compatible with the schema type ods:resource.

130.5 function toHtml(moduleName: string, filePath: string): string

Returns a reference to an image resource.

Parameters :

moduleName: the module name.

filePath: the local file path for the resource.

Return :

the resource string compatible with the schema type ods:resource.

CHAPITRE 131

Unit core.string

Script unit that provides string manipulation functions.

Methods
function isEmpty(value: string): boolean [p 987] Returns true if value is null or a zero length string.
function right(value: string, index: decimal): string [p 987] Returns a new string that contains the right characters from value string.
function substring(value: string, startIndex: decimal, endIndex: decimal): string [p 987] Returns a new string that contains a subsequence of characters from value string.
function toUpperCase(value: string): string [p 988] Returns a new string with all characters from input value string to upper case.
function toLowerCase(value: string): string [p 988] Returns a new string with all characters from input value string to lower case.
function replaceAll(value: string, pattern: string, replacement: string): string [p 988] Returns a new string with all pattern occurrences in input value string replaced with replacement.
function replaceFirst(value: string, pattern: string, replacement: string): string [p 988] Returns a new string with the first pattern occurrence in input value string replaced with replacement.
function startsWith(value: string, prefix: string): boolean [p 989] Returns true if a value string starts with a prefix.
function endsWith(value: string, suffix: string): boolean [p 989] Returns true if a value string ends with a suffix.
function contains(value: string, searchString: string): boolean [p 989] Returns true if a value string contains a searchString.
function join(separator: string [, substring: string]...): string [p 990] Returns a new string compose of all input substrings separated by as separator string.

Methods

Ce chapitre contient les sections suivantes :

1. [function isEmpty\(value: string\): boolean](#)
2. [function right\(value: string, index: decimal\): string](#)
3. [function substring\(value: string, startIndex: decimal, endIndex: decimal\): string](#)
4. [function toUpperCase\(value: string\): string](#)
5. [function toLowerCase\(value: string\): string](#)
6. [function replaceAll\(value: string, pattern: string, replacement: string\): string](#)
7. [function replaceFirst\(value: string, pattern: string, replacement: string\): string](#)
8. [function startsWith\(value: string, prefix: string\): boolean](#)
9. [function endsWith\(value: string, suffix: string\): boolean](#)
10. [function contains\(value: string, searchString: string\): boolean](#)
11. [function join\(separator: string \[, substring: string\]...\): string](#)

131.1 function isEmpty(value: string): boolean

Returns true if value is null or a zero length string.

Parameters :

value: a string value.

Return :

true if value is null or a zero length string, or else false. Never returns null.

131.2 function right(value: string, index: decimal): string

Returns a new string that contains the right characters from value string.

The starting starts from the specified index and extends to the end of the string.

Parameters :

value: the input string

index: the input start index. Must be a integer greater or equal to zero and less than the length of the string.

Return :

the new string or null if any parameter is null.

131.3 function substring(value: string, startIndex: decimal, endIndex: decimal): string

Returns a new string that contains a subsequence of characters from value string.

The sub string start from the specified startIndex and extends to the endIndex of the input string.

Parameters :

value: the input string

startIndex: the input start index. Must be a integer greater or equal to zero and less than the length of the string.

endIndex: the input end index. Must be a integer greater or equal to startIndex and less than the length of the string.

Return :

the new string or null if any parameter is null.

131.4 function toUpperCase(value: string): string

Returns a new string with all characters from input value string to upper case.

Parameters :

value: the input string

Return :

the upper cased string or null if parameter is null.

131.5 function toLowerCase(value: string): string

Returns a new string with all characters from input value string to lower case.

Parameters :

value: the input string

Return :

the lower cased string or null if parameter is null.

131.6 function replaceAll(value: string, pattern: string, replacement: string): string

Returns a new string with all pattern occurrences in input value string replaced with replacement.

Parameters :

value: the input string

pattern: the string pattern (regex) to replace

replacement: the replacement string

Return :

the new string or null if parameter is null.

131.7 function replaceFirst(value: string, pattern: string, replacement: string): string

Returns a new string with the first pattern occurrence in input value string replaced with replacement.

Parameters :

value: the input string

pattern: the string pattern (regex) to replace

replacement: the replacement string

Return :

the new string or null if parameter is null.

131.8 function **startsWith(value: string, prefix: string): boolean**

Returns true if a value string starts with a prefix.

Parameters :

value: the input string

prefix: the input prefix to look for

Return :

true if value starts with prefix or else false.

Returns null if any parameters is null.

131.9 function **endsWith(value: string, suffix: string): boolean**

Returns true if a value string ends with a suffix.

Parameters :

value: the input string

suffix: the input suffix to look for

Return :

true if value ends with suffix or else false.

Returns null if any parameters is null.

131.10 function **contains(value: string, searchString: string): boolean**

Returns true if a value string contains a searchString.

Parameters :

value: the input string

searchString: the input searchString to look for

Return :

true if value contains the searchString or else false.

Returns null if any parameters is null.

131.11 function join(separator: string [, substring: string]...): string

Returns a new string compose of all input substrings separated by as separator string.

Parameters :

separator: the input separator string

substring: a substring to add to the result string. All element must be string.

Return :

a new string compose of all input substrings separated by the separator input.

If any parameters is null or no substrings is provided return null.

CHAPITRE 132

Unit core.time

Script unit that provides time functions.

Methods
function addHours(value: time, hours: decimal): time [p 992] Adds hours to a time.
function addMinutes(value: time, minutes: decimal): time [p 992] Adds minutes to a time.
function addSeconds(value: time, seconds: decimal): time [p 992] Adds seconds to a time.
function of(hours: decimal, minutes: decimal, day: decimal): time [p 992] Creates a time given hours, minutes and seconds.
function fromDatetime(value: datetime): datetime [p 993] Converts a date and time to a time.
function seconds(startValue: time, endValue: time): decimal [p 993] Returns the number of seconds between two times.

Ce chapitre contient les sections suivantes :

1. [function addHours\(value: time, hours: decimal\): time](#)
2. [function addMinutes\(value: time, minutes: decimal\): time](#)
3. [function addSeconds\(value: time, seconds: decimal\): time](#)
4. [function of\(hours: decimal, minutes: decimal, day: decimal\): time](#)
5. [function fromDatetime\(value: datetime\): datetime](#)
6. [function seconds\(startValue: time, endValue: time\): decimal](#)

132.1 function addHours(value: time, hours: decimal): time

Adds hours to a time. The calculation wraps around midnight.

Parameters :

value: the input time

hours: the number of hours to be added. Must be an integer.

Return :

the new time or null if any parameter is null.

132.2 function addMinutes(value: time, minutes: decimal): time

Adds minutes to a time. The calculation wraps around midnight.

Parameters :

value: the input time

minutes: the number of minutes to be added. Must be an integer.

Return :

the new time or null if any parameter is null.

132.3 function addSeconds(value: time, seconds: decimal): time

Adds seconds to a time. Fractions of seconds are supported with millisecond precision. The calculation wraps around midnight.

Parameters :

value: the input time

seconds: the number of seconds to be added. Fractions are rounded to three decimal digits.

Return :

the new time or null if any parameter is null.

132.4 function of(hours: decimal, minutes: decimal, day: decimal): time

Creates a time given hours, minutes and seconds. Fractions of seconds are supported with millisecond precision.

Parameters :

hour: the input hour

minute: the input minute

second: the input second. Fractions are rounded to three decimal digits.

Return :

the new time or null if any parameter is null.

132.5 function fromDatetime(value: datetime): datetime

Converts a date and time to a time. The date part is ignored.

Parameters :

value: the input date and time

Return :

the new time or null if parameter is null.

132.6 function seconds(startValue: time, endValue: time): decimal

Returns the number of seconds between two times.

Parameters :

startValue: the input start date and time inclusive

endValue: the input end date and time exclusive

Return :

the number of seconds or null if any parameter is null. Return value may have a fractional part.

CHAPITRE 133

Unit core.uri

Script unit that provides functions for managing Uniform Resource Identifier (URI) references.

Methods
function of(value: string): uri (p 995) Creates an URI reference from a string.

Ce chapitre contient les sections suivantes :

1. [function of\(value: string\): uri](#)

133.1 function of(value: string): uri

Creates an URI reference from a string.

Parameters :

value: the input value, for example 'https://www.tibco.com'.

Return :

the URI.

