TIBCO EBX® Documentation Produit

Version 5.9.23
avril 2023

TIBCO®

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 Cloud Software Group, Inc.

TIBCO and TIBCO EBX are either registered trademarks or trademarks of Cloud Software Group, 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. Cloud Software Group, 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 Cloud Software Group, Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (*https://www.tibco.com/patents*) for details.

Copyright © 2006-2023. Cloud Software Group, Inc. All rights reserved.

Table des matières

Guide utilisateur

Introduction

1. Notions clés de TIBCO EBX®.... 13
2. Interface utilisateur.... 17
3. Glossaire.... 23

Modèles de données

4. Introduction aux modèles de données.... 34

Implémentation des modèles de données

5. Création du modèle de données.... 39
6. Configuration du modèle de données.... 41
7. Modélisation de la structure des données.... 49
8. Propriétés des éléments du modèle de données.... 55
9. Contrôles sur les éléments du modèle de données.... 71
10. Barres d'outils.... 79
11. Actions sur les modèles de données existants.... 87

Publication et gestion de versions des modèles de données

12. Publication du modèle de données.... 91
13. Gestion des versions de modèles de données embarqués.... 93

Espaces de données

14. Introduction aux espaces de données.... 98
15. Création d'un espace de données.... 101
16. Actions sur les espaces de données existants.... 103
17. Images.... 113

Jeux de données

18. Introduction aux jeux de données.... 120
19. Création du jeu de données.... 123
20. Visualisation des données.... 125
21. Edition des données.... 135
22. Actions sur les jeux de données existants.... 139
23. Héritage entre jeux de données.... 143

Modèles de workflow

24. Introduction aux modèles de workflow.... 148
25. Modélisation du workflow.... 153
26. Configuration du modèle de workflow.... 165
27. Publication d'un modèle de workflow.... 173

Workflows de données

28. Introduction aux workflows de données.... 176
29. Utilisation de l'interface utilisateur de la section Workflow de données.... 179
30. Bons de travail.... 185

Gestion de workflows de données
31. Lancement et monitoring de workflows de données....191
32. Administration de workflows de données....193

Services de données

33. Introduction aux services de données....198
34. Génération de WSDL pour services de données....201

Manuel de référence

Intégration

35. Présentation de l'intégration et des déploiements....207
36. Utiliser TIBCO EBX® comme composant web....211
37. Services UI prédéfinis....221

Services d'import et d'export

38. Import et export XML....235
39. Import et export CSV....241

40. Syntaxe XPath supportée....249

Localisation

41. Libellés et localisation....258
42. Extension de l'internationalisation de TIBCO EBX®....261

Persistance

43. Présentation de la persistance....264
44. Mode relationnel....267
45. Historique....273
46. Réplication....281
47. Évolutions du modèle de données....289

Divers

48. Héritage et résolution de valeur....296
49. Permissions....301
50. Éditeur de critères....319
51. Instructions relatives aux performances....321

Guide d'administration (en anglais)

52. Administration overview....332

Installation & configuration

53. Supported environments....336
54. Java EE deployment....343

Installation notes

55. Installation note for JBoss EAP 7.1.x....353
56. Installation note for Tomcat 8.x....357
57. Installation note for WebSphere AS 9....361
58. Installation note for WebLogic 12c R2....367

59. TIBCO EBX® main configuration file....371
60. Initialization and first-launch assistant....393
61. Deploying and registering TIBCO EBX® add-ons....395

Technical administration

62. Repository administration....398
63. UI administration....409
64. Users and roles directory....425
65. Data model administration....429
66. Database mapping administration....431
67. Workflow management....435
68. Task scheduler....439
69. Audit trail....445
70. Other....447

Distributed Data Delivery (D3)

71. Introduction to D3....450
72. D3 broadcasts and delivery dataspaces....455
73. D3 JMS Configuration....459
74. D3 administration....467

Guide de sécurité (en anglais)

75. Security Best Practices.....478

Guide du développeur (en anglais)

Introduction

76. Packaging TIBCO EBX® modules.....485
77. Mapping to Java.....491
78. Tools for Java developers.....497
79. Terminology changes.....499

Data model

80. Introduction.....502
81. Data types.....505
82. Tables and relationships.....519
83. Constraints, triggers and functions.....539
84. Labels and messages.....557
85. Additional properties.....563
86. Data services.....571
87. Toolbars.....573
88. Workflow model.....575

User interface

89. Interface customization.....586

User services

90. Overview.....589
91. Quick start.....593
92. Implementing a user service.....597
93. Declaring a user service.....611
94. Development recommendations.....617

SOAP data services

95. Introduction.....622
96. WSDL generation.....633
97. SOAP operations.....641

REST data services

98. Introduction.....676
99. Built-in RESTful services.....683
100. JSON format.....721
101. REST Toolkit.....747

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.

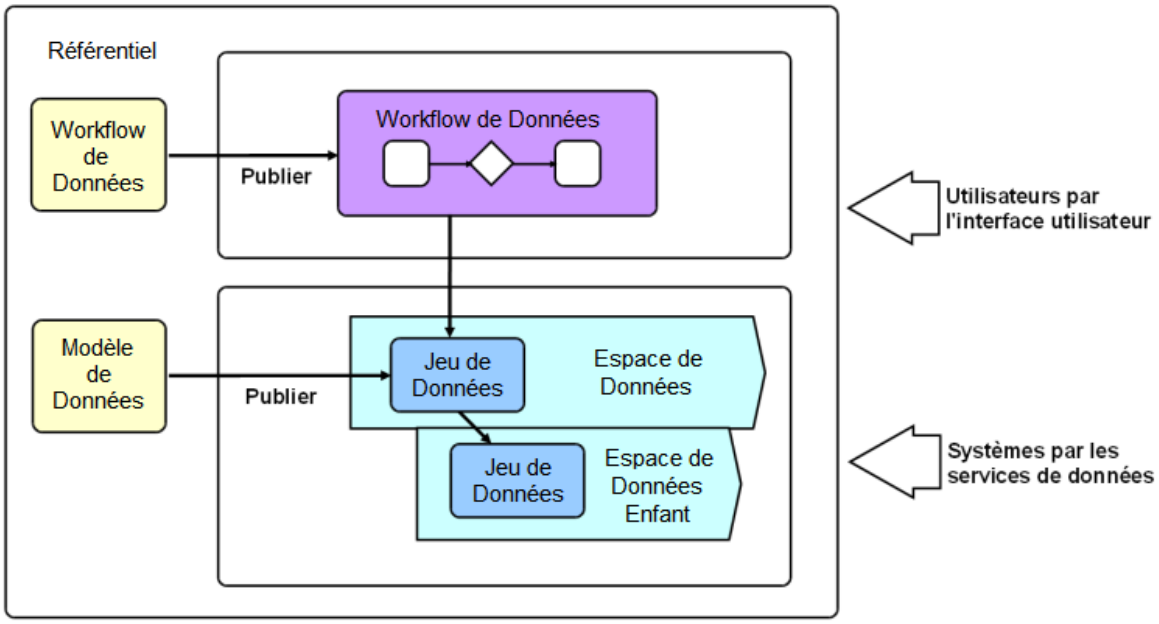
Un projet MDM basé sur EBX® commence par la création d'un *modèle de données*. Celui-ci définit les tables, les champs, les liens et les règles métiers permettant de décrire les données de référence. De bons exemples sont les catalogues de produits, les hiérarchies financières, les listes de fournisseurs 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 24]

Jeu de données [p 26]

Espace de données [p 28]

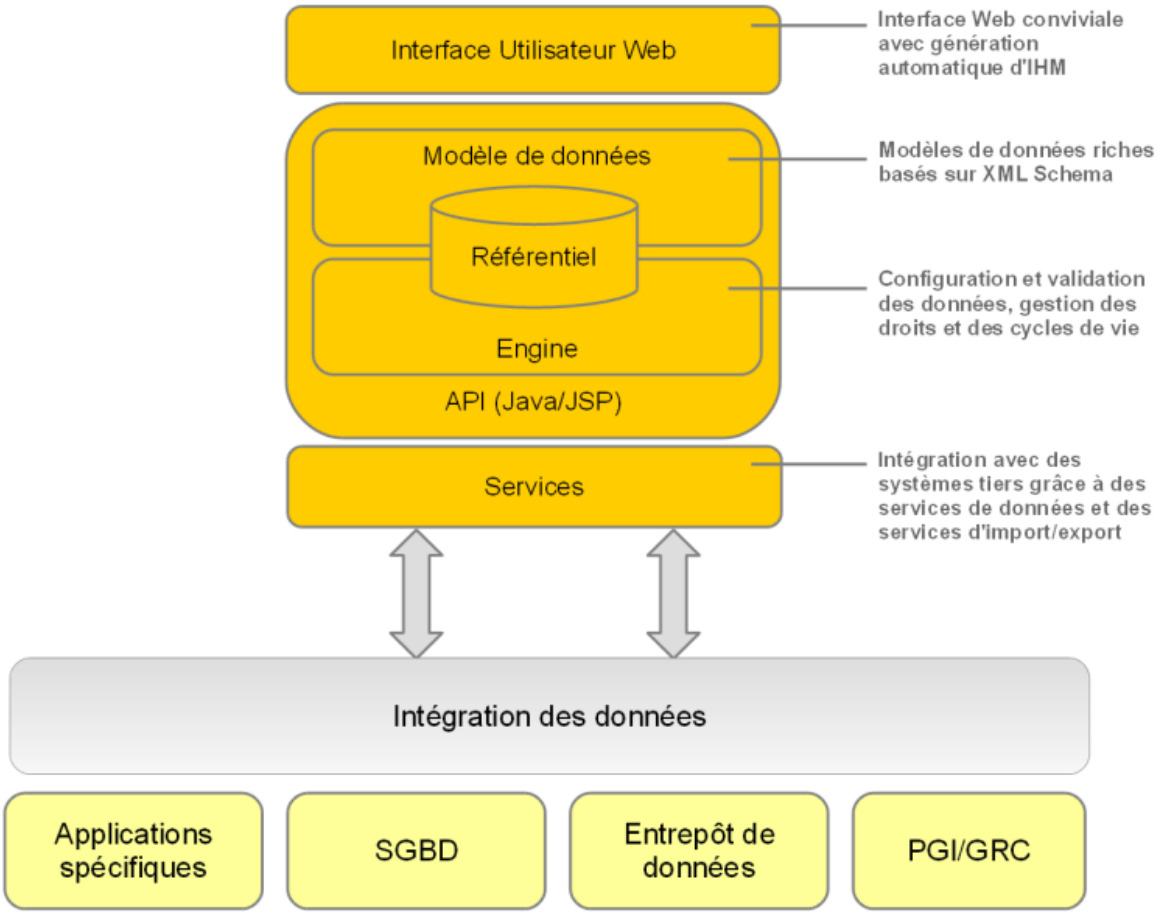
Modèle de workflow [p 29]

Workflow de données [p 30]

Service de données [p 31]

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 409]*

2.2 Perspective avancée

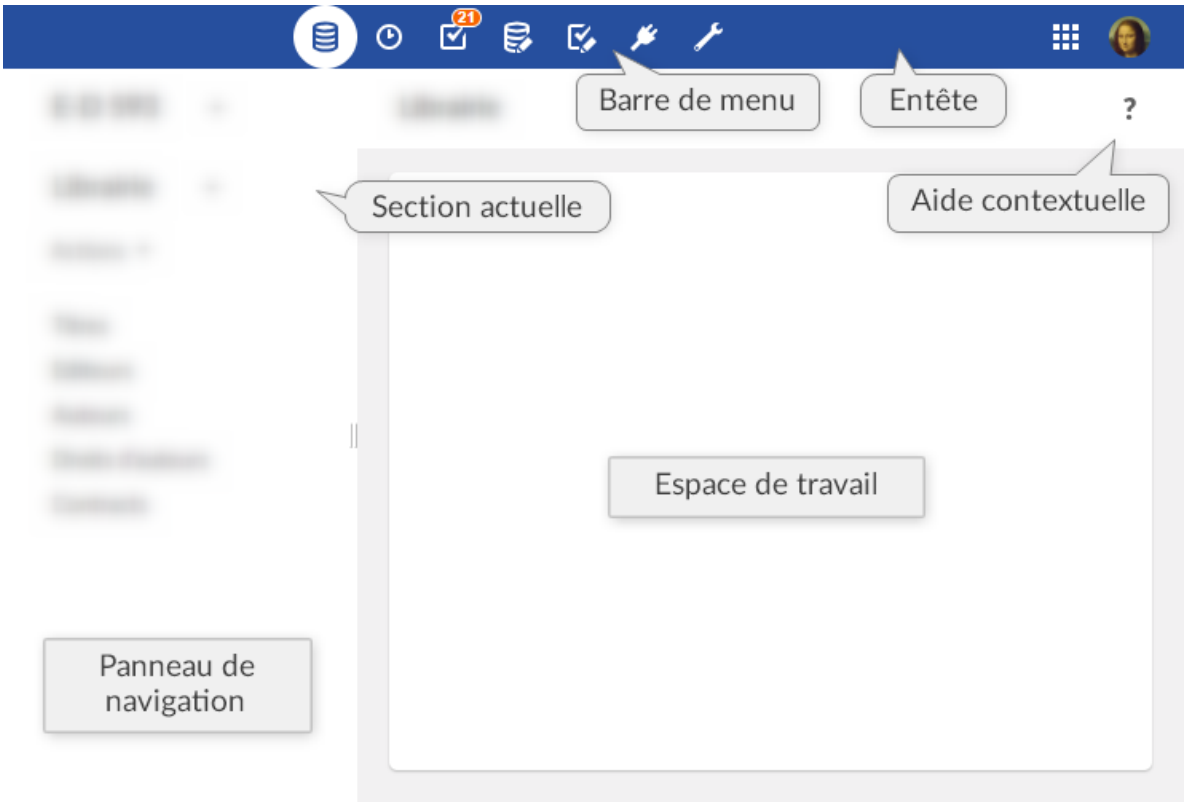
Par défaut, la perspective avancée d'EBX® est accessible à tous les utilisateurs. Cependant, son accès peut être restreint à certains profils uniquement. Cette vue est divisée en plusieurs zones principales, référencées dans la documentation sous les termes suivants:

> **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**

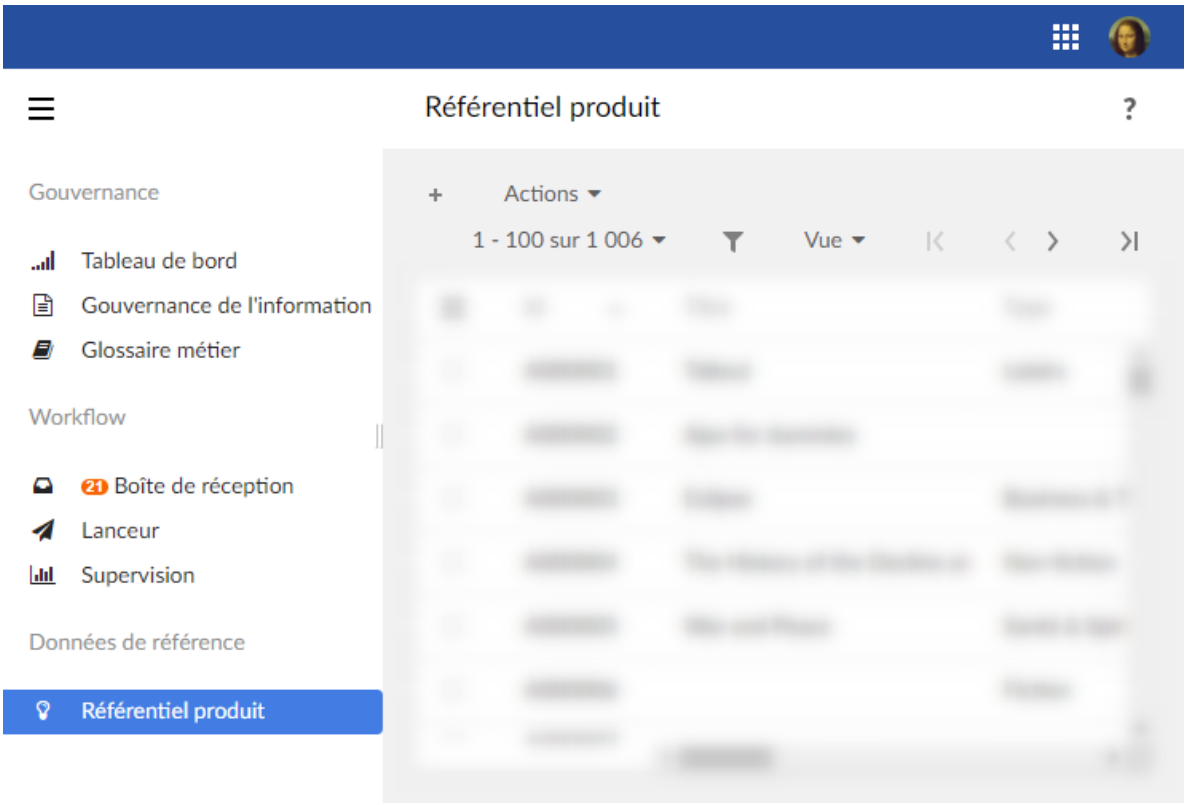
Les perspectives dans EBX® sont des vues configurables avec une audience définie. Les perspectives permettent aux utilisateurs métier de bénéficier d'une interface simplifiée. Une perspective peut être affectée à un ou plusieurs profils. Cette vue est divisée en plusieurs zones principales, référencées 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 410].

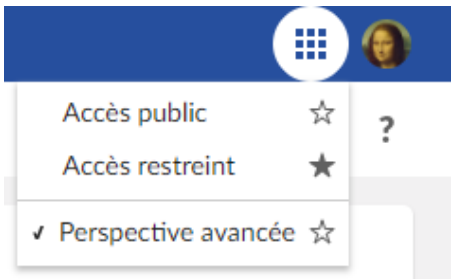
Exemple de menu hiérarchique :



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 421]*

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.addUserAvatar`[API].

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.

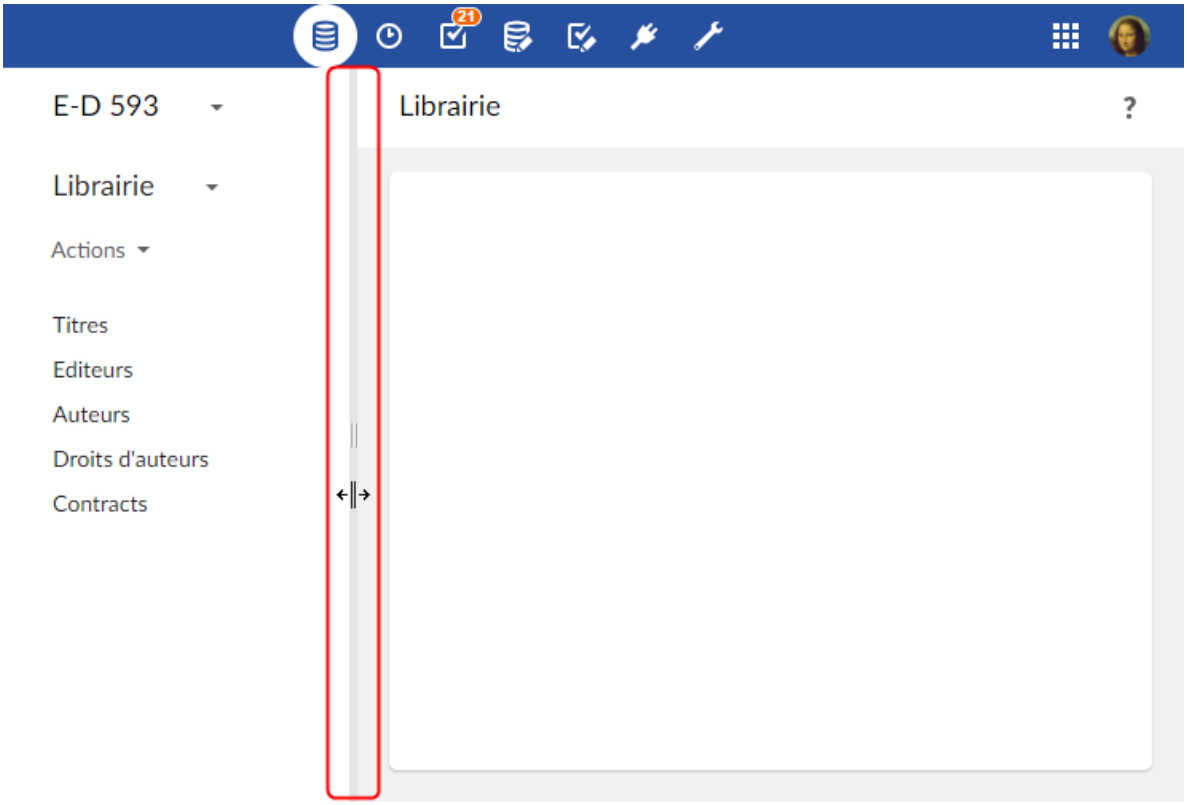
Densité

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®

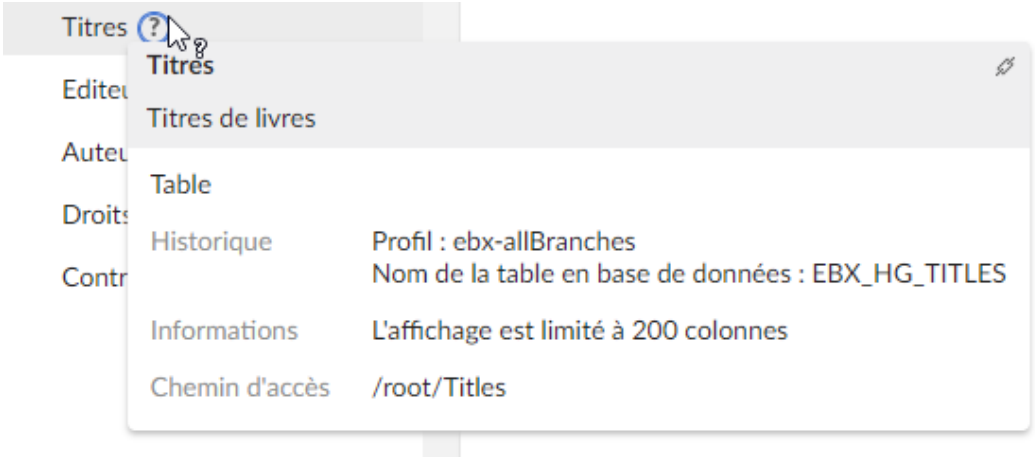
En plus de la documentation complète du produit accessible via le panneau utilisateur [p 20], l'aide est accessible de plusieurs façons dans l'interface.

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 28].

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 23], rôle [p 24].

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 24], profil [p 23].

Concept apparenté : User and roles directory [p 425].

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 24], profil [p 23].

Concept apparenté : User and roles directory [p 425].

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 23], rôle [p 24].

Concept apparenté : User and roles directory [p 425].

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 425].

API Java : `Session`[API].

3.2 Modélisation de données

Section de la documentation Modèles de données [p 34]

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 26].

Concept apparenté : Modèles de données [p 34].

champ

Elément 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 26], groupe [p 25], table (modèle de données) [p 25], règle de validation [p 26], héritage [p 27].

Concept apparenté : Propriétés des éléments de structure [p 55], Contrôles sur les champs de données [p 71].

API Java : `SchemaNode`[API].

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 519].

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 25].

Concept apparenté : Clé étrangère [p 524].

table (modèle de données)

Elément 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 26], clé primaire [p 25], type réutilisable [p 26].

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 26].

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 34].

3.3 Gestion des données

Section de la documentation Jeux de données [p 120]

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 26], clé primaire [p 25].

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 26], clé primaire [p 25].

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 26], champ [p 25], groupe [p 25], vues [p 27].

Les jeux de données sont representés par l'icône .

Concept apparenté : Jeux de données [p 120].

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 296].

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 127]

Hiérarchies [p 129]

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 131].

vue favorite

Lors de la consultation d'une table, l'utilisateur peut choisir de définir la vue actuelle comme sa vue favorite via le sous-menu 'Gérer les vues'.

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

Concept apparenté : Gérer les vues [p 132].

3.4 Cycle de vie des données

Section de la documentation Espaces de données [p 98]

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 27], référentiel [p 23], fusion [p 28].

Concept apparenté : Dataspaces [p 98].

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 28], fusion [p 28], référentiel [p 23].

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 107].

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 113]

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

3.5 Historique

Section de la documentation History [p 273]

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 29], vue historique des transactions [p 29], profil d'historisation [p 29].

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 29].

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 273].

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 273].

3.6 Modélisation de workflow

Section de la documentation Modèles de workflow [p 148]

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 148].

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 29].

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 29].

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 176]

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 29].

Concept apparenté : Workflows de données [p 176].

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 31].

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 30].

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 198]

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 203].

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 29], un noeud est une étape du workflow ou une condition.
- Dans le contexte du modèle de données [p 24], un noeud est un groupe, une table ou un champ.
- Dans le contexte des hiérarchies [p 27], un noeud représente une valeur d'une dimension.
- Dans un arbre d'héritage de jeux de données [p 27], un noeud est un jeu de données.
- Dans un jeu de données [p 26], 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 :

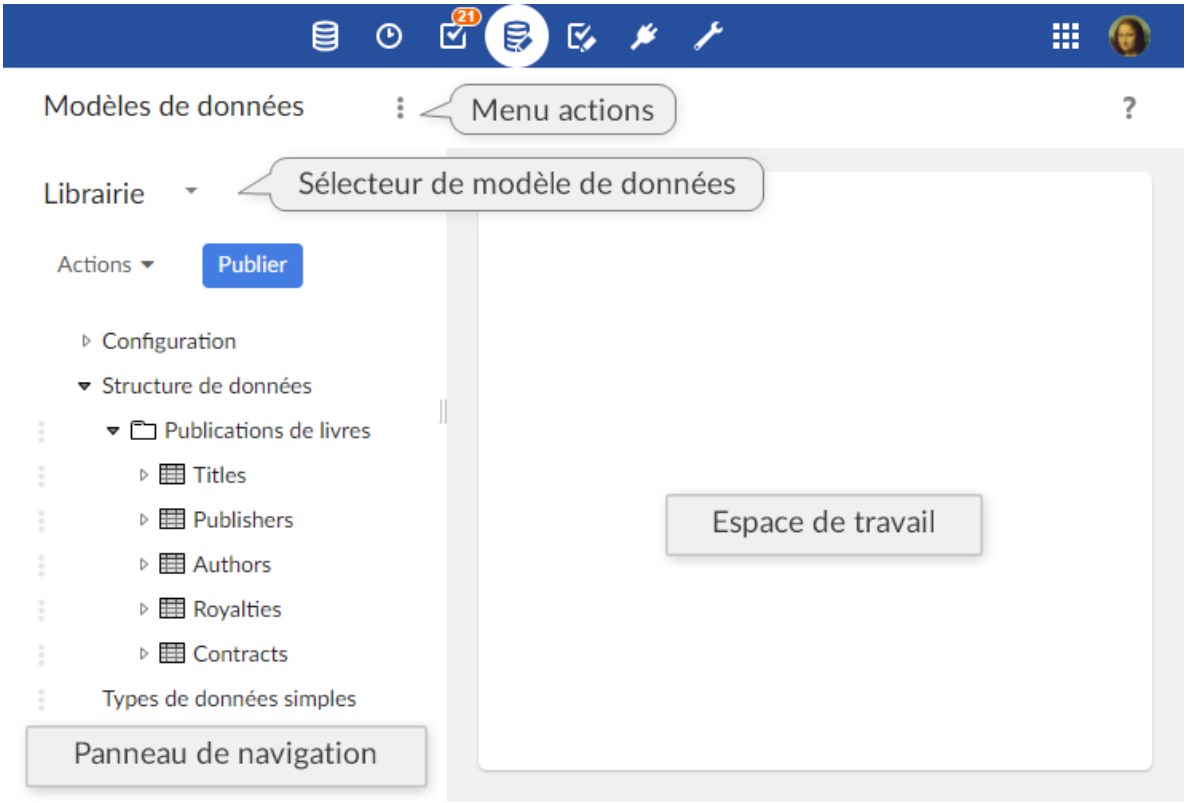
- champ [p 25]
- clé primaire [p 25]
- clé étrangère [p 25]
- table [p 25]
- groupe [p 25]
- type réutilisable [p 26]
- règle de validation [p 26]

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.
'Bindings' du modèle	Les bindings du modèle d'adaptation spécifient quels sont les types Java à générer à partir du modèle.
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.).
Barres d'outils	Les barres d'outils disponibles dans le modèle de données.
Composants Ajax	Définit les composants Ajax disponibles dans le modèle.
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).
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.
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.
Réplications	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.

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.

Voir aussi

Modélisation de la structure des données [p 49]

Configuration du modèle de données [p 41]

Types réutilisables [p 51]

Icônes des éléments du modèle de données

 champ [p 25]

clé primaire [p 25]

clé étrangère [p 25]

table [p 25]

groupe [p 25]

Concepts apparentés

Espaces de données [p 98]

Jeux de données [p 120]

CHAPITRE 5

Création du modèle de données

Ce chapitre contient les sections suivantes :

1. Création d'un modèle de données
2. Sélection d'un type de 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.

5.2 Sélection d'un type de modèle de données

Si l'utilisateur a le rôle "Administrateur", il pourra sélectionner le type de modèle de données. Un utilisateur avec le rôle "Administrateur" peut créer un modèle de données *sémantique* ou *relationnel*.

Modèles sémantiques

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

Modèles relationnels

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

> **Note**
>
> Un modèle de données relationnel peut être utilisé par un seul jeu de données. L'espace de données contenant le jeu de données doit également être déclaré comme relationnel.

Voir aussi

Relational mode [p 267]

Espaces de données [p 98]

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. Services de données
6. Réplication des données vers tables relationnelles
7. 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 35] 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 35] 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 140].

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.
Chemins des sources	Emplacements des codes sources utilisés pour configurer les composants Java dans la "Librairie de composants". Si un 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 91] 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 143] 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 558] 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 555] 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 `UserServiceDeclaration`[API] 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 517]*

6.5Services 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 571]*

6.6 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 "Réplications" 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 *Réplication [p 281]*

6.7 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 44] 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 51] 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 51] 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 25] 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 71]*

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 `Mot de passe`, 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 319]. 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 547] 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 547] 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 563] 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 319] 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 sélectionnnée, 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 sélectionnnée, 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.

- Outils de recherche

  Lorsque la propriété "Caché dans toutes les recherches" est sélectionnée, l'élément courant ne sera pas affiché dans les outils de recherche textuelle et typée d'un jeu de données. Lorsque la propriété "Caché uniquement dans une recherche textuelle" est sélectionnée, l'élément courant ne sera pas affiché dans l'outil de recherche textuelle d'un jeu de données. Si ces propriétés concernant les recherches ne sont pas renseignées, alors l'élément courant sera affiché par défaut dans les outils de recherche textuelle et 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 sélectionnnée, 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 565] 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 `UIWidgetFactory`[API] 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 567] 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 60] 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 563] 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 569] 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 568] 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 568] 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 551] 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 552] dans le Guide utilisateur.

UI bean

Voir Propriétés avancées communes [p 61].

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 61].

Propriétés d'accès

Voir Propriétés avancées communes [p 61].

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 555] dans le Guide du développeur.

Affichage par défaut

Voir Propriétés avancées communes [p 59].

Catégorie du noeud

Voir Propriétés avancées communes [p 62].

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 27] dans le glossaire.

Pour plus d'informations, voir aussi Champs hérités [p 298].

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, "/name /startDate".
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 `UILabelRenderer`[API] ou `UILabelRendererForHierarchy`[API] 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 416] 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 `UIForm`[API] et `UserServiceRecordFormFactory`[API] 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 273] pour plus d'informations.
Index	Propriété permettant d'indexer certains champs de la table. L'indexation améliore le temps d'accès des requêtes portant sur ces champs. La combinaison des champs dans chaque index doit être unique. **Nom de l'index**: Nom unique pour cet index. **Champs à indexer**: Les champs à indexer, où chaque champ d'index est identifié par un chemin XPath absolu qui débute sous la table.
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 554] dans le Guide du développeur.

Propriétés d'accès

Voir Propriétés avancées communes [p 61].

Affichage par défaut

Voir Propriétés avancées communes [p 59].

Catégorie du noeud

Voir Propriétés avancées communes [p 62].

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 61].

Transformation à l'export

Voir Propriétés avancées communes [p 61].

Propriétés d'accès

Voir Propriétés avancées communes [p 61].

Affichage par défaut

Visibilité	Voir Propriétés avancées communes [p 59].
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 62].

Concepts apparentés *Contrôles sur les éléments du modèle de données [p 71]*

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 55]*

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 539].

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 542]*

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 `TableRefDisplay`[API] 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 `TableRefFilter`[API] 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 `Constraint`[API] de l'API Java. Voir Programmatic constraints [p 546] pour plus d'informations.
Enumération spécifique (composant)	Spécifie une classe Java pour définir une énumération. La classe doit définir une liste ordonnée de valeurs, en implémentant l'interface `ConstraintEnumeration`[API] de l'API Java.
Enumération alimentée par un autre noeud	Spécifie un champ qui définit les valeurs autorisées pour cette énumération. Le champ spécifié doit être une liste ou doit définir une énumération.
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 `DataspaceSetFilter`[API].

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 55]*

CHAPITRE **10**

Barres d'outils

Ce chapitre contient les sections suivantes :

1. Définition

10.1 Définition

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 81]
- Bouton menu [p 82]

- Séparateur [p 82]
- Groupe de menu [p 83]
- Action de menu [p 84]
- Sous-menu [p 85]

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 :

Cible	Définit si le service s'exécute sur la sélection courante ou dans un composant web. Un service s'exécutant dans un composant web n'a pas accès à la sélection courante, mais il est possible de spécifier une sélection et des paramètres spécifiques.
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. **Voir aussi** *`WebComponentDeclarationContext.setAvailableAsToolbarAction`*[API]
Taille de la modale	Taille de la fenêtre modale affichant le composant web.
Redimensionnable	Indique si la fenêtre modale peut être redimensionnée.
Paramètres	Paramètres du service. Les paramètres ne peuvent être spécifiés que si la cible du service est 'Composant web'.
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.
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.
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.
Cible	Définit si le service s'exécute sur la sélection courante ou dans un composant web. Un service s'exécutant dans un composant web n'a pas accès à la sélection courante, mais il est possible de spécifier une sélection et des paramètres spécifiques.
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. **Voir aussi** `WebComponentDeclarationContext.setAvailableAsToolbarAction`[API]
Taille de la modale	Taille de la fenêtre modale affichant le composant web.
Redimensionnable	Indique si la fenêtre modale peut être redimensionnée.
Paramètres	Paramètres du service. Les paramètres ne peuvent être spécifiés que si la cible du service est 'Composant web'.

> **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 35] dans le panneau de navigation.

Voir aussi *Import et export de fichiers XML Schema Document (XSD)* *[p 87]*

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 35] dans le panneau de navigation.

Voir aussi *Import et export de fichiers XML Schema Document (XSD)* *[p 87]*

Voir aussi *Utilisation des barres d'outils* *[p 67]*

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 35] dans le panneau de navigation.

Ce chapitre contient les sections suivantes :

1. Validation du modèle de données
2. Import et export de fichiers XML Schema Document (XSD)
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 326] pour obtenir plus d'informations concernant la validation incrémentale de données.

11.2 Import et export de fichiers XML Schema Document (XSD)

TIBCO EBX® fournit des services intégrés pour importer et exporter des fichiers XML Schema Document (XSD). Les services d'import et d'export sont accessibles depuis le menu **"Actions" du modèle de données** [p 35] dans le panneau de navigation. Un import ou export est toujours effectué sur l'intégralité du modèle de données. Lors d'un import, la structure du modèle de données courant est entièrement remplacée par le contenu du document XML Schema importé. Lors d'un export, le modèle de données complet est exporté dans le document XML Schema cible.

Pour importer un fichier XSD, le fichier doit être valide et 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 43] pour plus d'informations sur la déclaration des modules.

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

Un document XML Schema (XSD) 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 à importer dans le système de fichiers local.
- **Barres d'outils :** chemin du document XML à importer dans le système de fichiers local qui contient les barres d'outils utilisées par le modèle de données.

Il est possible d'importer un document XML Schema (XSD) contenu dans le référentiel. Pour cela, sélectionnez *Importer à partir du référentiel de modèles* :

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

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 35]. 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 91] 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 41] 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 92] 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 93] 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 35]** 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 35]** 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 35].

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 402] 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

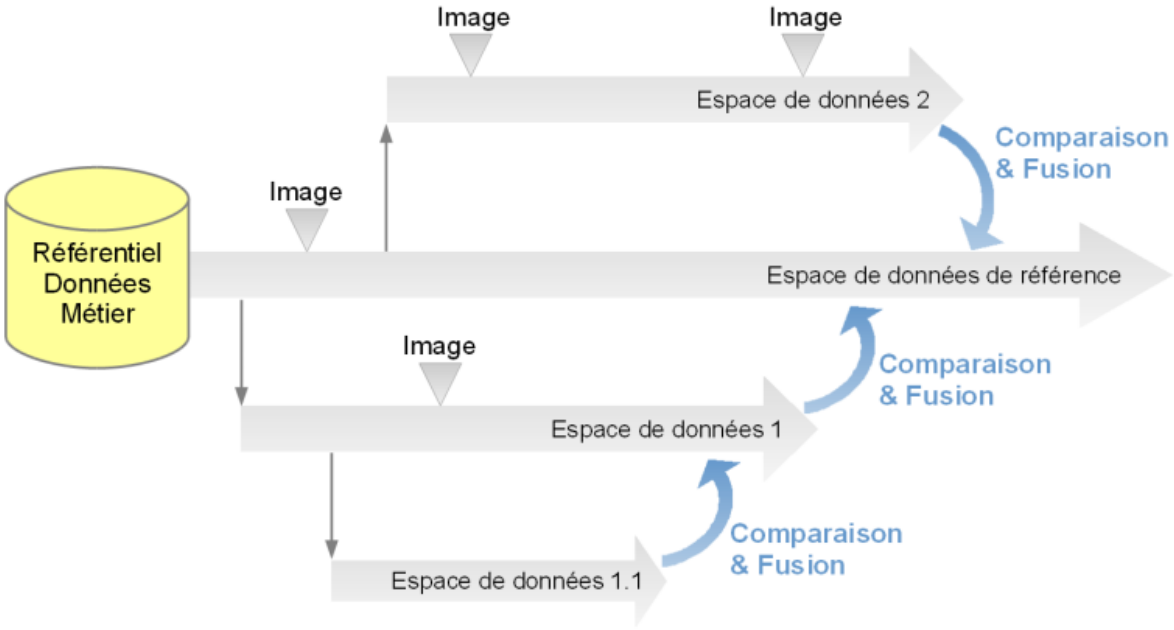
14.1 Présentation

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 :

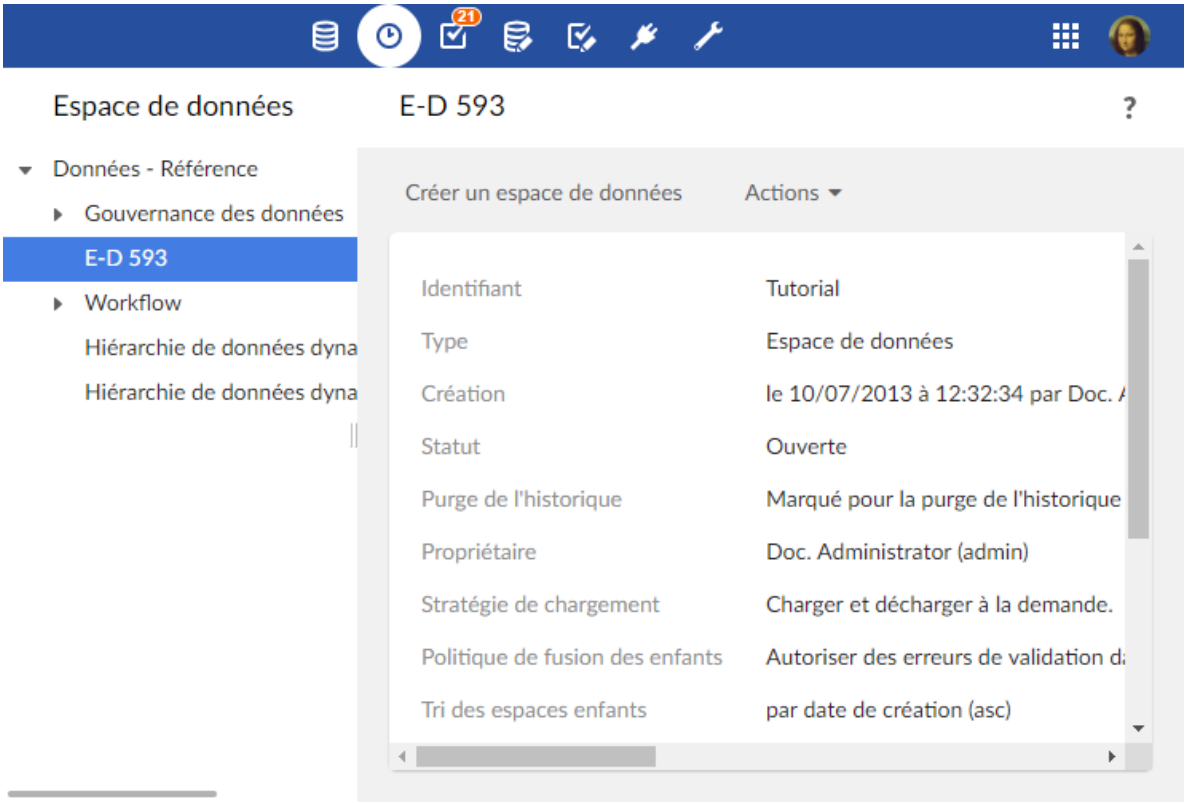
- espace de données [p 28]
- image [p 28]
- jeu de données [p 26]
- fusion [p 28]
- espace de données de référence [p 28]

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 101]*

Images *[p 113]*

Concepts apparentés *Jeux de données* *[p 120]*

CHAPITRE **15**

Création d'un espace de données

Ce chapitre contient les sections suivantes :

1. Création d'un espace de données
2. Propriétés
3. Mode relationnel

15.1 Création d'un espace de données

Par défaut, les espaces de données dans TIBCO EBX® sont en *mode sémantique*. Ce mode supporte toutes les fonctionnalités permettant la gestion du cycle de vie des données.

Pour créer un nouvel espace de données en mode sémantique, 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. Une image initiale représentant cet état est créée.

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

Voir aussi *Mode relationnel [p 102]*

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.
Mode relationnel	Spécifie si l'espace de données est en mode relationnel. Cette option existe seulement lors de la création d'un espace de données à partir de l'espace de données de référence.

15.3 **Mode relationnel**

Les espaces de données en mode relationnel ne peuvent être créés qu'à partir de l'espace de données de référence. Ils sont limités en fonctionnalités par rapport aux espaces de données en mode sémantique. Par exemple, les espaces de données en mode relationnel n'ont pas d'images et ne peuvent pas avoir d'espaces de données enfants.

Le mode relationnel implique que les tables de données de cet espace soient stockées directement dans un SGBDR.

Voir aussi *Mode relationnel [p 267]*

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	*Seul l'administrateur a le droit de modifier cette option.* • **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 309].
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 verifié 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 109]*

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 *Comparer deux contenus [p 232]*

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 402] 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 dataspaces and snapshots* *[p 404]*

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 28]*

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 104].
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 *Comparer deux contenus [p 232]*

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 402] 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 dataspaces and snapshots [p 404]*

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.

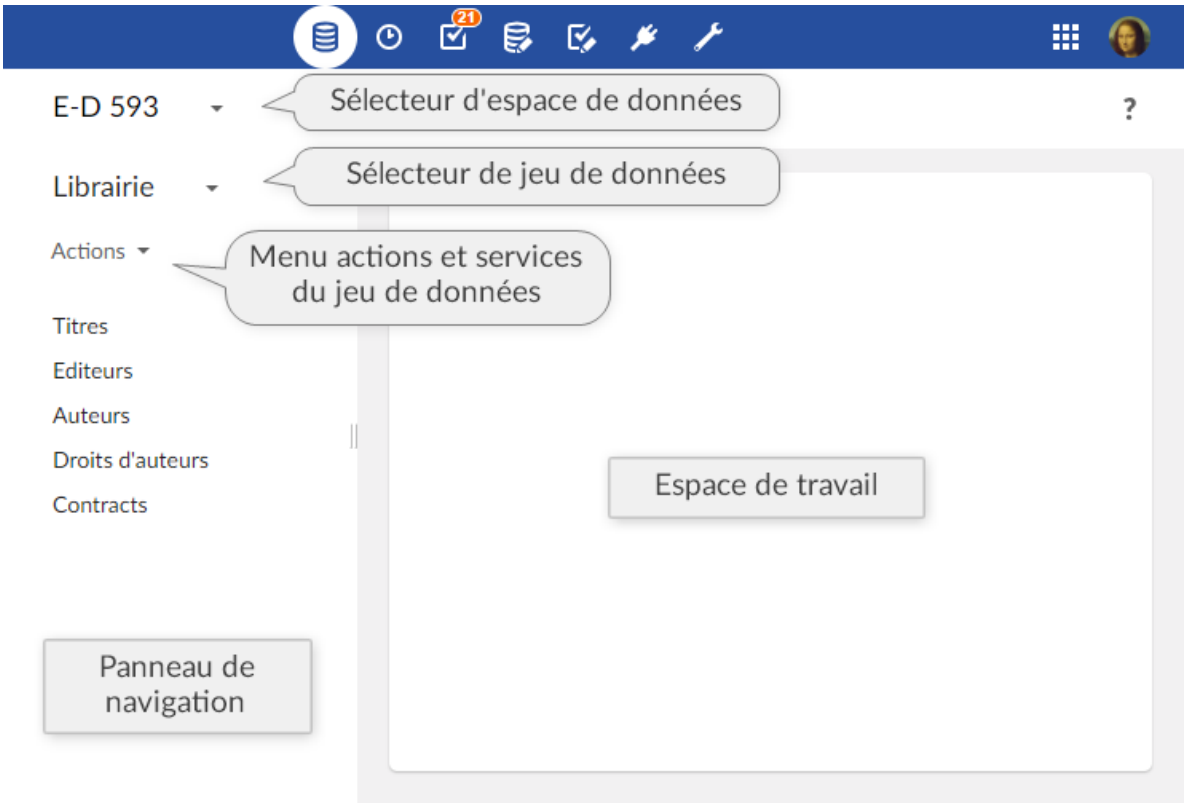
Concepts de base liés aux jeux de données

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

- espace de données [p 28]
- jeu de données [p 26]
- enregistrement [p 26]
- champ [p 25]
- clé primaire [p 25]
- clé étrangère [p 25]
- table (jeu de données) [p 26]
- groupe [p 25]

18.2 Utilisation de l'interface utilisateur de la section Données

Dans le cadre de l'utilisation de la Perspective avancée [p 17], 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 123]

Recherche et filtrage des données [p 126]

Actions sur les enregistrements dans l'interface utilisateur [p 135]

Héritage [p 27]

Concepts apparentés

Modèle de données *[p 34]*

Espace de données *[p 98]*

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 121]' 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 121] ▼ 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 143]*

CHAPITRE 20

Visualisation des données

TIBCO EBX® offre un mécanisme de personnalisation des tables via la fonctionnalité des vues. Une vue permet de spécifier les colonnes à afficher ainsi que l'ordre d'affichage. Les vues peuvent être gérées par profil grâce au concept de vues recommandées [p 131].

Ce chapitre contient les sections suivantes :

1. Menu 'Vue'
2. Tri des données
3. Recherche et filtrage des données
4. Vues
5. Gestion des vues
6. Édition tabulaire
7. 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 dans le sous-menu dédié 'Gestion de vues' [p 132].

Les vues peuvent également être regroupées. L'administrateur doit avoir défini des groupes au préalable 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 128] pour plus d'informations.

20.2 Tri des données

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

Par défaut, les enregistrements sont ordonnés par clé primaire ascendante.

Afin de définir des critères de tri spécifiques, cliquer sur le bouton 'Sélection et tri' dans l'espace de travail.

Chaque critère de tri est défini par un nom de colonne et un ordre de tri (ascendant ou descendant). Utiliser les flèches 'Déplacer à gauche/droite' pour ajouter ou enlever un critère dans la table 'Trié'. Une fois un critère sélectionné, il est possible de choisir son ordre de tri en cliquant sur le bouton 'ASC' ou 'DESC' à droite.

Pour changer la priorité d'un critère de tri, sélectionnez-le dans la liste, puis cliquez sur les flèches vers le haut ou vers le bas pour le déplacer.

Pour rétablir la vue sans critère de tri spécifique, sélectionner *Vue* > *Rétablir*.

20.3 Recherche et filtrage des données

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

Chaque panneau de filtre (ou de recherche) propose une case à cocher sur sa barre de titre : cocher cette case applique le filtre ; la décocher le désapplique.

> **Note**
>
> Appliquer une vue réinitialisera et retirera tous les filtres actuellement appliqués.

Recherche

En mode simple, la 'Recherche' permet d'ajouter des critères contextualisés sur un ou plusieurs champs. Lors de l'ajout d'un critère, les opérateurs proposés seront en adéquation avec le type du champ correspondant.

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

> **Note**
>
> En mode avancé, les critères dont l'opérateur est "correspond à" ou "correspond à (respecter la casse)" suivent la syntaxe standard des expressions régulières de Java.

Voir aussi *Regex pattern*

Recherche textuelle

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

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

Exemples :

- `aa bb` : le champ contient "aa" et "bb".
- `aa "bb cc"` : le champ contient "aa" et "bb cc".
- `aa*` : le champ commence par "aa".
- `*bb` : le champ se termine par "bb".

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

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

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

L'option *Sensible à la casse* oblige la recherche à tenir compte de la casse, en distinguant les majuscules des minuscules.

Recherche sur validation

La recherche par validation permet de voir les enregistrements en fonction de leur statut dans la dernière validation effectuée. Il est possible de voir les enregistrements avec les niveaux 'Erreur', 'Avertissements' et 'Informations'.

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

Recherches spécifiques sur tables

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

Voir aussi *Spécifier les filtres UI sur une table [p 208]*

20.4 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 129] et hiérarchique [p 129].

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, qui peut 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 423].
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 125].
Filtre	Définit les critères utilisés pour filtrer les enregistrements. Voir Editeur de critères [p 319].
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 caché 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.

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 319]*

Champ d'ordonnancement

Afin de pouvoir déplacer les noeuds dans la vue hiérarchique, il faut désigner un champ d'ordonnancement éligible. Celui-ci est défini dans la table sur laquelle est appliquée la vue hiérarchique. Un champ d'ordonnancement doit avoir le type de données 'Entier' et doit avoir la vue par défaut 'Caché' dans les propriétés avancées 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.

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

Attention

Le champ d'ordonnancement doit être un champ technique dédié, et non un champ contenant des données métier. En effet, ces données seront modifiées automatiquement par les actions de réordonnancement.

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, sélectionner 'Publier' dans le menu *Vue > Gérer les vues > nom de la vue > Publier*.

20.5 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.

Gérer les vues

Le sous-menu 'Gérer les vues' permet d'accéder aux actions suivantes :

Définir cette vue comme ma favorite	Uniquement disponible lorsque la vue en cours d'affichage n'est PAS la vue recommandée. La vue favorite sera automatiquement appliquée lors de l'accès à la table.
Définir la vue recommandée comme ma favorite	Uniquement disponible lorsqu'une vue favorite est définie. En cliquant sur cette action, la vue courante ne sera plus la vue favorite de l'utilisateur. Une vue recommandée, à l'instar de la vue favorite, est appliquée automatiquement lors de l'accès à la table. **Cette action n'est pas affichée si aucune vue favorite n'a été définie.**

20.6 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 129]*

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 énumerations qui ne sont pas des chaînes de caractères.

20.7 **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 65] 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 273]*

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 aggrégées 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 241]

Services XML [p 235]

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.isHistoryRestore`[API].

> **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 278].
> - 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 273]*

CHAPITRE 22

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

22.1 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 326] pour obtenir plus d'informations concernant la validation incrémentale de données.

22.2 Duplication d'un jeu de données

Pour dupliquer un jeu de données existant, sélectionnez-le dans le menu "*Sélectionner le jeu de données* [p 121]" ▼ dans le panneau de navigation, puis sélectionnez *Actions* > *Dupliquer*.

22.3 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*.

22.4 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 23]*

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 309].
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 139].
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 23

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 43]*

Ce chapitre contient les sections suivantes :

1. Structure de l'héritage entre jeux de données
2. Héritage de valeurs

23.1 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.

23.2 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 *Mécanisme de résolution d'enregistrement* *[p 298]*

Modèles de workflow

CHAPITRE **24**

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

24.1 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 29]*

Workflow de données (glossaire) *[p 31]*

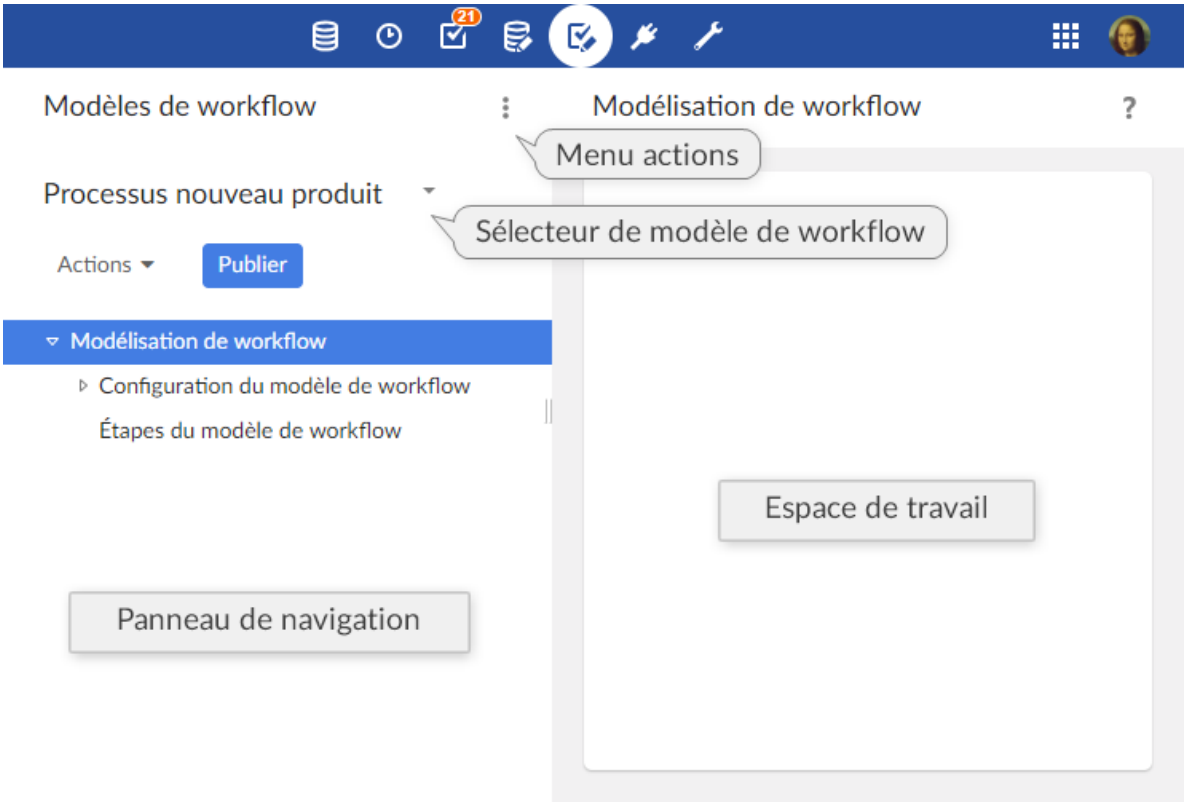
Concepts de base utilisés dans la modélisation des workflows

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

- tâche automatique [p 30]
- tâche utilisateur [p 30]
- bon de travail [p 31]

- condition de workflow [p 30]
- appel à des sous-workflows [p 30]
- tâche d'attente [p 30]
- contexte des données [p 30]

24.2 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.

24.3 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.
```

24.4 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.

Concepts apparentés *Workflows de données [p 176]*

CHAPITRE 25

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

25.1 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.

25.2 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 (excepté pour les appels de sous-workflows), 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. Les cas où cette propriété est ignorée sont les suivants : si l'étape suivante est un sous-workflow, ou 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.

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).

25.3 **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 compatiblité. Pour l'afficher, il faut activer une propriété dans le fichier `ebx.properties`. Pour plus d'informations, voir Disabling user task legacy mode [p 386].

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 157]*

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 221]*

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.handleWorkItemCompletion`[API] 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 `UserTask`[API].

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

Notification

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

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

Voir aussi *Modèles de message génériques* *[p 149]*

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.

25.4 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 `module.xml`. 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 485]

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 `ScriptTaskBean`[API]. 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.executeScript`[API] est appelée lorsque le workflow de données atteint l'étape correspondante.

Attention

La méthode `ScriptTaskBean.executeScript`[API] 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 576].

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 576].

La méthode `ScriptTask.executeScript`[API] est appelée lorsque le workflow de données atteint l'étape correspondante.

Attention

La méthode `ScriptTask.executeScript`[API] 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 576].

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.

25.5 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 `module.xml`. 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 485]

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 `ConditionBean`[API]. 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 579].

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 `Condition`[API].

Voir l'exemple [p 578] 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.

25.6 **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.

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 realisé.
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 `module.xml`. 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.

25.7 **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.

25.8 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.

Le service est disponible via un bouton dédié dans la vue hiérarchique. Le bouton est le suivant :

Ce service fournit une vue aux possibilités d'édition limitées : il est possible d'éditer les étapes existantes, mais pas d'en modifier les liens ni de créer de nouvelles étapes.

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 PDF sur une page

Vue

Mise en page > Mise en page par défaut	Applique la mise en page par défaut.
Grille > Afficher/Masquer la grille	Affiche la grille lorsque celle-ci est invisible, sinon masque la grille.

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.

Fonctionnalités additionnelles

La vue graphique offre également des fonctionnalités additionnelles

Annuler l'action précédente	CTRL + Z
Zoomer/Dezoomer	Bouton du milieu de la souris puis roulette / CTRL puis roulette
Sélection multiple	liquer 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 **26**

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

26.1 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 149] 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	*Cette propriété est obsolète.* Spécifie si un modèle de workflow est activé. Un modèle de workflow doit être activé pour pouvoir être publié.

26.2 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 149].
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 149].
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 149].
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 prioritié par défaut pour le référentiel sera appliquée à tous les workflows et bons de travail sans priorité. Voir Priorité de bons de travail [p 189] 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 workflowr	Composant interceptant les événements principaux d'un workflow. Ce bean doit être déclaré dans un fichier `module.xml`. Voir l'exemple [p 582].
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 `module.xml`. Voir l'exemple [p 581].
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'.

26.3 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.

26.4 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.

26.5 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 193]*

26.6 Historique des images du modèle de workflow

L'historique des images d'un modèle de workflow peut être géré sous **Actions > Historique**.

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**.

26.7 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 173]*

CHAPITRE 27

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

27.1 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.

27.2 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 171]*

27.3 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 **28**

Introduction aux workflows de données

Ce chapitre contient les sections suivantes :

1. Présentation

28.1 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 185]*

Lancement et monitoring de workflows de données *[p 191]*

Administration de workflows de données *[p 193]*

Permissions sur les workflows de données associés *[p 169]*

Concepts apparentés *Modèles de workflow* *[p 148]*

CHAPITRE **29**

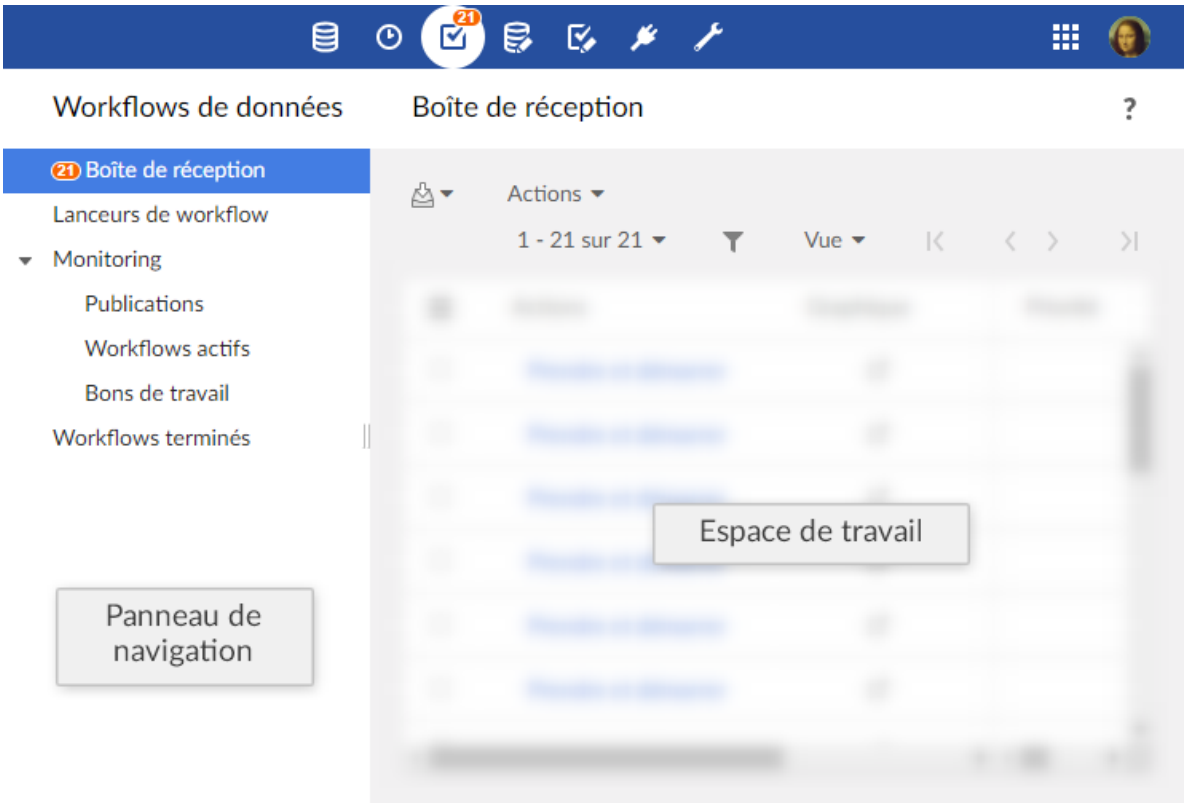
Utilisation de l'interface utilisateur de la section Workflow de données

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

29.1 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

> *Utiliser TIBCO EBX® comme composant web* *[p 211]*
>
> `ServiceKey`[API]

29.2 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 154] 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 167] dans la modélisation de workflow.

29.3 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 127] pour plus d'informations.

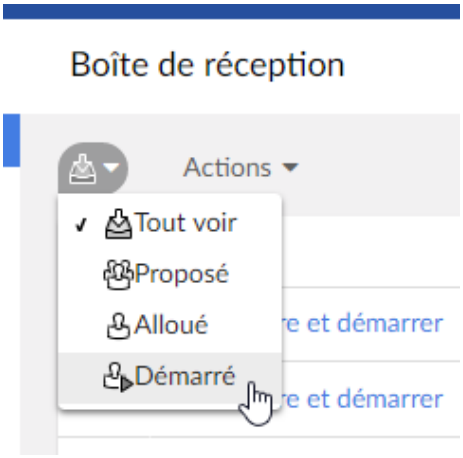
29.4 Colonnes spécifiques

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.

29.5 **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.



29.6 **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 décison qui a été prise pour le bon de travail (accepté ou rejeté).

CHAPITRE 30

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

30.1 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 589]*

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*.

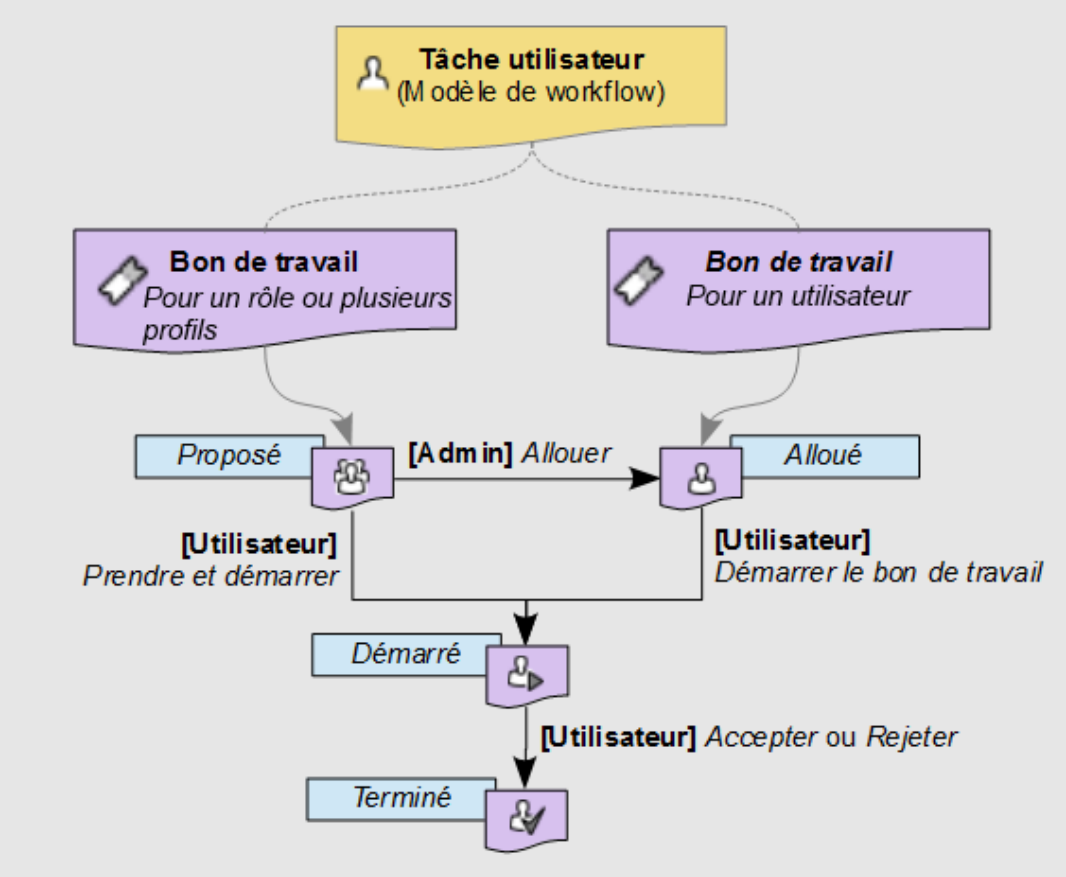
Lorsque le bon de travail est à l'état *proposé*, tout utilisateur ou tout membre des rôles auxquels il est proposé peut prendre le bon de travail en utilisant l'action 'Prendre et démarrer'. Le bon de travail passe ainsi à l'état *démarré*.

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*.

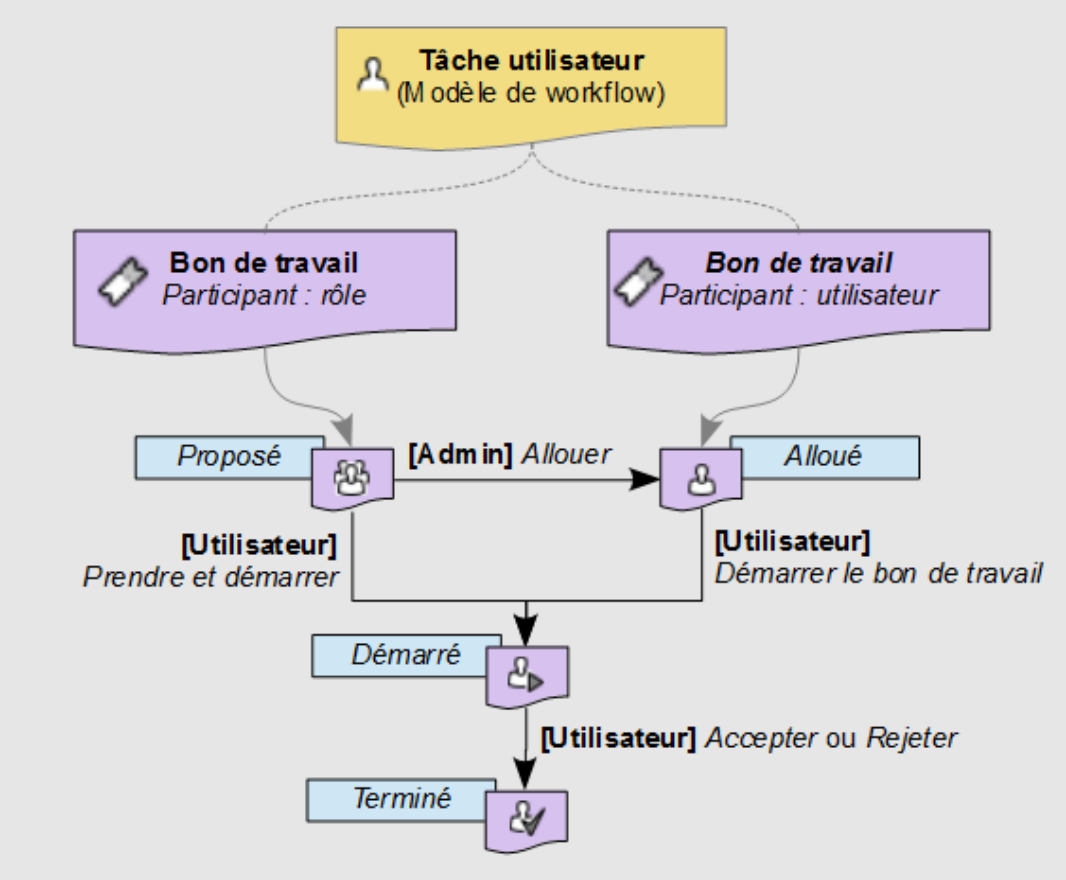
Une fois que l'utilisateur, qui a démarré le bon de travail, a réalisé l'action demandée, l'action terminale 'Accepter' ou 'Rejeter' place le bon de travail dans l'état *terminé*. Lorsqu'un utilisateur termine un bon de travail, le workflow de données passe automatiquement à l'étape suivante définie dans le modèle de workflow.

Diagramme des états de bon de travail

Mode par défaut



Mode de compatibilité



30.2 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.

30.3 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 30]*

Concepts apparentés *Tâche utilisateur* *[p 154]*

CHAPITRE **31**

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

31.1 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.

31.2 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.

31.3 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 *proposé*.
Désallouer	Remettre un bon de travail qui est actuellement dans l'état *alloué* dans l'état *proposé*.
Réallouer	Modifier l'utilisateur à qui le bon de travail est alloué. Cette action est disponible pour les bons de travail dans l'état *alloué*.

Voir aussi

Bons de travail *[p 185]*

Permissions sur les workflows de données associés *[p 169]*

Concepts apparentés *Modèles de workflow* *[p 148]*

CHAPITRE 32

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

32.1 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 pas une action 'Accepter' ou 'Rejeter', réalisée explicitement par l'utilisateur. La fin de la tâche utilisateur chapeau est déterminée en fonction du critère de fin de tâche défini pour la tâche utilisateur dans le modèle de workflow.
- une condition, qui est évaluée automatiquement afin de déterminer l'étape suivante de l'exécution du workflow de données.
- 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 executer :** 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 435]*

32.2 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.

Dépublication d'une publication de workflow

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

1. Désactivez la publication de workflow afin d'éviter que des utilisateurs continuent de lancer des nouveaux workflows de données sur cette publication. Pour cela, suivez le processus décrit dans la section Désactivation d'une publication de workflow [p 194].

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 repositonné au debut 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 435]*

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 169]*

Services de données

CHAPITRE 33

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

33.1 Présentation

Fonction du service de données

Un service de données [p 31] 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 622]

REST [p 676]

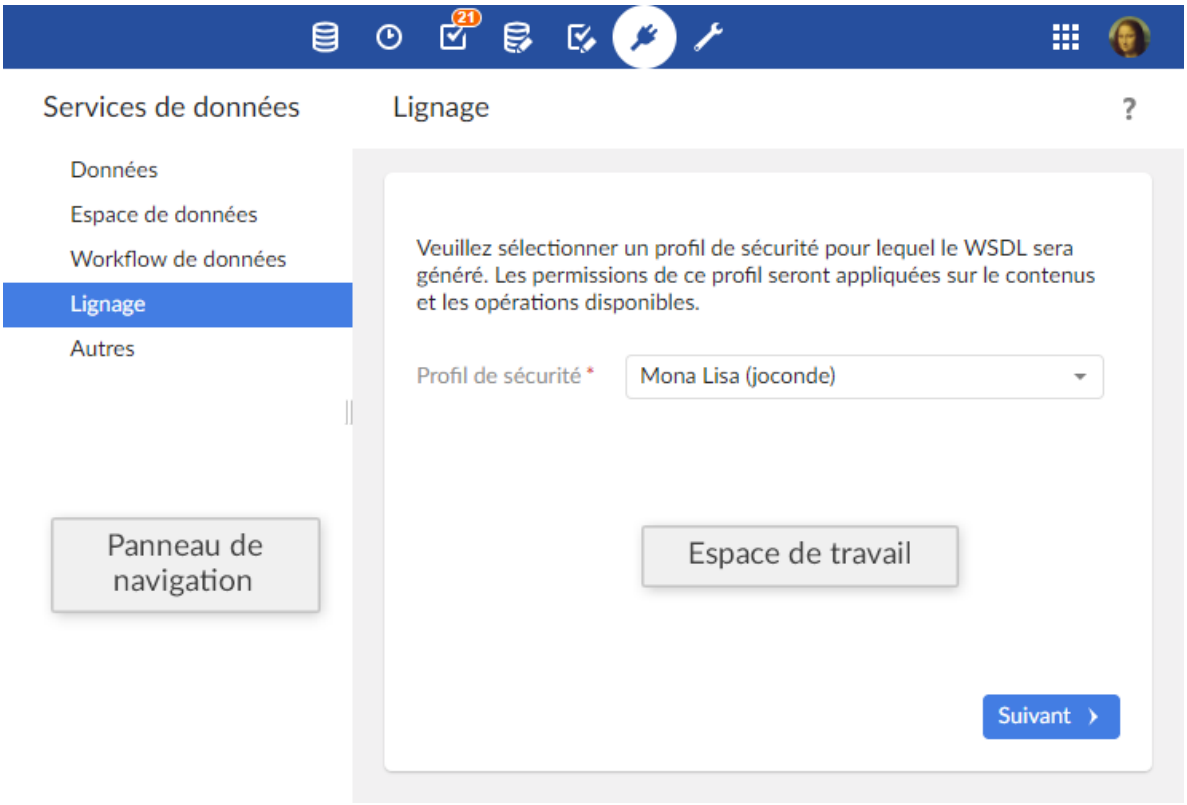
Lignage

Le lignage [p 32] 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 31]*

33.2 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 98]*

Jeux de données *[p 120]*

Workflows de données *[p 176]*

Introduction to data services *[p 622]*

CHAPITRE **34**

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

34.1 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 635]

Operations generated from a data model *[p 641]*

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 635]*

Operations generated from a data model *[p 641]*

34.2 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 635]*

Operations on datasets and dataspaces *[p 662]*

34.3 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 635]

Operations on data workflows [p 668]

34.4 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 201].

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 198]*

34.5 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 635]*

User interface operations *[p 672]*

System information operation *[p 672]*

34.6 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 201].

Voir aussi

WSDL download from HTTP protocol *[p 635]*

Directory services *[p 671]*

Manuel de référence

Intégration

CHAPITRE 35

Présentation de l'intégration et des déploiements

Il existe différents services et composants API permettant de développer des extensions personnalisées pour TIBCO EBX® et de les intégrer à d'autres systèmes.

Ce chapitre contient les sections suivantes :

1. Utiliser EBX® comme composant web
2. Personnalisation de l'interface utilisateur
3. Services de données
4. Services d'import/export XML et CSV
5. Services programmatiques

35.1 Utiliser EBX® comme composant web

Il est possible d'utiliser EBX® comme composant web d'une interface utilisateur, en lançant une requête HTTP. Ces composants web peuvent être intégrés à n'importe quelle application accessible via un navigateur web compatible [p 336].

Les écrans EBX® sont généralement intégrés à la structure intranet de l'organisation. Les composants web peuvent aussi être lancés à partir de l'interface utilisateur d'EBX® dans les Services utilisateurs [p 207].

Voir aussi *Utiliser EBX® comme composant web [p 211]*

35.2 Personnalisation de l'interface utilisateur

Services utilisateur (user services)

Un service utilisateur est une extension d'EBX® qui fournit une interface homme-machine (IHM) permettant aux utilisateurs d'accéder à des fonctionnalités spécifiques ou avancées.

Voir aussi *Vue d'ensemble des services utilisateurs [p 589]*

Mise en page personnalisée

Une couche de présentation permet de surcharger la mise en page par défaut des formulaires dans l'interface utilisateur.

Voir aussi *Form layout [p 587]*

Widgets personnalisés

Un widget personnalisé (custom widget) est un composant graphique développé spécifiquement pour personnaliser l'affichage de groupes ou de champs d'un modèle de données ou d'un schéma programmatique.

Voir aussi *Custom widgets [p 587]*

Ajax

EBX® supporte Ajax pour les échanges de données asynchrones avec le serveur sans avoir à rafraîchir la page courante.

Voir aussi

User service Ajax callbacks [p 586]

Ajax component `UIAjaxComponent`[API]

Spécifier les filtres UI sur une table

En complément des filtres par défaut et des panneaux de recherche de l'interface utilisateur, il est possible de définir des filtres additionnels en fonction de la structure de la table. Pour cela, cette classe spécifique doit être précisée lors de la définition de la table et doit étendre `UITableFilter`.

Voir `UITableFilter`[API] pour plus d'informations.

35.3 Services de données

Le module 'Services de données' permet aux systèmes externes d'interagir avec EBX® en utilisant, l'une des interfaces suivantes:

- Web Services Description Language (WSDL)
- Representational state transfer (REST)

Voir aussi

WSDL/SOAP [p 622]

REST [p 676]

35.4 Services d'import/export XML et CSV

EBX® contient des services intégrés pour l'import et l'export de données aux formats XML et CSV. Les imports et les exports en XML et CSV peuvent être effectués via l'interface utilisateur, les services de données, ou l'API Java.

Voir aussi

Import et export XML *[p 235]*

Import et export CSV *[p 241]*

35.5 Services programmatiques

Les services programmatiques permettent d'exécuter des procédures dans un contexte précis, par exemple dans une tâche planifiée ou un traitement par lot.

Quelques exemples de services programmatiques :

- Import de données à partir d'une source externe,
- Export de données vers des systèmes multiples,
- Historisation des données, lancée par un système de supervision,
- Optimisation et remaniement des données si les **services intégrés d'optimisation** `AdaptationTreeOptimizerSpec`[API] d'EBX® ne suffisent pas.

Voir aussi *`ProgrammaticService`*[API]

CHAPITRE **36**

Utiliser TIBCO EBX® comme composant web

Ce chapitre contient les sections suivantes :

1. Présentation
2. Intégration des composants web d'EBX® à des applications
3. Sélection de l'élément de référentiel et du périmètre
4. Sélection combinée
5. Caractéristiques de la requête
6. Exemples d'appels à un composant web EBX®

36.1 Présentation

EBX® peut être utilisé comme composant web d'une interface utilisateur, appelé par protocole HTTP. Un composant web EBX® peut être intégré à n'importe quelle application accessible via un navigateur web compatible. Ce mode d'accès permet de profiter des fonctionnalités majeures d'EBX®, telles que l'authentification utilisateur, la validation des données, ainsi que la génération automatique de l'interface utilisateur, tout en permettant également d'axer la navigation utilisateur sur certains éléments du référentiel.

Les composants web d'EBX® sont généralement intégrés à la structure intranet de l'organisation ou aux applications qui gèrent l'attribution de tâches spécifiques aux utilisateurs.

Voir aussi *Navigateurs web compatibles* *[p 336]*

36.2 Intégration des composants web d'EBX® à des applications

Une application web qui appelle un composant web d'EBX® peut être :

1. Une application non-Java, le cas le plus basique étant une page HTML statique.

   Dans ce cas, l'application doit envoyer une requête HTTP qui respecte les spécifications de requête [p 213] du composant web EBX®.
2. Une application Java, par exemple :

- Une application web Java exécutée sur la même instance de serveur d'application que le référentiel EBX® qu'il référence ou sur une instance de serveur d'application différente.
- Un Service utilisateur [p 207] ou un Widget personnalisé [p 208] EBX®, auquel cas la nouvelle session héritera automatiquement de la session parent d'EBX®.

> **Note**
>
> En Java, la méthode recommandée pour établir des requêtes HTTP qui appellent les composants web d'EBX® est la classe `UIHttpManagerComponent`[API] dans l'API.

36.3 Sélection de l'élément de référentiel et du périmètre

Quand un composant web d'EBX® est appelé, l'utilisateur doit d'abord être authentifié dans la session HTTP nouvellement instanciée. Le composant web sélectionne alors un élément du référentiel et l'affiche conformément au paramètre de mise en page du `scope` défini dans la requête.

Le paramètre `firstCallDisplay` peut changer cet affichage automatique en fonction de sa valeur.

Les éléments du référentiel qui peuvent être sélectionnés sont les suivants :

- Espace de données ou image
- Jeu de données
- Noeud
- Table ou vue publiée
- Enregistrement de table

Le périmètre, défini par le paramètre 'scope', détermine quelle partie de l'interface est affichée à l'utilisateur et définit les limites de navigation de l'utilisateur pendant la session. Le périmètre par défaut utilisé par le composant web est le plus petit possible dépendant de l'entité ou du service sélectionnés ou invoqués par la requête.

Voir aussi *scope [p 217]*

Voir aussi *firstCallDisplay [p 217]*

Il est également possible de sélectionner une perspective particulière ainsi qu'une action de perspective.

Par défaut, la sélection de l'élément est faite dans le cadre de la perspective de l'utilisateur si le périmètre de l'affichage ("scope") est complet ("full").

Voir aussi *Perspective [p 17]*

36.4 Sélection combinée

Une URL de composant Web peut spécifier une perspective ainsi qu'une action ou une entité (espace de données, jeu de données, etc). Ainsi, pour un composant Web ayant dans son URL une perspective et une entité (mais pas d'action), si une action de la perspective correspond à cette entité, alors cette action est automatiquement sélectionnée.

Sinon, si aucune action ne correspond à cette entité, aucune action n'est sélectionnée mais l'entité est ouverte.

Si une action est spécifié en même temps qu'une entité, cette dernière est ignorée et l'action sera sélectionnée.

Cas particulier

Si l'entité cible est un enregistrement et qu'une action correspond à la table de l'enregistrement, alors cette action est sélectionnée et l'enregistrement sera ouvert à l'intérieur de l'action.

De même, si un bon de travail est ciblé par le composant Web, et si une action de type « inbox » existe dans la perspective, alors cette action sera sélectionnée et le bon de travail sera ouvert à l'intérieur de l'action.

Limitations connues

Si le composant Web spécifie un prédicat pour filtrer une table, il faut que l'action de perspective spécifie exactement le même prédicat pour pouvoir être sélectionnée.

De même, si l'action spécifie un prédicat de filtrage sur une table, le composant Web doit spécifier exactement le même prédicat pour que la correspondance soit établie.

36.5 **Caractéristiques de la requête**

URL de base

Dans un déploiement par défaut, l'URL de base doit avoir la forme suivante :

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

> **Note**
>
> L'URL de base doit faire référence au servlet `FrontServlet`, défini dans le descripteur de déploiement `/WEB-INF/web.xml` de l'application web `ebx.war`.

Authentification de l'utilisateur et paramètres d'informations de la session

Paramètre	Description	Requis
`login` et `password`, ou un *jeton utilisateur lié au répertoire*	Définit les propriétés d'authentification de l'utilisateur. Si le login/mot de passe ou le jeton utilisateur lié au répertoire ne sont pas fournis, l'utilisateur sera invité à s'authentifier via la page de connexion du référentiel. Voir `Directory`[API] pour plus d'informations.	Non
`trackingInfo`	Définit les informations de suivi de la nouvelle session. Les informations de suivi sont recueillies dans les tables de l'historique. Parallèlement, ces informations peuvent être utilisées pour restreindre programmatiquement les autorisations d'accès. Voir `AccessRule`[API] pour plus d'informations.	Non
`redirect`	L'URL vers laquelle l'utilisateur sera redirigé à la fin de la session du composant, lorsqu'il clique sur le bouton 'Fermer'. Le bouton 'Fermer' est systématiquement affiché lors de la sélection d'enregistrements, cependant il faut toujours spécifier via le paramètre `closeButton` si l'affichage doit être maintenu dans les autres cas. Pour plus d'informations, voir Exit policy [p 413].	Non
`locale`	Spécifie la locale à utiliser. La valeur est soit `en-US` soit `fr-FR`.	Non, la locale par défaut est celle enregistrée pour l'utilisateur.

Entité et paramètres de sélection du service

Paramètre	Description	Requis
`branch`	Sélectionne l'espace de données spécifié.	Non
`version`	Sélectionne l'image d'espace de donnée spécifiée.	Non
`instance`	Sélectionne le jeu de données spécifié. La valeur doit être la référence d'un jeu de données qui existe dans l'espace de données ou dans l'image sélectionnés.	Seulement si `xpath` ou `viewPublication` sont spécifiés.
`viewPublication`	Définit le nom de publication de la vue tabulaire ou hiérarchique à appliquer au contenu choisi. Ce nom de publication est celui déclaré lors de la publication de la vue. Il se trouve dans l'espace 'Administration' sous *Configuration des vues > Vues*. Tous les paramètres de la vue, tels que ses filtres, l'ordre de tri et les colonnes affichées, sont appliqués au résultat. Un espace de données et un jeu de données doivent être sélectionnés afin que cette vue s'applique. Le choix d'une table cible n'est pas nécessaire, puisqu'elle peut être automatiquement déterminée sur la base de la définition de la vue. Ce paramètre peut être associé au prédicat spécifié dans le paramètre `xpath` en tant qu'opération logique 'AND'.	Non
`xpath`	Définit la sélection d'un noeud dans le jeu de données. La valeur peut être un chemin absolu valide situé dans le jeu de données sélectionné. L'écriture doit être conforme à un XPath simplifié, avec une syntaxe abrégée. Il peut aussi s'agir d'un prédicat entouré par "[" and "]" si une table peut être automatiquement sélectionnée à l'aide d'autres paramètres de composants web (par exemple, `viewPublication` ou `workflowView`). Voir aussi XPath supported syntax [p 249] pour la syntaxe XPath Voir `UIHttpManagerComponent.setPredicate`[API] pour plus d'informations.	Non
`service`	Définit le service auquel accéder. Pour plus d'informations sur les services utilisateurs intégrés, voir Services built-in [p 221]. Dans l'API Java, voir `ServiceKey`[API] pour plus d'informations.	Non
`workflowView`	Définit la section du workflow à sélectionner. Voir `WorkflowView`[API] pour plus d'informations.	Non
`perspectiveName`	Définit le nom de la perspective à sélectionner. Si ce paramètre est défini, le paramètre `scope` ne pourra prendre que deux valeurs : full et data.	Seulement si `perspectiveActionId` ou `perspectiveActionName` est spécifié.
`perspectiveActionId`	**Déprécié.** Merci de considérer l'utilisation de `perspectiveActionName` à la place.	Non

Paramètre	Description	Requis
	Définit l'identifiant de l'action de perspective à sélectionner.	
`perspectiveActionName`	Définit le nom unique de l'action de perspective à sélectionner.	Non

Paramètres de mise en page

Paramètre	Description	Requis
`scope`	Définit le périmètre utilisé par le composant web. Il peut prendre la valeur `full`, `data`, `dataspace`, `dataset` ou `node`. Voir `UIHttpManagerComponent.Scope`[API] pour plus d'informations.	Non, la valeur par défaut sera calculée pour être la plus petite possible en fonction de la sélection cible.

`firstCallDisplay`	Définit l'affichage qui doit être utilisé à la place de celui déterminé par la combinaison de la sélection et du paramètre `scope`.
	Les valeurs possibles sont :
	• `auto` : L'affichage est automatiquement positionné en fonction de la sélection.
	• `view` : Force l'affichage de la vue tabulaire ou de la vue hiérarchique.
	• `record` : Si le prédicat cible au moins un enregistrement, force l'affichage du premier enregistrement de la liste.
	Par exemple,
	`firstCallDisplay=view`
	`firstCallDisplay=view:hierarchyExpanded`
	`firstCallDisplay=record`
	`firstCallDisplay=record:{predicate}`
	Voir `UIHttpManagerComponent.setFirstCallDisplay`[API] pour plus d'informations.
	Voir `UIHttpManagerComponent.setFirstCallDisplayHierarchyExpanded`[API] pour plus d'informations.
	Voir `UIHttpManagerComponent.setFirstCallDisplayRecord`[API] pour plus d'informations.
		Non, la valeur par défaut sera calculée en fonction de la sélection cible.

`closeButton`	Définit la manière dont le bouton de fin de session sera affiché. La valeur peut être `logout` ou `cross`. Voir `UIHttpManagerComponent.CloseButtonSpec`[API] pour plus d'informations.	Non. Si le périmètre n'est pas `full`, aucun bouton de fin de session ne sera affiché par défaut.

`dataSetFeatures`	Définit les fonctionnalités à afficher dans un service utilisateur au niveau du jeu de données ou d'un formulaire en dehors d'une table.
	Ces options se rapportent uniquement aux fonctionnalités au sein de l'espace de travail. Il est recommandé d'utiliser cette propriété avec le plus petit `scope` possible, c'est à dire `dataset` ou `node`.
	Syntaxe :
	`<`*`prefixe`*`> ":" <`*`feature`*`> [ "," <`*`feature`*`>]*`
	où
	• `<`*`prefixe`*`>` est `hide` ou `show`,
	• `<`*`feature`*`>` est `services`, `title`, `save`, ou `revert`.
	Par exemple,
	`hide:title`
	`show:save,revert`
	Voir `UIHttpManagerComponent.DataSetFeatures`[API] pour plus d'informations.
		Non.

Paramètre	Description	Requis

`viewFeatures`	Définit les fonctionnalités à afficher dans une vue tabulaire ou hiérarchique (au niveau de la table).
	Ces options se rapportent uniquement aux fonctionnalités au sein de l'espace de travail. Il est recommandé d'utiliser cette propriété avec le plus petit `scope` possible, c'est à dire `dataset` ou `node`.
	Syntaxe :
	`<prefixe> ":" <feature> [ "," <feature>]*`
	où
	• `<prefixe>` est `hide` ou `show`,
	• `<feature>` est `create`, `views`, `selection`, `filters`, `services`, `refresh`, `title`, ou `breadcrumb`.
	Par exemple,
	`hide:title,selection`
	`show:service,title,breadcrumb`
	Voir `UIHttpManagerComponent.ViewFeatures`[API] pour plus d'informations.
		Non.

`recordFeatures`	Définit les fonctionnalités à afficher dans un formulaire au niveau de l'enregistrement.
	Ces options se rapportent uniquement aux fonctionnalités au sein de l'espace de travail. Il est recommandé d'utiliser cette propriété avec le plus petit `scope` possible, c'est à dire `dataset` ou `node`.
	Syntaxe:
	`<prefixe> ":" <feature> [ "," <feature>]*`
	où
	• `<prefixe>` est `hide` ou `show`,
	• `<feature>` est `services`, `title`, `breadcrumb`, `save`, `saveAndClose`, `close`, ou `revert`.
	Par exemple,
	`hide:title`
	`show:save,saveAndClose,revert`
	Voir `UIHttpManagerComponent.RecordFeatures`[API] pour plus d'informations.
		Non.

`pageSize`	Précise le nombre d'enregistrements affichés par page dans une vue table (qu'elle soit tabulaire ou hiérarchique).	Non.
`startWorkItem`	Précise qu'un élément de travail doit être automatiquement pris et démarré. La valeur peut être `true` ou `false`. Voir `ServiceKey.WORKFLOW`[API] pour plus d'informations.	Non. La valeur par défaut est `false`, l'état du bon de travail cible restant inchangé.

36.6 Exemples d'appels à un composant web EBX®

URI minimale :

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

Se connecte en tant qu"admin' et sélectionne l'espace de données 'Référence' :

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

Sélectionne l'espace de données 'Référence' et accède au service de validation intégré :

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

Sélectionne la table des rôles dans le répertoire par défaut :

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

Sélectionne l'enregistrement 'admin' dans le répertoire par défaut :

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

Accède à l'interface pour créer un nouvel utilisateur dans le répertoire par défaut :

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

Compare l'enregistrement 'admin' du répertoire par défaut avec l'enregistrement 'jSmith' :

Compare l'enregistrement 'R1' du jeu de données 'instanceId' dans l'espace de données 'Reference' avec l'enregistrement 'R0' :

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

CHAPITRE 37

Services UI prédéfinis

EBX® inclut des services UI prédéfinis. Ces services peuvent être utilisés :

- pour définir une tâche utilisateur [p 154]
- pour définir une action dans le menu d'une perspective [p 18]
- comme base pour définir un service UI étendu [p 613]
- pour utiliser EBX® en mode composant Web [p 211]

Cette page référence tous les services UI prédéfinis et leurs paramètres.

Ce chapitre contient les sections suivantes :

1. Accéder à des données (service par défaut)
2. Créer un nouvel enregistrement
3. Dupliquer un enregistrement
4. Exporter les données depuis une table au format XML
5. Exporter les données depuis une table au format CSV
6. Importer les données dans une table depuis un fichier XML
7. Importer les données dans une table depuis un fichier CSV
8. Accéder à un espace de données
9. Valider un espace de données, une image ou un jeu de données
10. Fusionner un espace de données
11. Accéder à l'interface de fusion des espace de données
12. Comparer deux contenus
13. Workflows de données

37.1 Accéder à des données (service par défaut)

Par défaut, un workflow considère automatiquement ce service comme terminé. Le bouton "Accepter" est donc toujours disponible.

Ce service est utilisé par défaut si aucun service n'est spécifié.

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Un espace de données est obligatoire pour ce service.
`disableAutoComplete`	Désactiver Accepter au démarrage	Par défaut, l'interaction associée à ce service est directement considérée comme terminée. Par conséquent, le bouton Accepter est automatiquement affiché à l'ouverture du bon de travail. Ce paramètre est utile pour désactiver ce comportement. Si la valeur est "true", le développeur a la charge de terminer l'interaction en utilisant SessionInteraction dans un service UI ou un trigger, par exemple. La valeur par défaut est "false". Les perspectives n'utilisent pas ce paramètre.
`firstCallDisplay`	Mode d'affichage du premier appel	Définit le mode d'affichage qui doit être utilisé lorsqu'une table filtrée ou un enregistrement est affiché lors du premier appel. Défaut (valeur = 'auto') : l'affichage est défini automatiquement en fonction de la sélection. Vue (valeur = 'view') : force l'affichage de la vue tabulaire ou hiérarchique. Enregistrement (valeur = 'record') : si le prédicat a au moins un enregistrement, force l'affichage du formulaire de l'enregistrement.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`version`	Image	Identifiant de l'image concernée - Un espace de données est obligatoire pour ce service.

Paramètre	Libellé	Description
`viewPublication`	Vue	Le nom de publication de la vue à afficher. La vue doit être configurée pour la table sélectionnée.
`xpath`	Noeud du jeu de données (XPath)	La valeur saisie doit être un chemin absolu d'un noeud dans le jeu de données saisi. La notation doit être conforme à la convention XPath, sous sa forme abrégée.

37.2 Créer un nouvel enregistrement

Pour un workflow, le service creation est considéré terminé après la première soumission sans erreur (enregistrement créé). Si ce service est appelé alors qu'il est déjà terminé, l'enregistrement créé est affiché en mode modification ou lecture seule (selon les droits de l'utilisateur).

Paramètre d'appel du service : `service=@creation`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Ce champ est obligatoire pour ce service.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi - Ce champ est obligatoire pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`xpath`	Table du jeu de données (XPath)	La valeur saisie doit être un chemin absolu d'une table dans le jeu de données saisi. La notation doit être conforme à la convention XPath, sous sa forme abrégée - Ce champ est obligatoire pour ce service.

Paramètres en sortie

Paramètre	Libellé	Description
`created`	Enregistrement créé	Contient le xPath de l'enregistrement créé.

37.3 Dupliquer un enregistrement

Pour un workflow, le service duplicate est considéré terminé après la première soumission sans erreur (enregistrement créé). Si ce service est appelé alors qu'il est déjà terminé, l'enregistrement créé est affiché en mode modification ou lecture seule (selon les droits de l'utilisateur).

Paramètre d'appel du service : `service=@duplicate`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Ce champ est obligatoire pour ce service.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi - Ce champ est obligatoire pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`xpath`	Enregistrement à dupliquer (XPath)	La valeur saisie doit être un chemin absolu d'un enregistrement existant. La notation doit être conforme à la convention XPath, sous sa forme abrégée - Ce champ est obligatoire pour ce service.

Paramètres en sortie

Paramètre	Libellé	Description
`created`	Enregistrement créé	Contient le xPath de l'enregistrement créé.

37.4 Exporter les données depuis une table au format XML

Le service exportToXML est considéré terminé une fois l'export terminé et le fichier téléchargé.

Paramètre d'appel du service : `service=@exportToXML`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Un espace de données est obligatoire pour ce service.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi - Ce champ doit être renseigné pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`version`	Image	Identifiant de l'image concernée - Un espace de données est obligatoire pour ce service.
`xpath`	Table du jeu de données à exporter (XPath)	La valeur saisie doit être un chemin absolu d'une table dans le jeu de données saisi. La notation doit être conforme à la convention XPath, sous sa forme abrégée - Ce champ est obligatoire pour ce service.

37.5 Exporter les données depuis une table au format CSV

Un workflow considère automatiquement le service exportToCSV comme terminé une fois l'export achevé et le fichier téléchargé.

Paramètre d'appel du service : `service=@exportToCSV`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Un espace de données est obligatoire pour ce service.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi - Ce champ doit être renseigné pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`version`	Image	Identifiant de l'image concernée - Un espace de données est obligatoire pour ce service.
`xpath`	Table du jeu de données à exporter (XPath)	La valeur saisie doit être un chemin absolu d'une table dans le jeu de données saisi. La notation doit être conforme à la convention XPath, sous sa forme abrégée - Ce champ est obligatoire pour ce service.

37.6 Importer les données dans une table depuis un fichier XML

Un workflow considère le service importFromXML comme terminé une fois l'import réussi.

Paramètre d'appel du service : `service=@importFromXML`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Un espace de données est obligatoire pour ce service.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi - Ce champ doit être renseigné pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`xpath`	Table du jeu de données dans laquelle importer (XPath)	La valeur saisie doit être un chemin absolu d'une table dans le jeu de données saisi. La notation doit être conforme à la convention XPath, sous sa forme abrégée - Ce champ est obligatoire pour ce service.

37.7 **Importer les données dans une table depuis un fichier CSV**

Un workflow considère le service importFromCSV comme terminé une fois l'import réussi.

Paramètre d'appel du service : `service=@importFromCSV`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Un espace de données est obligatoire pour ce service.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi - Ce champ doit être renseigné pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`xpath`	Table du jeu de données dans laquelle importer (XPath)	La valeur saisie doit être un chemin absolu d'une table dans le jeu de données saisi. La notation doit être conforme à la convention XPath, sous sa forme abrégée - Ce champ est obligatoire pour ce service.

37.8 **Accéder à un espace de données**

Un workflow considère automatiquement le service de sélection d'espace de données comme terminé.

Paramètre d'appel du service : `service=@selectDataSpace`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Un espace de données est obligatoire pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`version`	Image	Identifiant de l'image concernée - Un espace de données est obligatoire pour ce service.

37.9 **Valider un espace de données, une image ou un jeu de données**

Un workflow considère automatiquement le service validation comme terminé.

Paramètre d'appel du service : `service=@validation`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Un espace de données ou une image est obligatoire pour ce service.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`version`	Image	Identifiant de l'image concernée - Un espace de données ou une image est obligatoire pour ce service.

Paramètres en sortie

Paramètre	Libellé	Description
`hasError`	Erreurs trouvées	Contient 'true' si la validation a généré des erreurs.
`hasFatal`	Erreurs fatales trouvées	Contient 'true' si la validation a généré des erreurs fatales.
`hasInfo`	Informations trouvées	Contient 'true' si la validation a généré des informations.
`hasWarning`	Avertissements trouvées	Contient 'true' si la validation a généré des avertissements.

37.10 **Fusionner un espace de données**

Un workflow considère le service merge comme terminé une fois la fusion effectuée et l'espace de données fermé.

Paramètre d'appel du service : `service=@merge`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Ce champ est obligatoire pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.

Paramètres en sortie

Paramètre	Libellé	Description
`mergeResult`	Réussite de la fusion	Contient 'true' si la fusion a réussi, 'false' sinon.
`mergeState`	État de la fusion	Contient le code de retour de la fusion. Il est fortement recommandé de parser cette valeur en utilisant le classe UIHttpManagerComponentReturnCode.

37.11 **Accéder à l'interface de fusion des espace de données**

Le service merge.view est automatiquement considéré comme terminé.

Paramètre d'appel du service : `service=@merge.view`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Ce champ est obligatoire pour ce service.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.

37.12 **Comparer deux contenus**

Un workflow considère automatiquement le service compare comme terminé.

Paramètre d'appel du service : `service=@compare`

Paramètres en entrée

Paramètre	Libellé	Description
`branch`	Espace de données	Identifiant de l'espace de données concerné - Un espace de données ou une image, et un espace de données ou une image à comparer sont obligatoires pour ce service.
`compare.branch`	Espace de données à comparer	Identifiant de l'espace de données à comparer - Un espace de données et un espace de données à comparer sont obligatoires pour ce service.
`compare.filter`	Filtre de comparaison	Pour ignorer l'héritage et les champs de valeurs calculées lors de la comparaison (désactiver le mode résolu), le filtre "persistedValuesOnly" doit être spécifié. Par défaut, lorsqu'aucun filtre n'est défini, la comparaison utilise le mode résolu.
`compare.instance`	Jeu de données à comparer	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données à comparer saisi.
`compare.version`	Image à comparer	Identifiant de l'image à comparer - Un espace de données et un espace de données à comparer sont obligatoires pour ce service.
`compare.xpath`	Table ou enregistrement à comparer (XPath)	La valeur saisie doit être un chemin absolu d'une table ou d'un enregistrement dans le jeu de données à comparer saisi. La notation doit être conforme à la convention XPath, sous sa forme abrégée.
`instance`	Jeu de données	La valeur saisie doit être une référence vers un jeu de données qui existe dans l'espace de données saisi.
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service, c'est-à-dire les entités que l'utilisateur peut sélectionner dans sa session. Si non renseignée, la valeur utilisée sera celle par défaut. Pour les perspectives, la valeur par défaut est toujours 'node'. Pour les workflows, la valeur par défaut dépend des entités ou du service sélectionnés.
`trackingInfo`	Informations de suivi	Les informations de suivi sont enregistrées dans les logs 'historique'. Ces informations peuvent également être utilisées dans d'autres buts comme le contrôle d'accès ou une mise à disposition d'informations supplémentaires.
`version`	Image	Identifiant de l'image concernée - Un espace de données ou une image, et

Paramètre	Libellé	Description
		un espace de données ou une image à comparer sont obligatoires pour ce service.
`xpath`	Table ou enregistrement (XPath)	La valeur saisie doit être un chemin absolu d'une table ou d'un enregistrement dans le jeu de données saisi. La notation doit être conforme à la convention XPath, sous sa forme abrégée.

37.13 **Workflows de données**

Ce service fournit un accès aux interfaces utilisateurs des workflows de données.

Paramètre d'appel du service : `service=@workflow`

> **Note**
>
> Ce service est destiné aux perspectives uniquement.

Paramètres en entrée

Paramètre	Libellé	Description
`scope`	Périmètre	Définit le périmètre de la navigation utilisateur pour ce service.
`viewPublication`	Publication de vue	Définit le nom de publication de la vue à appliquer pour ce service.
`workflowView`	Type d'affichage	Spécifie le type d'affichage du workflow. La valeur peut être une des suivantes: "inbox" (boîte de réception), "launcher" (lanceurs de workflows), "monitoringPublications" (publications), "monitoringWorkflows" (workflows actifs), "monitoringWorkItems" (bons de travail) ou "completedWorkflows" (workflows terminés).
`xpath`	Filtre (XPath)	Un paramètre optionnel permettant de filtrer les données affichées. La syntaxe doit respecter celle d'un prédicat XPath entouré par "[" et "]".

CHAPITRE **38**

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

38.1 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*.

38.2 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 649] et insertion [p 651], ainsi que `ImportSpec.setByDelta`[API] 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 `null`. • 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 `null` d'une chaîne de caractères vide, la valeur du champ cible prend la valeur `null`. 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 566] **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 `update`, la valeur de champ reste inchangée. • Pour l'opération `insert`, le comportement est le même que lorsque le mode `byDelta` est désactivé.
L'élément existe tout en étant vide (par exemple, `<fieldA/>`)	• Pour des nœuds de type `xs:string` (ou un de ses sous-types), le champ cible prend la valeur `null` s'il distingue `null` d'une chaîne vide. Autrement, la valeur est une chaîne vide. • Pour les types de nœuds non`xs:string`, une exception est lancée conformément au XML Schema. Voir aussi *TIBCO EBX® whitespace management for data types* [p 551]
L'élément est présent et de valeur `null` (par exemple, `<fieldA xsi:nil="true"/>`)	Le champ cible prend toujours la valeur `null` sauf dans le cas des listes, pour lesquelles il n'est pas supporté. Afin d'utiliser l'attribut `xsi:nil="true"`, il est nécessaire d'ajouter la déclaration du namespace `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`.

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`, i l 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.

38.3 **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.

38.4 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

38.5 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 39

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

39.1 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 417]*

39.2 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 27] lors d'un export CSV, l'option doit être spécifiée au préalable dans le modèle. Pour plus d'informations sur l'héritage, voir Héritage et résolution de valeur [p 296]. 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 `ExportSpec`[API] et `ExportImportCSVSpec`[API] dans l'API Java.

39.3 **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 *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*.
Activer l'héritage	Afin de prendre en compte l'héritage [p 27] lors d'un export CSV, l'option doit être spécifiée au préalable dans le modèle. Pour plus d'informations sur l'héritage, voir Héritage et résolution de valeur [p 296] et `ExportImportCSVSpec.setInheritanceEnabled`[API]. 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 `ImportSpec`[API] et `ExportImportCSVSpec`[API] dans l'API Java.

39.4 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.setNullStringEncoded`[API] 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

39.5 **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 235].

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 40

Syntaxe XPath supportée

Ce chapitre contient les sections suivantes :

1. Vue d'ensemble
2. Exemple d'expressions
3. Spécifications de syntaxe pour les expressions XPath
4. API Java

40.1 Vue d'ensemble

L'écriture XPath utilisée dans TIBCO EBX® doit être conforme à la *syntaxe abrégée* du XML Path Language (XPath) Version 1.0 standard, avec certaines restrictions. Ce document décrit la syntaxe abrégée supportée.

40.2 Exemple d'expressions

L'expression générale XPath est :

```
path[predicate]
```

Chemin absolu

```
/bibliotheque/livres/
```

Chemins relatifs

```
./Auteur
../Titre
```

Racine et chemins descendants

```
//livres
```

Chemins de table avec prédicats

```
../../livre/[auteur_id = 0101 and (editeur = 'harmattan')]
/bibliotheque/livres/[not(editeur = 'dumesnil')]
```

Prédicats complexes

```
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') )
```

Prédicats avec paramètres

`author_id = $param1 and publisher = $param2` où les paramètres `$param1` et `$param2` ont respectivement pour valeurs `0101` et `'harmattan'`

`col1 < $param1 and col4 = $param2` où les paramètres `$param1` et `$param2` ont respectivement pour valeurs `100` et `'true'`

`contains(col3,$param1) and date-greater-than(col2,$param2)` où les paramètres `$param1` et `$param2` ont respectivement pour valeurs `'xxx'` et `'2007-12-30'`

> **Note**
>
> L'utilisation de cette notation est uniquement limitée à l'API Java puisque les valeurs des paramètres peuvent uniquement être définies par la méthode `Request.setXPathParameter`[API].

Prédicats sur libellé

```
osd:label(./delivery_date)='12/30/2014' and ends-with(osd:label(../adress),'Beijing - China')
```

Prédicats sur libellé d'un enregistrement

```
osd:contains-record-label('dumesnil') or osd:contains-record-label('harmattan')
```

Prédicats pour la recherche sur validation

```
osd:has-validation-item()
osd:has-validation-item('error,info')
osd:contains-validation-message('xxx')
osd:contains-validation-message('xxx','info,warning')
```

> **Note**

- Les fonctions XPath de validation sont interdites pour les associations et pour les filtres de clé étrangère.
- Les prédicats `osd:label`, `osd:contains-record-label` et `osd:contains-validation-message` sont localisés. La locale peut être définie par l'une des méthodes de l'API Java `Request.setLocale`[API] ou `Request.setSession`[API].

Attention

Afin de garantir que le rapport de validation est à jour lors de la recherche, il est nécessaire de valider explicitement la table juste avant d'utiliser ces prédicats.

40.3 Spécifications de syntaxe pour les expressions XPath

Vue d'ensemble

Expression	Format	Exemple
Expression XPath	<*chemin container*>[*predicate*]	`/livres[titre='xxx']`
<*container path*>	<*chemin absolu*> ou <*chemin relatif*>	
<*absolute path*>	`/a/b` ou `//b`	`//livres`
<*relative path*>	`../../b`, `./b` ou `b`	`../../livres`

Spécification du prédicat

Expression	Format	Notes/Exemple
<*predicate*>	Exemple : A et (B et non(C)) A,B,C : <*expression atomique*>	Composition de : parenthèses d'opérateurs logiques, not() et expressions atomiques.

<*atomic expression*>	<*chemin*><*comparateur*><*critère*> ou méthode(<*chemin*>,<*critère*>)	`royalty = 24.5`
		`starts-with(title, 'Johnat')`
		`booleanValue = true`

<*path*>	<*chemin relatif*> *ou osd:label(*<*chemin relatif*>*)*	Relatif à la table qui le contient : `../authorstitle`
<*comparator*>	<*comparateur booléen*>, <*comparateur numérique*> ou <*comparateur 'string'*>	
<*boolean comparator*>	`=` ou `!=`	
<*numeric comparator*>	`=` , `!=` , `<`, `>`, `<=`, ou `>=`	
<*string comparator*>	`=`	
<*method*>	<*méthode date*>, <*méthode 'string'*>, méthode `osd:is-null` ou méthode `osd:is-not-null`	
<*date, time & dateTime method*>	`date-less-than`, `date-equal` ou `date-greater-than`	
<*string method*>	`matches`, `starts-with`, `ends-with`, `contains`, `osd:is-empty`, `osd:is-not-empty`, `osd:is-empty-or-nil`, `osd:is-neither-empty-nor-nil`, `osd:is-equal-case-insensitive`, `osd:starts-with-case-insensitive`, `osd:ends-with-case-insensitive`, `osd:contains-case-insensitive`, ou `osd:contains-record-label`	
<*criterion*>	<*critère booléen*>, <*critère numérique*>, <*critère 'string'*>, <*critère date*>, <*critère 'time'*>, ou <*critère 'dateTime'*>	
<*boolean criterion*>	`true` , `false`	
<*numeric criterion*>	Un entier ou un décimal	`-4.6`
<*string criterion*>	Chaîne de caractères entre apostrophes	`'azerty'`

Expression	Format	Notes/Exemple
<*date criterion*>	Entre apostrophes et formaté comme suit 'aaaa-MM-jj'	`'2007-12-31'`
<*time criterion*>	Entre apostrophes et formaté comme suit 'HH:mm:ss' ou 'HH:mm:ss.SSS'	`'11:55:00'`
<*dateTime criterion*>	Entre apostrophes et formaté comme suit 'aaaa-MM-jjTHH:mm:ss' ou 'aaaa-MM-jjTHH:mm:ss.SSS'	`'2007-12-31T11:55:00'`

Formule XPath 1.0

Il est possible d'utiliser une formule XPath 1.0 dans la partie valeur du critère d'une expression de prédicat atomique (côté droit).

Par exemple, au lieu de `[./a=3]`, il est possible d'utiliser l'expression `[./a=(floor(./d)+ 2.0)]`.

En raison de la forte dépendance des prédicats et le type du noeud du critère, la portion de chemin de l'expression du prédicat atomique (côté gauche) doit être un chemin de noeud et ne peut être une formule XPath. Par exemple, l'expression `/table[floor(./a) > ceiling(./d)]` n'est pas valide.

Prédicat sur libellé

La fonction `osd:label()` peut être appliquée à la portion de chemin du prédicat atomique, afin de résoudre le prédicat sur le libellé et non sur la valeur. Dans ce cas, seuls les opérateurs et les critères de chaîne peuvent être utilisés, i.e. `ends-with(osd:label(./price),'99')`.

Un prédicat sur libellé est localisé, le critère doit donc être exprimé dans la même locale que la requête filtrée par prédicat. Par exemple : `request.setLocale(Locale.FRENCH); request.setXPathFilter("osd:label(./delivery_date)='30/12/2014'");`

> **Note**
>
> Il est interdit d'utiliser la fonction `osd:label` si la partie droite du prédicat est une valeur contextuelle.

> **Note**
>
> Si la fonction `osd:label` est utilisée dans un modèle de données, par exemple dans une sélection ou dans le prédicat de filtre du noeud de référence d'une table, la locale par défaut du modèle de données (telle que définie dans sa déclaration de module) doit être utilisée pour le format du critère (même si cela n'est généralement pas recommandé).

Voir aussi *`SchemaNode.displayOccurrence`*[API]

Valeurs contextuelles

Pour les prédicats se rapportant à un noeud de sélection, la valeur du critère (à savoir, le coté droit du prédicat) peut être remplacée par un chemin contextuel utilisant la syntaxe `${<relative-path>}` où *<relative-path>* est l'emplacement de l'élément lié au noeud de sélection.

> **Note**
>
> Lors de l'appel à une méthode, le critère est le deuxième paramètre, et le premier paramètre ne peut pas être une valeur relative.

Listes groupées

Pour les prédicats sur listes groupées, le prédicat renvoie `true` indépendamment du comparateur si l'un des éléments de la liste vérifie le prédicat.

> **Note**
>
> Une attention particulière doit être portée au comparateur `!=`. Par exemple, pour une liste groupée, `./list != 'a'` est différent de `not(./list = 'a')`. Là où la liste contient les éléments `(e1,e2,..)`, le premier prédicat équivaut à `e1 != 'a' or e2 != 'a' ...`, alors que le second équivaut à `e1 != 'a' and e2 != 'a' ...`.

Valeurs 'Null'

Les valeurs Null doivent être traitées explicitement dans uns prédicat utilisant les opérateurs `osd:is-null` et `osd:is-not-null`.

Par exemple, `/root/products[./price<100]` ou `/root/products[./price!=100]` ne retournera aucun produit dont les prix ne sont pas définis (`null`). Dans ce cas, pour retourner également les valeurs non définies, le prédicat devrait plutôt être : `/root/products[./price!=100 or osd:is-null(./price)]`.

Comment gérer les apostrophes simples et doubles dans les expressions littérales

Par défaut, une expression littérale est délimitée par des apostrophes simples (`'`). Si l'expression littérale contient des apostrophes simples et pas d'apostrophes doubles, l'expression doit être délimitée par des apostrophes doubles (`"`). Si l'expression littérale contient des apostrophes simples et doubles, les apostrophes simples doivent être doublées.

Géré par la méthode `XPathExpressionHelper.encodeLiteralStringWithDelimiters`[API] dans l'API Java.

Exemples d'utilisation de `encodeLiteralStringWithDelimiters`

Valeur de l'expression littérale	**Résultat de cette méthode**
`Coeur`	`'Coeur'`
`Coeur d'Alene`	`"Coeur d'Alene"`
`Il a dit: "Ils vivent dans Coeur d'Alene".`	`'Il a dit: "Ils vivent dans Coeur d''Alene".'`

Extraction de clés étrangères

Dans EBX®, les clés étrangères sont regroupées dans un champ unique avec la déclaration `osd:tableRef` [p 524].

La syntaxe XPath standard a été étendue afin d'extraire la valeur de n'importe quel champ de clé primaire ciblé.

Exemple

Si la table `/root/tableA` a un champ `osd:tableRef` nommé 'fkB' dont la cible est `/root/tableB` et la clé primaire de `tableB` a deux champs, `id` de type `xs:int` et `date` de type `xs:date`, alors les expressions suivantes sont valides :

- `/root/tableB[ fkB = '123|2008-01-21' ]`, où la chaîne "123|2008-01-21" est une représentation de l'intégralité de la valeur de la clé primaire.

  Voir **Syntaxe de la représentation de la chaîne interne des clés primaires** `PrimaryKey.syntax`[API] pour plus d'informations.
- `/root/tableA[ fkB/id = 123 and date-equal(fkB/date, '2008-01-21') ]`, où ce prédicat est un équivalent plus efficace que celui de l'exemple précédent.
- `/root/tableA[ fkB/id >= 123 ]`, où n'importe quel opérateur numérique pourrait être utilisé, puisque le champ de clé primaire cible est de type `xs:int`.
- `/root/tableA[ date-greater-than( ./fkB/date, '2007-01-01' ) ]`, où n'importe quel opérateur date pourrait être utilisé, puisque le champ de clé primaire cible est de type `xs:date`;
- `/root/tableA[ fkB = "" ]` n'est pas valide puisque la clé primaire cible possède deux colonnes.
- `/root/tableA[ osd:is-null(fkB) ]` vérifie si une clé étrangère est `null` (non définie).

40.4 API Java

Utilisation de XPath dans l'API Java :

En API Java, la classe `XPathFilter` permet de définir des prédicats XPath et d'exécuter des requêtes sur ceux-ci.

La classe `XPathExpressionHelper` offre des méthodes utilitaires pour manipuler les prédicats et les chemins XPath.

Localisation

CHAPITRE 41

Libellés et localisation

Ce chapitre contient les sections suivantes :

1. Présentation
2. Règles de formatage des valeurs
3. Syntaxe des locales

41.1 Présentation

TIBCO EBX® donne la possibilité de gérer les libellés et l'internationalisation des modèles de données.

Localisation de l'interface utilisateur

Dans EBX®, les préférences de langue peuvent êtres définies sur deux périmètres distincts :

1. La session : chaque utilisateur peut sélectionner une locale par défaut à partir du panneau utilisateur.
2. Le fichier principal de configuration d'EBX®, intitulé `ebx.properties` par défaut. Voir Extension de l'internationalisation de TIBCO EBX® [p 261] pour plus d'informations.

Informations textuelles

Dans EBX®, la plupart des entités de données de référence ont un libellé et une description, ou sont liées à un message utilisateur. Par exemple :

- Les espaces de données, les images et les jeux de données peuvent avoir leurs propres libellés et descriptions. Le libellé est indépendant du nom unique, ce qui permet la traduction ainsi que la modification ;
- N'importe quel noeud du modèle de données peut avoir un libellé et une description statiques ;
- Les valeurs énumérées peuvent avoir un libellé statique ;
- Les messages de validation peuvent être personnalisés, et les restrictions de permission peuvent fournir un texte expliquant le motif ;
- Chaque enregistrement est affiché dynamiquement en fonction de son contenu, ainsi que le contexte dans lequel il est affiché (dans une hiérarchie, en tant que clé étrangère, etc.) ;

Toutes ces informations textuelles peuvent être traduites dans les locales déclarées dans `ebx.properties`.

Voir aussi

Libellés et messages *[p 557]*

Déclaration de tables *[p 519]*

Déclaration de clés étrangères *[p 524]*

41.2 Règles de formatage des valeurs

Lorsqu'une valeur est affichée pour l'utilisateur, elle est formatée conformément à son type et à la règle de formatage de la locale actuelle. Par exemple, une date sera affichée dans certaines locales "jj/MM/aaaa" et "MM/jj/aaaa" dans d'autres.

Une règle de formatage est utilisée pour définir la manière dont sont affichées les valeurs des types simples [p 506].

Pour chaque locale déclarée dans `ebx.properties`, sa règle de formatage est configurée dans un fichier : `/WEB-INF/ebx/{locale}/frontEndFormattingPolicy.xml`. Par exemple, pour définir la règle de formatage destinée au grec (`el`), le système recherche le chemin suivant dans le module :

```
/WEB-INF/ebx/el/frontEndFormattingPolicy.xml
```

Si le fichier correspondant n'existe pas, la règle de formatage est recherchée dans le class-path de EBX®. Si la règle de formatage spécifique à la locale n'est pas trouvée, la règle de formatage `en_US` sera appliquée.

Le contenu du fichier `frontEndFormattingPolicy.xml` est le suivant :

```
<?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>
```

Les éléments `date`, `dateTime` et `time` sont obligatoires.

Les séparateurs de groupes et de décimaux qui apparaissent lors du formatage des nombres peuvent être modifiés en définissant les attributs `groupingSeparator` et `decimalSeparator` pour les éléments `decimal` et `int`.

41.3 Syntaxe des locales

Il y a deux manières d'exprimer une locale :

1. La recommandation XML suit la recommandation IETF BCP 47, qui utilise le trait d'union "-" comme séparateur.
2. La spécification Java utilise le tiret bas "_" au lieu du trait d'union.

Dans n'importe quel fichier XML (XSD, fichier de règle de formatage, etc.) lu par EBX®, les deux syntaxes sont autorisées.

Pour un chemin web, à savoir un chemin contenu dans l'application web, seule la syntaxe Java est autorisée. Ainsi, les fichiers de règles de formatage doivent se trouver dans des répertoires dont le nom de locale respecte la syntaxe Java.

Voir aussi *Extension de l'internationalisation de TIBCO EBX®* *[p 261]*

CHAPITRE **42**

Extension de l'internationalisation de TIBCO EBX®

Ce chapitre contient les sections suivantes :

1. Localisation native d'EBX®
2. Extension de la localisation de l'interface utilisateur d'EBX®
3. Résolution des ressources localisées
4. Limitations connues

42.1 Localisation native d'EBX®

Par défaut, l'interface utilisateur intégrée d'EBX® est fournie en anglais (en-US) et en français (fr-FR). La localisation est constituée des règles de formatage de données et d'un ensemble de fichiers de messages (fichier de ressources) :

- Pour l'anglais, la localisation est fournie par une règle de formatage de données et un ensemble de fichiers de messages sans locale définie,
- Pour le français, la localisation est fournie par une règle de formatage et un ensemble de fichiers de messages avec une locale définie à "fr".

EBX® offre la possibilité d'ajouter des locales pour étendre la localisation de l'interface utilisateur et d'internationaliser la documentation des modèles de données et services associés.

42.2 Extension de la localisation de l'interface utilisateur d'EBX®

EBX® prend en charge la localisation de son interface utilisateur vers n'importe quelle langue ou région compatible.

> **Note**
>
> Actuellement, seuls les caractères latins et cyrilliques sont acceptés. Les locales utilisant d'autres jeux de caractères sont utilisables, mais ne sont pas prises en charge.

Ajouter une nouvelle locale

Pour ajouter une nouvelle locale, suivre les étapes suivantes :

- Déclarer la nouvelle locale dans le fichier principal de configuration d'EBX®. Par exemple : ebx.locales.available=en-US, fr-FR, xx
  - La première locale est toujours considérée comme étant celle par défaut.
  - Les locales intégrées, en-US et fr-FR, peuvent être supprimées si besoin.

  Voir Configuring EBX® localization [p 375].
- Déployer les fichiers suivants dans le class-path d'EBX® :
  - Un fichier de règles de formatage, nommé `com.orchestranetworks.i18n.frontEndFormattingPolicy_xx.xml`,
  - Un ensemble de fichiers de messages localisés (*_xx.mxml) dans un fichier de ressources.

    > **Note**
    >
    > Les fichiers doivent terminer par ".mxml".

42.3 Résolution des ressources localisées

Depuis la version 5.7.0, les ressources localisées sont résolues sur la base de la proximité de locale, grâce au mécanisme de résolution suivant :

- resourceName + "_" + langue + "_" + pays + "_" + variante + ".mxml"
- resourceName + "_" + langue + "_" + pays + ".mxml"
- resourceName + "_" + langue + ".mxml"
- resourceName + ".mxml"

> **Note**
>
> La résolution s'effectue au niveau du message localisé. Il est ainsi possible de définir un ou plusieurs fichiers pour une locale n'incluant que les messages dont la localisation spécifique est souhaitée.

42.4 Limitations connues

Éléments non extensibles

La localisation des éléments suivants ne peut pas être étendue :

- Documentation produit EBX®,
- Visualiseur et éditeur HTML EBX®.

Persistance

CHAPITRE **43**

Présentation de la persistance

Ce chapitre décrit la façon dont les données de référence, l'historique et les tables répliquées sont conservés. Une table peut utiliser n'importe quelle combinaison de modes de persistance des données de référence, d'historisation et de réplication.

Bien que toutes les informations conservées dans TIBCO EBX® soient ensuite stockées dans la base de données sous-jacente en tant que tables relationnelles, le fait qu'elles se trouvent de façon accessible en dehors d'EBX® dépendra du mode mappé.

> **Note**
>
> Le terme *mode mappé* [p 265] fait référence à toute table stockée telle quelle et dont le contenu est accessible directement en base de données.

Ce chapitre contient les sections suivantes :

1. Persistance des données de référence gérées
2. Historisation
3. Réplication
4. Mode mappé

43.1 Persistance des données de référence gérées

Les données modélisées et gouvernées par le référentiel EBX® peuvent être conservées dans l'un des deux modes suivants : sémantique (par défaut) ou relationnel, selon ce qui a été spécifié dans le modèle de données sous-jacent. Des tables distinctes définies dans l'un ou l'autre mode peuvent coexister et collaborer dans le même référentiel EBX®.

Pour comparer le mode relationnel et le mode sémantique, voir le chapitre Présentation des modes [p 267]

43.2 Historisation

Les tables de données de référence peuvent activer l'historisation afin de tracer les modifications sur leurs données, indépendamment du fait qu'elles soient conservées en mode sémantique ou relationnel et qu'elles soient répliquées ou non.

L'historique est en mode mappé, ce qui signifie qu'il peut être consulté directement dans la base de données sous-jacente.

Voir aussi *Historique* *[p 273]*

43.3 Réplication

La réplication permet un accès SQL direct aux tables, en faisant une copie des données du référentiel vers les tables relationnelles de réplique en base de données. La réplication peut être activée sur n'importe quelle table, indépendamment de son mode de persistance (sémantique ou relationnel) et du fait que l'historique soit activé ou non.

Les répliques sont conservées en mode mappé, leur but principal étant de rendre les données de référence accessibles par requête directe en dehors d'EBX®.

Voir aussi *Réplication* *[p 281]*

43.4 Mode mappé

Présentation du mode mappé

Le mode mappé fait référence aux cas où les tables sont conservées dans la base de données relationnelle sous-jacente dans un format permettant d'accéder directement aux données, en dehors d'EBX®. Les données de référence déclarées en mode relationnel, l'historique et les copies de tables sont autant d'exemples de tables en mode mappé.

Tous les cas de mode mappé impliquent une modification automatique du schéma de la base de données (les tables, les index, etc.) le cas échéant, en exécutant automatiquement les commandes DDL nécessaires en arrière-plan. Ce type de procédure est toujours déclenché au moment de la compilation du modèle de données, et le rapport de compilation du modèle de données signale toute erreur éventuelle.

Autre remarque d'ordre général concernant les modes mappés : la plupart du temps, lorsqu'une entité d'un modèle de données est supprimée, l'objet correspondant en base de données n'est pas supprimé immédiatement. Il est noté comme désactivé, ce qui laisse la possibilité de réactiver l'objet par la suite. Afin de supprimer définitivement de la base de données l'objet et les données et ressources qui y sont associées, il faut le marquer pour la purge. La suppression a alors lieu lors de la purge générale suivante.

Voir aussi

Database mapping administration *[p 431]*

Évolutions du modèle de données *[p 289]*

Restrictions sur le modèle de données suite au mode mappé

En raison de la nature de la persistance directe en base de données sous-jacente, certaines restrictions s'appliquent à toutes les tables stockées en mode mappé :

- Limitations of supported databases [p 339]
- Longueur illimitée pour les chaînes de caractères : tous les champs chaînes de caractère, excepté les clés étrangères, de type `xs:string`, leurs types dérivés et `xs:anyURI`, doivent définir une facette 'maxLength' ou 'length'. Puisqu'un champ de clé étrangère est composé des champs de la clé primaire finale de ses tables cibles, ce critère sur les facettes s'applique à chacun des champs de clé primaire finale, et non au champ de clé étrangère lui-même. De plus, les limitations de la base

de données sous-jacente concernant la longueur maximale de ses types de caractères s'appliquent, telles que *VARCHAR* et *NVARCHAR2*.

- Les longues listes de colonnes sont parfois non-indexables. Exemple sur Oracle : la base de données exige une limite de taille maximale cumulée des colonnes comprises dans un index. Pour les chaînes de caractères, cette taille dépend également du jeu de caractères. Si le serveur de base de données échoue lors de la création de l'index, il sera nécessaire d'envisager de revoir la conception des index, typiquement en réduisant la longueur des colonnes concernées, ou en incluant moins de colonnes dans l'index. Le raisonnement est qu'un index menant à cette situation aurait des en-têtes tellement grands qu'il ne pourrait de toute façon pas être efficace.
- Les champs de type `type="osd:password"` sont ignorés.
- Les types complexes terminaux sont supportés ; cependant, ils ne peuvent être définis à `null` de manière globale, au niveau de l'enregistrement.

D'une façon plus générale, les tables en mode mappé sont soumises aux limitations du SGBDR sous-jacent. Par exemple, le nombre maximum de colonnes dans une table doit être respecté (1000 pour Oracle 11g R2, 1600 pour PostgreSQL). Remarque : une table d'historique contient deux fois plus de champs que ceux déclarés dans le schéma (un champ fonctionnel, plus un champ généré pour le code de l'opération).

Les évolutions du modèle de données peuvent aussi être limitées par le SGBDR sous-jacent, en fonction du modèle de données existant.

Voir aussi *Évolutions du modèle de données* *[p 289]*

CHAPITRE 44

Mode relationnel

Ce chapitre contient les sections suivantes :

1. Présentation des modes
2. Activation du mode relationnel pour un modèle de données
3. Validation
4. Accès SQL aux données en mode relationnel
5. Limitations du mode relationnel

44.1 Présentation des modes

Le mode sémantique en détail

Le mode sémantique présente toutes les fonctionnalités avancées de TIBCO EBX® en matière de gestion de données de référence. En particulier les espaces de données, l'héritage des jeux de données et les champs hérités.

Le mode sémantique est le mode par défaut de persistance des données gouvernées par le référentiel EBX®. Les modèles de données sont en mode sémantique, à moins que le mode relationnel [p 268] ne soit spécifié [p 269] de manière explicite.

En interne, les données de référence gérées en mode sémantique sont représentées en XML standard, conformément au document XML Schema de son modèle de données. La représentation XML est compressée à nouveau et segmentée pour être stockée dans des tables génériques de bases de données relationnelles. Ce mode permet de stocker et d'accéder efficacement aux données, notamment pour :

- les espaces de données: lors de la création d'un espace de données enfant, aucune donnée n'est dupliquée, et
- l'héritage : aucune donnée n'est dupliquée lors de la création d'une instance héritée.

Le mode sémantique permet aussi de maintenir un nombre illimité de jeux de données pour chaque modèle de données, organisés dans un nombre illimité d'espaces de données et d'images. Tout cela sans impacter le schéma de base de données.

Comme ce mode n'utilise que des tables communes internes et génériques, les modifications de structure du modèle de données n'ont pas non plus d'impact sur le schéma de base de données. Les évolutions du modèle de données n'impactent que le contenu des tables génériques de la base de données.

Voir aussi

Services de données *[p 98]*

dataset inheritance *[p 297]*

inherited fields *[p 298]*

Le mode relationnel en détail

Le mode relationnel, qui est un mode mappé, conserve les données de référence directement dans la base de données. Le but premier du mode relationnel est de pouvoir tirer parti des performances et des capacités de montée en charge de la base de données relationnelle sous-jacente. Cependant, le mode relationnel ne gère pas les fonctionnalités avancées de gouvernance proposées en mode sémantique.

Dans certains cas où les avantages de la gestion en mode sémantique ne sont pas indispensables, comme pour les tables "à un instant donné", ou les tables mises à jour régulièrement par des systèmes externes, les gains de performance offerts en mode relationnel peuvent être plus utiles.

De manière générale, quand un jeu de données est en mode relationnel, chaque table de ce jeu de données possède une table correspondante dans la base de données, et tous les champs de son modèle de données correspondent à une colonne de table relationnelle.

Comparaison directe du mode sémantique et du mode relationnel

Le tableau suivant résume les différences entre les deux modes de persistance :

	Mode sémantique	**Mode relationnel**
Espaces de données	Oui	Non
Héritage de jeu de données	Oui	Non
Champs hérités	Oui	Non
Modèle de données	Toutes les fonctionnalités sont supportées.	Des restrictions s'appliquent, voir Restrictions du modèle de données pour les tables en mode relationnel [p 272].
Lectures directes en SQL	Non	Oui, voir Lectures SQL [p 271].
Écritures directes en SQL	Non	Oui, mais uniquement sous certaines conditions spécifiques, voir Écritures SQL [p 271].
Validation des données	Oui, active le mode tolérant.	Oui, certaines contraintes deviennent bloquantes, voir Validation [p 269].
Transactions	Voir Concurrency and isolation levels [p 492].	Voir Concurrency and isolation levels [p 492].
Évolutions du modèle de données	Voir Évolutions du modèle de données [p 289] dans le Manuel de référence.	Voir Évolutions du modèle de données [p 289] dans le Manuel de référence.

44.2 Activation du mode relationnel pour un modèle de données

Le modèle de données déclare qu'il est en mode relationnel. En raison des contraintes intrinsèques au mode relationnel, telles que l'absence d'espace de données enfant ou d'image, un espace de données relationnel spécifique devra être fourni, sur lequel le modèle de données sera publié. Les espaces de données relationnels ne permettent pas de créer de sous-espaces de données ou d'images.

Exemple d'une déclaration de mode relationnel :

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

avec les éléments :

Élément	Description	Requis
`dataSpace`	Désigne l'espace de données dans lequel le modèle de données doit être publié. Cet espace de données doit être lui-même en mode relationnel. Aucun espace de données ou image ne peut être créé à partir de l'espace de données déclaré dans ce mode.	Oui
`dataSet`	Désigne le jeu de données où le modèle de données doit être publié.	Oui
`tablesPrefix`	Désigne le préfixe commun utilisé pour nommer les tables générées dans la base de données.	Oui

44.3 Validation

Cette section détaille l'impact du mode relationnel sur la validation de données.

Contraintes structurelles

Certaines contraintes du modèles de données d'EBX® vont générer une "contrainte structurelle" sur le schéma du SGBDR sous-jacent du mode relationnel. C'est également le cas quand l'historique de table est activé [p 273]. Les facettes suivantes sont concernées :

- les facettes `xs:maxLength` et `xs:length` sur les éléments `string` ;
- les facettes `xs:totalDigits` et `xs:fractionDigits` sur les éléments `xs:decimal`.

Les bases de données n'offrent pas le mode de validation tolérant proposé par EBX®. Les contraintes citées précédemment deviennent alors des *contraintes bloquantes*. Une contrainte bloquante signifie

que les mises à jour non conformes seront rejetées. De plus, ces contraintes ne sont plus vérifiées au cours du processus de validation, à l'exception des contraintes de clés étrangères dans certains cas (voir Mode de blocage de clé étrangère [p 270]). Quand une transaction n'est pas conforme à une contrainte bloquante, elle est annulée et une `ConstraintViolationException`[API] est levée.

Voir aussi *Contraintes bloquantes et non bloquantes [p 548]*

Mode bloquant de clé étrangère

Afin de réduire le temps de validation, les contraintes de clé étrangère sont automatiquement définies en mode bloquant si :

- Les contraintes de clé étrangère sont définies dans une table en mode relationnel,
- Les contraintes de clé étrangère sont définies dans une table en mode sémantique ou relationnel référençant une table en mode relationnel.

Pour cette contrainte, le mode bloquant implique que la tentative d'exécution des actions suivantes provoque une `ConstraintViolationException`[API] :

- effacer un enregistrement référencé par une contrainte de clé étrangère,
- effacer une instance référencée par une contrainte de clé étrangère,
- fermer un espace de données référencé par une contrainte de clé étrangère.

Il est cependant possible de surcharger ce comportement par défaut en définissant un mode de validation spécifique. Voir Blocking and non-blocking constraints [p 548] pour plus d'informations.

Afin de garantir l'intégrité des contraintes de clé étrangère après une écriture SQL directe qui contourne le cadre de gouvernance d'EBX®, les contraintes de clé étrangère seront validées dans les cas suivants :

- à la première validation explicite via l'interface utilisateur ou l'API,
- à la première validation explicite via l'interface utilisateur ou l'API après actualisation du schéma,
- à la première validation explicite via l'interface utilisateur ou l'API après la réinitialisation du rapport de validation d'un jeu de données dans l'interface utilisateur.

Important :

- L'aspect bloquant de la contrainte de clé étrangère ne concerne pas les filtres qui peuvent être définis. C'est à dire qu'une contrainte de clé étrangère est non bloquante si un enregistrement référencé existe mais ne répond pas aux critères de filtre de la clé étrangère. Dans ce cas, les mises à jour ne sont pas rejetées et une erreur sera alors ajoutée au rapport de validation.
- Les contraintes de clé étrangère ne sont pas en mode bloquant lors de l'import d'archives. En effet, toutes les contraintes bloquantes, à l'exception des contraintes structurelles, sont toujours désactivées lors de l'import d'archives. Ceci permet d'avoir une certaine souplesse lors de l'import d'archives lorsque, sous certaines conditions, l'import de clés étrangères référençant des enregistrements qui n'ont pas encore été importés doit être tolérant.

Contraintes sur l'ensemble de la table

Les **contraintes** `Constraint`[API] programmatiques sont vérifiées pour chaque enregistrement de la table à la validation. Si la table définit des millions d'enregistrements, des problèmes de performance se poseront. Il est alors recommandé de définir une **contrainte au niveau table** `ConstraintOnTable`[API].

Dans les cas où une contrainte au niveau table ne peut être définie, il est recommandé de définir au moins une **dépendance locale ou explicite** `DependenciesDefinitionContext.dependencies`[API], afin de réduire le coût de la validation incrémentale.

Voir aussi *`ConstraintOnTable`*[API]

44.4 Accès SQL aux données en mode relationnel

Cette section décrit comment accéder directement aux données en mode relationnel via SQL.

Voir aussi *Accès SQL à l'historique* *[p 276]*

Trouver la table dans la base de données

Pour chaque table EBX® en mode relationnel, une table correspondante est générée dans le SGBDR. En utilisant l'interface utilisateur d'EBX®, il est possible de trouver le nom de cette table en cliquant sur le panneau de documentation de la table.

Lectures SQL

Les lectures directes en SQL sont possibles sur des transactions bien gérées et à durée de vie courte de préférence. Cependant, pour ce type d'accès, les permissions d'EBX® ne sont pas prises en compte. Les applications autorisées à réaliser des lectures doivent être identifiées à travers d'autres processus d'authentification et permissions.

Écritures SQL

Les écritures directes en SQL contournent le cadre de gouvernance d'EBX®. Par conséquent, elles doivent être utilisées avec une ***grande précaution***. Elles peuvent être à l'origine des situations suivantes :

- échec de l'historisation des tables d'EBX® ;
- échec de l'exécution des triggers d'EBX® ;
- échec de la vérification des permissions et des contraintes d'EBX® ;
- modifications qui échappent au processus de validation incrémentale ;
- perte de visibilité des tables sémantiques d'EBX®, qui peuvent être référencées par des clés étrangères.

Par conséquent, les écritures directes en SQL doivent être réalisées *si, et seulement si, toutes les conditions suivantes sont remplies* :

- les tables modifiées ne sont pas historisées et n'ont pas de triggers EBX®.
- l'application réalisant les écritures peut être authentifiée grâce aux permissions associées, afin de garantir l'intégrité des données. Plus particulièrement, l'intégrité des clés étrangères (`osd:tableRef`) doit être préservée à tout moment. Voir Mode bloquant de clé étrangère [p 270] pour plus d'informations.
- le serveur d'application qui exécute EBX® doit être arrêté *lorsqu'une écriture est réalisée*. Ceci permet de garantir que la validation incrémentale ne devienne obsolète, ce qui serait le cas dans un contexte batch.

44.5 Limitations du mode relationnel

Le mode relationnel est pleinement opérationnel mais fait l'objet de quelques limitations, listées ci-dessous. Dans le cas de l'utilisation du mode relationnel, il est fortement recommandé de prendre connaissance de ces limitations et de contacter l'équipe support de TIBCO EBX® à https://support.tibco.com en cas de questions.

Voir Supported databases [p 339] pour les bases de données supportant le mode relationnel.

Restrictions du modèle de données pour les tables en mode relationnel

Certaines restriction s'appliquent aux modèles de données en mode relationnel :

- Restrictions sur le modèle de données suite au mode mappé [p 265]
- Les listes agrégées [p 516] ne sont pas supportées dans les tables relationnelles. Un tel schéma provoquera une erreur de compilation.
- Les attributs définis par l'utilisateur sur des tables relationnelles provoquent des erreurs de compilation dans le modèle de données.
- Héritage de jeu de données [p 297].
- Champs hérités [p 298].
- Des contraintes programmatiques, puisque le coût de calcul de la validation serait trop important. Cependant, les **contraintes sur tables** `ConstraintOnTable`[API] restent disponibles.

Les évolutions de schéma peuvent aussi être limitées par le SGBDR sous-jacent selon les données déjà contenues dans les tables concernées.

Voir aussi *Évolutions du modèle de données [p 289]*

Autres limitations du mode relationnel

- Limitations du mode mappé [p 265]
- Il n'est pas possible de créer d'espaces de données enfants ou d'images à partir d'un espace de données contenant des jeux de données en mode relationnel.
- Pour D3, il n'est pas possible de diffuser un espace de données défini en mode relationnel.
- Pour de très grands volumes de données, la validation aura une faible performance si la table relationnelle déclare une de ces fonctionnalités : `osd:function`, `osd:select`, `osd:uiFilter`, `osd:tableRef/filter`. De plus, aucun tri ne peut être appliqué sur une colonne `osd:function`.
- Il est impossible de définir `AdaptationValue.INHERIT_VALUE` sur un noeud appartenant à un modèle de données en mode relationnel.

CHAPITRE 45

Historique

Ce chapitre contient les sections suivantes :

1. Présentation
2. Configuration de l'historique
3. Affichage d'historique et permissions
4. Accès SQL à l'historique
5. Impacts et limitations du mode historisé

45.1 Présentation

L'historique est une fonctionnalité permettant de suivre toutes les modifications effectuées sur les données d'une table (création d'enregistrements, mise à jour ou suppression).

Ceci constitue une version améliorée de la piste d'audit XML [p 445]. La piste d'audit XML est activée par défaut, mais peut être désactivée en toute sécurité si l'historique est mis en place pour les tables concernées.

Voir aussi

Historique [p 29]

Mode relationnel [p 267]

Réplication [p 281]

Évolutions du modèle de données [p 289]

45.2 Configuration de l'historique

Afin d'activer l'historisation d'une table, un profil d'historique doit être défini pour cette table dans le modèle de données. Cette section décrit les profils d'historique et comment ils sont associés aux tables.

Configuration de l'historique dans le référentiel

Un profil d'historique spécifie le moment où l'historisation doit être mise en place. Pour modifier les profils d'historique, sélectionner *Administration > Historique et journaux*.

Un profil d'historique est identifié par un nom et définit les informations suivantes :

- un libellé internationalisé,

- une liste d'espaces de données (branches) pour lesquels l'historique est activé. Il est possible de spécifier si seuls les enfants directs ou si tous les descendants sont aussi concernés.

Certains profils sont déjà présents lors de l'installation du référentiel. Ces profils ne peuvent être ni supprimés ni modifiés.

Id profil	Description
`ebx-referenceBranch`	Ce profil est uniquement activé pour l'espace de données de référence.
`ebx-allBranches`	Ce profil est activé pour tous les espaces de données.
`ebx-instanceHeaders`	Ce profil historise les en-têtes des jeux de données. Cependant, ce profil ne sera paramétrable que dans une future version, étant donné que le modèle de données interne ne définit que les noeuds des jeux de données.

Configuration de l'historique dans le modèle de données

Activation de l'historique de table

L'historique peut être activé pour une table via l'assistant de modèle de données, ou en éditant le modèle de données sous-jacent.

Pour activer l'historique en éditant le modèle de données, un profil d'historique doit être déclaré pour la table en utilisant l'élément `historyProfile`.

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

L'assistant de modèle de données permet de visualiser le profil d'historisation défini dans le référentiel.

L'historisation peut être activée séparément pour chaque table. Voir la documentation de conception du modèle [p 502] pour plus d'informations.

Désactivation de l'historique d'un champ ou d'un groupe spécifique

Pour une table historisée, la règle par défaut est l'historisation de tous les éléments qu'elle prend en charge, (voir Impacts et limitations du mode historisé [p 278]).

Il est possible de désactiver l'historique d'un champ ou d'un groupe spécifique, que ce soit via l'assistant du modèle de données ou via la modification du modèle de données sous-jacent.

Pour désactiver l'historique d'un champ ou d'un groupe en éditant le modèle de données, utiliser l'élément `osd:history` avec l'attribut `disable="true"`.

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

Pour désactiver l'historique d'un champ ou d'un groupe via l'assistant du modèle de données, utiliser la propriété `History` dans les `Advanced properties` de l'élément.

Quand cette propriété est définie pour un groupe, l'historique est désactivé récursivement pour tous ses descendants. Lorsque l'on désactive l'historique d'un groupe, il est impossible de réactiver spécifiquement l'historique pour un descendant.

> **Note**
>
> Si la table contenant le champ ou le groupe n'est pas historisée, cette propriété n'aura aucun effet.
>
> Il n'est pas possible de désactiver l'historique pour les champs de clé primaire.

Intégrité

Si des problèmes sont détectés lors de la compilation du modèle de données, des messages d'alerte ou d'erreur seront ajoutés au rapport de validation associé à ce modèle de données. En outre, si la moindre erreur est détectée, chaque instance (jeu de données) associée sera inaccessible. Les cas d'erreur les plus répandus sont les suivants :

- Une table fait référence à un profil qui n'est pas défini dans le référentiel.
- Un profil d'historique qui est référencé dans le modèle de données mentionne un espace de données non défini ou fermé du référentiel courant.

> **Note**
>
> Le déploiement d'un modèle de données dans un référentiel qui ne dispose pas des profils requis nécessitera l'intervention de l'administrateur pour les ajouter.

45.3 Affichage d'historique et permissions

Vue historique de table

Lorsque l'historique a été activé pour une table dans un modèle de données, il est possible d'accéder à la vue de l'historique à partir de l'interface utilisateur depuis : un enregistrement, une sélection d'enregistrements, une table et un jeu de données.

La section suivante explique comment les permissions sont déterminées.

Pour plus d'informations, voir la section vue historique de table [p 29] Pour accéder à la vue historique de table à partir de Java, la méthode `AdaptationTable.getHistory`[API] doit être invoquée.

Permissions pour historique de table

Les permissions relatives aux données sont aussi appliquées à l'historique des données. Les permissions de l'historique sont définies automatiquement, la permission la plus restrictive parmi les permissions des données et les droit d'accès en *lecture seule* sera appliquée.

C'est le cas des permissions définies par l'utilisateur et des règles de permission programmatiques.

Lors de la définition d'une règle programmatique, il peut être nécessaire de faire la distinction entre le contexte du jeu de données fonctionnel et le contexte de vue historique. Soit parce que les permissions attendues sont différentes, soit parce que certains champs ne sont pas présents dans la structure de l'historique. C'est le cas en ce qui concerne les champs de jeux de données, les valeurs calculées, ainsi que les champs dont l'historique a été désactivé [p 274]. Les méthodes `Adaptation.isHistory`[API] et `AdaptationTable.getHistory`[API] peuvent alors être utilisées dans la règle programmatique afin d'implémenter un comportement spécifique pour l'historique.

Vue historique des transactions

La vue historique des transactions permet d'accéder aux transactions exécutées, indépendamment d'une table, du jeu de données ou du modèle de données, directement à partir de l'interface utilisateur.

Pour afficher la table 'Historique des transactions', aller dans 'Administration' et sélectionner 'Historique et journaux' à l'aide du menu déroulant symbolisé par une flèche dans le panneau de navigation. On peut aussi accéder à l'historique des transactions à partir des 'Espaces de données' en sélectionnant un espace de données historisé et en utilisant le menu 'Actions' dans l'espace de travail.

Pour plus d'informations, voir vue historique des transactions [p 29].

45.4 Accès SQL à l'historique

Cette section décrit le moyen d'accéder directement à l'historique grâce à SQL.

Voir aussi *Accès SQL aux données en mode relationnel [p 271]*

Restrictions d'accès

L'accès aux tables de la base de données n'est autorisé qu'en mode lecture seule. C'est à l'administrateur de la base de données d'en interdire l'accès en écriture, sauf pour l'utilisateur de base de données de TIBCO EBX®, tel qu'indiqué dans la section Rules for the database access and user privileges [p 399].

Présentation du schéma relationnel

Voici une description des tables d'historique dans la base de données.

Le schéma de la base de données contient (voir aussi le diagramme dans la section suivante) :

Des tables communes et génériques	La table principale est `HV_TX` ; chaque enregistrement de cette table représente une transaction. Seules les transactions qui impliquent au moins une table historisée sont enregistrées. Ces tables communes ont toutes le préfixe "HV".
Tables spécifiques générées	Pour chaque table historisée, une table d'historique spécifique est générée. Cette table contient l'historique des modifications de données sur la table. Dans l'interface utilisateur d'EBX®, le nom de cette table en base de données peut être connu en cliquant sur le panneau de documentation de la table (mode avancé). Toutes les tables d'historique spécifiques ont le préfixe "HG".

Exemple de table d'historique générée

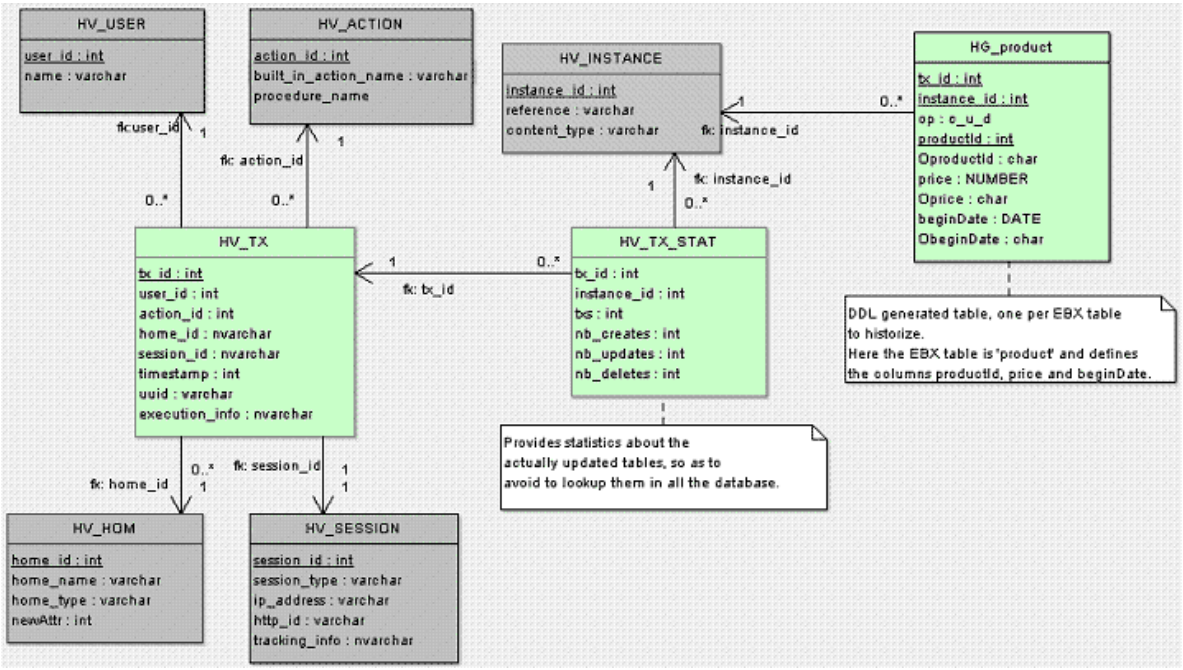
Dans l'exemple suivant, une table nommée `product` est historisée. Supposons que cette table déclare trois champs dans le modèle de données EBX® :

Produit

- productId : int

- price : int
- beginDate : Date

Le diagramme ci-dessous montre le schéma relationnel qui en résulte :



L'activation de l'historique sur cette table génère la table HG_product affichée dans la structure du schéma de l'historique ci-dessus. Voici la description de ses différents champs :

- `tx_id`: ID de la transaction.
- `instance`: ID de l'instance.
- `op`: type d'opération - C (créer), U (mettre à jour) ou D (effacer).
- `productId`: `productId` valeur de champ.
- `OproductId`: champ d'opération pour `productId`, voir section suivante.
- `price`: `price` valeur de champ.
- `Oprice`: champ d'opération pour `price`, voir section suivante.
- `beginDate`: `date` valeur de champ.
- `ObeginDate`: champ d'opération pour `beginDate`, voir section suivante.

Combinaison doperations

Si plusieurs opérations ont lieu au cours de la même transaction, le champ d'opération est résolu selon la règle suivante :

- `C + U -> C`

  `D + U -> D`

  `D + C -> U`

  `C + D -> {} (pas d'enregistrement dans l'historique)`

Valeurs de champs d'opération

Pour chaque champ fonctionnel, un champ d'opération supplémentaire est défini. Il est composé du nom du champ dont le préfixe est `O`. Ce champ précise si le champ fonctionnel a été modifié et prend une des valeurs suivantes :

- `null`: si la valeur du champ fonctionnel n'a pas été modifiée (et que sa valeur n'est pas INHERIT).
- `M`: si la valeur du champ fonctionnel a été modifiée (mais ne prend pas la valeur INHERIT).
- `D`: si l'enregistrement a été effacé.

Si inheritance [p 296] est activé, le champ d'opération peut avoir trois valeurs supplémentaires :

- `T`: si la valeur du champ fonctionnel n'a pas été modifiée et que sa valeur est INHERIT.
- `I`: si la valeur du champ fonctionnel a pris la valeur INHERIT.
- `O`: si l'enregistrement a été mis en mode OCCULTING.

45.5 Impacts et limitations du mode historisé

La fonctionnalité d'historique a certains impacts et des limitations connues, qui sont décrits dans cette section. Lors de l'utilisation du mode historisé, il est fortement conseillé de lire attentivement ces limitations et de contacter le support de Cloud Software Group, Inc. pour toute question.

Validation

Certaines contraintes du modèle de données d'EBX® deviennent bloquantes quand l'historique de la table est activé. Pour plus d'informations, voir la section Contraintes structurelles [p 269].

Restrictions du modèle de données pour les tables historisées

Certaines restrictions s'appliquent aux modèles de données qui contiennent des tables historisées :

- Restrictions sur le modèle de données suite au mode mappé [p 265]
- Certaines limitations existent pour deux types de listes agrégées : les listes agrégées sous une autre liste agrégée, et les listes agrégées sous un groupe terminal. Les modèles de données contenant ce type de listes agrégées peuvent être utilisés, cependant ces listes seront ignorées (non historisées).
- Les valeurs calculées sont ignorées.
- La présence d'attributs définis par l'utilisateur sur des tables historisées provoque des erreurs lors de la compilation du modèle de données.

Les évolutions du modèle de données peuvent aussi être restreintes par le SGBDR sous-jacent, en fonction des données déjà présentes dans les tables concernées.

Voir aussi *Évolutions du modèle de données [p 289]*

Autres limitations du mode historisé

- Aucune donnée n'est copiée lorsque l'historique est activé sur une table contenant des données existantes.
- Les opérations d'ensemble sur les jeux de données ne sont pas historisées (créer et supprimer une instance), même si elles déclarent une table historisée.

- Les libellés par défaut référençant un champ non historisé ne sont pas pris en charge pour les tables historisées.

  Par conséquent, les libellés par défaut référençant un champ calculé ne sont pas pris en charge pour les tables historisées.

  Pour contourner ce problème, il suffit d'implémenter l'interface `UILabelRenderer` et d'adapter le calcul du libellé pour l'historique.
- D3 : l'historique peut être activé dans l'espace de données de destination d'un noeud maître. Mais, dans l'espace de données de destination des noeuds esclaves, la fonction historisation est toujours désactivée.
- Utilisateur enregistré dans l'historique : pour certaines opérations spécifiques, l'utilisateur réalisant la dernière opération et celui qui est enregistré dans l'historique correspondant peuvent être différents.

  Ceci est dû au fait que ces opérations sont un rapport du statut des données à un état antérieur :

  - Import d'archives : lors de l'import d'archive dans un espace de données, l'heure et le nom de l'utilisateur de la dernière opération réalisée dans l'espace de données sont conservés, alors que le nom de l'utilisateur enregistré dans l'historique est celui qui réalise l'import.
  - Fusion programmatique : lors de la fusion programmatique d'un espace de données, l'heure et le nom de l'utilisateur de la dernière opération réalisée dans l'espace de données enfant sont conservés, alors que le nom de l'utilisateur enregistré dans l'historique est celui qui réalise la fusion.
  - D3 : en ce qui concerne la fonctionnalité de "distributed data delivery", lorsqu'une diffusion est réalisée, les données du noeud maître sont reportées dans le noeud esclave. L'heure, ainsi que le nom de l'utilisateur qui a réalisé la dernière opération dans l'espace de données enfant, sont conservés. Le nom de l'utilisateur enregistré dans l'historique est "ebx-systemUser" ; c'est lui qui procède à la déclaration sur le noeud esclave lors de la diffusion.

CHAPITRE 46

Réplication

Ce chapitre contient les sections suivantes :

1. Présentation
2. Configuration de la réplication
3. Accéder à une table répliquée à l'aide de SQL
4. Requête d'actualisation des réplications 'onDemand'
5. Impact et limitations de la réplication

46.1 Présentation

Les données stockées dans le référentiel de TIBCO EBX® peuvent être copiées en miroir dans des tables relationnelles dédiées afin d'offrir un accès direct aux données via des requêtes et des vues SQL.

Tout comme l'historique et le mode relationnel, cette réplication des données est transparente pour l'utilisateur final et pour les applications clientes. Certaines actions provoquent des modifications automatiques des répliques dans la base de données :

- L'activation de la réplique, au niveau du modèle, met à jour le schéma de la base de données en exécutant automatiquement les déclarations de DDL nécessaires.
- Les évolutions du modèle de données impactant les tables répliquées, telles que la création d'une nouvelle colonne, actualisent aussi le schéma de la base de données par le biais de déclarations DDL.
- En utilisant le mode d'actualisation 'onCommit' : la mise à jour des données du référentiel d'EBX® déclenche les insertions, mises à jour et suppressions associées aux répliques en base de données.

Voir aussi

Mode relationnel *[p 267]*

Historique *[p 273]*

Évolutions du modèle de données *[p 289]*

Repository administration *[p 398]*

> **Note**
>
> *table répliquée* : désigne une table de données de référence qui a fait l'objet d'une réplication
>
> *table de réplique* (ou *réplique*) : désigne une table en base de données qui est la cible de la réplication

46.2 Configuration de la réplication

Activer la réplication

Pour définir une *unité de réplication* sur un modèle de données, utiliser l'élément `osd:replication` sous les éléments `annotation/appinfo`. Chaque unité de réplication désigne des tables dans un jeu de données unique dans un espace de données spécifique.

Les éléments imbriqués sont les suivants :

Élément	Description	Requis
`name`	Nom de l'unité de réplication. Ce nom identifie l'unité de réplication dans le modèle de données. Ce nom doit être unique.	Oui
`dataSpace`	Indique l'espace de données concerné par la réplication. Cet espace de données ne peut ni être une version ni être relationnel.	Oui
`dataSet`	Indique le jeu de données concerné par la réplication.	Oui
`refresh`	Spécifie la politique de synchronisation des données. Les politiques disponibles sont : • `onCommit`: Dans la base de données, le contenu de la réplique est toujours à jour par rapport à sa table source. Chaque transaction qui met à jour la table source d'EBX® déclenche les insertions, mises à jour et suppressions correspondantes sur la réplique. • `onDemand` : La réplication des tables spécifiées n'est effectuée que lorsqu'une opération d'actualisation explicite est réalisée. Voir Requête d'actualisation des réplications 'onDemand' [p 285].	Oui
`table/path`	Indique le chemin de la table dans le modèle de données qui doit être répliquée dans la base de données.	Oui
`table/nameInDatabase`	Indique le nom de la table dans la base de données qui contiendra les données répliquées. Ce nom doit être unique par rapport à toutes les unités de réplication.	Oui
`table/element/path`	Indique le chemin de la liste agrégée dans la table qui doit être répliquée dans la base de données.	Oui
`table/element/ nameInDatabase`	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.	Oui

Par exemple :

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

```
 </xs:annotation>
 ...
</xs:schema>
```

Remarques :

- Voir Restrictions du modèle de données pour les tables répliquées [p 285]
- Si, lors de la compilation du modèle de données, le jeu de données et/ou l'espace de données spécifiés n'existent pas dans le référentiel courant, un avertissement est émis, mais la table des réplicats est créée dans la base de données. Une fois que l'espace de données et le jeu de données sont créés, la réplication est activée.
- Si, lors de la compilation d'un modèle de données, une table répliquée est effacée, ou si certaines des propriétés ci-dessus ont été modifiées, la réplique est effacée de la base de données, puis recréée avec la nouvelle définition si besoin.

Désactiver la réplication pour un champ ou un groupe donné

Le comportement par défaut d'une table répliquée est la copie de tous ses éléments pris en charge (voir Restrictions du modèle de données pour les tables répliquées [p 285]).

Il est possible de désactiver la réplication pour un champ ou un groupe spécifique, soit grâce à l'assistant de modèle de données, soit en modifiant le modèle de données sous-jacent.

Pour désactiver la réplication d'un champ ou d'un groupe en modifiant le modèle de données, utiliser l'élément `osd:replication` avec l'attribut `disable="true"`.

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

Pour désactiver la réplication d'un champ ou d'un groupe à l'aide de l'assistant du modèle de données, utiliser la propriété `Replication` des `Advanced properties` de l'élément.

Lorsque cette propriété est définie sur un groupe, la réplication est désactivée récursivement pour tous ses descendants. Une fois que la réplication a été désactivée sur un groupe, il n'est pas possible de la ré-activer sur un descendant.

> **Note**
>
> Si la table contenant le champ ou le groupe n'est pas répliquée, cette propriété n'aura aucun effet.
>
> Il n'est pas possible de désactiver la réplication pour les champs de clé primaire.

46.3 Accéder à une table répliquée à l'aide de SQL

Cette section décrit comment accéder directement à une réplique en utilisant SQL.

Voir aussi *Accès SQL à l'historique [p 276]*

Trouver la réplique dans la base de données

Pour chaque table de réplique d'EBX®, une table correspondante est générée dans le SGBDR. Grâce à l'interface utilisateur d'EBX®, il est possible de trouver le nom de cette table de base de données en cliquant sur le panneau de documentation de la table.

Restrictions d'accès

L'accès direct aux répliques en base de données n'est autorisé qu'en mode lecture-seule. Il est de la responsabilité de l'administrateur de la base de données de bloquer l'accès en écriture à tous les utilisateurs de la base de données à l'exception de celui utilisé par EBX®.

Voir aussi *Rules for the database access and user privileges [p 399]*

Lectures SQL

Les lectures directes en SQL sont possibles sur des transactions bien encadrées et de préférence de courte durée. Toutefois, pour de tels accès, les permissions d'EBX® ne sont pas prises en compte. Par conséquent, les applications ayant le droit d'effectuer des lectures doivent être sécurisées par le biais de permissions et de processus d'authentification tiers.

46.4 Requête d'actualisation des réplications 'onDemand'

La politique d'actualisation 'onDemand' nécessite une requête explicite pour actualiser les données de la table répliquée.

Il y a plusieurs moyens pour demander une actualisation de réplication :

- **Interface utilisateur** : Dans le menu d'actions du jeu de données, utiliser l'action 'Rafraîchir les répliques' sous le groupe 'Réplication' pour lancer l'assistant d'actualisation de réplication.
- **Services de données**: Utiliser l'opération de services de données de l'actualisation des réplications. Voir Replication refresh [p 668] pour davantage d'informations concernant les services de données.
- **API Java** : Appeler les méthodes `ReplicationUnit.performRefresh`[API] dans l'API `ReplicationUnit` afin d'actualiser l'unité de réplication.

46.5 Impact et limitations de la réplication

La fonctionnalité de réplication a certains impacts et des limitations connues, listés ci-dessous. En cas d'utilisation de la réplication, il est fortement conseillé de lire attentivement ces limitations et de contacter le support de Cloud Software Group, Inc. pour toute question.

Voir Supported databases [p 339] pour les bases de données qui prennent en charge la réplication.

Validation

Certaines contraintes de modèle de données d'EBX® deviennent bloquantes lorsque la réplication est activée. Pour plus d'informations, voir Contraintes structurelles [p 269].

Restrictions du modèle de données pour les tables répliquées

Certaines restrictions s'appliquent aux modèles de données contenant des tables répliquées :

- Restrictions sur le modèle de données suite au mode mappé [p 265]
- L'héritage de jeu de données n'est pas pris en charge dans le cas de la politique d'actualisation 'onCommit' si le jeu de données spécifié n'est pas un jeu de données racine ou s'il n'a pas encore été créé. Voir héritage de jeu de données [p 297] pour plus d'informations.

- De la même manière, l'héritage de champ est uniquement pris en charge pour la politique d'actualisation 'onDemand'. Cela signifie qu'à la compilation du modèle de données, une erreur est émise si le mode d'actualisation est 'onCommit' et que la table devant être répliquée a un champ hérité. Voir champs hérités [p 298] pour plus d'informations.
- Les valeurs calculées sont ignorées.
- Il existe des limitations pour deux catégories de listes agrégées : les listes agrégées sous une autre liste agrégée, et les listes agrégées sous un groupe terminal. Les modèles de données contenant de telles listes agrégées peuvent être utilisées, toutefois ces listes seront ignorées (non repliquées).
- Les attributs personnalisés ne sont pas pris en charge. Une erreur de compilation est émise s'ils sont inclus dans une unité de réplication.

Les évolutions du modèle de données peuvent aussi être limitées par le SGBDR sous-jacent, en fonction des données déjà contenues dans les tables concernées.

Voir aussi *Évolutions du modèle de données [p 289]*

Configuration de la base de données

L'opération d'actualisation est optimisée pour transmettre uniquement les lignes de la table source qui ont été modifiées (création et suppression) depuis la dernière actualisation. Toutefois, en fonction du volume de données échangées, cette opération peut se révéler volumineuse et nécessiter des transactions importantes. En particulier, la première opération d'actualisation peut concerner un grand nombre de lignes. Il est nécessaire que la base de données soit configurée correctement afin de permettre l'exécution de ces transactions de manière optimale.

Par exemple, avec Oracle :

- Il est impératif que l'ensemble des tables de réplique dans une unité de réplication puisse être contenu dans l'espace de table 'UNDO'.
- Il est recommandé de fournir suffisamment d'espace dans le cache de données afin de permettre à ces transactions de s'exécuter avec un minimum d'accès disque.
- Il est recommandé de réserver suffisamment d'espace pour les groupes de logs 'REDO' pour que les transactions n'aient pas à attendre le processus 'db_writer'.

Distributed data delivery (D3)

La réplication est disponible sur le master D3 ainsi que sur les espaces de données de livraison esclaves. Sur le master, le comportement de réplication est le même que dans un espace de données sémantique standard, mais sur les esclaves, le contenu répliqué est celui de la dernière image diffusée.

Dans un espace de données de livraison esclave, certaines restrictions s'appliquent :

- La politique d'actualisation définie dans le modèle de données n'a aucune influence sur le comportement décrit plus haut : la réplication se déclenche toujours à la suite de la création de l'image.
- L'action `Refresh replicas` n'est pas disponible.
- L'invocation de la méthode `ReplicationUnit.performRefresh`[API] n'est pas autorisée.

Voir aussi *Présentation D3 [p 450]*

Autres limitations de la réplication

- Limitations des bases de données prises en charge [p 339]
- Dans le cas de l'héritage, un champ d'enregistrement répliqué ne peut être une "valeur héritée" (`AdaptationValue.INHERIT_VALUE`). Il ne fait que contenir la valeur héritée dans ce genre de cas. Plus généralement, il n'est pas possible de distinguer l'état héritage de l'état réécriture.

CHAPITRE 47

Évolutions du modèle de données

Ce chapitre présente les modifications qui peuvent être réalisées sur les modèles de données, ainsi que les éventuelles limitations. Les restrictions et/ou impacts potentiels des évolutions du modèle de données dépendent du mode de persistance. Les principes applicables à chaque mode sont les suivants :

- Mode sémantique : flexible et non-bloquant. Peut conduire à une perte de données, par exemple une définition de clé primaire peut évoluer librement, mais tout enregistrement existant, dans un espace de données ou dans un snapshot, qui enfreint la contrainte de clé primaire ne sera plus chargé.
- Tout mode mappé : restrictif et donc bloquant si les données existent et si l'évolution viendrait enfreindre leur intégrité.

Attention

A chaque évolution du modèle de données, il est crucial d'anticiper la perte potentielle de données. Si la perte est avérée, dans le cas où les données existantes doivent être conservées, un plan de migration de données doit être mis en place et appliqué avant la publication ou le déploiement du nouveau modèle de données. Il est important de noter également que les données ne sont pas détruites immédiatement après l'évolution du modèle de données ; en mode sémantique, du moment qu'aucune mise à jour n'est effectuée sur une table dont la définition a évolué, si le modèle de données est publié à un état précédent, alors les données précédentes sont récupérées.

> **Note**
>
> Certains types d'évolutions du modèle de données ne peuvent être réalisés directement dans l'interface utilisateur. Le modèle de données doit alors être exporté, modifié au format XSD, puis ré-importé. En ce qui concerne les modifications d'un modèle de données impactant également sa configuration, en plus de sa structure, le XSD doit être importé dans TIBCO EBX® à partir d'un module. Sans quoi, les modifications de configuration ne seront pas prises en compte.

Voir aussi *Mode mappé* *[p 265]*

Ce chapitre contient les sections suivantes :

1. Types d'évolutions autorisés
2. Limitations/restrictions

47.1 Types d'évolutions autorisés

Cette section décrit les modifications possibles sur les modèles de données après leur création.

Évolutions au niveau du modèle

Les modifications suivantes peuvent être effectuées sur les modèles de données existants :

- Un modèle de données, en mode sémantique, peut être déclaré comme étant en mode relationnel. Les données doivent être transférées manuellement, en exportant puis ré-important un fichier XML ou une archive.
- Le mode relationnel peut être désactivé sur le modèle de données. Les données doivent être transférées manuellement, en exportant puis ré-important un fichier XML ou une archive.
- Des unités de réplication peuvent être ajoutées au modèle de données. Si leur politique d'actualisation est 'onCommit', les répliques correspondantes seront créées et actualisées lors de la prochaine compilation de schéma.
- Les unités de réplication peuvent être retirées du modèle de données. Les répliques correspondantes seront immédiatement abandonnées.
- Le modèle de données peut être supprimé. S'il déclare des unités de réplication, les répliques correspondantes seront immédiatement abandonnées. S'il est relationnel ou contient des tables historisées, cette modification marque les tables mappées associées comme désactivées. Voir Mapping en bases de données [p 431] pour la suppression effective des objets de base de données associés.

Évolutions au niveau de la table

Le modèle de données peut subir les modifications suivantes au niveau de la table :

- Une nouvelle table peut être ajoutée. Lors de la création, la table peut aussi déclarer un ou plusieurs modes mappés.
- Une table existante peut être supprimée. Si elle déclare des unités de réplication, les répliques correspondantes seront immédiatement abandonnées. Cette modification marque la table mappée comme désactivée, si elle est historisée ou relationnelle. Voir Mapping en bases de données [p 431] pour la suppression effective des objets de base de données associés.
- Une table existante en mode sémantique peut être déclarée comme étant en mode relationnel. Les données doivent être transférées manuellement, en exportant puis ré-important un fichier XML ou une archive.
- L'historique peut être activé ou désactivé sur une table. Il ne prendra pas en compte les opérations réalisées pendant sa période de désactivation.
- Une table peut être renommée. Les données doivent être transférées manuellement, en exportant puis ré-important un fichier XML ou une archive, car cette modification est considérée comme étant une combinaison de suppression et de création.

Évolutions au niveau du champ

Le modèle de données peut subir les modifications suivantes au niveau du champ :

- Un nouveau champ peut être ajouté.

- Un champ existant peut être supprimé. En mode sémantique, les données du champ supprimé seront retirées de chaque enregistrement lors de sa prochaine mise à jour. Pour une table de réplique, la colonne correspondante est automatiquement supprimée. En mode historique ou relationnel, le champ est marqué comme désactivé.
- Un champ peut être spécifiquement désactivé de l'historique ou de la réplication qui s'appliquent à sa table, en utilisant l'attribut `disable="true"`. Pour une table répliquée, la colonne correspondante est automatiquement retirée. Pour une table d'historique, la colonne reste présente mais est marquée comme désactivée. Voir Désactivation de l'historique d'un champ ou d'un groupe spécifique [p 274] et Désactiver la réplication pour un champ ou un groupe donné [p 284].
- Les facettes d'un champ peuvent être modifiées, excepté celles listées sous Limitations/restrictions [p 291].

Les modifications énoncées ci-dessus sont acceptées, mais peuvent donner lieu à une perte de données. Les données doivent être transférées manuellement, en exportant puis en ré-important un fichier XML ou une archive, car ces modifications sont considérées comme étant la combinaison d'une suppression et d'une création.

- Un champ peut être renommé.
- Le type d'un champ peut être changé.

Évolutions au niveau de l'index

- Un index peut être ajouté ou renommé.
- Un index peut être modifié, en modifiant ses champs ou en changeant leur ordre. En mode mappé, l'index existant est effacé et un autre est créé.
- Un index peut être supprimé. En mode mappé, un index supprimé est aussi supprimé de la base de données.

47.2 Limitations/restrictions

> **Note**
>
> Toutes les limitations listées dans cette section, qui impactent le mode mappé, peuvent être contournées en purgeant les ressources de base de données de la table mappée. Pour la procédure de purge des ressources de base de données de la table mappée, voir Mapping en bases de données [p 431].

Limitations liées aux évolutions de clé primaire

Lorsqu'une définition de clé primaire est modifiée :

- En mode sémantique, les enregistrements existants sont chargés dans le cache uniquement s'ils :
  - Respectent la contrainte d'unicité de la clé primaire,
  - Sont conformes à la nouvelle structure de la clé primaire.
- En mode mappé, le SGBDR sous-jacent accepte uniquement une modification de clé primaire si tous les enregistrements de la table respectent ses contraintes d'unicité et de non-nullité. En particulier, si une table contient déjà des enregistrements :

- L'ajout d'un nouveau champ à la clé primaire nécessite l'attribution d'une valeur par défaut pour ce champ. Contournement : d'abord ajouter le champ, le valoriser pour les enregistrements existants, puis ajouter le champ à la clé primaire.
- La suppression d'un champ existant de la clé primaire sera rejetée si elle implique de rendre non unique la clé primaire d'enregistrements existants (l'attribution d'une valeur par défaut ne change rien dans ce cas).
- En règle générale, il n'est pas possible de renommer un champ de clé de primaire ; ceci est uniquement possible si ce champ n'était pas nécessaire pour rendre toutes les clés primaires uniques. En effet, le renommage d'un champ se traduit par une combinaison de suppression et de création ; par conséquent, l'opération sera rejetée si elle implique de rendre non unique la clé primaire d'enregistrements existants (l'attribution d'une valeur par défaut ne change rien dans ce cas).

Limitations liées aux évolutions de clé étrangère

- Lorsque la déclaration d'une facette `osd:tableRef` est ajoutée ou modifiée, ou que la clé primaire de la table cible d'une facette `osd:tableRef` est modifiée :
  - En mode sémantique, les valeurs du champ sont uniquement chargées dans le cache si elles respectent la nouvelle structure de la clé primaire cible.
  - En mode mappé, la structure d'un champ de clé étrangère est définie pour correspondre à celle de la clé primaire cible. Un champ unique déclarant une contrainte `osd:tableRef` peut alors être divisé en colonnes, dont le nombre et les types correspondent à ceux de la clé primaire cible. Par conséquent, les cas d'évolution suivants auront un impact sur la structure de la table mappée :
    - déclaration d'une nouvelle contrainte `osd:tableRef` sur un champ de table ;
    - suppression d'une contrainte `osd:tableRef` existante sur un champ de table ;
    - ajout (respectivement suppression) d'une colonne à (respectivement d') une clé primaire référencée par une contrainte `osd:tableRef` existante ;
    - modification du type ou du chemin de n'importe quelle colonne d'une clé primaire référencée par une contrainte `osd:tableRef` existante.

    Ces différents cas d'évolution se traduisent par une combinaison de créations et/ou de suppressions de champs. Par conséquent, les données existantes doivent être transférées manuellement.

Limitations liées aux évolutions au niveau du champ

- En mode mappé, lorsqu'une facette `maxLength`, `length`, `totalDigits` ou `fractionDigits` est modifiée :

  L'acceptation ou non de cette modification dépend du SGBD sous-jacent, ainsi que du type de champ et du contenu de la table.

  Par exemple, Oracle acceptera de changer une VARCHAR(20) en VARCHAR(50), mais ne changera une VARCHAR(50) en VARCHAR(20) que si la table ne contient pas de valeurs faisant plus de 20 caractères.

  PostgreSQL connaît les mêmes limitations. Par ailleurs, toute modification d'un type de champ (incluant les modifications de sa longueur) invalidera toutes les déclarations pré-établies qui y sont liées et nécessitera un redémarrage du serveur d'application.

- Lorsque la cardinalité d'un élément est modifiée :
  - En mode sémantique, ce changement est pris en charge. Cependant, deux cas sont à distinguer :
    - Lors de la transformation d'un élément unique vers une liste agrégée, la valeur unique précédente est conservée et ajoutée à la nouvelle liste agrégée.
    - Lors de la transformation d'une liste agrégée en un élément unique, seule la dernière valeur de la liste agrégée est conservée dans l'élément unique. Les autres valeurs sont perdues.
  - En mode relationnel, les listes agrégées ne sont pas prises en charge. Un message d'erreur est ajouté au rapport de compilation du modèle de données si un élément est changé en liste agrégée.
  - En mode historisé, lorsqu'un élément unique est transformé en liste agrégée, la modification est prise en compte, mais la valeur unique précédente est perdue.

Divers

CHAPITRE **48**

Héritage et résolution de valeur

Ce chapitre contient les sections suivantes :

1. Présentation
2. Héritage de jeu de données
3. Champs hérités
4. Service d'optimisation et refactoring

48.1 Présentation

Le principe de l'héritage est de mutualiser les ressources qui sont partagées par de multiples contextes ou entités. TIBCO EBX® propose des mécanismes pour définir, factoriser et résoudre les valeurs de données : *héritage de jeu de données* et *champs hérités*.

De plus, des *fonctions* peuvent être définies pour calculer des valeurs.

> **Note**
>
> Les mécanismes d'héritage décrit dans ce chapitre ne doivent pas être confondus avec l'héritage structurel, lequel s'applique généralement aux modèles et est proposé dans des diagrammes de classes UML par exemple.

Voir aussi *Héritage (glossaire)* *[p 27]*

Héritage de jeu de données

L'héritage de jeu de données est particulièrement utile quand les données s'appliquent à des contextes globaux d'entreprise, tels que filiales ou partenaires.

En se basant sur une hiérarchie de jeux de données, il est possible de factoriser les données communes au niveau de la racine ou des jeux de données intermédiaires et de définir des données spécialisées dans des contextes donnés.

Les mécanismes d'héritage du jeu de données sont détaillés ci-dessous dans Héritage de jeu de données [p 297].

Champs hérités

Contrairement à l'héritage de jeu de données, qui utilise les relations globales entre jeux de données, les champs hérités utilisent des dépendances plus fines qui sont spécifiques à la structure des données. Cela permet de factoriser et de spécialiser les données au niveau des entités métier.

Par exemple, si le modèle précise qu'un "Produit" est associé à une "FamilleDeProduits", il est possible que certains attributs du "Produit" héritent leurs valeurs des attributs définis dans sa "FamilleDeProduits" associée.

> **Note**
>
> Lors de l'utilisation des deux héritages sur un même jeu de données, l'héritage de champ est prioritaire sur celui du jeu de données.

Valeurs calculées (fonctions)

Dans le modèle de données, il est aussi possible de définir qu'un noeud contient une *valeur calculée*. Dans ce cas, la fonction JavaBean spécifiée sera exécutée chaque fois que la valeur est requêtée.

La fonction peut prendre en compte le contexte courant, tel que les valeurs de l'enregistrement courant ou des calculs utilisant une autre table et envoyer des requêtes à des systèmes tiers.

Voir aussi *Computed values [p 553]*

48.2 Héritage de jeu de données

Déclaration d'héritage de jeu de données

Le mécanisme d'héritage du jeu de données est déclaré dans un modèle de données de la façon suivante :

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

L'élément `osd:inheritance` définit la propriété `dataSetInheritance` qui permet d'activer ou non l'héritage pour les jeux de données utilisant ce modèle de données. Les valeurs suivantes peuvent être spécifiées :

- `all`, indique que l'héritage est activé pour tous les jeux de données qui instancient le modèle de données.
- `none`, indique que l'héritage est désactivé pour tous les jeux de données qui instancient le modèle de données.

Si rien n'est spécifié, le mécanisme d'héritage est désactivé.

Mécanisme de résolution

Le mécanisme de résolution des valeurs héritées du jeu de données a le comportement suivant :

1. Si la valeur est définie localement, elle est retournée.

   Elle peut être explicitement `null`.
2. Sinon, la première valeur définie localement est recherchée en remontant la relation enfant-parent dans la hiérarchie des jeux de données.
3. Si aucune valeur définie n'est trouvée, la valeur par défaut est retournée.

Si aucune valeur par défaut n'est définie, `null` est retourné.

Note : Les valeurs par défaut ne peuvent pas être définies sur :

- Un noeud de clé primaire unique
- Des noeuds auto-incrémentés
- Des noeuds définissant une valeur calculée

Mécanisme de résolution d'enregistrement

Tout comme les valeurs, les enregistrements d'une table peuvent aussi être hérités en tant qu'unité par des contextes multiples, mais aussi être partiellement redéfinis (*écrasé*), définis pour un contexte spécifique (*mode racine*), ou être *occultés*.

Un enregistrement se définit suivant l'un des quatre modes de définition distincts :

enregistrement racine	Défini localement dans la table et n'a aucun parent. Cela signifie qu'aucun enregistrement ayant la même clé primaire n'existe dans la table parente, ou que ce parent est un enregistrement occulté.
enregistrement surchargé	Défini localement dans la table et possède un enregistrement parent. Cela signifie qu'un enregistrement ayant la même clé primaire existe dans la table parente et que ce parent n'est pas un enregistrement occulté. Les valeurs de l'enregistrement surchargé sont héritées de son parent, sauf pour les valeurs qu'il redéfinit explicitement.
enregistrement hérité	Non défini localement dans la table courante et possède un enregistrement parent. Toutes les valeurs sont héritées. Les fonctions sont toujours résolues dans le contexte de l'enregistrement courant et ne sont pas héritées.
enregistrement occulté	Précise que si un parent ayant la même clé primaire est défini, ce parent ne sera pas visible dans les descendants de la table.

Voir aussi *Héritage entre jeux de données [p 143]*

Définir le comportement de l'héritage au niveau de la table

Il est aussi possible de définir des règles de gestion dans la déclaration d'une table dans le modèle de données.

Voir aussi *Properties related to dataset inheritance [p 523]*

48.3 Champs hérités

Le mécanisme d'héritage spécifique permet de récupérer la valeur d'un champ selon ses liens avec les autres tables.

Déclaration d'héritage de champ

L'héritage spécifique doit être défini sur les noeuds terminaux dans le modèle de données sous-jacent et est déclaré de cette façon :

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

L'élément `sourceRecord` est une expression qui décrit comment retrouver l'enregistrement à partir duquel la valeur est héritée. Il s'agit d'une clé étrangère, ou d'une séquence de clés étrangères, de l'élément courant à la table source.

Si `sourceRecord` n'est pas défini dans le modèle de données, les champs hérités sont récupérés à partir de l'enregistrement courant.

L'élément `sourceNode` est le chemin du noeud à partir duquel hériter dans l'enregistrement source.

Les conditions suivantes doivent être remplies pour l'héritage spécifique :

- L'élément `sourceNode` est obligatoire.
- L'expression du chemin de l'enregistrement source doit être un chemin de clés étrangères cohérent, à partir de l'élément courant vers l'enregistrement source. Cette expression doit impliquer uniquement des relations un-à-un et zéro-à-un.
- Le `sourceRecord` ne peut pas contenir d'éléments de liste agrégée.
- Chaque élément du `sourceRecord` doit être une clé étrangère.
- Si le champ hérité est aussi une clé étrangère, le `sourceRecord` ne peut pas faire référence à lui même pour obtenir le chemin vers l'enregistrement source de la valeur héritée.
- Chaque élément du `sourceRecord` doit exister.
- Le noeud source doit appartenir à la table contenant l'enregistrement source.
- Le noeud source doit être terminal.
- Le noeud source doit être accessible en écriture.
- Le type de noeud source doit être compatible avec le type de noeud courant.
- Les cardinalités du noeud source doivent être compatibles avec celles du noeud courant.
- Le noeud source ne peut être le même que celui du champ hérité si les champs desquels on hérite sont récupérés dans le même enregistrement.

Mécanisme de résolution de valeur

Le mécanisme de résolution pour les valeurs de champs hérités a le comportement suivant :

1. Si la valeur est définie localement, elle est retournée.

   Elle peut être explicitement `null`.

2. Autrement, résolution de l'enregistrement source ainsi que la valeur à hériter, en fonction des propriétés définies dans le modèle de données.

3. Le processus est récursif ; si le noeud source ne définit pas de valeur localement, elle est alors résolue en accord avec le comportement d'héritage du noeud source.

48.4 Service d'optimisation et refactoring

EBX® offre un service IU intégré pour optimiser l'héritage des jeux de données dans la hiérarchie des jeux de données. Ce service permet les fonctions suivantes :

- **Gestion des valeurs en doublon :** Détecte et supprime toutes les valeurs qui sont des doublons de la valeur héritée.
- **Mutualisation des valeurs communes :** Détecte et mutualise les valeurs communes parmi les descendants du même ancêtre.

Détails de la procédure

Les jeux de données sont traités de bas en haut, ce qui signifie que si le service est exécuté sur le jeu de données à un niveau *N*, avec *N+1* étant le niveau de ses enfants et *N+2* étant le niveau des petits-enfants, le service traitera d'abord les jeux de données au niveau *N+2* afin de déterminer s'ils peuvent être optimisés par rapport aux jeux de données du niveau *N+1*. Ensuite, une optimisation du niveau *N+1* sera exécutée par rapport au niveau *N*.

> **Note**
>
> - Ces fonctions d'optimisation et de refactoring ne gèrent pas les valeurs par défaut qui sont déclarées dans le modèle de données.
> - Le niveau le plus élevé pris en compte pendant la procédure d'optimisation est toujours le jeu de données sur lequel s'exécute le service. Cela signifie que l'optimisation et le refactoring ne sont pas réalisés entre le jeu de données cible et ses propres ancêtres.
> - L'optimisation de table est réalisée sur des enregistrements ayant la même clé primaire.
> - Les champs hérités ne sont pas optimisés.
> - *Les fonctions d'optimisation et de refactoring ne modifient pas la vue résolue d'un jeu de données, si celle-ci est activée.*

Disponibilité du service

Le service 'Optimize & Refactor' est disponible sur les jeux de données qui ont des jeux de données enfants et dont la propriété 'Activée' est réglée sur 'Non' dans les informations du jeu de données.

Le service est disponible pour tout profil ayant des droits en écriture sur les valeurs du jeu de données courant. Il peut être désactivé en limitant les droits d'accès d'un profil.

> **Note**
>
> Pour des raisons de performance, les droits d'accès ne sont pas vérifiés pour chaque noeud et enregistrement de table.

CHAPITRE 49

Permissions

Les permissions définissent l'accès aux données et les actions disponibles pour chaque utilisateur.

Ce chapitre contient les sections suivantes :

1. Présentation
2. Définition des règles utilisateur
3. Définition des règles programmatiques
4. Résolution des permissions sur données
5. Résolution de permissions pour les services
6. Résolution de permissions pour les actions

49.1 Présentation

Les permissions permettent de définir si une action est autorisée. Elles définissent aussi les droits d'accès d'une entité : cachée, en lecture, ou en lecture-écriture. Les entités principales contrôlées par les permissions sont les suivantes :

- Espace de données
- Jeu de données
- Table
- Groupe
- Champ

Utilisateurs, fonctions et profils

La définition et la résolution des permissions sont basées sur la notion de *profils*, terme générique faisant référence aux utilisateurs ou aux rôles.

Chaque utilisateur peut avoir plusieurs rôles, et un même rôle peut être partagé par plusieurs utilisateurs.

Ces relations sont définies dans le référentiel des rôles et utilisateurs. Voir Users and roles directory [p 425].

Définitions particulières:

- Un *administrateur* est un membre du rôle built-in 'ADMINISTRATEUR'.

- Un *propriétaire d'un jeu de données* est un membre de l'attribut *propriétaire* spécifié dans les informations d'un jeu de données racine. Dans ce cas, le rôle built-in 'PROPRIÉTAIRE' est activé lorsque les permissions sont résolues dans le contexte du jeu de données.
- Un *propriétaire d'un espace de données* est un membre de l'attribut *propriétaire* spécifié pour un espace de données. Dans ce cas, le rôle built-in 'PROPRIÉTAIRE' est activé lorsque les permissions sont résolues dans le contexte de l'espace de données.

Règles de permission

Une règle de permission définit les autorisations accordées à un profil pour une entité donnée.

Les règles de permission définies par l'utilisateur sont créées via l'interface utilisateur. Voir la section Définition des règles utilisateur [p 304].

Les règles de permission programmatiques peuvent être créées par des développeurs. Voir la section Définition des règles programmatiques [p 308].

Résolution des permissions

Les permissions sont toujours résolues dans le contexte d'une session utilisateur authentifiée. Ainsi, les permissions sont principalement basées sur les profils de l'utilisateur.

En général, la résolution est restrictive entre un niveau donné et son niveau parent. Ainsi, pour n'importe quel niveau donné, un utilisateur ne peut pas avoir de permission supérieure à celle résolue au niveau d'un de ses parents.

Les permissions programmatiques sont toujours considérées comme étant restrictives.

> **Note**
>
> Dans l'API Java, la classe `SessionPermissions`[API] permet d'accéder aux permissions résolues.

Voir aussi

Résolution des permissions sur données [p 309]

Résolution de permissions pour les services [p 313]

Résolution de permissions pour les actions [p 315]

Permissions spéciales du propriétaire et de l'administrateur

Sur un jeu de données

Un administrateur ou un propriétaire de jeu de données peut effectuer les actions suivantes sur le jeu de données :

- Gérer ses permissions
- Changer son propriétaire, s'il est un jeu de données racine

- Modifier ses informations générales (libellés et descriptions localisés)

Attention

Bien que la définition des permissions puisse restreindre les droits d'affichage ou d'exécution de certaines actions pour l'administrateur ou le propriétaire du jeu de données, il reste possible pour ces derniers de modifier leur propre accès, puisqu'ils auront toujours accès à la gestion des permissions.

Sur un espace de données

Pour devenir le *super propriétaire* d'un espace de données, un utilisateur doit :

- Être propriétaire de l'espace de données et être autorisé à gérer ses permissions, ou
- Être propriétaire d'un espace de données qui est l'ancêtre de l'espace de données courant et être autorisé à gérer les permissions de cet espace de données ancêtre.

L'administrateur ou le super propriétaire d'un espace de données peut réaliser les actions suivantes sur l'espace de données :

- Gérer ses permissions
- Changer son propriétaire
- Le verrouiller ou le déverrouiller
- Modifier ses informations générales (libellés et descriptions localisés)

De plus, dans un workflow, lors de l'utilisation des scripts task intégrés 'Créer un espace de données' ou 'Créer une image', les permissions résolues sont calculées en utilisant le propriétaire défini dans la configuration du script task plutôt que dans la session courante. Cela s'explique par le fait que, pour ces cas, la session courante est associée à un utilisateur système.

Attention

Bien que la définition des permissions puisse restreindre les droits d'affichage ou d'exécution de certaines actions pour l'administrateur ou le propriétaire de l'espace de données, il reste possible pour ces derniers de modifier leur propre accès, puisqu'ils auront toujours accès à la gestion des permissions.

Impact de la fusion sur les permissions

Lorsqu'un espace de données est fusionné, les permissions du jeu de données de l'espace de données enfant sont fusionnées avec celles du jeu de données de l'espace de données parent si et seulement si l'utilisateur le précise pendant le processus de fusion. Les permissions de l'espace de données parent ne sont jamais impactées.

Si certains éléments sont masqués pour le profil qui tente de réaliser une fusion, l'action ne sera pas réalisable, car l'impact de la fusion sur les données ne sera pas entièrement visible.

Aspects importants concernant les permissions

Il faut garder à l'esprit les recommandations suivantes lorsqu'on travaille avec les permissions :

- Lorsqu'une permission `hidden` est retournée sur un champ pour une session donnée, un utilisateur pourrait "deviner" des informations normalement interdites en filtrant ou en triant sur ce champ dans les cas suivants :

  - En utilisant une `Request`[API] sur cette table avec **les permissions activées** `Request.setSession`[API]. Pour éviter cela, les permissions sur ce noeud doivent être vérifiées explicitement avant d'appliquer le filtre ou le critère de tri.
  - En définissant une vue personnalisée sur cette table dans l'IHM. Pour éviter cela, la permission de définir des vues personnalisées devrait être restreinte pour ces utilisateurs.
- La résolution des libellés personnalisés de tables (propriété 'defaultlabel') et de relations (propriété 'display') ne tient pas compte des permissions. Les champs habituellement masqués du fait des restrictions de droits d'accès, seront affichés dans ces libellés. Aucun de ces libellés ne devrait donc contenir de champ confidentiel. Sinon, une stratégie de permission doit être définie afin de restreindre l'affichage de l'intégralité du libellé.
- Lorsqu'une procédure désactive toutes les vérifications de permission en utilisant `ProcedureContext.setAllPrivileges`[API], le code client doit vérifier que la session de l'utilisateur courant est autorisée à exécuter la procédure.
- Lors de l'exécution d'une action sur table (création, suppression, surcharge, occultation) dans une procédure, le droit d'accès du noeud de la table pour la session utilisateur est ignoré durant la résolution des permissions. Lorsque ce contrôle doit être effectué, le code client doit au préalable appeler explicitement `SessionPermissions.getNodeAccessPermission`[API] dans la procédure.
- Afin d'optimiser la résolution des permissions, un cache dédié est mis en place au niveau de la session ; il ne prend en compte que les règles définies par l'utilisateur, et non les règles programmatiques (celles-ci ne sont pas mises en cache car elles sont contextuelles et dynamiques). Le cycle de vie du cache de session dépend du contexte, voir l'explication ci-dessous :
  - Dans l'IU, le cache est vidé pour chaque événement non-ajax (i.e affichage de page, ouverte de pop-up, etc.).
  - Dans les procédures programmatiques, le cache n'est vidé qu'à la fin de la procédure, à moins d'être explicitement vidé (voir ci-dessous).

> **Attention**
>
> Lors de la modification de permissions dans un contexte de procédure (import d'une archive EBX® ou fusion programmatique d'un espace de données), le cache de la session **doit** être vidé via un appel à `Session.clearCache`[API]. Autrement, ces modifications ne seront pas prises en compte avant la fin de la procédure.

49.2 Définition des règles utilisateur

Tous les niveaux utilisent un schéma similaire, qui permet la définition de règles de permission pour les profils.

Définition des règles utilisateur dans les espaces de données

Pour un espace de données, les permissions administrables pour chaque profil sont les suivantes :

Mode	Autorisation
Écriture	• Peut voir l'espace de données. • Peut accéder aux jeux de données en fonction des permissions du jeu de données.
Lecture seulement	• Peut voir l'espace de données et ses images. • Peut voir les espaces de données enfants, si autorisé par les permissions. • Peut voir le contenu de l'espace de données, sans pouvoir le modifier.
Masqué	• Ne peut voir ni l'espace de données ni ses images. • Si autorisé à voir l'espace de données enfant, peut voir l'espace de données courant sans pouvoir le sélectionner. • Ne peut accéder au contenu de l'espace de données, ni aux jeux de données. • Ne peut réaliser aucune action sur l'espace de données.

Restriction d'accès	Indique si cette association profil-permission de l'espace de données doit être prioritaire sur les autres règles de permissions.
Créer un espace de données enfant	Indique si le profil peut créer des espaces de données enfants à partir de l'espace de données courant.
Créer une image enfant	Indique si le profil peut créer une image de l'espace de données courant.
Initialiser la fusion	Indique si le profil peut fusionner l'espace de données courant avec son espace de données parent.
Exporter une archive	Indique si le profil peut exporter l'espace de données courant comme archive.
Importer une archive	Indique si le profil peut importer une archive dans l'espace de données courant.
Fermer un espace de données	Indique si le profil peut fermer l'espace de données courant.
Fermer une image	Indique si le profil peut fermer une image de l'espace de données courant.
Droits sur les services	Indique si un profil a le droit d'exécuter des services sur l'espace de données. Par défaut, tous les services d'espaces

	de données sont autorisés. Un administrateur ou super propriétaire de l'espace de données courant ou un utilisateur donné autorisé à modifier les permissions sur l'espace de données courant peut modifier ces permissions afin de restreindre les services de l'espace de données pour certains profils.
Permissions de l'espace de données enfant à la création	Lorsqu'un utilisateur crée un espace de données enfant, les permissions de ce nouvel espace de données sont automatiquement attribuées au propriétaire du profil, conformément aux permissions définies sous 'Permissions d'un espace de données enfant à la création' dans l'espace de données parent. Si plusieurs permissions sont définies pour le propriétaire via différents rôles, le profil du propriétaire se comporte de la même manière qu'un autre profil et les permissions sont résolues [p 302] de la même manière.

Définition des règles utilisateur dans les jeux de données

Pour un jeu de données, les permissions administrables pour chaque profil sont les suivantes :

Actions sur les jeux de données

Restriction d'accès	Indique si l'association profil-permission du jeu de données doit être prioritaire sur les autres règles de permissions.
Créer un jeu de données enfant	Indique si le profil a le droit de créer un jeu de données enfant à partir du jeu de données courant.
Dupliquer un jeu de données	Indique si le profil a le droit de dupliquer le jeu de données courant.
Modifier le jeu de données parent	Indique si le profil a le droit de changer le jeu de données parent d'un jeu de données enfant.

Actions sur les tables

Les droits d'actions sur les tables par défaut sont définis au niveau du jeu de données. Il est ensuite possible de surcharger ces droits par défaut pour une ou plusieurs tables données. Les permissions administrables pour chaque profil sont les suivantes :

Créer un enregistrement	Indique si le profil a le droit de créer des enregistrements dans la table.
Surcharger un enregistrement	Indique si le profil a le droit de surcharger les enregistrements de la table.
Occulter un enregistrement	Indique si le profil a le droit d'occulter les enregistrements de la table.
Supprimer un enregistrement	Indique si le profil a le droit de supprimer des enregistrements dans la table.

Droits d'accès sur les valeurs de noeud

Les permissions définies sur des noeuds terminaux spécifiques surchargent les droits d'accès par défaut.

Lecture-écriture	Peut afficher et modifier les valeurs du noeud.
Lecture	Peut afficher les noeuds, mais ne peut pas modifier leurs valeurs.
Masqué	Ne peut pas voir les noeuds.

Permissions sur les services

Un administrateur ou propriétaire de l'espace de données courant peut modifier les permissions par défaut d'un service afin de permettre ou limiter son usage pour certains profils.

Activé	Rend le service disponible pour le profil.
Désactivé	Rend le service indisponible pour le profil. Il ne sera ni affiché dans les menus, ni exécutable depuis un composant web.
Par défaut	Rend le service activé ou désactivé selon la permission par défaut définie lors de sa déclaration. Voir `ActivationContext.setDefaultPermission`[API] pour plus d'informations.

49.3 Définition des règles programmatiques

Les règles programmatiques permettent de définir plus précisément les conditions d'accès à une donnée ou à un service utilisateur en fonction du contexte.

Il existe plusieurs types de règles programmatiques :

- les `AccessRule`[API], décrites ci-après dans la section Définition des règles d'accès aux données [p 308].
- les ServiceActivationRule [p 308], décrites ci-après dans la section Définition des règles d'activation sur service [p 308].
- les `ServicePermissionRule`[API], décrites ci-après dans la section Définition des règles de permission sur service [p 309].

Définition des règles d'accès aux données

Les `AccessRules` sont des règles permettant de définir programmatiquement et en fonction du contexte les droits de lecture/écriture sur un noeud du modèle de données ou sur les enregistrements d'une table.

La définition d'une `AccessRule` s'effectue ainsi :

1. Création d'une règle sous la forme d'une classe Java implémentant l'interface `AccessRule`[API] ou `AccessRuleForCreate`[API].
2. Assignation de cette règle aux noeuds concernés dans l'extension du schéma : `SchemaExtensions`[API].

   Selon la cible (noeud(s) du modèle ou enregistrements) ou le type (`AccessRule` oi `AccessRuleForCreate`) de la règle, plusieurs méthodes comme `SchemaExtensionsContext.setAccessRuleForCreateOnNode`[API] ou `SchemaExtensionsContext.setAccessRuleOnOccurrence`[API] sont disponibles.

   La règle ainsi assignée est dite "locale" et n'est exécutée que lorsque l'entité cible est sollicitée. Voir Résolution des permissions sur données [p 309] pour plus d'informations.

> **Attention**
>
> Une seule `AccessRule` peut être définie pour chaque noeud, espace de données ou enregistrement. Une seule `AccessRuleForCreate` peut être définie pour chaque noeud d'une table. Ainsi, la définition d'une nouvelle règle programmatique d'un certain type remplacera l'existante.

Définition des règles d'activation sur service

Les `ServiceActivationRules` sont des règles permettant de définir si un service est activé ou non pour un espace de données ou un jeu de données. Un service désactivé par ce moyen n'est jamais disponible dans l'entité pour laquelle il est désactivé, quel que soit le profil courant et ce pour l'exécution ou l'affichage, même dans les écrans de permissions.

La définition d'une `ServiceActivationRule` s'effectue ainsi :

1. Création d'une règle sous la forme d'une classe Java implémentant l'interface `ServiceActivationRuleForDataspace`[API] ou `ServiceActivationRuleForDataset`[API], selon le type de service.

2. Assignation de cette règle aux services concernés au niveau de leur déclaration, selon le type de service via les méthodes `ActivationContextOnDataspace.setActivationRule`[API] ou `ActivationContextWithDatasetSet.setActivationRule`[API].

   La règle ainsi assignée sera évaluée lors de l'évaluation de l'activation du service. Voir Résolution de permissions pour les services [p 313] pour plus d'informations.

Définition des règles de permission sur service

Les `ServicePermissionRules` sont des règles avancées permettant de définir dynamiquement les conditions d'affichage et d'exécution d'un service en fonction du contexte (session courante, entité sélectionnée, etc.). Le service doit au préalable être activé pour le contexte courant pour que ce type de règle se déclenche.

La définition d'une `ServicePermissionRule` s'effectue de la façon suivante :

1. Création d'une règle sous la forme d'une classe Java implémentant l'interface `ServicePermissionRule`[API].
2. Assignation de cette règle aux services concernés :
   - Soit, pour les nouveaux services, au niveau de leur déclaration via la méthode `ActivationContext.setPermissionRule`[API].

     La règle ainsi assignée est dite "globale", et est exécutée dés lors que le service est activé pour le contexte courant. Voir Résolution de permissions pour les services [p 313] pour plus d'informations.
   - Soit, pour les services existants, dans **l'extension du schéma** `SchemaExtensions`[API] via les méthodes `SchemaExtensionsContext.setServicePermissionRuleOnNode`[API] et `SchemaExtensionsContext.setServicePermissionRuleOnNodeAndAllDescendants`[API]. Il est ainsi possible d'assigner une règle à n'importe quel service, y compris les services standards fournis par EBX®, sur un ou plusieurs noeuds du modèle de données : un noeud table, un noeud d'association, etc.

     La règle ainsi assignée est dite "locale" et n'est exécutée que dans le contexte du schéma étendu et lorsque le noeud concerné est celui spécifié. Voir Résolution de permissions pour les services [p 313] pour plus d'informations.

> **Attention**
> Une seule `ServicePermissionRule` peut être définie pour chaque noeud du modèle. Ainsi, la définition d'une nouvelle règle programmatique remplacera l'existante.

49.4 Résolution des permissions sur données

Résolution des règles utilisateur

Les droits d'accès définis via l'interface utilisateur sont résolus sur quatre niveaux : espace de données, jeu de données, enregistrement (le cas échéant) et noeud.

Si un profil est associé à des droits d'accès restrictifs à un niveau donné, le minimum de tous les droits restrictifs définis à ce niveau est résolu. Si aucune restriction n'est définie pour ce niveau, le maximum de tous les droits d'accès définis pour ce niveau est résolu.

Lorsqu'une permission restrictive est définie pour un profil, elle devient prioritaire sur les autres permissions potentiellement accordées par les autres rôles de l'utilisateur. D'une manière générale, pour toutes les règles de permission définies par l'utilisateur qui s'appliquent à la session utilisateur courante :

- Si certaines règles avec restrictions sont définies, les permissions minimales de ces règles restreintes sont appliquées.
- Si aucune règle avec restrictions n'est définie, les permissions maximales de toutes les règles correspondantes sont appliquées.

Exemples :

Étant donné deux profils *P1* et *P2* concernant le même utilisateur, le tableau suivant présente les possibilités de résolution de permission de cet utilisateur à un service.

Autorisation de P1	Autorisation de P2	Résolution de permission du service
Activé	Activé	Activé. Les restrictions ne changent rien.
Désactivé	Désactivé	Désactivé. Les restrictions ne changent rien.
Activé	Désactivé	Activé, à moins que l'autorisation de P2 ne soit une restriction.
Désactivé	Activé	Activé, à moins que l'autorisation de P1 ne soit une restriction.

Cette politique de restriction est identique pour les droits d'accès aux données.

Dans un autre exemple, un espace de données peut être masqué pour tous les utilisateur en définissant une association restrictive entre le profil intégré "Profile.EVERYONE" et le droit d'accès "masqué".

Quel que soit le niveau, les droits d'accès les plus restrictifs entre ceux résolus à ce niveau et ceux résolus à un niveau plus élevé sont appliqués. Par exemple, si les permissions d'accès au jeu de données d'un utilisateur sont résolues en accès lecture-écriture, mais que l'espace de données qui le contient n'autorise que l'accès en lecture, l'utilisateur ne bénéficiera que de l'accès en lecture seule pour ce jeu de données.

> **Note**
>
> Le mécanisme d'héritage de jeu données s'applique aux valeurs et aux droits d'accès. Plus précisément, les droits d'accès définis sur un jeu de données seront appliqués sur ses jeux de données enfants. Il est possible de surcharger ces droits dans le jeu de données enfant.

Exemple de résolution des droits d'accès

Dans cet exemple, trois utilisateurs appartiennent aux rôles définis et profils suivants :

Utilisateur	Profil
Utilisateur 1	• utilisateur1 • rôle A • rôle B
Utilisateur 2	• utilisateur2 • rôle A • rôle B • rôle C
Utilisateur 3	• utilisateur3 • rôle A • rôle C

Les droits d'accès des profils pour un élément donné sont les suivants :

Profil	Droits d'accès	Politique de restriction
utilisateur1	Masqué	Oui
utilisateur3	Lecture	Non
Rôle A	Lecture/Écriture	Non
Rôle B	Lecture	Oui
Rôle C	Masqué	Non

Après la résolution basée sur les droits d'accès du rôle et du profil ci-dessus, les droits appliqués à chaque utilisateur sont les suivants :

Utilisateur	Droits d'accès résolus
Utilisateur 1	Masqué
Utilisateur 2	Lecture
Utilisateur 3	Lecture/Écriture

Résolution des droits d'accès aux espaces de données et aux images

Au niveau de l'espace de données, les droits d'accès sont résolus de la manière suivante :

- Si un utilisateur a plusieurs droits définis via plusieurs profils :
  - Si les droits incluent des restrictions, le minimum des associations restrictives profil-droits est appliqué.
  - Autrement, le maximum des associations profil-droits est appliqué.
- Si l'utilisateur n'a pas de droits définis :
  - Si l'utilisateur est un administrateur ou le propriétaire de l'espace de données, l'accès en lecture-écriture est accordé pour cet espace de données.
  - Autrement, l'espace de données sera masqué.

Résolution des droits d'accès aux jeux de données

Au niveau du jeu de données, le même principe qu'au niveau de l'espace de données s'applique. Après la résolution des droits d'accès au niveau du jeu de données seul, les droits d'accès définitifs sont déterminés en prenant les droits les plus bas entre les droits résolus de l'espace de données et les droits résolus du jeu de données. Par exemple, si un espace de données est résolu en lecture seulement pour un utilisateur et qu'un de ses jeux de données est résolu en lecture-écriture, l'utilisateur ne bénéficiera que de l'accès en lecture seule pour ce jeu de données.

Résolution des droits d'accès aux noeuds

Au niveau du noeud, le même principe qu'aux niveaux de l'espace de données et du jeu de données s'applique. Après la résolution des droits d'accès au niveau du noeud seul, les droits d'accès définitifs sont déterminés en prenant les droits les plus bas entre les droits résolus du jeu de données et les droits résolus du noeud.

Des droits d'accès spécifiques peuvent être définis au niveau du noeud. Si aucun droit d'accès spécifique n'est défini, le droit d'accès par défaut est utilisé pour le processus de résolution.

> **Note**
>
> La procédure de résolution diffère légèrement entre les tables et leurs noeuds enfants.

Cas particulier des tables et noeuds de tables

Cette section décrit le processus de résolution utilisé pour un noeud de table donné ou un enregistrement *N* de la table.

Pour chaque règle de permission définie par l'utilisateur correspondant à un des profils de l'utilisateur, les droits d'accès pour *N* seront soit :

1. Les droits d'accès définis localement pour *N* ;
2. Hérités des droits d'accès définis dans le noeud de table ;
3. Hérités des droits d'accès par défaut pour les valeurs du jeu de données.

Toutes les règles de permission correspondantes définies par l'utilisateur sont utilisées pour résoudre les droits d'accès pour *N*. La résolution est effectuée en fonction de la politique de restriction [p 309].

Les droits d'accès résolus définitifs seront les plus bas entre l'espace de données, le jeu de données et le droit d'accès résolu pour *N*.

Résolution des règles programmatiques

Il existe trois niveaux de résolution pour les règles de droit d'accès programmatiques : le jeu de données, l'enregistrement et le noeud. Puisqu'une seule règle d'accès programmatique peut être configurée pour un niveau donné, la dernière règle configurée sera celle utilisée par la procédure de résolution.

Résolution de règle sur jeu de données

Pour un jeu de données, la dernière règle est considérée comme la règle résolue.

Résolution de règle sur enregistrement

Pour un enregistrement, la règle résolue est la plus basse entre la règle résolue configurée sur le jeu de données et la règle configurée sur cet enregistrement. Voir `SchemaExtensionsContext.setAccessRuleOnOccurrence`[API] pour plus d'informations.

Résolution de règle sur noeud

Pour un noeud enfant d'un enregistrement, la règle résolue est la plus basse entre la règle résolue sur l'enregistrement et la règle configurée pour ce noeud.

Pour un noeud enfant d'un jeu de données, la règle résolue est la plus basse entre la règle résolue configurée sur le jeu de données et la règle configurée sur ce noeud. Voir `SchemaExtensionsContext.setAccessRuleOnNode`[API] pour plus d'informations.

Politique d'affichage pour les menus déroulant des clés étrangères

Si un enregistrement est masqué du fait des règles d'accès, il n'apparaîtra pas dans le menu déroulant des clés étrangères.

> **Attention**
> Les droits d'accès résolus sur un jeu de données ou un noeud de jeu de données sont les droits minimum entre les droits d'accès résolus définis dans l'interface utilisateur et les règles programmatiques résolues, le cas échéant.

49.5 Résolution de permissions pour les services

Les services utilisateurs permettent d'exécuter des fonctions spécifiques et avancées à partir de l'interface utilisateur. Selon leur définition, ces services peuvent être appelés depuis un menu, comme action dans un workflow, comme item de perspective, ou encore être exécutés directement depuis une URL comme composant web [p 215].

Voir aussi *Overview [p 589]*

Les permissions d'un service sont résolues dés lors que celui-ci est sollicité dans l'interface utilisateur, c'est à dire :

- Lors de son exécution, juste avant l'affichage du service.

  Si la permission résolue dans le contexte pour l'utilisateur n'est pas `enabled`, un message d'interdiction s'affiche à la place du service.
- Lors de l'affichage des menus si le service est défini comme affichable dans les menus.

Si la permission résolue dans le contexte pour l'utilisateur n'est pas `enabled`, le service n'apparaît pas dans le menu.

Ainsi, à chaque sollicitation la résolution de permissions d'un service se déroule de la manière suivante, dans l'ordre et tant que les conditions sont respectées :

1. L'activation du service doit correspondre au contexte courant. Cette activation tient compte :
   - du type d'entité sélectionnée (jeu de données, table, enregistrement, etc.) ;
   - des règles statiques d'activation définies au sein de la méthode `UserServiceDeclaration.defineActivation`[API];
   - de l'éventuelle règle dynamique d'activation (ServiceActivationRule [p 308]) également définie au sein de la méthode `UserServiceDeclaration.defineActivation`[API].
2. Lorsque le service est activé pour le contexte courant, les permissions pour la session utilisateur sont alors évaluées :
   - Si des permissions ont été définies via l'interface utilisateur pour l'utilisateur courant (ou ses rôles), leur résolution doit retourner `enabled`.

     Pour plus d'informations, se reporter à la section Résolution des règles utilisateur [p 314] ci-après.
   - Si une règle de permission globale [p 309] est définie pour le service, elle doit retourner `enabled` pour le contexte fourni (voir `ServicePermissionRuleContext`[API]).
   - Si une règle de permission locale [p 309] est définie pour le noeud sélectionné, elle doit retourner `enabled` pour le contexte fourni (voir `ServicePermissionRuleContext`[API]).

Résolution des règles utilisateur

Exemple

Dans cet exemple, nous avons deux utilisateurs appartenant à différents rôles et profils :

Utilisateur	Profils
Utilisateur 1	• utilisateur1 • rôle A • rôle B
Utilisateur 2	• rôle C • rôle D

Les permissions associées aux rôles et aux profils au niveau du jeu de données sont les suivantes :

Profil	**Service prédéfini de création** (`@creation`)	**Service prédéfini de duplication** (`@duplicate`)	**Service prédéfini de comparaison** (`@compare`)	**Service personnalisé 1** (`custom1`)	**Service personnalisé 2** (`custom2`)	**Politique de restriction**
utilisateur1	Activé	Désactivé	Activé	Désactivé	Activé	Non
Rôle A	Activé	Activé	Désactivé	Activé	Désactivé	Oui
Rôle B	Activé	Désactivé	Activé	Activé	Désactivé	Oui
Rôle C	Activé	Activé	Désactivé	Désactivé	Désactivé	Non
Rôle D	Activé	Désactivé	Désactivé	Activé	Désactivé	Non

Les services disponibles pour chaque utilisateur après la résolution des droits sont les suivantes :

Utilisateurs	**Services disponibles**
Utilisateur 1	Service prédéfini de création (`@creation`)
	Service personnalisé 1 (`custom1`)
Utilisateur 2	Service prédéfini de création (`@creation`)
	Service prédéfini de duplication (`@duplicate`)
	Service personnalisé 1 (`custom1`)

Voir aussi *Résolution des règles utilisateur* *[p 309]*

49.6 Résolution de permissions pour les actions

Les actions sont les opérations bas-niveau de manipulation d'objets EBX® sur lesquels il est possible de spécifier les droits d'exécution pour un profil. A la différence des permissions sur services utilisateurs, qui ne concernent que l'interface utilisateur, ces droits sont également applicables lorsqu'une opération est effectuée programmatiquement (c'est à dire via une `Procedure`[API]) ou indirectement (par exemple lors d'un import de données, les actions sur table (création, surcharge, occultation et suppression) sont évaluées).

Voici la liste des actions sur lesquelles des droits sont définissables :

Objet de l'action	Actions disponibles
Espace de données	Créer un espace de données enfant
	Créer une image
	Initier une fusion
	Exporter une archive
	Importer une archive
	Fermer l'espace de données
	Fermer l'image
	Créer un jeu de données
Jeu de données	Dupliquer le jeu de données
	Supprimer le jeu de données
	Activer/désactiver le jeu de données
	Créer une vue
Table	Créer un nouvel enregistrement
	Surcharger des enregistrements
	Occulter des enregistrements
	Supprimer des enregistrements

Pour la résolution des permissions sur actions, seules les permissions définies via l'interface utilisateur pour l'utilisateur courant (ou ses rôles) sont prises en compte, la politique de restriction s'appliquant comme pour toute permission définie via l'interface utilisateur.

Pour plus d'informations, se reporter à la section Résolution des règles utilisateur [p 317] ci-après.

Résolution des règles utilisateur

Exemple

Dans cet exemple, nous avons deux utilisateurs appartenant à différents rôles et profils :

Utilisateur	Profils
Utilisateur 1	• utilisateur1 • rôle A • rôle B
Utilisateur 2	• rôle C • rôle D

Les droits associés aux rôles et aux profils sur les actions d'une table donnée sont les suivants :

Profil	Créer un enregistrement	Surcharger un enregistrement	Occulter un enregistrement	Supprimer un enregistrement	Politique de restriction
utilisateur1	Non	Oui	Non	Oui	Non
Rôle A	Oui	Non	Oui	Non	Oui
Rôle B	Non	Oui	Oui	Non	Oui
Rôle C	Oui	Non	Non	Non	Non
Rôle D	Non	Non	Oui	Non	Non

Les actions disponibles pour chaque utilisateur après la résolution des droits sont les suivantes :

Utilisateurs	Actions disponibles
Utilisateur 1	Occulter un enregistrement
Utilisateur 2	Créer un enregistrement
	Occulter un enregistrement

Voir aussi *Résolution des règles utilisateur* *[p 309]*

CHAPITRE **50**

Éditeur de critères

Ce chapitre contient les sections suivantes :

1. Présentation
2. Blocs conditionnels
3. Critères atomiques

50.1 Présentation

L'éditeur de critères est présent dans différentes sections de l'interface utilisateur. Il permet de définir des filtres pour les tables, ainsi que des règles de validation et de calcul de données. Cet éditeur est basé sur la Recommandation W3C XPath 1.0.

Deux types de critères existent : les critères atomiques et les blocs conditionnels.

Voir aussi *Syntaxe XPath supportée [p 249]*

50.2 Blocs conditionnels

Les blocs conditionnels sont constitués de critères atomiques et d'autres blocs conditionnels. Ils expriment une condition basée sur des critères. Les types de blocs suivants existent :

- **Tous les critères sont faux**: Aucun des critères du bloc n'est vrai.
- **Un critère au moins est faux**: Au moins un critère dans le bloc est faux.
- **Tous les critères sont vrais**: Tous les critères dans le bloc sont vrais.
- **Un critère au moins est vrai**: Un ou davantage de critères sont vrais.

50.3 Critères atomiques

Un prédicat atomique est défini par un champ, un opérateur et une expression (soit une valeur ou une formule XPath).

Champ	Spécifie le champ de la table sur lequel le critère s'applique.
Opérateur	Spécifie l'opérateur utilisé. Les opérateurs disponibles dépendent du type de données du champ.
Valeur	Spécifie la valeur ou l'expression. Voir Expression [p 320] ci-dessous.
Code uniquement	Lorsqu'il est coché, il spécifie la recherche des valeurs sous-jacentes pour le champ au lieu des libellés, recherchés par défaut.

Expression

L'expression peut être une valeur fixe ou une formule. Lors de la création d'un filtre, seules les valeurs fixes sont autorisées. Au cours de la création d'une règle de validation ou de calcul, une formule peut être créée à l'aide de l'assistant.

Limitation connue : Le champ de la formule ne valide pas les valeurs en entrée, seuls la syntaxe et les chemins sont vérifiés.

CHAPITRE **51**

Instructions relatives aux performances

Ce chapitre contient les sections suivantes :

1. Principales vérifications sur les performances
2. Vérifications pour l'utilisation d'un espace de données
3. Gestion de la mémoire
4. Validation
5. Mises à jour de masse
6. Accès aux tables

51.1 Principales vérifications sur les performances

Bien qu'TIBCO EBX® soit conçu pour prendre en charge de grands volumes de données, plusieurs facteurs courants peuvent engendrer de faibles performances. Les points clé abordés dans cette section permettront de résoudre les éventuels problèmes de performance.

Extensions programmatiques coûteuses

Pour information, la table ci-dessous détaille les extensions programmatiques qui peuvent être implémentées.

Cas d'utilisation	Extensions programmatiques pouvant être impliquées
Validation	• **contraintes programmatiques** `Constraint`[API] • **valeurs calculées** `ValueFunction`[API]
Accès table	• **règles de permission au niveau de l'enregistrement** `SchemaExtensionsContext.setAccessRuleOnOccurrence`[API] • **filtres programmatiques** `AdaptationFilter`[API]
Affichage du contenu EBX®	• **valeurs calculées** `ValueFunction`[API] • **Composants UI** `UIBeanEditor`[API] • **règles de permission au niveau du noeud** `SchemaExtensionsContext.setAccessRuleOnNode`[API]
Mise à jour des données	• **triggers** `Package com.orchestranetworks.schema.trigger`[API]

Pour les grands volumes de données, les algorithmes lourds ont un impact important sur les performances. Par exemple, la complexité d'un algorithme de contrainte est $O(n^2)$. Si la taille des données est 100, le coût résultant sera proportionnellement de 10 000 (produisant généralement un résultat immédiat). Cependant, si la taille des données est 10 000, le coût résultant sera proportionnellement de 10 000 000.

Une autre cause de baisse des performances est l'appel à des ressources externes. L'utilisation d'un cache local résout généralement ce type de problèmes.

Si l'un des cas d'utilisation ci-dessus mène à une baisse des performances, il est recommandé d'identifier le problème soit via l'analyse du code, soit en utilisant un outil de profiling Java.

Intégration de l'annuaire

L'authentification et la gestion des permissions font appel à l'annuaire des rôles et utilisateurs [p 425].

Si l'implémentation spécifique d'un annuaire est déployée et accède à un annuaire externe, il peut être utile de mettre en place un cache local. En particulier, une des méthodes les plus fréquemment appelées est `Directory.isUserInRole`[API].

Listes agrégées

Dans un modèle de données, lorsque la contrainte de cardinalité d'un élément `maxOccurs` est supérieure à 1 et qu'aucune `osd:table` n'est déclarée pour cet élément, elle est implémentée en tant que `List` Java. Ce type d'élément est appelé liste agrégée [p 516], par opposition à une table.

Il est important de prendre en compte qu'il n'existe aucune optimisation spécifique lors de l'accès aux listes agrégées en termes d'itérations, d'affichage utilisateur, etc. En dehors des questions de

performance, les listes agrégées sont limitées en ce qui concerne les nombreuses fonctionnalités prises en charge par les tables. Voir introduction aux tables [p 519] pour une liste de ces fonctionnalités.

Attention

Pour les raisons indiquées ci-dessus, les listes agrégées devraient être utilisées uniquement sur de petits volumes de données simples (une ou deux douzaines d'enregistrements), sans exigence spécifique concernant leur identification, résolution, permissions, etc. Pour des volumes de données plus importants (ou des fonctionnalités plus avancées), il est conseillé d'utiliser les déclarations `osd:table`.

51.2 Vérifications pour l'utilisation d'un espace de données

Les espaces de données [p 98] disponibles en mode sémantique sont des outils précieux pour gérer les cycles de vie des données complexes. Bien que cette fonctionnalité apporte une grande flexibilité, elle implique aussi un coût supplémentaire, qui doit être pris en compte pour l'optimisation des processus métiers.

Cette section résume les problèmes de performance les plus courants qui peuvent apparaître dans le cas d'une utilisation intensive de plusieurs espaces de données contenant de grandes tables ; et de quelle façon les éviter.

Note

Parfois, l'utilisation d'espaces de données n'est pas strictement nécessaire. Citons le cas extrême où chaque transaction déclencherait les actions suivantes :

1. Un espace de données est créé.
2. La transaction modifie certaines données.
3. L'espace de données est fusionné, fermé, puis supprimé.

Dans ce cas, aucune référence à l'espace de données ne sera nécessaire par la suite ; son utilisation pour modifier de données isolées n'est donc pas nécessaire. Ainsi, l'utilisation de `Procedure`[API] fournit déjà une isolation suffisante pour éviter les conflits entre opérations concurrentes. Il serait alors plus efficace de réaliser les modifications directement dans l'espace de données cible, et de ne pas conserver les étapes de branche et de fusion.

Voici une analogie à l'attention des développeurs, portant sur un outil de gestion de code source (CVS, SVN, etc.) : pour réaliser une modification simple n'impactant que quelques fichiers, il suffit de le faire directement dans la branche principale. En fait, ce ne serait ni pratique ni durable, en ce qui concerne le tag/la copie des fichiers, si chaque modification de fichier impliquait de brancher l'ensemble du projet, de modifier les fichiers, puis de fusionner la branche dédiée.

Mémoire insuffisante

Lorsqu'une table est en mode sémantique (par défaut), le cache de la mémoire Java d'EBX® est utilisé. Cela garantit un accès aux données bien plus efficace lorsque ces données sont déjà chargées dans le cache. Toutefois, s'il n'y a pas assez d'espace pour les données courantes, des rechargements intempestifs de données depuis la base de données sous-jacente vers le cache de la JVM peuvent avoir un lourd impact sur les performances globales.

Cette surcharge d'échanges de mémoire (swap) ne peut se produire que pour les tables dans un espace de données ayant une stratégie de chargement à la demande [p 325].

Ce type de problème peut être détecté via le fichier log de monitoring [p 325]. Si cela se produit, différentes actions peuvent être envisagées :

- réduire le nombre d'espaces de données enfants contenant des tables de grande taille ;
- réduire le nombre d'index spécifiquement définis pour les tables de grande taille ;
- utiliser le mode relationnel au lieu du mode sémantique ;
- ou (de manière évidente) allouer davantage de mémoire, ou optimiser la mémoire utilisée par les applications pour les objets non-EBX®.

Voir aussi

Gestion de la mémoire [p 324]

Mode relationnel [p 267]

Annulation de transaction

En mode sémantique, lorsqu'une transaction a réalisé des mises à jour dans l'espace de données courant et qu'elle est interrompue, les index chargés des tables modifiées sont réinitialisés. Si les mises à jour d'une table de grande taille sont souvent annulées et, qu'en même temps, les accès à cette table sont fréquents, alors le travail relatif à la reconstruction de l'index provoquera un ralentissement de l'accès à la table. Par ailleurs, l'allocation de mémoire induite et l'activité de "garbage collector" peuvent mener à une réduction globale des performances.

Voir aussi

Garde fonctionnelle et exceptions `TableTrigger.guardAndException`[API]

`Procedure`[API]

Réorganisation des tables de la base de données

Comme avec toute base de données, l'insertion et la suppression de volumes de données importants peuvent mener à une fragmentation des données, qui peut détériorer les performances avec le temps. Pour résoudre ce problème, la réorganisation des tables impactées de la base de données est nécessaire. Voir Monitoring and cleanup of the relational database [p 403].

Une spécificité d' EBX® est que la création d'espaces de données et d'images ajoute de nouvelles entrées aux tables `HTA` et `ATB`. Si l'on constate une baisse des performances, il peut donc être nécessaire de planifier une réorganisation de ces tables, pour les référentiels de taille importante dans lesquels de nombreux espaces de données sont créés et supprimés.

Voir aussi *Monitoring and cleanup of the relational database [p 403]*

51.3 Gestion de la mémoire

Stratégie de chargement

L'administrateur peut préciser la stratégie de chargement de l'espace de données ou de l'image dans ses informations. La stratégie par défaut est de charger et de décharger les ressources à la demande.

Pour les ressources fréquemment utilisées, une stratégie de *chargement forcé* est généralement recommandée.

La table suivante détaille les modes de chargement disponibles en mode sémantique. Le serveur d'applications doit être redémarré afin de prendre en compte tout changement de stratégie de chargement.

Chargement et déchargement à la demande	Dans ce mode par défaut, chaque ressource d'un espace de données est uniquement chargée ou construite si nécessaire. Les ressources de l'espace de données sont référencées par le logiciel via la classe Java standard `SoftReference`. Par conséquent, chaque ressource peut être déchargée "à la discrétion du "garbage collector" en réponse à la demande mémoire". L'avantage principal de ce mode est sa capacité à libérer de la mémoire lorsque cela est nécessaire. En contrepartie, cela implique un coût de chargement/construction lorsque la ressource accédée n'a pas encore été chargée depuis le démarrage du serveur, ou si elle a été déchargée depuis.
Chargement forcé	Si la stratégie de chargement forcé est activée pour un espace de données ou une image, ses ressources sont chargées de manière asynchrone au démarrage du serveur. Chaque ressource de l'espace de données est conservée en mémoire jusqu'à l'arrêt du serveur ou la fermeture de l'espace de données. Ce mode est particulièrement recommandé pour les espaces de données ayant une longue durée de vie et/ou ceux qui sont fréquemment utilisés, à savoir tout espace de données utilisé comme référence.
Chargement forcé et pré-validation	Cette stratégie est similaire à la stratégie de chargement forcé, la différence étant que le contenu de l'espace de données chargé ou de l'image sera aussi validé au démarrage du serveur.

Monitoring

Les indications relatives au chargement/déchargement du cache d'EBX® sont fournies par le monitoring de la base de données sous-jacente, ainsi que par la catégorie de logging du 'monitoring' [p 377].

Si le nombre d'objets *cleared* et *built* demeure élevé sur une longue période, cela signifie qu'EBX® effectue de nombreux chargements et déchargements d'objets dans le cache.

Réglage de la mémoire

La mémoire maximum pouvant être allouée à la JVM est habituellement spécifiée en utilisant l'option de ligne de commande Java `-Xmx`. Comme c'est le cas pour tout processus intensif, il est important

que la taille spécifiée par cette option n'excède pas la mémoire physique disponible, ainsi le processus Java n'utilisera pas le disque pour des échanges mémoire (swap) au niveau du système d'exploitation.

Le réglage du "garbage collector" peut aussi améliorer les performances globales. Ce réglage doit être adapté au cas d'utilisation ainsi qu'à l'environnement spécifique Java Runtime utilisé.

51.4 **Validation**

Le mécanisme de validation incrémentale permet d'optimiser le travail nécessaire lors des mises à jour. Le processus de validation incrémentale a le comportement suivant :

- Le premier appel au rapport de validation d'un jeu de données effectue une validation complète du jeu de données. La stratégie de chargement [p 324] peut aussi spécifier qu'un espace de données doit être prévalidé au démarrage du serveur.
- Les mises à jour de données maintiendront le rapport de validation de manière transparente et asynchrone, dans la mesure où les noeuds mis à jour définiront des dépendances explicites. Par exemple, les facettes standard et statiques, les contraintes de clé étrangère, les facettes dynamiques, les noeuds de sélection définissent des dépendances explicites.
- Si une mise à jour de masse est effectuée ou s'il y a trop de messages de validation, le processus de validation incrémentale est interrompu. Le prochain appel au rapport de validation déclenchera alors une validation complète.
- Si une transaction est annulée, l'état de validation du jeu de données mis à jour est réinitialisé. Le prochain appel au rapport de validation déclenchera aussi une validation complète.

Certains noeuds sont systématiquement revalidés, et ce, même si aucune mise à jour n'a eu lieu depuis la dernière validation. Ce sont des noeuds à *dépendances inconnues*. Un noeud a des dépendances inconnues si :

- Il définit une **contrainte programmatique** `Constraint`[API] dans le mode *dépendances inconnues* par défaut,
- Il déclare une **valeur calculée** `ValueFunction`[API], ou il déclare une facette dynamique qui dépend d'un noeud qui est lui-même une **valeur calculée** `ValueFunction`[API].
- Il est un Champs hérités [p 298], ou il déclare une facette dynamique qui dépend d'un noeud qui est lui-même un Champs hérités [p 298].

Par conséquent, pour les tables de grande taille (au delà de l'ordre de grandeur 10^5), il est recommandé d'éviter les noeuds ayant des dépendances inconnues (ou au moins d'en minimiser le nombre). Pour les contraintes programmatiques, le développeur peut définir deux modes alternatifs qui réduiront de manière drastique les coûts de validation incrémentale : mode *dépendance locale* et *dépendances explicites*. Pour plus d'informations, voir **Dépendances et validation** `DependenciesDefinitionContext.dependencies`[API].

> **Note**
>
> Il est possible, pour un utilisateur administrateur, de réinitialiser manuellement le rapport de validation d'un jeu de données. Cette option est disponible dans la section du rapport de validation dans EBX®.

51.5 **Mises à jour de masse**

Les mises à jour de masse peuvent inclure plusieurs centaines de milliers d'insertions, de modifications et de suppressions. Ces mises à jour sont assez rares (habituellement imports de données initiales),

ou sont réalisées de manière non-interactive (traitements par lot de nuit). Ainsi, l'importance des performances pour ces mises à jour est moins critique que pour des opérations interactives ou fréquentes. Toutefois, tout comme le traitement par lot classique, elles peuvent donner lieu à certains problèmes spécifiques.

Mode batch

Pour les tables en mode relationnel, l'implémentation des insertions, mises à jours et suppressions s'appuie sur la fonction `JDBC` de traitement par lot (batch). Pour des procédures à fort volume, ceci améliore grandement les performances, en limitant le nombre d'allers-retours entre le serveur d'applications et la base de données.

Afin d'exploiter totalement cette fonctionnalité, il est possible d'activer le mode batch sur des procédures à fort volume. Voir `ProcedureContext.setBatch`[API]. Ceci désactive le test d'existence avant insertion d'un enregistrement, et réduit ainsi le nombre de requêtes émises vers la base de données. Le traitement par lot en est rendu encore plus efficace.

Limites de transaction

Il n'est généralement pas recommandé d'utiliser une seule transaction lorsque le nombre de mises à jour atomiques dans la transaction est supérieur à l'ordre de grandeur 10^4. Les transactions de taille importante nécessitent beaucoup de ressources, particulièrement mémoire, d'EBX® ainsi que de la base de données sous-jacente.

Afin de réduire la taille de la transaction, il est possible de :

- Définir la propriété ebx.manager.import.commit.threshold [p 386]. Cependant, cette propriété n'est utilisée que pour les imports d'archive interactifs réalisés à partir de l'interface utilisateur d'EBX®.
- Définir explicitement un **commit threshold** `ProcedureContext.setCommitThreshold`[API] à l'intérieur de la procédure batch.
- Limiter structurellement le périmètre de la transaction en implémentant `Procedure`[API] pour une partie de la tâche et l'exécuter aussi souvent que nécessaire.

D'autre part, définir une taille de transaction très basse peut aussi altérer les performances à cause des tâches persistantes qui doivent être exécutées sur chaque commit.

> **Note**
>
> Si des commits intermédiaires posent problème du fait que l'atomicité transactionnelle n'est plus garantie, il est recommandé d'exécuter la mise à jour de masse à l'intérieur d'un espace de données dédié. Cet espace de données sera créé juste avant la mise à jour de masse. Si la mise à jour ne s'achève pas avec succès, l'espace de données doit être fermé, et il faudra rententer la mise à jour après avoir corrigé la cause de l'échec initial. Si la mise à jour aboutit avec succès, l'espace de données peut alors être fusionné sans risque avec l'espace de données d'origine.

Triggers

Si nécessaire, les triggers peuvent être désactivés en utilisant la méthode `ProcedureContext.setTriggerActivation`[API].

51.6 Accès aux tables

Fonctionnalités

On accède généralement aux tables via EBX® et aussi via l'API `Request`[API] et les services de données. Cet accès implique un ensemble unique de fonctions, incluant un processus de *résolution dynamique*. Ce processus a le comportement suivant :

- **Héritage** : L'héritage dans l'arborescence du jeu de données prend en compte les enregistrements et les valeurs définies dans le jeu de données parent, en utilisant un processus récursif. De même, dans un jeu de données racine, un enregistrement peut hériter de certaines de ses valeurs à partir des valeurs par défaut du modèle de données, définies par l'attribut `xs:default`.
- **Calcul de valeur** : Un noeud déclaré en tant que `osd:function` est toujours calculé à la volée lorsque l'on accède à la valeur. Voir `ValueFunction.getValue`[API].
- **Filtrage** : Un prédicat XPath [p 249], un **filtre programmatique** `AdaptationFilter`[API], ou une **règle de permission** `SchemaExtensionsContext.setAccessRuleOnOccurrence`[API] niveau enregistrement nécessitent une sélection d'enregistrements.
- **Tri** : Un tri des enregistrements qui en résultent peut être réalisé.

Accès aux tables en mode sémantique

Architecture et conception

Afin d'améliorer la vitesse des opérations sur les tables, des index sont gérés par le moteur d'EBX®.

Les fonctions avancées d'EBX®, telles que le cycle de vie avancé (images et espaces de données), l'héritage de jeu de données, et la modélisation souple de schéma XML, ont conduit à une conception spécialisée des mécanismes d'indexation. Cette conception peut être résumée de la façon suivante :

- Les *index* maintiennent une structure de données en mémoire, relative à une table complète.
- Un index n'est pas sauvegardé et sa création nécessite le chargement de tous les blocs de table de la base de données.

Attention

Un accès plus rapide aux tables est assuré si les index sont prêts et maintenus dans le cache mémoire. Comme mentionné ci-dessus, il est important pour la Machine Virtuelle Java d'avoir un espace alloué suffisant pour qu'elle ne libère pas les index trop rapidement.

Facteurs de performance

L'optimiseur de requêtes préfère l'utilisation des index lors du calcul du résultat d'une requête.

Attention

- Seuls les filtres XPath sont pris en compte pour l'optimisation de l'index.
- Les index des clés non-primaires ne sont pas pris en compte pour les jeux de données enfants.

En supposant que les index soient déjà construits, les impacts sur les performances sont les suivants :

1. Si la requête n'inclut ni filtrage, ni règles programmatiques, ni tri, alors l'accès à ses premières lignes (celles récupérées par une vue paginée) est presque instantané.
2. Si la requête peut être résolue sans étape de tri supplémentaire (c'est le cas s'il n'y a pas de critère de tri, ou si ses critères de tri se rapportent à ceux de l'index utilisé pour le calcul de la requête), l'accès aux premières lignes d'une table devrait être rapide. Plus précisément, cela dépend du coût de l'algorithme de filtrage spécifique exécuté lors de la récupération d'au moins 2000 enregistrements.
3. Les deux cas ci-dessus garantissent un temps d'accès indépendant de la taille de la table, tout en fournissant une vue triée par l'index utilisé. Si un tri supplémentaire est requis, le temps nécessaire au premier accès dépend de la taille de la table, conformément à une fonction `Nlog(N)`, où `N` est le nombre d'enregistrements dans la vue résolue.

> **Note**
>
> Les requêtes paginées ajoutent automatiquement la clé primaire à la fin du critère spécifié, afin d'assurer un tri cohérent. Ainsi, les champs de clé primaire devraient aussi être ajoutés à la fin de tout index destiné à améliorer les performances des requêtes paginées. Celles-ci incluent les vues tabulaires et hiérarchiques, ainsi que les menus déroulants pour les clés étrangères.

Si les index ne sont pas encore construits, ou ont été déchargés, un temps supplémentaire est nécessaire. Le temps de construction est `O(Nlog(N))`.

L'accès aux blocs de données de la table est nécessaire lorsque la requête ne peut pas être calculée sur un index unique (que ce soit pour résoudre une règle, filtrer ou trier), ainsi que pour la construction de l'index. Si les blocs de table ne sont pas présents en mémoire, un temps supplémentaire est nécessaire afin d'aller les chercher dans la base de données.

Il est possible d'obtenir des informations via les logs de catégorie "request" et "monitoring" [p 325].

Autres opérations sur les tables

La création de nouveaux enregistrements ou l'insertion d'enregistrements dépend de l'index de la clé primaire. Ainsi, une création devient quasi immédiate si cet index est déjà chargé.

Accès REST aux tables historisées

Les informations de fusion dans la table d'historique (le champ `merge_info`) a un coût d'accès potentiellement élevé. Pour améliorer les performances et si le code client n'a pas besoin de ce champ, le paramètre `includeMergeInfo` [p 699] doit être défini à `false`.

Pour plus d'informations, voir Historique [p 273].

Accès aux tables en mode relationnel

Lors du calcul du résultat d'une requête, le moteur d'EBX® délègue ce qui suit au SGBDR :

- Le traitement de tous les critères de tri des requêtes, en les traduisant en clause `ORDER BY`.

- Dans la mesure du possible, le traitement des filtres de requêtes, en les traduisant en clause `WHERE`.

> **Attention**
> Seuls les filtres XPath sont pris en compte pour l'optimisation de l'index. Si la requête comprend des filtres non optimisables, les lignes de la table seront récupérées dans la base de données et filtrées dans la mémoire Java par EBX®, jusqu'à ce que la taille de page demandée soit atteinte. Ceci n'est pas aussi efficace que le filtrage côté base de données (en particulier en ce qui concerne les E/S).

L'information sur la requête SQL transmise est enregistrée dans les logs de catégorie *persistance*. Voir Configuring the EBX® logs [p 377].

Indexation

Afin d'améliorer la vitesse des opérations sur les tables, des index peuvent être déclarés sur une table au niveau du modèle de données. Cela déclenchera la création d'un index de la table correspondante en base de données.

Lors de la conception d'un index visant à améliorer les performances d'une requête donnée, les mêmes règles que pour la conception d'un index de base de données classique s'appliquent.

Définir une taille de récupération (fetch size)

Afin d'améliorer les performances, la taille de récupération (fetch size) doit être définie en fonction de la taille attendue pour le résultat de la requête sur une table. Si aucune taille de récupération n'est définie, la valeur par défaut sera utilisée.

- En mode sémantique, la valeur par défaut est 2000.
- En mode mappé, la valeur par défaut est attribuée par le pilote JDBC : 10 pour Oracle et 0 pour PostgreSQL.

> **Attention**
> Dans PostgreSQL, la valeur par défaut de 0 charge le pilote JDBC de récupérer l'ensemble des résultats en une seule fois, ce qui pourrait causer une `OutOfMemoryError` lors de la récupération de grandes quantités de données. D'autre part, l'utilisation de fetchSize sur PostgreSQL invalidera les curseurs côté serveur en fin de transaction. Si, dans le même thread, on récupère d'abord un résultat défini avec un fetchSize et que l'on exécute ensuite une procédure qui committe la transaction, alors l'accès au résultat suivant générera une erreur.

Voir aussi

`Request.setFetchSize`[API]

`RequestResult`[API]

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 dataspaces and to consider using a backup file system.

See also Cleaning up dataspaces, snapshots, and history [p 403] and Deleting dataspaces, snapshots, and history [p 404].

It is also possible to archive files of the file system type in order to reduce the storage costs, see EBX® monitoring [p 402].

Administration tasks can be scheduled by means of the task scheduler, using built-in tasks, see Task scheduler [p 439].

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 dataspaces, points of reference, and monitoring), while continuously monitoring the repository health reports (`./ebxLog/monitoring.log`).

See Gestion de la mémoire [p 324].

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 395].

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 438] 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 436] 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 376] and Audit trail [p 445] for more information.

52.3 Data model

Publication management

The management of publications of embedded data models [p 91]. See Data model administration [p 429] 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 497] for more information.

52.4 Perspectives

EBX® offers extensive UI customization options. Simplified interfaces (Recommended perspectives) [p 421] 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 410] 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 409] 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 375]*

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	Minimum supported version is 44 Compatibility mode is not supported.
Mozilla Firefox ESR 68 (see details)	As Mozilla Firefox is updated frequently, Cloud Software Group, 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, Cloud Software Group, 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.

Zoom troubleshooting

Zooming in or out may cause some minor display issues (for example extra scrollbar or misalignment). Those issues can be fixed by refreshing the screen using the provided navigation links.

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®.

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.R5 or higher, WebLogic Application Server 12cR2 or higher, JBoss EAP 6.0 or higher. See Java EE deployment overview [p 351].
- 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 398] 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.

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 `xs:string` elements, Oracle does not support the distinction between empty strings and `null` values. See Empty string management [p 553] 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 11 or higher.	When using PostgreSQL as the underlying database, a request fetch size must be set, otherwise the JDBC driver will fetch the whole result set at once. This could lead to an `OutOfMemoryError` when retrieving large amounts of data. Also, see this limitation [p 292] regarding the evolution of datamodels in mapped modes. See `Request.setFetchSize`[API]. 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.
Amazon Aurora PostgreSQL (compatible with PostgreSQL 11.16 or higher).	The comments in the above section for PostgreSQL apply.
Google Cloud SQL for PostgreSQL 11 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."

	The SAP Hana JDBC driver uses the local timezone of the JVM to handle timestamp SQL columns. Hence, for the specific use cases described in the section Accès SQL aux données en mode relationnel [p 271], the JVM powering the Hana JDBC driver - that is the JVM powering EBX® - should be started with the property `user.timezone` set to UTC. This configuration is not free of side effects: for example, the timestamps shown in the EBX® logs will be in UTC instead of the local timezone.
Microsoft SQL Server 2014 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 relational and 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 relational and history tables. The default database collation can be specified when the database is created. Otherwise, the database engine server collation is used. To avoid naming conflicts or unexpected 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 2.1.212 or higher.	H2 is not supported for production environments. The default H2 database settings do not allow consistent reads when records are modified. Relational tables are locked following a pessimistic model. To prevent concurrency issues, it is possible to activate the MVCC feature. Note, however, that the H2 documentation states this feature is not yet fully tested.

For other relational databases, 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), except in the specific use cases described in the section Accès SQL aux données en mode relationnel [p 271].

Voir aussi

Repository administration [p 398]

Data source of the EBX® repository [p 349]

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 336]

54.2 Software components

EBX® uses the following components:

- Library `ebx.jar`
- Embedded [p 344] and required [p 344] third-party Java libraries
- EBX® built-in web applications [p 347] and optional custom web applications [p 347]
- EBX® main configuration file [p 371]
- EBX® repository [p 398]
- Default user and roles directory [p 425], integrated within the EBX® repository, or a third-party system (LDAP, RDBMS) for the user authentication

Voir aussi *Supported environments [p 336]*

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_5.9.23_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.

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 2.1.210 validated. Note that H2 is not supported in production environments. https://www.h2database.com/ If the repository has been created using H2 version 1.x, this process must be applied, to migrate to the H2 v2 format.
Oracle JDBC	Oracle database 12cR2 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 `ojdbc8.jar` 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 `mssql-jdbc-8.4.1.jre8.jar` or `mssql-jdbc-8.4.1.jre11.jar` library, depending on the Java runtime environment version you use. Download Microsoft JDBC Driver 8.4.1 for SQL Server (zip).
PostgreSQL	PostgreSQL 9.6 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 349]*

Configuring the EBX® repository *[p 373]*

SMTP and emails

The library for JavaMail 1.5.6 email management is required.

The following libraries are used by email features in EBX®. See Activating and configuring SMTP and emails [p 380] for details on the configuration.

- `javax.mail.jar`, version 1.5.6, from August 10, 2016
- `smtp.jar`, version 1.5.6, from August 10, 2016
- `pop3.jar`, version 1.5.6, from August 10, 2016

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
- `ibmjsse.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 http://www.oracle.com/technetwork/java/javaee/overview for more information.
- For a Servlet/Java Server Pages (JSP) implementation using Java SE (Java Platform Standard Edition), for example Apache Tomcat, a JMS provider library such as Apache ActiveMQ may need to be added. See http://www.oracle.com/technetwork/java/javase/overview for more information.

> **Note**
>
> In 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, introduces 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 12c R2 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 351] 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.

Attention

Web application deployment on `/` path context is no more supported. The path context must not be empty nor equals to `/`. Moreover, web applications deployment on paths of different depth is deprecated. Every web application path context must be set on the same path depth.

For more information, see the note on repackaging the EBX® EAR [p 352] 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 348] 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 641] and Web Services Description Language WSDL generation [p 633] standards or using the Built-in RESTful services [p 683]. **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 485]

Declaring modules as undeployed [p 387]

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 337]

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`:

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

Configuring the application server for 'FrontServlet'

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

  For example, on Tomcat, this configuration is 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. 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
`jdbc/EBX_REPOSITORY`	Weblogic: `EBX_REPOSITORY` JBoss: `java:/EBX_REPOSITORY`	JDBC data source for EBX® Repository. Java type: `javax.sql.DataSource`

Voir aussi

Configuring the EBX® repository [p 373]

Rules for the database access and user privileges [p 399]

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 380] 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
`mail/EBX_MAIL_SESSION`	Weblogic: `EBX_MAIL_SESSION` JBoss: `java:/EBX_MAIL_SESSION`	Java Mail session used to send emails from EBX®. Java type: `javax.mail.Session`

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 381] 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 350] 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 381] 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 381] in the EBX® main configuration properties for more information on this property.

Reserved resource name	Default JNDI name	Description	Required
`jms/EBX_QueueIn`	Weblogic: `EBX_QueueIn` JBoss: `java:/jms/EBX_QueueIn`	JMS queue for incoming SOAP requests sent to EBX® by other applications. Java type: `javax.jms.Queue`	No
`jms/EBX_QueueFailure`	Weblogic: `EBX_QueueFailure` JBoss: `java:/jms/EBX_QueueFailure`	JMS queue for failures. It contains incoming SOAP requests for which an error has occurred. This allows replaying these messages if necessary. Java type: `javax.jms.Queue` **Note:** For this property to be read, the main configuration must also activate the queue for failures through the property `ebx.jms.activate.queueFailure`. See JMS [p 381] 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 353]
- Installation note for Tomcat 8.x [p 357]
- Installation note for WebSphere AS 9 [p 361]

- Installation note for WebLogic 12c R2 [p 367]

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 485] section.

CHAPITRE **55**

Installation note for JBoss EAP 7.1.x

Ce chapitre contient les sections suivantes :

1. Overview
2. Requirements
3. Installation
4. Configuration for EBX®
5. Updating EBX® Enterprise Application aRchive
6. Deploying EBX®
7. Start EBX®

55.1 Overview

Attention

- This chapter describes a quick installation example of TIBCO EBX® on the JBoss Application Server.
- It does not replace the documentation of this 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.
- The complete description of the components required by EBX® is given in the following chapter: Java EE deployment [p 343].
- 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.

- JBoss Application Server installation
- `EBX_HOME` directory configuration: copy `ebx.properties`
- Java Virtual Machine properties configuration
- JNDI entries configuration
- Data source and JDBC provider creation

- EBX.ear application update
- EBX.ear application deployment
- EBX® application start

55.2 Requirements

- 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 disable)

Voir aussi *Supported environments [p 336]*

55.3 Installation

This quick installation example is performed for a Linux operating system.

1. To download JBoss EAP 7.1, please first download `Installer` jar version 7.1.0 from: https://developers.redhat.com/products/eap/download/
2. Run the `Installer` using *java -jar* command line.
   For further installation details, please 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 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' at JBoss EAP 7.1 installation root path,
   12. Click on 'done'.

55.4 Configuration for EBX®

EBX® home directory creation and 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 then have the following text file:

   `/opt/ebx/home/ebx.properties`.

3. Edit the `ebx.properties` file to override the default database if needed. By default, the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported database and it is required to comment the `h2.standalone` one.

Java Virtual Machine properties 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 a running server as a service).
2. Add 'ebx.properties' and 'ebx.home' properties to 'JAVA_OPTS' respectively set with `ebx.properties` file's path and *EBX_HOME* directory's path.

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"/>
```

   Warning: 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"/>
```

Data source and JDBC provider creation

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, be sure 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>
```

55.5 Updating EBX® Enterprise Application aRchive

1. Copy from *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.

   Mail: see SMTP and emails [p 345] for more information.

   SSL: see Secure Socket Layer (SSL) [p 346] for more information.

   JMS: see Java Message Service (JMS) [p 346] for more information.

   XML Catalog API: see XML Catalog API [p 346] for more information.
3. Compress anew the ear archive.

55.6 Deploying EBX®

1. Copy EBX.ear into *`JBOSS_HOME`*`/standalone/deployments` folder.

55.7 Start EBX®

1. After the launch of the JBoss Server, run the EBX® web application: http://localhost:8080/ebx/.
2. At first launch, EBX® Wizard [p 393] helps you to configure the default properties of your initial repository.

CHAPITRE **56**

Installation note for Tomcat 8.x

Ce chapitre contient les sections suivantes :

1. Overview
2. Requirements
3. Installation
4. Configuration for EBX®
5. Deploying EBX®
6. Start EBX®

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 343].
- 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.
- The description below uses the variable name `$CATALINA_HOME` to refer to the directory into which you have installed Tomcat, and from which most relative paths are resolved. However, if you have configured Tomcat for multiple instances by setting a `$CATALINA_BASE` directory, you should use `$CATALINA_BASE` instead of `$CATALINA_HOME` for each of these references.

- Create `EBX_HOME` directory: copy `ebx.properties`
- Copy EBX® and third-party libraries: add libraries (*jar files*) to Tomcat lib directory

- Configure JVM arguments (Java system properties): create the `JAVA_OPTS` environment variable
- Deploy EBX® application: copy all war files to Tomcat webapps directory

56.2 Requirements

- Java SE 8 or 11
- Apache Tomcat 8.x
- Database and JDBC driver
- EBX® CD

Voir aussi *Supported environments* *[p 336]*

56.3 Installation

1. To download Tomcat 8.x, choose a core binary distributions from https://tomcat.apache.org/download-80.cgi
2. Run the installer or extract the archive and perform standard installation with default options

56.4 Configuration for EBX®

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 then have the following file:

   `C:\EBX\home\ebx.properties`, or `/home/ebx/ebx.properties`, a text file
3. Edit the `ebx.properties` file to override the default database if needed, by default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and it is required to comment the `h2.standalone` one.
4. Copy third-party library files to `$CATALINA_HOME\lib\` (or `$CATALINA_BASE\lib\`) directory. In our example, we will have:

   `$CATALINA_HOME\lib\mail.jar`

   `$CATALINA_HOME\lib\h2.jar` (default persistence factory)

   `$CATALINA_HOME\lib\xml-apis-1.4.01.jar` (coming from the *EBX® CD* directory `ebx.software\lib\lib-xml-apis\`)

   The exact description of these components is given in chapter Components [p 343]. Obviously, if those components are already deployed on the class-loading system, they do not have to be duplicated
5. Modify the `$CATALINA_HOME\conf\server.xml` (or `$CATALINA_BASE\conf\server.xml`) file. Add the following line to the <Host> element

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

   After this modification, we will have:

   ```
   <Host name=...>

   ... ...
   ```

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

... ...

</Host>
```

6. Modify the `$CATALINA_HOME\conf\catalina.properties` (or `$CATALINA_BASE\conf\catalina.properties`) file. Add the following lines to the `tomcat.util.scan.DefaultJarScanner.jarsToSkip` property:

   ```
   ebx.jar,\

   ebx-addons.jar,\
   ```

7. Configure the launch properties

   If our Tomcat is launched by a command in Windows' Command Prompt or Unix shell, we can create another launcher file:

   For Windows, edit the launcher file `%CATALINA_HOME%\bin\startup.bat`, and add the following command 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%
   ```

   or for Linux, edit the launcher file `$CATALINA_HOME/bin/startup.sh`, and add the following command 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}"
   ```

   (!) Accounts used to launch EBX® must have create/update/delete rights on *EBX_HOME* directory.

   Windows users that have installed Tomcat as a service may 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
   ```

   where <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`

56.5 **Deploying EBX®**

1. 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`

2. Copy from *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

56.6 **Start EBX®**

1. After Tomcat launch, run EBX® web application: http://localhost:8080/ebx/
2. At first launch, EBX® Wizard [p 393] helps you to configure the default properties of your initial repository.

CHAPITRE 57

Installation note for WebSphere AS 9

Ce chapitre contient les sections suivantes :

1. Overview
2. Requirements
3. Installation
4. Configuration for EBX®
5. Deploying EBX®
6. Start EBX®

57.1 Overview

Attention

- This chapter describes a quick installation example of TIBCO EBX® on the WebSphere Application Server.
- It does not replace the documentation of this 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.
- The complete description of the components required by EBX® is given in the following chapter: Java EE deployment [p 343].
- 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.

- Install the WebSphere Application Server
- Create the `EBX_HOME` directory: copy `ebx.properties`
- Create a new profile by using the 'WebSphere Customization Toolbox'
- Create a data source and JDBC provider
- Configure the Web container
- Configure the Java Virtual Machine

- Install the EBX.ear application
- Start the EBX® application

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 336]*

57.3 Installation

This quick installation example is performed for a Linux operating system.

1. To download WebSphere AS 9, please first download the latest 'Installation Manager' from http://www-01.ibm.com/support/docview.wss?uid=swg27025142
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, please 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

      Then click 'Next'
   4. Profile Name and Location: enter a profile name (example: 'EbxAppSrvProfile') and directory
      `/opt/IBM/WebSphere/AppServer/profiles/EbxAppSrvProfile`
      further correspond to `PROFILE_HOME` and click 'Next'
   5. Node and Host Names: enter the node name (example: 'Node1'), the server name (example: 'EbxServer'), the host name (example: 'localhost'), and then 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 personnal 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 Configuration for EBX®

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 then have the following text file:

   `/opt/ebx/home/ebx.properties`.
3. Edit the `ebx.properties` file to override the default database if needed. By default, the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported database and it is required to comment the `h2.standalone` one
4. Create the `EBX_LIB` directory, for example `/opt/ebx/lib`
5. Copy third-party library files from the *EBX® CD*, or from other sources, to the `<EBX_LIB>` directory. In our example, for a PostgreSQL database, we will have:

   `postgresql-X.X.X-driver.jar` (coming from another source than the *EBX® CD*).

   `xml-apis-1.4.01.jar` (coming from the *EBX® CD* directory `ebx.software/lib/lib-xml-apis/`).

   The exact description of these components is given in the chapter Components [p 343]. 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, and the database driver is configured at the data source level).
6. Start the server (for example 'EbxServer'),

   sudo <PROFILE_HOME>/bin/startServer.sh <serverName>

   ```
   cd /opt/IBM/WebSphere/AppServer/profiles/EbxAppSrvProfile
   ```

   ```
   sudo bin/startServer.sh EbxServer.
   ```
7. Connect into the 'WebSphere Integrated Solutions Console' using the user name and password typed during the profile creation (Administrative Security step), enter the following url in the browser:

   https://localhost:9043/ibm/console
8. Create a data source and JDBC provider
   1. On the left menu, go to 'Resources > JDBC > Data Sources', to configure your database access. Choose the jdbc 'Scope' (for example use 'Cell'), and click 'New'
   2. Enter basic data source information:

- Data source name: `EBX_REPOSITORY`
- JNDI name: `jdbc/EBX_REPOSITORY`

click on 'Next'

3. Select the JDBC provider: select 'Create new JDBC provider', and click 'Next'
4. Create a new JDBC provider: (example with a PostgreSQL database)
   - Database type: `User-defined`
   - Implementation class name: `org.postgresql.ds.PGConnectionPoolDataSource`
   - Name: `PostgreSQL`

   and click 'Next'
5. Enter database class path information: (example with a PostgreSQL database)
   - Class path: `/opt/ebx/lib/postgresql-X.X.X-driver.jar`

   and click 'Next'
6. Enter database specific properties for the data source: keep as default and click 'Next'
7. Setup security aliases: keep as default and click 'Next'
8. Summary: click 'Finish'
9. Save the master configuration

9. Configure the data source: `jdbc/EBX_REPOSITORY`
   1. Click on 'Data Sources > `EBX_REPOSITORY`'
   2. On the right in the 'Configure additional properties' section, click on 'Additional Properties' and define the database account access:
      - Define `user` value to the according user
      - Define `password` value to the according password
   3. Save the master configuration
   4. Test the connection
10. Configure the Web container
    1. Click on 'Application Servers'
    2. Click on the server name (for example: 'EbxServer')
    3. Click on 'Custom Properties' under 'Web Container Settings > Web container > Additional Properties'
    4. Add 'com.ibm.ws.webcontainer.AllowQueryParamWithNoEqual' property, and set it to `true`.

       Note

       The `com.ibm.ws.webcontainer.AllowQueryParamWithNoEqual` attribute must be set to `true` since, according to the Servlet API specification, a value of a request parameter must be returned as a `String`, or `null` if the parameter does not exist. Thus, EBX® expects a non `null`value for an existing request parameter.

11. Configure the Java Virtual Machine
    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. Enter in 'Generic JVM arguments' the value:

   ```
   -Debx.properties=/opt/ebx/home/ebx.properties -Debx.home=/opt/ebx/home
   ```
6. Enter in 'Classpath' the paths to the third-party library files placed in the `<EBX_LIB>` directory except for the JDBC driver
7. click 'Ok'
8. Save the master configuration

57.5 **Deploying EBX®**

1. Copy from the *EBX® CD* the `ebx.software/webapps/ear-packaging/EBX.ear` to the *EBX_HOME* directory. In our example, we will have:

   `/opt/ebx/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), enter 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® application
   1. New Application: On the right panel, click on 'New Enterprise Application'
   2. Preparing for the application installation: Browse your EBX.ear file, located under `/opt/ebx/ear/EBX.ear`, then 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 all modules, then click 'Next'
   7. Warnings will appear related to `JNDI:mail/EBX_MAIL_SESSION` and `JNDI:jms/EBX_JMSConnectorFactory`. This behavior is normal since we had not configured these resources. 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 all modules, then click 'Next'
   9. Warnings will appear related to non-available resources. This behavior is normal since we had not configured these resources, then 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, it logs 'Application EBX® installed successfully', then 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', select 'Class loading and update detection'
    3. Under 'General Properties', change 'Class loader order' to 'Classes loaded with local class loader first (parent last)'
    4. Return to 'Applications > Enterprise Applications'
7. Enterprise Applications: select EBX®, and then click 'Start'

   The EBX® 'Application status' will be changed into a green arrow

57.6 **Start EBX®**

1. After the launch of the WebSphere Server, run the EBX® web application: http://localhost:9080/ebx/
2. At first launch, EBX® Wizard [p 393] helps you to configure the default properties of your initial repository

CHAPITRE **58**

Installation note for WebLogic 12c R2

Ce chapitre contient les sections suivantes :

1. Overview
2. Requirements
3. Installation
4. Configuration for EBX®
5. Deploying EBX®
6. Start EBX®

58.1 Overview

Attention

- This chapter describes a quick installation example of TIBCO EBX® on the WebLogic 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 the following chapter: Java EE deployment [p 343].
- 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.

- Install the Java Virtual Machine: Create the JAVA_HOME environment variable
- Install the WebLogic Server
- Create the `EBX_HOME` directory: copy `ebx.properties`
- Create a new domain by using the 'Configuration Wizard'
- Configure the domain: Declare EBX® `JAVA_OPTIONS` and copy the database JDBC driver
- Install and configure the JDBC driver
- Deploy the EBX® application: install dedicated ear file

58.2 Requirements

- Java SE 8 or 11
- WebLogic Server 12c R2
- Database and JDBC driver
- EBX® CD

Voir aussi *Supported environments [p 336]*

58.3 Installation

1. To download WebLogic 12c R2, please refer to https://edelivery.oracle.com/osdc/faces/Home.jspx
2. Run the 'Fusion Middleware Configuration Wizard' (`<weblogic_home>/oracle_common/common/bin/config.sh`) and perform a standard installation with default options:
   1. Create Domain: choose 'Create a new domain' and specify the domain home folder, 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, and 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. Machines: keep as default and click 'Next'
   10. Virtual Targets: keep as default and click 'Next'
   11. Partitions: keep as default and click 'Next'
   12. Configuration Summary: click 'Create'
   13. Configuration Process: click 'Next'
   14. End Of Configuration: click 'Finish'

58.4 Configuration for EBX®

1. Create the *EBX_HOME* directory, for example `C:\EBX\home`, or `/home/ebx`
2. Copy from the *EBX® CD* the `ebx.software\files\ebx.properties` file to *EBX_HOME*. In our example, we will then have the following file:

`C:\EBX\home\ebx.properties`, or `/home/ebx/ebx.properties`, a text file

3. Edit the `ebx.properties` file to override the default database if needed, by default the standalone H2 database is defined. The property key `ebx.persistence.factory` must be uncommented for other supported databases and it is required to comment the `h2.standalone` one.
4. 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}"
   ```
5. Copy third-party library files to the `<DOMAIN_HOME>/lib` directory. In our example, for an H2 standalone data base, we will have:

   `h2.jar` (default persistence factory)

   The exact description of these components is given in chapter Components [p 343]. Obviously, if those components are already deployed on the class-loading system, they do not have to be duplicated (ex: mail.jar and xml-apis-1.4.01.jar are already present in the WebLogic Server).
6. Start the 'Administration server' (for example 'AdminServer'), `<DOMAIN_HOME>/bin/startWebLogic.sh`
7. Launch the 'WebLogic Server Administration Console', enter the following url in the browser:

   http://localhost:7001/console.

   Log in with the domain administrator username and password
8. Click on 'Services > Data sources' in the 'Domain Structure' panel, then click on the 'New > Generic Data Source' button
   1. Type Name: `EBX_REPOSITORY`, JNDI Name: `EBX_REPOSITORY` Database Type: *Your database type* and 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')

      `<DOMAIN_HOME>/bin/stopWebLogic.sh`

      `<DOMAIN_HOME>/bin/startWebLogic.sh`

58.5 Deploying EBX®

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', enter 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 on `C:\EBX\home\`, or `/home/ebx/` folder and click 'Next'
    2. Choose the installation type and scope: Click on 'Install this deployment as an application' and 'Global' default scope and click 'Next'
    3. Select the deployment targets: Select a node name (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 on 'Start' > 'Servicing all requests'
7. Start the application node name (for example 'EbxServer'),

    ```
    <DOMAIN_HOME>/bin/startManagedWebLogic.sh EbxServer http://localhost:7001
    ```

58.6 **Start EBX®**

1. After WebLogic Server launch, run the EBX® web application: http://localhost:7003/ebx/
2. At first launch, EBX® Wizard [p 393] helps you to configure the default properties of your initial repository

CHAPITRE **59**

TIBCO EBX® main configuration file

Ce chapitre contient les sections suivantes :

1. Overview
2. Setting an EBX® license key
3. Setting automatic installation on first launch
4. Setting the EBX® root directory
5. Configuring the EBX® repository
6. Configuring the user and roles directory
7. Configuring EBX® localization
8. Setting temporary files directories
9. Activating the XML audit trail
10. Configuring the EBX® logs
11. Activating and configuring SMTP and emails
12. Configuring data services
13. Activating and configuring JMS
14. Configuring distributed data delivery (D3)
15. Configuring REST toolkit services
16. Configuring Web access from end-user browsers
17. Configuring failover
18. Tuning the EBX® repository
19. 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 348]. Administrators can also perform further configuration through the user interface, which is then stored in the EBX® repository.

Voir aussi

Deployment details *[p 348]*

UI administration *[p 409]*

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 an EBX® license key

Voir aussi *Initialization and first-launch assistant* *[p 393]*

The license key can be retrieved from the TIBCO eDelivery site.

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

59.3 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).
## If true, property ebx.license should also be set to a valid license.
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 none 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
```

59.4 Setting the EBX® root directory

The EBX® root directory contains 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
```

59.5 Configuring the EBX® repository

Before configuring the persistence properties of the EBX® repository, carefully read the section Technical architecture [p 398] in the chapter 'Repository administration'.

The required library (driver) for each supported database is described in the chapter Database drivers [p 344].

Voir aussi

Repository administration [p 398]

Rules for the database access and user privileges [p 399]

Supported databases [p 339]

Data source of the EBX® repository [p 349]

Database drivers [p 344]

```
################################################################
## 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=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


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

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


## Activate to use VARCHAR2 instead of NVARCHAR2 on Oracle; 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 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=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


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

## Specific properties to be set only only if you want to ignore the standard
## deployment process of 'ebx' web application in the target operational environment
## (see the deployment descriptor 'web.xml' of 'ebx' web application).
#ebx.persistence.url= \
#jdbc:sqlserver://127.0.0.1:1036;databasename=ebxDatabase
#ebx.persistence.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy


######################################################################
## Case EBX® persistence system is Microsoft Azure SQL database.
######################################################################
#ebx.persistence.factory=azure.sql

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


######################################################################
## Case EBX® persistence system is PostgreSQL.
######################################################################
#ebx.persistence.factory=postgresql

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

```
#ebx.persistence.url=jdbc:postgresql://127.0.0.1:5432/ebxDatabase
#ebx.persistence.driver=org.postgresql.Driver
#ebx.persistence.user=xxxxxxxxx
#ebx.persistence.password=yyyyyyyy
```

59.6 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 425]*

`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
```

59.7 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 *Extension de l'internationalisation de TIBCO EBX®* *[p 261]*

59.8 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
```

59.9 Activating the XML audit trail

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

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

Voir aussi *Audit trail* *[p 445]*

59.10 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 380]). **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 325].
ebx.log4j.category.log.request	The optimization strategy for every `Request`[API] issued on a semantic table in the EBX® repository.
ebx.log4j.category.log.restServices	Logs for REST services events in EBX®, including those from the REST Toolkit [p 747].

Some of these categories can also be written to through custom code using the `LoggingCategory`[API] interface.

```
 ################################################
## Log4J properties:
##
## We have some specific syntax extensions:
## - Appender ebxFile:<aFileName>
## Defines a file appender with default settings (threshold=DEBUG)
##
```

```
## - property log.defaultConversionPattern is set by Java
################################################
#ebx.log4j.debug=true
#ebx.log4j.disable=
ebx.log4j.rootCategory= INFO
ebx.log4j.category.log.kernel= INFO, Console, ebxFile:kernel, kernelMail
ebx.log4j.category.log.workflow= INFO, ebxFile:workflow
ebx.log4j.category.log.persistence= INFO, ebxFile:persistence
ebx.log4j.category.log.setup= INFO, Console, ebxFile:kernel
ebx.log4j.category.log.mail= INFO, Console, ebxFile:mail
ebx.log4j.category.log.frontEnd= INFO, Console, ebxFile:kernel
ebx.log4j.category.log.frontEnd.incomingRequest= INFO
ebx.log4j.category.log.frontEnd.requestHistory= INFO
ebx.log4j.category.log.frontEnd.UIComponentInput= INFO
ebx.log4j.category.log.fsm= INFO, Console, ebxFile:fsm
ebx.log4j.category.log.fsm.dispatch= INFO
ebx.log4j.category.log.fsm.pageHistory= INFO
ebx.log4j.category.log.wbp= FATAL, Console
#-----------------------------------------------
ebx.log4j.appender.Console.Threshold = INFO
ebx.log4j.appender.Console=com.onwbp.org.apache.log4j.ConsoleAppender
ebx.log4j.appender.Console.layout=com.onwbp.org.apache.log4j.PatternLayout
ebx.log4j.appender.Console.layout.ConversionPattern=${log.defaultConversionPattern}
#-----------------------------------------------
ebx.log4j.appender.kernelMail.Threshold = ERROR
ebx.log4j.appender.kernelMail = com.onwbp.org.apache.log4j.net.SMTPAppender
ebx.log4j.appender.kernelMail.To = admin@domain.com
ebx.log4j.appender.kernelMail.From = admin${ebx.site.name}
ebx.log4j.appender.kernelMail.Subject = 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, ebxFile:monitoring
ebx.log4j.category.log.dataServices= INFO, ebxFile:dataServices
ebx.log4j.category.log.d3= INFO, ebxFile:d3
ebx.log4j.category.log.request= INFO, ebxFile:request
ebx.log4j.category.log.restServices= INFO, ebxFile: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`.

The property `ebx.log4j.appender.ebxFile.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.ebxFile.backup.Threshold=-1
```

Custom SMTP appender

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

Voir aussi *Activating and configuring SMTP and emails* *[p 380]*

Module's appenders configuration

The configurations of module logging categories are declared according to a hierarchical pattern. Furthermore, every enabled logging request for a given category will be forwarded to all the appenders defined for that category, as well as to the appenders higher in the hierarchy. This fact means that appenders are inherited additively.

The root module logging category can be customized by setting the property `ebx.log4j.category.log.wbp`. For a given module, if the root module logging category configuration is the only one applicable, then a `DailyRollingFileAppender` is automatically added to the module's appenders list. The log file name will be derived from the module's name and from an optional specific sub-category.

```
#########################################################################
## Root module logging category configuration
#########################################################################
ebx.log4j.category.log.wbp = FATAL, Console
```

Every module logging category can be customized by setting the property `ebx.log4j.category.log.module.xxxxxx`, where `xxxxxx` corresponds to the module's name. The explicit configuration of the module logging category will disable the automatic addition of the `DailyRollingFileAppender` previously described. However, any other defined appenders will be inherited additively.

Since the inheritance mechanism may not be suitable for every case, the additivity can be broken by setting to `false` the property `ebx.log4j.additivity.log.wbp.xxxxxx`, where `xxxxxx` corresponds to the module's name.

```
#############################################################################
## Module logging category configuration
##
## The module's log messages will only be written to mycompany-module log file
## since additivity has been broken
#############################################################################
ebx.log4j.category.log.module.mycompany-module = INFO, ebxFile:mycompany-module

ebx.log4j.additivity.log.wbp.mycompany-module = false
```

Module's log threshold

The inheritance mechanism described in Module's appenders configuration [p 379] is applied to the module log threshold as well. Actually, the inherited level for a given logger, is equal to the first non-null level in the hierarchy, starting from this logger and proceeding up to the root module logging category.

Custom module log threshold

There is an exception to the inheritance, for custom module, since the log level threshold of their logging category, by default, is set to `INFO`. This threshold can be customized by setting the property `ebx.log4j.category.log.module.xxxxxx`, where `xxxxxx` corresponds to the custom module's name.

Example: `ebx.log4j.category.log.module.mycompany-module=DEBUG`.

Voir aussi *`ModuleContextOnRepositoryStartup.getLoggingCategory`*[API]

Add-on module log threshold

Like custom module, 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` where `xxxxxx` corresponds to the add-on module's name.

Example: `ebx.log4j.category.log.addon.daqa=DEBUG`

Modules log threshold overrides summary

Thus and considering EBX® logging features, the log threshold defined at the root level in the logger hierarchy may be overridden by (by order from least to most weighted):

- the module logging category explicit configuration,
- the module's `log.threshold.xxxx` or `log.threshold.category.xxxx` log configuration properties, where `xxxx` corresponds to the sub category,
- the following log configuration properties: `ebx.log4j.category.log.module.xxxxxx`, where `xxxxxx` corresponds to the custom module's name, or `ebx.log4j.category.log.addon.xxxxxx`, where `xxxxxx` corresponds to the add-on module's name.

59.11 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.onwbp.org.apache.log4j.net.SMTPAppender`.

Voir aussi *Mail sessions* *[p 349]*

```
################################################
## 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 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
```

59.12 Configuring data services

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

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

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

```
# If unspecified, default is false.
#ebx.dataservices.dataDeletionOnCloseOrMerge.default=false
#ebx.dataservices.historyDeletionOnCloseOrMerge.default=false

# Upon WSDL generation, specifies if the target namespace value
# corresponds to the content before 5.5.0 'ebx-services'
# or 'urn:ebx:ebx-services' in conformity with the URI syntax.
# If the parameter is set to true, there is no check of the target
# namespace as URI at the WSDL generation.
# If unspecified, default is false.
#ebx.dataservices.wsdlTargetNamespace.disabledCheck=false

######################################################################
## 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, it tries authentication 'Basic Authentication Scheme'
# method and set 'Basic' value in 'WWW-Authenticate' header of HTTP
# response.
# 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
```

59.13 Activating and configuring JMS

Voir aussi *JMS for data services* *[p 350]*

```
######################################################################
## 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
```

59.14 Configuring distributed data delivery (D3)

See Configuring D3 nodes [p 469] for the main configuration file properties pertaining to D3.

Voir aussi

JMS for distributed data delivery (D3) *[p 459]*

Introduction to D3 *[p 450]*

59.15 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
```

59.16 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 412]*

```
######################################################################
## 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. Default value is false.
##
## 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`.

Voir aussi *Path recommendations*

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`.

Voir aussi *Path recommendations*

Path recommendations

It is recommended to deploy webapps of EBX® modules on the same path level. For example:

- `/module1`
- `/module2`

or:

- `/path/module1`
- `/path/module2`

It avoids difficulties when configuring the advanced features of Proxy mode and Reverse-proxy mode.

Consequently, it is not recommended to use the root path `"/"` as the servlet's HTTP or HTTPS path in the following properties:

- `ebx.servlet.http.path`
- `ebx.servlet.http.ui.path`
- `ebx.servlet.https.path`
- `ebx.servlet.https.ui.path`

59.17 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 399]*

```
################################################
## 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
```

59.18 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

# A validation messages threshold allows specifying the maximum number of
# messages to consider per constraint when performing a validation.
# This threshold is considered for each constraint defined in a data model
# and for each severity in each dataset validation report.
# When the threshold is reached by a constraint and a severity:
# - the validation of the constraint is stopped
# - an error message indicating that the threshold has been reached
# is added to the validation report.
# When set to 0 or a negative value, the threshold is not considered.
# Default value is 0.
#
ebx.validation.constraints.messages.threshold = 100


# Specifies whether the validation report should be kept in memory,
# regardless of the loading strategy of the dataspace.
# Default value is true. However, it is recommended to deactivate it
# when the repository contains a large number of open dataspaces and
# datasets.
#ebx.validation.report.keepInMemory=false
```

Voir aussi *Validation report page* *[p 391]*

59.19 Miscellaneous

Activating data workflows

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

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

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_MODE`[API] 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
```

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
```

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 undeployed

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 486]*

Dynamically reloading the main configuration *[p 387]*

```
################################################
## 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 382] 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 497] 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 dataspaces.
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 must be of type "m:[pattern]:[options]".
## the list can be void
################################################
```

```
ebx.resource.exclude=m: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=100
```

Voir aussi *Tuning the EBX® repository* *[p 386]*

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 386]*

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 343].

> **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. License key
2. Configuration steps

60.1 License key

When launching EBX®, the license key page displays automatically if no valid license key is available, that is, if there is no license key entered in the EBX® main configuration file, or if the current license key has expired.

The license key can be retrieved from the TIBCO eDelivery site (an account is mandatory in order to access eDelivery, please contact https://support.tibco.com to request one).

The license key is available in the `TIB_ebx_{ebx.version.public}_license_key.pdf` file.

60.2 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 425]*

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. 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 379].

61.2 Registering an add-on module

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, and enter its license key.

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

   The license key can be retrieved from the TIBCO eDelivery site.
5. Click on **Save**.

61.3 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 dataspaces.

   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 dataspaces. When an add-on module is no longer deployed, it is thus necessary to close/delete and purge manually all data/dataspaces related to the add-on.

Technical administration

CHAPITRE 62

Repository administration

Ce chapitre contient les sections suivantes :

1. Technical architecture
2. Auto-increments
3. Repository management
4. Monitoring management
5. Dataspaces

62.1 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 339], 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 399] and Failover with hot-standby [p 399]. Furthermore, to achieve horizontal scalability, an alternative is to deploy a distributed data delivery (D3) [p 450] 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 339]

Data source of the EBX® repository [p 349]

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**, except for specific use cases described in the section Accès SQL aux données en mode relationnel [p 271].

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 401].

Voir aussi

Accès SQL à l'historique [p 276]

Accéder à une table répliquée à l'aide de SQL [p 284]

Accès SQL aux données en mode relationnel [p 271]

Data source of the EBX® repository [p 349]

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. 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 400] 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 385].

  The format of the request URL must be:

  `http[s]://<host>[:<port>]/ebx?activationKeyFromStandbyMode={value}`
- Using the Java API, call the method `RepositoryStatus.wakeFromStandby`[API].

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 273]' > 'Repository connection log'.

The status of the repository may be retrieved using the methods in the `RepositoryStatus`[API] API.

It is also possible to get 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.

62.2 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.

62.3 Repository management

Installation and upgrades

Automatic installation and upgrades

By complying with the Rules for the database access and user privileges [p 399], 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 dataspaces, 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:

- For technical reasons, migration to an Oracle database is only supported from another Oracle database.
- The names of the database objects representing the mapped tables (history, replication, relational) 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

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 *RFC 4122*.
repository label	Provides a user-friendly label that identifies the purpose and context of the repository. For example: "Production environment".
store format	Identifies the underlying persistence system, including the current version of its structure.

62.4 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.

Monitoring and reorganization

The persistence data source of the repository must be monitored through RDBMS monitoring.

The EBX® tables specified in the default semantic mode [p 267] have their content persisted in a set of generic database tables:

- The table `${ebx.persistence.table.prefix}HOM`, in which each record represents a dataspace or a snapshot (its name is `EBX_HOM` if the property `ebx.persistence.table.prefix` is unset).
- The table `${ebx.persistence.table.prefix}BLK`, where the data of EBX® tables in semantic mode are segmented into blocks of at most 100 EBX® records (its name is `EBX_BLK` if the property `ebx.persistence.table.prefix` is unset).
- The tables ${ebx.persistence.table.prefix}HTA, ${ebx.persistence.table.prefix}TAR and ${ebx.persistence.table.prefix}ATB, defining which blocks belong to a given EBX® table in a given dataspace or snapshot.

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).

Impact on UI

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.closeHome`[API].
- Using the data service "close dataspace" and "close snapshot" operations. See Closing a dataspace or snapshot [p 667] for more information.

Once the dataspaces and snapshots have been closed, the data can be safely removed from the repository.

> **Note**
>
> Closed dataspaces and snapshots can be reopened in the 'Administration' area, under 'Dataspaces'.

Deleting dataspaces, snapshots, and history

Dataspaces, 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 dataspaces, snapshots, and history can be performed in a number of different ways:

- From the 'Dataspaces/Snapshots' table under 'Dataspaces' 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.deleteHome`[API] and `RepositoryPurge.markHomeForHistoryPurge`[API].
- 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.purgeAll`[API].
- Using the task scheduler. See Task scheduler [p 439] for more information.

The purge process is logged in the directory `${ebx.repository.directory}/db.purge/`.

Cleaning up tables having unreadable records

Some data model evolutions can lead to unreadable data:

- A column containing null values is added to the primary key of a table.
- The type of a column has changed to a different type with no possible conversion.

In these situations, records that do not match the new table definition are no longer visible, but remain persisted in the table. (This allows retrieving records if switching back to the previous table definition). When such records are encountered, an informative error is recorded in EBX® logs.

EBX® provides the option to clean the records that no longer conform to the model, once the new version of the data model is stabilized. This allows recovering space in the database and getting rid of error messages. Please proceed carefully, as this operation permanently removes all unreadable records from the selected table, and cannot be undone.

Cleaning up unreadable records is done by selecting in the 'Administration' area **Actions > Clean up unreadable records** in the navigation pane.

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 dataspaces, 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 404]*

Monitoring and clean up of 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:

- **XML audit trail**: `${ebx.repository.directory}/History/`
- **Archives**: `${ebx.repository.directory}/archives/`
- **Logs**: ebx.logs.directory [p 377]
- **Temporary directory**: ebx.temp.directory [p 375]

Attention

For **XML audit trail**, if large transactions are executed with full update details activated (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 445]*

Tuning the EBX® repository *[p 386]*

62.5 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 104]*

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 dataspaces, snapshots, and history* *[p 403]*

Dataspace permissions

This table lists all the existing permission rules defined on all the dataspaces in the repository. You can view the permission rules and modify their information.

Voir aussi *Permissions sur un espace de données* *[p 106]*

Repository history

The table 'Deleted dataspaces/snapshots' lists all the dataspaces that have already been purged from the repository.

From this section, it is also possible to delete the history of purged dataspaces.

CHAPITRE 63

UI administration

TIBCO EBX® comes with a full user interface called Advanced perspective [p 410] that includes all available features. The interface is fully customizable [p 413] (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 409], giving the possibility to grant or restrict access to functional categories. Administrators can create simplified perspectives called recommended perspectives [p 421] 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

63.1 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 309] 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 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.

63.2 Advanced perspective

The advanced perspective and its parameterization are unique. It is the parent perspective from which any new perspective [p 421] 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 27]*

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 421]*

Favorite perspectives *[p 19]*

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 124]*

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 412]. 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 382]. 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 *http-equiv="X-UA-Compatible" content="IE=EmulateIE8"* is added to the header of pages. However, in some local environments, this setting may conflict with existing policies, in which case this header must be omitted by setting the parameter to 'No'. The default value is 'Yes'. See Specifying Document Compatibility Modes for more information.
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	En fonction des performances du réseau et du navigateur, ajustez la manière d'afficher chaque noeud non terminal du formulaire d'enregistrement. En ce qui concerne le poids de la page téléchargée, le mode lien est léger, les modes développé et réduit sont plus lourds. En cas de modification de cette propriété, les utilisateurs doivent se déconnecter et se reconnecter afin que la nouvelle valeur soit prise en compte.
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 413] 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 27]*

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 419]*
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 589]* 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 415] 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 418] 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.

63.3 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 19] 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 421] 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
```

63.4 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'.

63.5 **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 414]. 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 **64**

Users and roles directory

Ce chapitre contient les sections suivantes :

1. Overview
2. Concepts
3. Default directory
4. Custom directory

64.1 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 375]*

Custom directory *[p 428]*

64.2 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 301].

Voir aussi

profil [p 23]

rôle [p 24]

utilisateur [p 23]

administrateur [p 24]

annuaire des utilisateurs et des rôles [p 24]

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.

64.3 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 570].

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.

64.4 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 65

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

65.1 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.

65.2 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.

65.3 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 links describe how the evolutions of data models are managed at the persistence level: Cleaning up tables having unreadable records [p 405] and Purging master tables in the database [p 433].

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 **66**

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

66.1 Overview

Information and services relative to database mapping can be found in the *Administration* area.

Voir aussi

Mapped modes *[p 265]*

`DatabaseMapping`[API]

66.2 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]

66.3 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 274]*

Disabling replication on a specific field or group *[p 284]*

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.

66.4 Renaming master tables in the database

This feature allows renaming master tables for both relational and history modes in the database. However, 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 that the repository prefix is followed by the history tables prefix.
>
> For relational tables, it is also required that the repository prefix is followed by the relational tables prefix.
>
> Besides, the new name must be a valid table identifier, which depends on the naming rules of the underlying RDBMS.

66.5 Renaming auxiliary tables in the database

This feature allows renaming history auxiliary tables in the database. This feature is neither available for replicated tables since their names are specified in the data model, nor for the relational mode, as aggregated lists are never supported in this mode.

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 that the new identifier begins with a letter.
>
> It is required that the new identifier begins with the repository prefix.
>
> It is also required that the repository prefix is 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.

66.6 Purging master tables in the database

This feature allows purging history and relational tables in the database if they are 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.

For history mode:

- Deactivate historization of the table in the data model, or
- Remove the table from the data model

For relational mode:

- Remove the table from the data model, or
- Deactivate the relational mode on the data model. As data models in semantic mode cannot be used for relational dataspaces, it is thus necessary to create a dataset on a semantic dataspace using this modified data model. TIBCO EBX® will then detect that the relational mode was previously used, and will therefore propose the relational table database resources for purge.

CHAPITRE 67

Workflow management

Ce chapitre contient les sections suivantes :

1. Workflows
2. Interactions
3. Workflow history

67.1 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 193]*

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 31]*

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 31]*

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 30]*

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.

67.2 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.

67.3 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 406]*

> **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 68

Task scheduler

Ce chapitre contient les sections suivantes :

1. Overview
2. Configuration from EBX®
3. Cron expression
4. Task definition
5. Task configuration

68.1 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.

68.2 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.

68.3 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 comprised 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 comprised 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.

68.4 Task definition

EBX® scheduler comes with some predefined tasks.

Custom scheduled tasks can be added by the means of **scheduler** `Package` `com.orchestranetworks.scheduler`[API] Java API.

The declaration of schedules and tasks is done by selecting 'Task scheduler' in the 'Administration' area.

68.5 Task configuration

A user must be associated with a task definition; this user will be used to generate the **session** `Session`[API] that will run the task.

> **Note**
>
> The user will not be authenticated, and no password is required. As a consequence, a user with no password set in the directory can only be used to run scheduled tasks.

A custom task can be 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 69

Audit trail

Ce chapitre contient les sections suivantes :

1. Overview
2. Update details and disk management
3. File organization

69.1 Overview

XML audit trail is a feature that allows logging updates to XML files. An alternative history feature is also available to record table updates in the relational database; see Historique [p 273].

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.

69.2 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.setHistoryActivation`[API].

Voir aussi *EBX® monitoring* *[p 406]*

69.3 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 70

Other

Ce chapitre contient les sections suivantes :

1. Lineage
2. Event broker

70.1 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.

70.2 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.

Distributed Data Delivery (D3)

CHAPITRE 71

Introduction to D3

Ce chapitre contient les sections suivantes :

1. Overview
2. D3 terminology
3. Known limitations

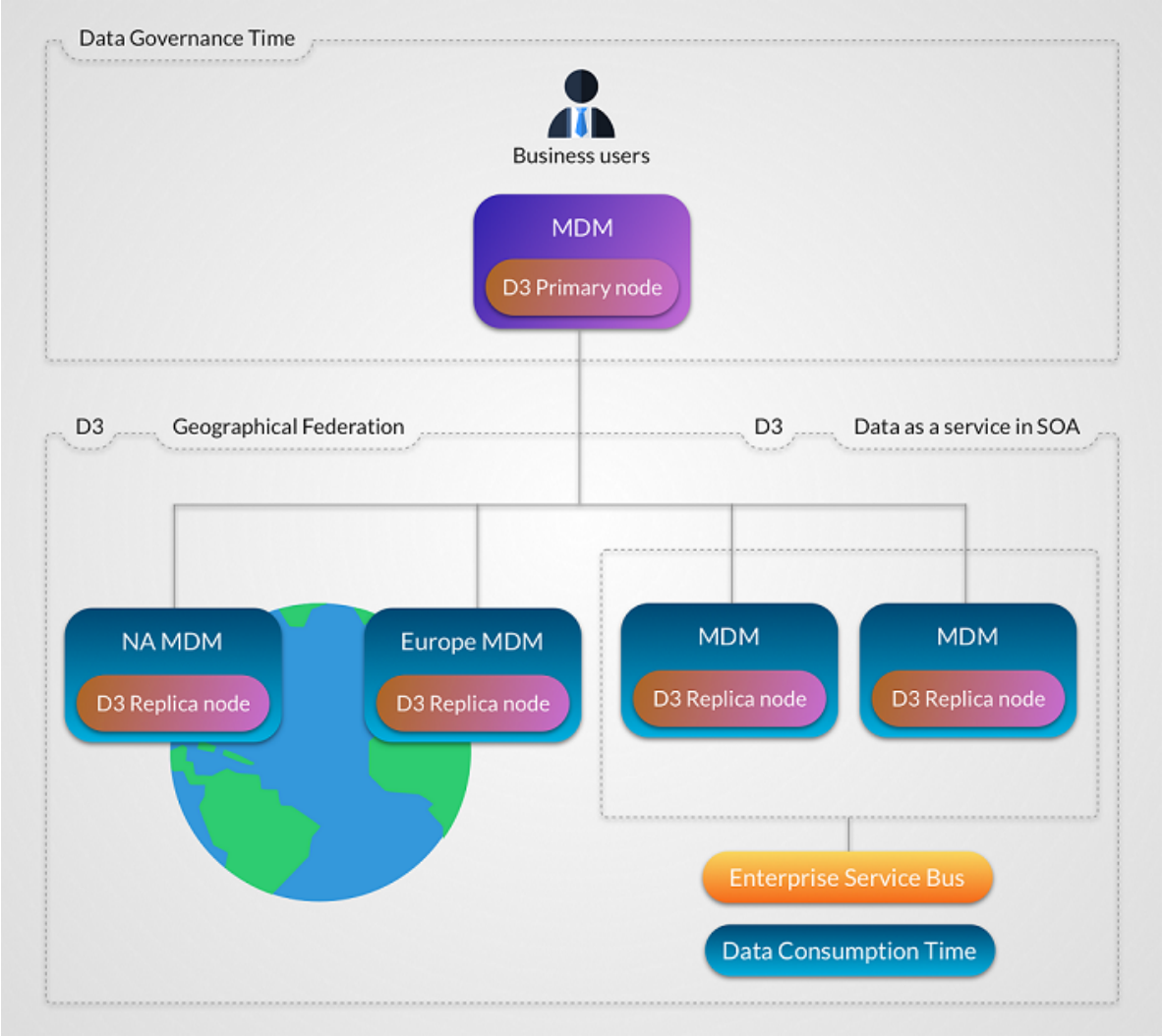
71.1 Overview

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 dataspaces 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 dataspaces.

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 459]*

71.2 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 457].
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 dataspaces. 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 dataspaces, 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 dataspaces, 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 dataspaces and replica node delivery dataspaces **must** be disjoint.

71.3 Known limitations

General limitations

- Each replica node must have only one primary node.
- Embedded data models cannot be used in D3 dataspaces. 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 dataspaces are not broadcast, whereas access rights on datasets are.
- Dataspace information is not broadcast.
- Dataspaces defined in relational mode cannot be 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 dataspaces cannot be broadcast, thus the EBX® default user directory cannot be synchronized using D3.

CHAPITRE 72

D3 broadcasts and delivery dataspaces

Ce chapitre contient les sections suivantes :

1. Broadcast
2. Replica node registration
3. Accessing delivery dataspaces

72.1 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 472]*

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 `D3NodeAsMaster`[API].

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).
- The dataspace or snapshot does not contain any tables in relational mode.
- 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 474]*

> **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 473].

Destination

In the target replica or hub node side:

- The `ebx-d3-reference` dataspace identifier is the common parent of all the delivery dataspaces.
- 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 is already existing on the first broadcast or full synchronization it will be overridden.
- If an existing dataspace with the same identifier of the delivery one is detected outside of the `ebx-d3-reference`, An error will be raisen.

Voir aussi *Known limitations* *[p 454]*

72.2 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 469].

72.3 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 644]*

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.getBroadcastVersion`[API].

CHAPITRE 73

D3 JMS Configuration

Ce chapitre contient les sections suivantes :

1. JMS for distributed data delivery (D3)

73.1 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 350] 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 381] and Distributed data delivery (D3) [p 381] 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.

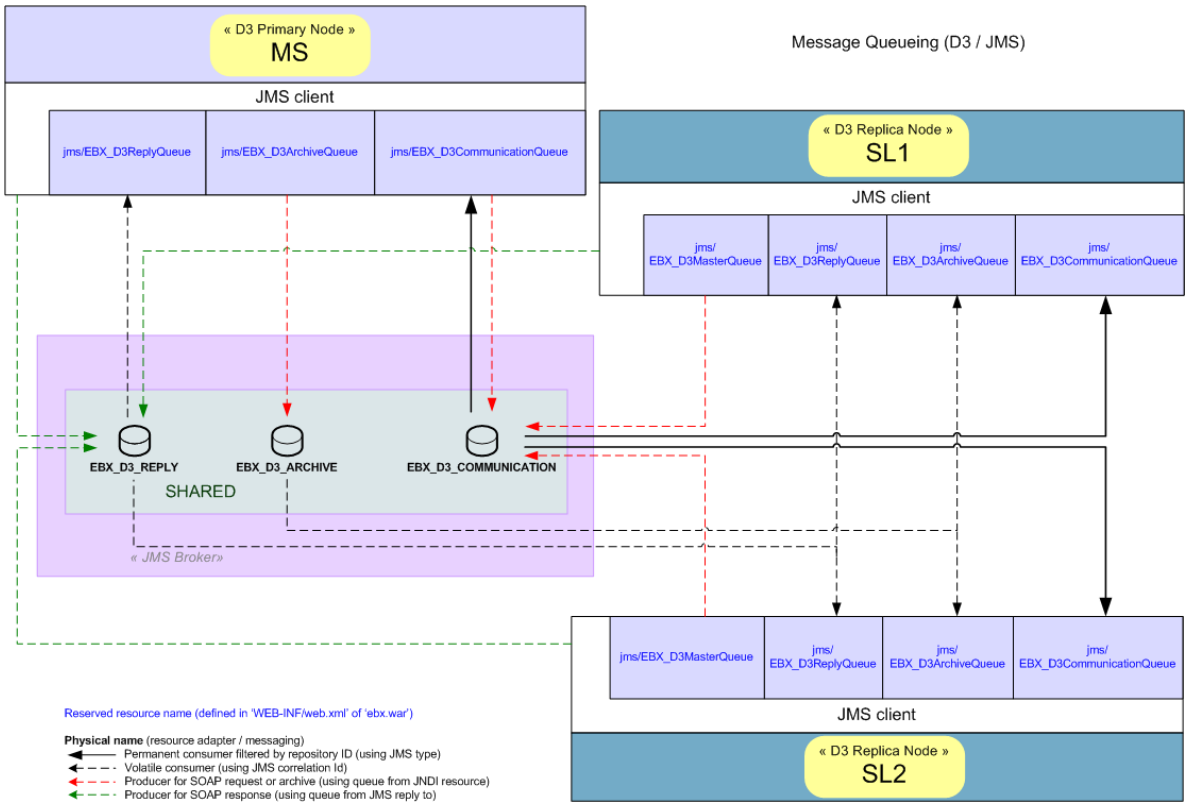
Resources can be freely named, but the physical names of their associated queue must correspond to the definition of replica nodes for resources `jms/EBX_D3ArchiveQueue` and `jms/EBX_D3CommunicationQueue`.

> **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.

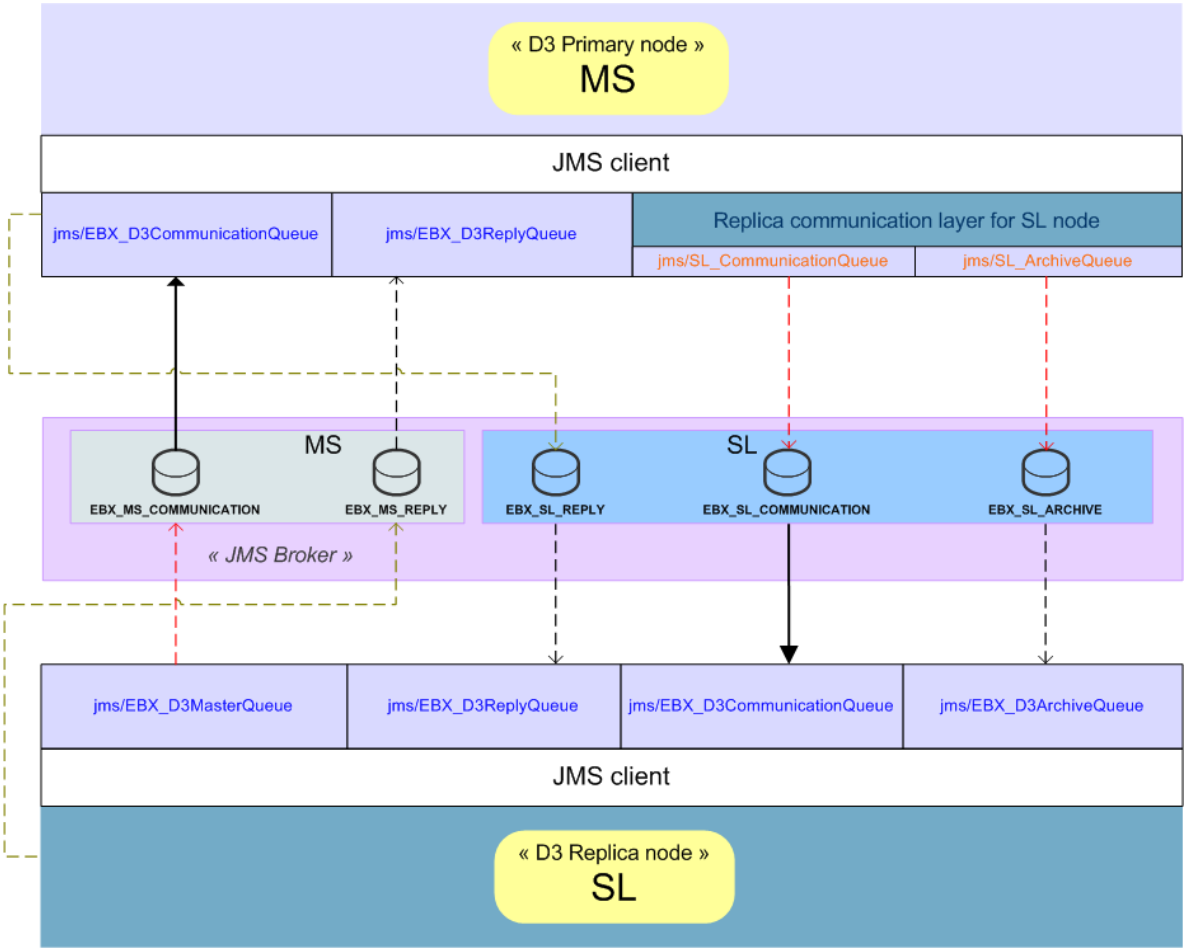
Examples of JMS configuration

	Shared queues	**Specific queues**
Primary-Replica nodes architecture	Between a primary node and two replica nodes with shared queues [p 462]	Between a primary node and a replica node with replica-specific queues [p 463]
Hub-Hub architecture	Between two hub nodes with shared queues [p 464]	Between two hub nodes with replica-specific queues [p 465]

Between a primary node and two replica nodes with shared queues



Between a primary node and a replica node with replica-specific queues

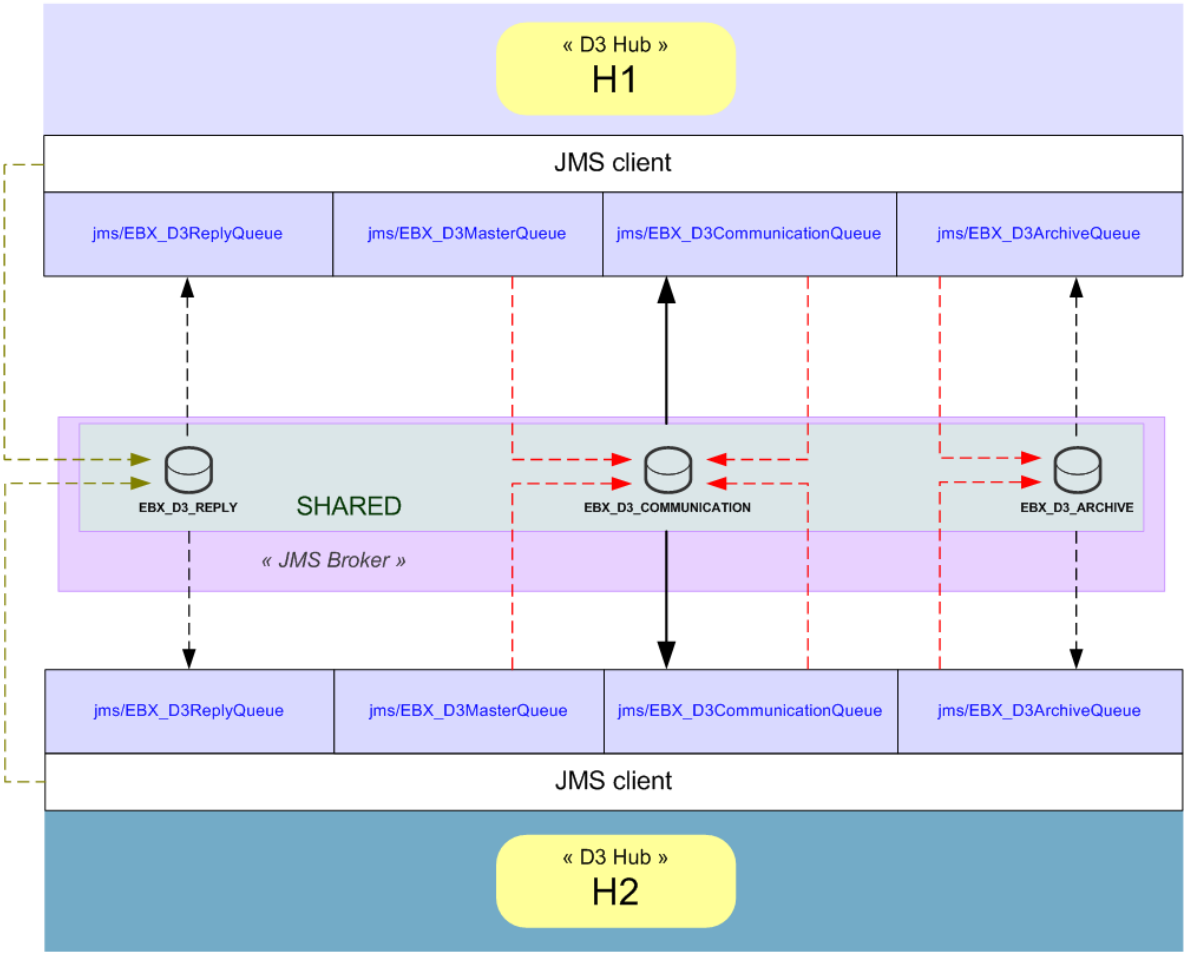


Reserved or custom resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

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

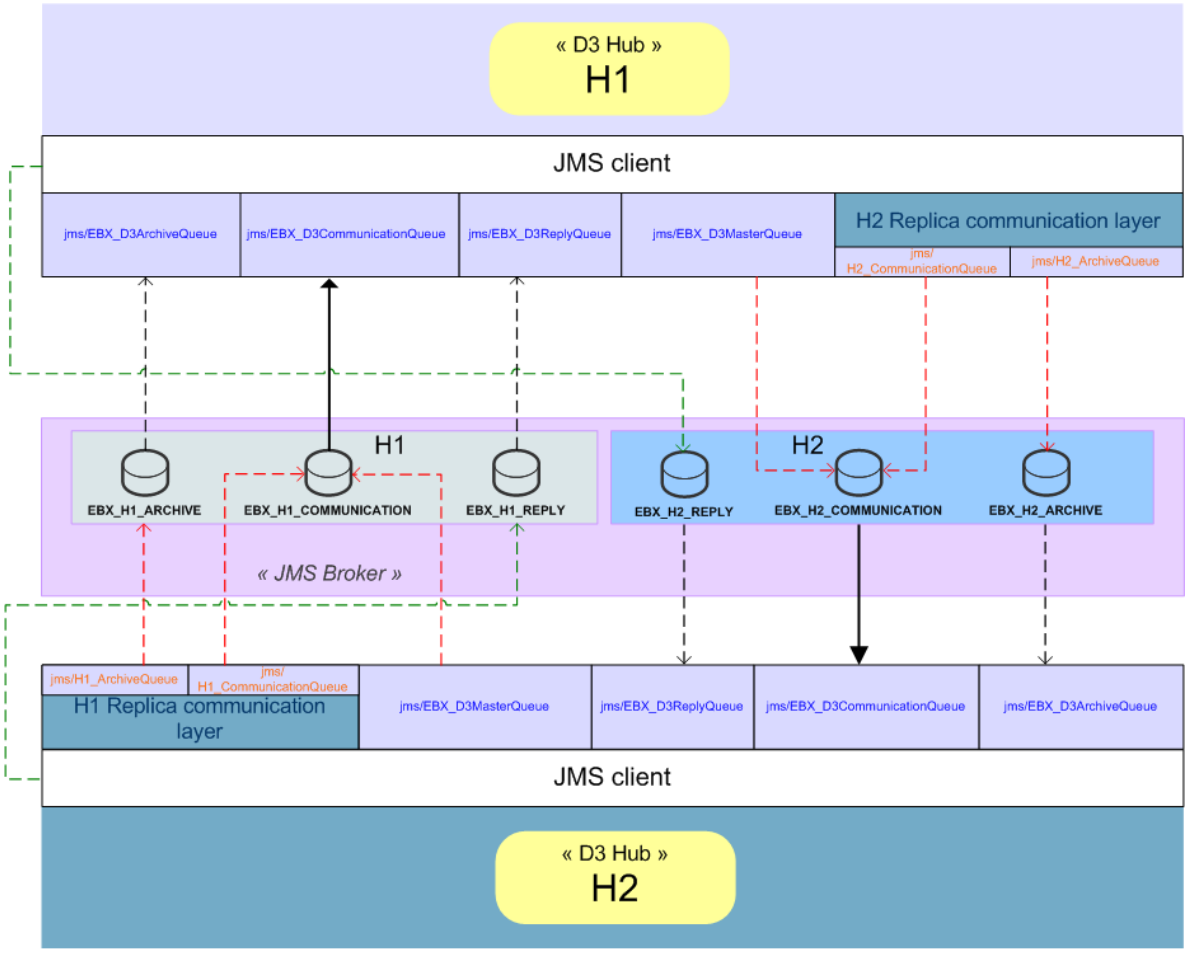


Reserved resource name (defined in 'WEB-INF/web.xml' of 'ebx.war')

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 74

D3 administration

Ce chapitre contient les sections suivantes :

1. Quick start
2. Configuring D3 nodes
3. Supervision

74.1 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 453] and replica [p 453] nodes included).

Update the `ebx.properties`*primary* 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 453] to the 'Delivery profiles' table (it must correspond to a logical name) and declare the delivery mode. Possible values are: cluster mode [p 453] or federation mode [p 453].

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 453] 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 471] 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 471] 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.

74.2 Configuring D3 nodes

Runtime configuration of primary and hub nodes through the user interface

The declaration of delivery dataspaces and delivery profiles is done by selecting the '[D3] Primary configuration' feature from the 'Administration' area, where you will find the following tables:

Delivery dataspaces	Declarations of the dataspaces 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 dataspaces 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 453]*

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 454]*

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 453]*

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 459]*

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 dataspaces' 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 dataspaces' 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 dataspaces' 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.

74.3 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 *`NodeMonitoring`*[API]

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 377]*

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 75

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. These are best practices in general, and 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. User directory and Administration rights

75.1 Encryption algorithms

Web Server or Application Server may specify encryption algorithms when setting HTTPS parameters. Some recommendations on these algorithms are provided in section HTTPS [p 478]. Password and fields having osd:password as a type are storing hash of their value with SHA_512 as algorithm. That includes the password of users of the default directory.

75.2 HTTPS

Using HTTPS for communication with clients (GUI and REST or SOAP) is recommended. 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's 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, getting too restrictive on allowed cyphers may prevent some clients to connect as they may not be able to negotiate the 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

75.3 Installation

Deployed components 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.

75.4 Web Server

If you have to expose web applications on the Internet, it's a good practice to protect them with a Web Server in a demilitarized zone while EBX® and the database server may be in a production zone. Here are some best practices for the configuration.

The secure cipher suite and protocols should be set according to the above section HTTPS [p 478].

It is also a best practice not to use the default configuration, and to remove any banner that may 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
```

The Web Server is the recommended way for setting restrictions with HTTP security headers. Be aware that headers related to the origin will 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, EBX® should be configured to not set any HTTP security headers. To do so, the property `ebx.security.headers.activated` must be set to 'false'.

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
```

For version 5.9.4, if the property `ebx.security.headers.activated` is not set or set to `true`, the security header must also be unset beforehand. For version 5.9.4, if the property `ebx.security.headers.activated` is set to `false`, the security header does not need to be unset, so ignore the first line in following snippets. For previous versions, the security header must be unset beforehand.

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 shuould 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"
```

75.5 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 478].

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.

75.6 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.

75.7 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.

75.8 User directory and Administration rights

For production and test platforms, EBX® must be integrated with a custom directory [p 428] 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.

Guide du développeur (en anglais)

Introduction

CHAPITRE 76

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

76.1 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 486].
`/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 486].
`/META-INF/MANIFEST.MF`	Optional. If present, EBX® reports the 'Implementation-Title' and 'Implementation-Version' values to *Administration > Configuration technique > Modules et modèles de données*.
`/www/`	This optional directory contains all packaged resources, which are accessible via public URL. See Packaged resources [p 488].

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.

76.2 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 *Declaration and configuration* of legacy user services. From the version 5.8.0, it is strongly advised to use the new user services [p 589].	No.
`beans`	Declares reusable Java bean components. See the workflow package [p 575].	No.
`ajaxComponents`	Declares Ajax components. See **Declaring an Ajax component in a module** `UIAjaxComponent.declareInModule`[API] in the Java API.	No.

76.3 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 `ModuleRegistrationListener`[API].

> **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: JavaTM Servlet 3.0 Specification for more information.

or:

- contain a Servlet extending the class `ModuleRegistrationServlet`[API];
- 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 `ModuleRegistrationServlet`[API] 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 387].
- 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 337].

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
                          http://java.sun.com/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
  ...
 }

}
```

76.4 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`[API]

Type `osd:resource` *[p 510]*

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 77

Mapping to Java

Ce chapitre contient les sections suivantes :

1. How to access data from Java?
2. Concurrency and isolation levels
3. Mapping of data types
4. Java bindings

77.1 How to access data from Java?

Read access

Data can be read from various generic Java classes, mainly `Adaptation`[API] and `ValueContext`[API]. The getter methods for these classes return objects that are typed according to the mapping rules described in the section Mapping of data types [p 492].

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 `ValueContextForUpdate`[API], or
- During the user input validation, by calling the method `setNewValue` of the class `ValueContextForInputValidation`[API].

Modification of mutable objects

According to the mapping that is described in the Mapping of data types [p 492] 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.

77.2 Concurrency and isolation levels

Highest isolation level

The highest isolation level in ANSI/ISO SQL is `SERIALIZABLE`. Three execution methods guarantee the `SERIALIZABLE` isolation level within the scope of a dataspace:

- If the client code is run inside a `Procedure`[API] container. This is the case for every update, for exports to XML, CSV or archive, and for data services.
- If the client code accesses a dataspace that has been explicitly locked. See `LockSpec`[API].
- If the client code accesses data in a snapshot.

> **Note**
>
> For custom read-only transactions that run on a dataspace, it is recommended to use `ReadOnlyProcedure`[API].

Default isolation level

If the client code is run outside the contexts that enable `SERIALIZABLE`, its isolation level depends on the persistence mode:

- In semantic mode, the default isolation level is `READ UNCOMMITTED`.
- In relational mode, the default isolation level is the database default isolation level.

Voir aussi *Présentation des modes* *[p 267]*

Java access specificities

In a Java application, a record is represented by an instance of the `Adaptation` class. This object is initially linked to the corresponding persisted record. However, unless the client code is executed in a context that enables the SERIALIZABLE [p 492] isolation level, the object can become "disconnected" from the persisted record. If this occurs and concurrent updates have been performed, they will not be reflected in the `Adaptation` object.

Therefore, it is important for the client code to either be in a `SERIALIZABLE` context, or to regularly look up or refresh the `Adaptation` object.

Voir aussi

`AdaptationHome.findAdaptationOrNull`[API]

`AdaptationTable.lookupAdaptationByPrimaryKey`[API]

`Adaptation.getUpToDateInstance`[API]

77.3 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 506].

Voir aussi *`SchemaNode.createNewOccurrence`*[API]

Multiple cardinality on a simple element

If the attribute `maxOccurs` is greater than 1, the element is an aggregated list and the corresponding instance in Java is an instance of `java.util.List`.

Elements of the list are instances of the Java class that is determined from the mapping of the simple type (see previous section).

Complex data types

Complex type definitions without a class declaration

By default (no attribute `osd:class`), a terminal node of a complex type is instantiated using an internal class. This class provides a generic JavaBean implementation. However, if a custom client Java code has to access these values, it is recommended to use a custom JavaBean. To do so, use the `osd:class` declaration described in the next section.

It is also possible to transparently instantiate, read and modify the mapped Java object, with or without the attribute `osd:class`, by invoking the methods `SchemaNode.createNewOccurrence`[API], `SchemaNode.executeRead`[API] and `SchemaNode.executeWrite`[API].

Mapping of complex types to custom JavaBeans

It is possible to map an XML Schema complex type to a custom Java class. This is done by adding the attribute `osd:class` to the complex node definition. Unless the element has `xs:maxOccurs > 1`, you must also specify the attribute `osd:access` for the node to be considered a *terminal* node. If the element has `xs:maxOccurs > 1`, it is automatically considered to be terminal.

The custom Java class must conform to the JavaBean protocol. This means that each child of the complex type must correspond to a JavaBean property of the class. Additionally, each JavaBean property must be a read-write property, and its implementation must ensure that the value set by the setter method is returned, as-is, by the getter method. Contextual computations are not allowed in these methods.

Example

In this example, the 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 `UIBeanEditor`[API].

```
<xs:element name="customer" osd:access="RW">
  <xs:complexType name="subscriber" osd:class="com.carRental.Customer">
 <xs:sequence>
   <xs:element name="firstName" type="xs:string"/>
   ...
 </xs:sequence>
  </xs:complexType>
</xs:element>
```

Multiple cardinality on a complex element

If the attribute `maxOccurs` is greater than 1, then the corresponding instance in Java is:

- An instance of `java.util.List` for an aggregated list, where every element in the list is an instance of the Java class determined by the mapping of simple types [p 506], or
- An instance of `AdaptationTable`[API], if the property `osd:table` is specified.

77.4 Java bindings

Java bindings allow generating Java types that reflect the structure of the data model. The Java code generation can be done in the user interface. See Generating Java bindings [p 497].

Benefits

Ensuring the link between XML Schema structure and Java code provides a number of benefits:

- **Development assistance:** Auto-completion when typing an access path to parameters, if supported by your IDE.
- **Access code verification:** All accesses to parameters are verified at code compilation.
- **Impact verification:** Each modification of the data model impacts the code compilation state.
- **Cross-referencing:** By using the reference tools of your IDE, it is easy to verify where a parameter is used.

Consequently, it is strongly recommended to use Java bindings.

XML declaration

The specification of the Java types to be generated from the data model is included in the main schema.

Each binding element defines a generation target. It must be located at, in XPath notation, `xs:schema/xs:annotation/xs:appinfo/ebxbnd:binding`, where the prefix `ebxbnd` is a reference to the namespace identified by the URI `urn:ebx-schemas:binding_1.0`. Several binding elements can be defined if you have different generation targets.

The attribute `targetDirectory` of the element `ebxbnd:binding` defines the root directory used for Java type generation. Generally, it is the directory containing the project source code, `src`. A relative path is interpreted based on the current runtime directory of the VM, as opposed to the XML schema.

See bindings XML Schema.

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/">
    <javaPathConstants typeName="com.creditonline.RulesPaths">
     <nodes root="/rules" prefix="" />
    </javaPathConstants>
    <javaPathConstants typeName="com.creditonline.StylesheetConstants">
     <nodes root="/stylesheet" prefix="" />
    </javaPathConstants>
   </ebxbnd:binding>
  </xs:appinfo>
 </xs:annotation>
 ...
```

```
</xs:schema>
```

Java constants can be defined for XML schema paths. To do so, generate one or more interfaces from a schema node, including the root node `/`. The example generates two Java path constant interfaces, one from the node `/rules` and the other from the node `/stylesheet` in the schema. Interface names are described by the element `javaPathConstants` with the attribute `typeName`. The associated node is described by the element `nodes` with the attribute `root`.

CHAPITRE 78

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

78.1 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 389] using the property `backend.mode=development`.

78.2 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.

78.3 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 494]*

78.4 **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.

78.5 **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 79

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 23]*

Ce chapitre contient les sections suivantes :

1. Terminology changes in version 5.9
2. Terminology changes in version 5.0

79.1 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

79.2 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 80

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 505], such as integer, boolean, decimal, date, time;
- The ability to define additional simple types [p 507] and complex types [p 507];
- The ability to define simple lists of items, called aggregated lists [p 516];
- Validation constraints [p 539] (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 508], for example: locale, resource, html;
- Definition of tables [p 519] and foreign key constraints [p 524];
- Mapping data in EBX® to Java beans;
- Advanced validation constraints [p 539] (extended facets), such as dynamic enumerations;
- Extensive presentation information [p 557], 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

80.1 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 34]. 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.

80.2 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

80.3 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 123]*

80.4 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>
```

80.5 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`

80.6 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 81

Data types

This chapter details the data types supported by TIBCO EBX®.

Voir aussi *Tables and relationships* *[p 519]*

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

81.1 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	java.lang.String	
xs:boolean	java.lang.Boolean	
xs:decimal	java.math.BigDecimal	A totalDigits facet with a value equal to 15 is added by default to decimal fields that are contained in a mapped table (relational, historized or replicated table). However, this facet can be overwritten with a greater value in the data model.
xs:dateTime	java.util.Date	
xs:time	java.util.Date	The date portion of the returned Date is always set to '1970/01/01'.
xs:date	java.util.Date	The time portion of the returned Date is always the beginning of the day, that is, '00:00:00'.
xs:anyURI	java.net.URI	
xs:Name (xs:string restriction)	java.lang.String	
xs:int (xs:decimal restriction)	java.lang.Integer	
xs:integer (xs:decimal restriction)	java.lang.Integer	This mapping does not comply with the XML Schema recommendation. Although the XML Schema specification states that xs:integer has no value space limitation, this value space is, in fact, restricted by the Java specifications of the java.lang.Integer object.

The mapping between XML Schema types and Java types are detailed in the section Mapping of data types [p 492].

81.2 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.

81.3 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®.

81.4 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. For a basic `xs:string`, its default user interface in EBX® consists of a dedicated editor with several lines for input and display.

```
<xs:simpleType name="text">
 <xs:restriction base="xs:string" />
</xs:simpleType>
```

`osd:html`

This represents a character string with HTML formatting. A WYSIWYG editor is provided in 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.encryptString`[API].

It is also possible for the default editor to use a different encryption mechanism by specifying a class that implements the interface `Encryption`[API].

```
<xs:element name="password" type="osd:password">
 <xs:annotation>
  <xs:appinfo>
   <osd:uiBean class="com.orchestranetworks.ui.UIPassword">
    <encryptionClass>package.EncryptionClassName</encryptionClass>
   </osd:uiBean>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

It is possible to specify some salt by referencing a path to another field, and by using a class the implements the interface `HashComputation`[API].

```
<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 488]. This type requires the definition of the facet FacetOResource [p 544].

```
<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>
    </include>
    <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 dataspaces that match the specified pattern are included in the set. If `none` then neither children nor descendants of the dataspaces that match the specified pattern are included. If `allDescendants` then all descendants of the dataspaces that match the specified pattern are included. If `allBranchDescendants` then all descendant branches of the dataspaces that match the specified pattern are included. If `allSnapshotDescendants` then all descendant snapshots of the dataspaces that match the specified pattern are included. If `directBranchChildren` then only direct branches of the dataspaces that match the specified pattern are included. If `directSnapshotChildren` then only direct snapshots of the dataspaces 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 dataspaces that match the specified pattern are excluded. If `allDescendants` then all descendants of the dataspaces that match the specified pattern are excluded. If `allBranchDescendants` then all descendant branches of the dataspaces that match the specified pattern are excluded. If `allSnapshotDescendants` then all descendant snapshots of the dataspaces that match the specified pattern are excluded. If `directBranchChildren` then only direct branches of the dataspaces that match the specified pattern are excluded. If `directSnapshotChildren` then only direct snapshots of the dataspaces 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 `DataspaceSetFilter`[API].

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 560] and Control policy [p 548] 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 542].

- `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 `DatasetSetFilter`[API]. 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 560] and Control policy [p 548] for more information.

81.5 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.

81.6 **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: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 519].

81.7 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 **82**

Tables and relationships

Ce chapitre contient les sections suivantes :

1. Tables
2. Foreign keys
3. Associations

82.1 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 516]. Beyond relational capabilities, some features that tables provide are:

- filters and searches;
- sorting, views and hierarchies;
- identity constraints: primary keys, foreign keys [p 524] and uniqueness constraints [p 541];
- 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 298]);
- performance and memory optimization.

Voir aussi

Foreign keys [p 524]

Associations [p 527]

Actions sur les jeux de données existants [p 139]

Vue tabulaire simple [p 129]

Vues hiérarchiques [p 129]

Historique [p 273]

Declaration

A table element, which is an element with *maxOccurs > 1*, is declared by adding the following annotation:

```
<xs:annotation>
 <xs:appinfo>
  <osd:table>
   <primaryKeys>/pathToField1 /pathToField...n</primaryKeys>
  </osd:table>
 </xs:appinfo>
</xs:annotation>
```

Common properties

Element	Description	Required
`primaryKeys`	Specifies the primary key fields of the table. Each field of the primary key must be denoted by its absolute XPath notation that starts just under the root element of the table. If there are multiple fields in the primary key, the list is delimited by whitespace. **Note:** Whitespaces in primary keys of type `xs:string` are handled differently. See Whitespace handling for primary keys of type string [p 552].	Yes.

`defaultLabel`	Defines the end-user display of records. Multiple variants can be specified:
	• A static non-localized expression is defined using the `defaultLabel` element, for example:
	`<defaultLabel>Product: ${./productCode}</defaultLabel>`
	• Static localized expressions are specified using the `defaultLabel` element with the attribute `xml:lang`, for example:
	`<defaultLabel xml:lang="fr-FR">Produit : ${./productCode}</defaultLabel>`
	`<defaultLabel xml:lang="en-US">Product: ${./productCode}</defaultLabel>`
	• A JavaBean that implements the interface `UILabelRenderer`[API] and/or the interface `UILabelRendererForHierarchy`[API]. The JavaBean is specified by means of the attribute `osd:class`, for example:
	`<defaultLabel osd:class="com.wombat.MyLabel"></defaultLabel>`
	Note: The priority of the tags when displaying the user interface is the following:
	1. `defaultLabel` tags with a JavaBean (but it is not allowed to define several renderers of the same type);
	2. `defaultLabel` tags with a static localized expression using the `xml:lang` attribute;
	3. `defaultLabel` 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.

`index`	Specifies an index for speeding up requests that match this index (see performances [p 321]). The attribute `name` is mandatory. Each field of the index must be denoted by its absolute XPath notation, which starts just under the root element of the table. If there are multiple fields in the index, the list is delimited by whitespace. **Note:** • Indexing only concerns semantic and relational tables. History and replica tables are not affected. • It is possible to define multiple indexes on a table. • It is not possible to define two indexes with the same name.	No.

Element	Description	Required
	• It is not possible to declare two indexes containing the exact same fields. • An indexed field must be terminal. • An indexed field cannot be a list nor under a list. • A field declared as an *inherited field* cannot be indexed. • A field declared as a function cannot be indexed. For performance purposes, the following nodes are automatically indexed: • Primary keys nodes. See primary keys [p 519]. • Nodes defining a foreign key constraint. See foreign key constraint [p 524]. • Nodes declared as being unique. See uniqueness constraint [p 541]. • Auto-incremented nodes. See auto-incremented values [p 555].	
`recordForm`	Defines a specific component for customizing the record form in a dataset. This component is defined using a JavaBean that extends `UIForm`[API] or implements `UserServiceRecordFormFactory`[API]. The JavaBean is specified by means of the attribute `osd:class`, for example: `<recordForm osd:class="com.wombat.MyRecordForm"/>`	No.

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 298].	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 298].	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 298].	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 *Héritage de jeu de données [p 297]*

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
`tabularViewTop`	Définit la barre d'outils à afficher en entête de la vue table par défaut.	No.
`tabularViewRow`	Définit la barre d'outils à afficher pour chaque ligne de la vue table par défaut.	No.
`recordTop`	Définit la barre d'outils à afficher en entête du formulaire d'un enregistrement.	No.
`hierarchyViewTop`	Définit la barre d'outils à afficher en entête de la hiérarchie par défaut de la table.	No.

Voir aussi *Toolbars [p 573]*

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.

82.2 Foreign keys

Declaration

A reference to a table [p 519] 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 `display` and `pattern` elements. These static expressions can be localized using the additional attribute `xml:lang` on the `pattern` element, for example:
	`<display>`
	`<pattern>Product : ${./productCode}</pattern>`
	`<pattern xml:lang="fr-FR">Produit : ${./productCode}</pattern>`
	`<pattern xml:lang="en-US">Product: ${./productCode}</pattern>`
	`</display>`
	• A JavaBean that implements the interface `TableRefDisplay`[API]. It is specified using the attribute `osd:class`. For example:
	`<display osd:class="com.wombat.MyLabel"></display>`
	It is not possible to define both variants on the same foreign key element.
	Attention: 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 `display` property is not specified, the table's record rendering [p 521] is used.

`filter`	Specifies an additional constraint that filters the records of the target table. Two types of filters are available: • An XPath filter is an XPath predicate in the target table context. It is specified using the `predicate` element. For example: `<filter><predicate>type = ${../refType}</predicate></filter>` A localized validation message can be specified using the element `validationMessage`, which will be displayed to the end-user at the validation time if a record is not accepted by the filter. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute.	No.

Element	Description	Required
	To specify a default message for unsupported locales, define a `message` element with no `xml:lang` attribute. 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 `TableRefFilter`[API]. It is specified using the attribute `osd:class`. For example: `<filter osd:class="com.wombat.MyFilter"></filter>` Additional validation messages can be specified during the setup of the programmatic filter. 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 `TableRefFilterContext.setFilterForSearch`[API]. **Note:** The attributes `osd:class` and the property `predicate` cannot be set simultaneously. The validation search XPath functions are forbidden on a `tableRef` filter.	
`validation`	Specifies localized validation messages for the `osd:tableRef` and error management policy. A specific severity level can be defined in a nested `severity` element. The default severity is 'error'. An error management policy can be defined in a nested `blocksCommit` element. The error management policy that blocks all operations does not apply to filters. That is, a foreign key constraint is not blocking if a referenced record exists but does not satisfy a foreign key filter. In this case, updates are not rejected, and a validation error will be reported. Each localized message variant is defined in a nested `message` element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a `message` element with no `xml:lang` attribute.	No.

Attention

You can create a 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 519]

Primary key syntax `PrimaryKey.syntax`[API]

Extraction of foreign keys (XPath predicate syntax) [p 255]

Associations [p 527]

View for advanced selection [p 567]

`SchemaNode.getFacetOnTableReference`[API]

`SchemaFacetTableRef`[API]

82.3 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 524].
- *'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 566].
- 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 *foreign key*. Element `fieldToSource` defines the foreign key that refers to the source table of the association. The element `fieldToSource` is mandatory and must specify a foreign key field that refers to the table containing the association.	Yes if the association is the inverse relationship of a *foreign key*, otherwise no.
`linkTable`	Defines the properties of an association over a link table. The element `table` specifies the link table used by the association. The element `table` is mandatory and must refer to an existing table. **Important:** In order to be used by an association, a link table must define a primary key that is composed of auto-incremented fields and/or the foreign key to the source or target table of the association. The element `fieldToSource` defines the foreign key that refers to the source table of the association. The element `fieldToSource` is mandatory and must specify a foreign key field that refers to the table containing the association. The element `fieldToTarget` defines the foreign key that refers to the target table of the association. The element `fieldToTarget` 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 *XPath predicate*. The `predicate` element specifies the criteria of the association, relative to the current node. Examples: `/root/Products[catalog_ref =${../catalogId}]` or `//Products[catalog_ref = ${../catalogId}]` or `../Products[catalog_ref =${../catalogId}]`. The path to the predicate, for example `../Products`, 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 using the notation `${<relative-path>}` where `relative-path` is a path that identifies the element relative to the association node. See EBX® XPath supported syntax [p 249]. **Note**	Yes if the association is based on an *XPath predicate*, otherwise no.

Element	Description	Required
	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: `<filter><predicate>type = ${../refType}</predicate></filter>` 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 `checkOnAssociatedRecordCreation/message`. Each localized message variant is defined in a nested message element with its locale in an `xml:lang` attribute. To specify a default message for unsupported locales, define a message element with no `xml:lang` attribute. See Examples [p 536] 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 `filter`. 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 `UserServiceAssociationRecordFormFactory`[API]. The JavaBean is specified by means of the attribute `osd:class`, for example: `<recordForm osd:class="com.wombat.MyRecordFormFactory"/>`	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
`schemaLocation`	Defines the data model containing the fields used by the association. The data model is defined using a specific URN that allows referring to embedded data models and data models packaged in modules. See `SchemaLocation`[API] for more information about specific URNs supported by EBX®.	Yes.
`dataSet`	Defines the dataset used by the association. This dataset must use the data model specified by the element `schemaLocation`.	Yes.
`dataSpace`	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.

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 573]*

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 `nodePath` defines the path of the field that can be used to sort associated objects. The element `isAscending` specifies whether the sort order is ascending (true) or descending (false).	No.

The following example shows how to define a tabular view from the previous association between a catalog and its products:

```
<xs:element name="Products" minOccurs="0" maxOccurs="0" type="xs:string">
 <xs:annotation>
  <xs:appinfo>
   <osd:association>
    <tableRefInverse>
     <fieldToSource>/root/Products/catalog_ref</fieldToSource>
    </tableRefInverse>
   </osd:association>
   <osd:defaultView>
    <tabularView>
     <column>/productRange</column>
     <column>/productCode</column>
     <column>/productLabel</column>
     <column>/productDescription</column>
     <sort>
        <nodePath>/productLabel</nodePath>
        <isAscending>true</isAscending>
     </sort>
    </tabularView>
   </osd:defaultView>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Actions in the user interface

In the user interface, it is possible to perform the following actions:

- **Create**: it allows directly creating an object in the target table of the association. When a new object is created, it is automatically associated with the current record.
- **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
`viewForAssociateAction`	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.
`viewForMoveAction`	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>
   <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
`minOccurs`	Specifies the minimum number of associated objects that are required for this association. This minimum number is defined using the element `value` and must be a positive integer.	No, by default the minimum is not restricted.
`maxOccurs`	Specifies the maximum number of associated objects that are allowed for this association. This maximum number is defined by the element `value` and must be either a positive integer or the raw string `unbounded` which indicates that this maximum is not restricted. The maximum number of associated objects must be greater than the minimum number of associated objects.	No, by default the maximum is not restricted.
`constraint`	Defines an XPath predicate for restricting associated records. It is only possible to use fields from the source and the target table when defining an XPath predicate. That is, if it is an association over a link table, it is not possible to use fields of the link table in the XPath predicate. In associated 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.
`validation`	A validation message can be defined under the elements `minOccurs`, `maxOccurs` and `constraint`, using the element `validation`. The severity of the validation message is specified using the element `severity`. Possible severities are: `error`, `warning` and `info`. If the severity is not specified then, by default, the severity `error` is used. A localized validation message can be specified using the element `message`, which will be displayed to the end-user at 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 `xml:lang` attribute. To specify a default message for unsupported locales, define a message element with no `xml:lang` attribute.	No.

Data services integration

It is possible to define whether associated objects must be hidden in the Data service `select` operation. For this, the property `osd:defaultView/hiddenInDataServices` under `xs:annotation/xs:appinfo` can be set on the association. Setting the property to 'true' will hide associated objects in the Data

service `select` operation. If this property is not defined then, by default, associated objects will be shown in the Data service `select` operation.

Voir aussi

Hiding a field in Data Services [p 566]

Association field [p 642]

Examples

For example, the product catalog data model defined previously [p 522] 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>
```

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>
```

CHAPITRE 83

Constraints, triggers and functions

Facets allow you to define data constraints in your data models. TIBCO EBX® supports XML Schema 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
5. Triggers and functions

83.1 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 542].

- **4 -** `osd:dataspaceKey`, `osd:datasetName` and `osd:color` types do not support facets. Only Programmatic constraints [p 546] 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 508][3]						1
osd:dataspaceKey [p 510][4]						1
osd:datasetName [p 512][4]						1
osd:color [p 510][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 508][3]						
osd:dataspaceKey [p 510][4]						
osd:datasetName [p 512][4]						
osd:color [p 510][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.

Limitations:

1. The target of the `xs:field` element must be in a table.
2. The uniqueness constraint does not apply to fields inside an aggregated list.
3. The uniqueness constraint does not apply to computed fields.

Voir aussi ***Uniqueness constraint in the Java API*** `UniquenessConstraint`[API]

83.2 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 524] 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 510]. 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 `<dependencies>` in the file `module.xml`.
`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 www/common/images/, situé au même niveau que le répertoire WEB-INF/ 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 (www/fr/ par exemple), elle sera utilisée uniquement si une ressource du même nom est définie dans le répertoire www/common/.

This facet has the same behavior as an enumeration facet: the values are collected by recursively listing all the files in the local path in the specified resource type directory in the specified module.

Example:

```
<xs:element name="promotion" type="osd:resource">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:FacetOResource osd:moduleName="wbp"
     osd:resourceType="ext-images" osd:relativePath="promotion/" />
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</ xs:element>
```

For an overview of the standard directory structure of an EBX® module (Java EE web application), see Module structure [p 485].

excludeValue constraint

This facet verifies that a value is not the same as the specified excluded value.

In this example, the empty string is excluded from the allowed values.

Example:

```
<xs:element name="roleName">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:excludeValue value="">
     <osd:validation>
```

```
      <severity>error</severity>
       <message>Please select address role(s).</message>
     </osd:validation>
    </osd:excludeValue>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType type="xs:string" />
</xs:element>
```

excludeSegment constraint

This facet verifies that a value is not included in a range of values. Boundaries are excluded.

Example:

In this example, values between 20000 and 20999 are not allowed.

```
<xs:element name="zipCode">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:excludeSegment minValue="20000" maxValue="20999">
     <osd:validation>
      <severity>error</severity>
       <message>Postal code not valid.</message>
     </osd:validation>
    </osd:excludeSegment>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 <xs:simpleType type="xs:string" />
</xs:element>
```

Enumeration facet defined using another node

By default, an enumeration facet is described statically in XML Schema.

The content of an enumeration facet can also be provided dynamically by a list of simple elements in the data model.

Example:

In this example, the content of an enumeration facet is sourced from the node `CountryList`.

```
<xs:annotation>
 <xs:appinfo>
  <osd:otherFacets>
   <osd:enumeration osd:path="../CountryList" />
  </osd:otherFacets>
 </xs:appinfo>
</xs:annotation>
```

The 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>
```

83.3 Programmatic facets

A programmatic constraint can be added to any XML element declaration for a simple type.

In order to guarantee XML Schema conformance, programmatic constraints are specified under the element `annotation/appinfo/otherFacets`.

Programmatic constraints

A programmatic constraint is defined by a Java class that implements the interface `Constraint`[API].

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol.

Example:

In the example below, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

```
<xs:element name="amount">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraint class="com.foo.CheckAmount">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraint>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Voir aussi ***JavaBean specifications*** *`Package com.orchestranetworks.schema.JavaBeans`*[API]

Programmatic enumeration constraints

An enumeration constraint adds an ordered list of values to a basic programmatic constraint. This facet allows selecting a value from a list. It is defined by a Java class that implements the interface `ConstraintEnumeration`[API].

Example:

```
<xs:element name="amount">
 <xs:annotation>
```

```
  <xs:appinfo>
   <osd:otherFacets>
    <osd:constraintEnumeration class="com.foo.CheckAmountInEnumeration">
     <param1>...</param1>
     <param...n>...</param...n>
    </osd:constraintEnumeration>
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

Constraint on 'null' values

In some cases, a value is only mandatory if some conditions are satisfied, for example, if another field has a given value. In this case, the standard XML Schema attribute `minOccurs` is insufficient because it is static.

In order to check if a value is mandatory according to its context, the following requirements must be satisfied:

1. A programmatic constraint must be defined by a Java class (see above).
2. This class must implement the interface `ConstraintOnNull`[API].
3. The XML Schema cardinality attributes must specify that the element is optional (`minOccurs="0"` and `maxOccurs="1"`).

> **Note**
>
> By default, constraints on 'null' values are not checked upon user input. In order to enable a check at the input, the 'checkNullInput' property [p 551] 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>
```

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 [p 326].

Voir aussi ***JavaBeans Specifications*** *Package com.orchestranetworks.schema.JavaBeans*[API]

83.4 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:

`onInsertUpdateOrDelete`	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. This is also the default policy for foreign key constraints that are automatically set in blocking mode (because of a table in relational mode involved in the relationship) and that do not define a control policy.
`onUserSubmit-checkModifiedValues`	Specifies that the constraint must remain valid whenever a user modifies the associated value and submits a form. In this case, any form input that would violate the constraint is rejected and the values remain unchanged. This is the default policy for all blocking constraints mentioned in the previous case. For example, a foreign key constraint is by default not blocking (a record referred to by other records can be deleted, etc.), except in the context of a form submit.
`never`	Specifies that the constraint must never block operations. In this case, any operation that would violate the constraint is allowed. In the context of the user interface, this constraint does not block 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 on relational data models or in mapped tables. That is, this property is not available for fixed length, maximum length, maximum number of digits, and decimal place constraints due to the validation policy of the underlying RDBMS blocking constraints.

This property does not apply to archive imports and when merging dataspaces. That is, all blocking constraints, except structural constraints, are always disabled when importing archives and merging dataspaces.

Voir aussi

Facet validation message with severity *[p 560]*

Foreign keys *[p 524]*

Mode relationnel [p 267]

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.setBlocksCommitToNever`[API] and `ConstraintContextOnTable.setBlocksCommitToNever`[API] in the Java API for more information.

Example:

```
<xs:element name="price" type="xs:decimal">
 <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:minInclusive path="../priceMin">
     <osd:validation>
      <blocksCommit>onInsertUpdateOrDelete</blocksCommit>
     </osd:validation>
    </osd:minInclusive>
```

```
   </osd:otherFacets>
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

Check 'null' input

According to the 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"`.

> **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 547]*

Whitespace management *[p 551]*

Empty string management *[p 553]*

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 `#x20`s 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 519]*

Empty string management

Default conversion

For nodes of type `xs:string`, no distinction is made at user input between an empty string and a `null` value. That is, an empty string value is automatically converted to `null` at user input.

Distinction between empty strings and 'null' value

There are certain cases where the distinction is made between an empty string and the `null` value, such as when:

- A primary key defines a pattern that allows empty strings.
- An element defines a foreign key constraint and the target table has a single primary key defining a pattern that allows empty strings.
- An element defines a static enumeration that contains an empty string.
- An element defines a dynamic enumeration to another element with one of the aforementioned cases.

If the distinction is made between an empty string and a `null` value, this implies the following behaviors:

- An empty string will not be converted to `null` at user input,
- Input fields for nodes of type `xs:string` will display an additional button for setting the value of the node to `null`,
- At validation time, an empty string will be considered to be a compliant value with regard to the `minOccurs="1"` property.

Attention

In relational mode, the Oracle database does not support the distinction between empty strings and `null` values, and these specific cases are not supported.

Voir aussi *Mode relationnel [p 267]*

83.5 Triggers and functions

Computed values

By default, data is read and persisted in the XML repository. Nevertheless, data may be the result of a computation and/or external database access, for example, an RDBMS or a central system.

EBX® allows taking into account other data in the current dataset context.

This is made possible by defining *functions*.

A function is specified in the data model using the `osd:function` element (see example below).

- The value of the *class* attribute must be the qualified name of a Java class that implements the Java interface `ValueFunction`[API].

- Additional parameters may be specified at the data model level, in which case the JavaBean convention is applied.

Example:

```
<xs:element name="computedValue">
 <xs:annotation>
  <xs:appinfo>
   <osd:function class="com.foo.ComputeValue">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:function>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

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">
    <param1>...</param1>
    <param...n>...</param...n>
   </osd:function>
  </xs:appinfo>
 </xs:annotation>
 ...
</xs:element>
```

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.

For dataset triggers, a Java class that extends the abstract class `InstanceTrigger`[API] 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>
```

For the definition of table record triggers, a Java class that extends the abstract class `TableTrigger`[API] must be defined inside the `osd:trigger` element. It is advised to define the `annotation/appinfo/osd:trigger` elements just under the element describing the associated table or table type.

Examples:

On a table element:

```
<xs:element name="myTable" type="MyTableType" minOccurs="0" maxOccurs="unbounded">
 <xs:annotation>
  <xs:appinfo>
   <osd:table>
    <primaryKeys>/key</primaryKeys>
   </osd:table>
   <osd:trigger class="com.foo.MyTableTrigger" />
  </xs:appinfo>
 </xs:annotation>
</xs:element>
```

On a table type element:

```
<xs:complexType name="MyTableType">
   ...
   <xs:annotation>
    <xs:appinfo>
    <osd:trigger class="com.foo.MyTableTrigger">
    <param1>...</param1>
    <param...n>...</param...n>
    </osd:trigger>
    </xs:appinfo>
   </xs:annotation>
   ...
</xs:complexType>
```

As additional parameters can be defined, the implemented Java class must conform to the JavaBean protocol. In the example above, the Java class must define the methods: `getParam1()`, `setParam1(String)`, `getParamX()`, `setParamX(String)`, etc.

Auto-incremented values

It is possible to define auto-incremented values. Auto-incremented values are only allowed inside tables, and they must be of 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 dataspaces. 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 dataspaces).

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 **84**

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

84.1 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 `<osd:label>` and `<osd:description>` respectively.
Simple	The label is extracted from the text content, ending at the first period ('.'), with a maximum of 60 characters. The description uses the remainder of the text.

The description may also have a hyperlink, either a standard HTML `href` to an external document, or a link to another node of the adaptation within 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`*[API]

84.2 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>
```

84.3 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.

84.4 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>
```

84.5 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 85

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

85.1 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" />
```

85.2 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`	*Adaptation terminal node*: value can be read and written in EBX®.
`R-`	*Adaptation terminal node*: value can only be read in EBX®.
`CC`	*Cut*: This is not an adaptation terminal node and none of its children are adaptation terminal nodes. This "instruction" has priority over any child node regardless of the value of their *access* attribute.
`--`	If the node is a simple type, it is not adaptable. If the node is a complex type, it is not an adaptation terminal node and does not define any child nodes. The `root` node of a data model must specify this access mode.
Default	If the *access* attribute is not defined: • If the node is a computed value, it is considered to be `R-` • If the node is a simple type and its value is not computed, it is considered to be `RW` • If the node is an aggregated list, it is then a terminal value and is considered to be `RW` • Otherwise, it is not an adaptation terminal node and it does not define anything about its child nodes.

Example:

In this example, the element is adaptable because it is an adaptation terminal node.

```
<xs:element name="proxyIpAddress" type="xs:string" osd:access="RW"/>
```

85.3 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>
```

85.4 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 search tools

To specify whether or not to hide an element in search tools, use the element `osd:defaultView/hiddenInSearch="true|false|textSearchOnly"`.

If this property is set to `true`, then the field will not be selectable in the text and typed search tools of a 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.

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 641].

> **Note**
>
> - If a group is configured as being hidden, then all the fields nested under this group will be considered as hidden by data services.

Example:

```
<xs:element name="hiddenFieldInDataService" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
    <osd:defaultView>
    <hiddenInDataServices>true</hiddenInDataServices>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

In this example, the element is hidden in the Data Service select operation.

Defining a view for the combo box selector of a foreign key

It is possible to specify a published view that will be used to display the target table or the hierarchical view of a foreign key for a smoother selection. If a view has been defined, the selector will be displayed in the user interface in the combo box of this foreign key. The definition of a view can be done by using the XML Schema element `osd:defaultView/widget/viewForAdvancedSelection`.

> **Note**
>
> - This property can only be defined on foreign key fields.
> - The published view must be associated with the target table of the foreign key.
> - If the published view does not exist, then the advanced selection is not available in the foreign key field.

Example:

In this example, the name of a published view is defined to display the target table of a foreign key in the advanced selection.

```
<xs:element name="catalog_ref" type="xs:string" minOccurs="0"/>
   <xs:annotation>
  <xs:appinfo>
   <osd:otherFacets>
    <osd:tableRef>
     <tablePath>/root/Catalogs</tablePath>
    </osd:tableRef>
     </osd:otherFacets>
    <osd:defaultView>
    <widget>
     <viewForAdvancedSelection>catalogView</viewForAdvancedSelection>
    </widget>
    </osd:defaultView>
  </xs:appinfo>
   </xs:annotation>
</xs:element>
```

See Combo-box selector [p 60] for more information.

Customizing a default widget

A widget can be defined using the data model assistant. See Default view > Widget [p 60] for more information.

85.5 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"/>
```

85.6 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"/>
```

85.7 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 **86**

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

86.1 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 44]*

86.2 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 overrided 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.

86.3 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.

86.4 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 622]*

86.5 Limitations

WSDL operations' names are not available with `external` data models.

CHAPITRE 87

Toolbars

This chapter details how toolbars are defined and managed by TIBCO EBX®.

Ce chapitre contient les sections suivantes :

1. Definition
2. Using toolbars

87.1 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 523]*

Toolbar API `ToolbarFactory`[API]

87.2 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 523]*

Associations *[p 527]*

CHAPITRE 88

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

88.1 Bean categories

Step	**Library**	Specific
Scripts	ScriptTaskBean	ScriptTask
Conditions	ConditionBean	Condition
User task	UserTask	

88.2 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]

88.3 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>
```

88.4 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]

88.5 **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*** `Condition`[API]

88.6 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*** `ConditionBean`[API]

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>
```

88.7 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>
```

88.8 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>
```

88.9 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>
```

88.10 **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>
```

88.11 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 89

Interface customization

The TIBCO EBX® graphical interface can be customized through various EBX® APIs.

Ce chapitre contient les sections suivantes :

1. How to embed a Web Component
2. User services
3. Form layout
4. Custom widgets
5. Table filter
6. Record label
7. CSS and JavaScript

89.1 How to embed a Web Component

EBX® can be integrated into any application that is accessible through a supported web browser, thanks to the Web Component API.

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 *Utiliser TIBCO EBX® comme composant web [p 211]*

89.2 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 589]*

89.3 Form layout

It is possible to override the default layout of forms in the user interface by highly customizing it with the `UIForm` API. 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

`UIForm`[API]

`UIFormPaneWriter`[API]

`UIWidget`[API]

`UIFormHeader`[API]

`UIFormBottomBar`[API]

89.4 Custom 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]

89.5 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 55]

`UILabelRendererForHierarchy`[API]

89.6 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]

89.7 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 486]*

Development recommendations *[p 617]*

`UIDependencyRegisterer`[API]

CHAPITRE 90

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

90.1 **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 *single* 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 *record* 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 *single* association target record.

90.2 **Declaration**

A user service can be declared at two levels:

- Module,
- Data model.

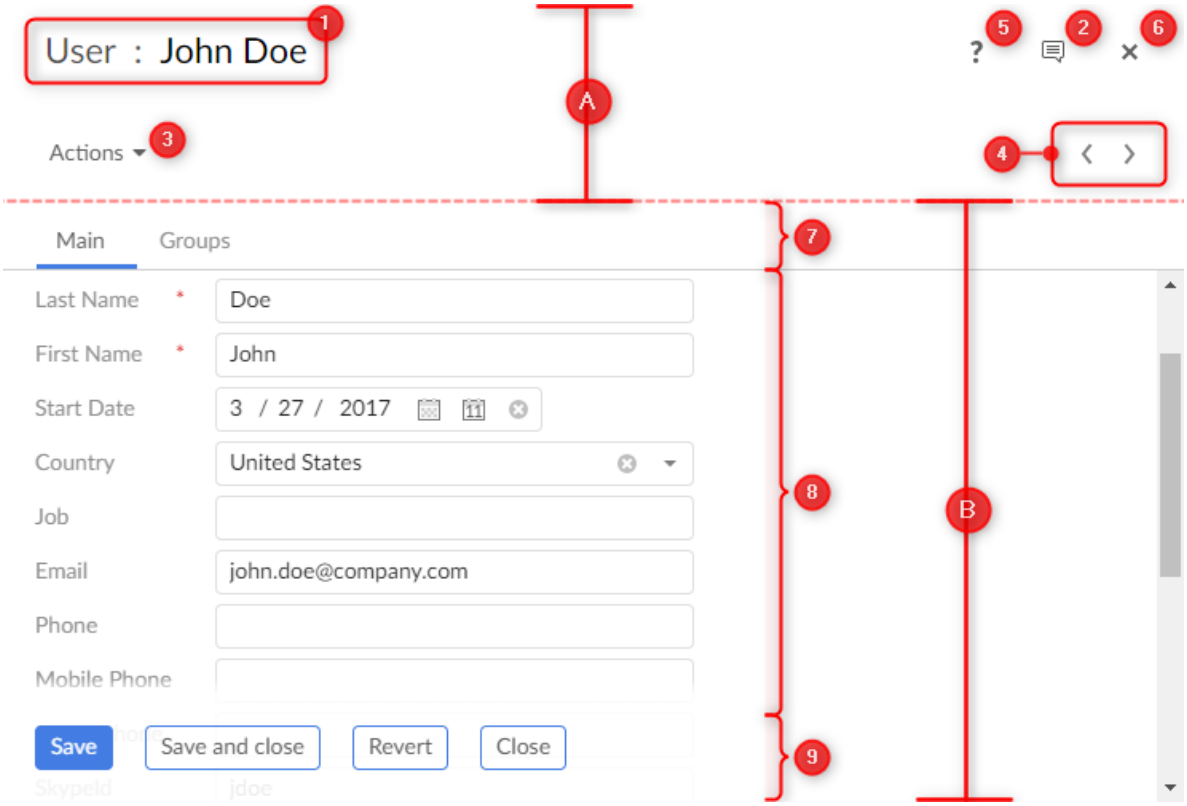
A service declared by a data model can only be launched when the current selection includes a dataset of this model. The user service cannot be of the `Dataspace` nature.

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.

90.3 Display

On the following figure are displayed the functional areas of a user service.



A. Header B. Form	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

Most areas are optional and customizable. Refer to Quick start [p 593], Implementing a user service [p 597] and Declaring a user service [p 611] for more details.

90.4 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 **91**

Quick start

Ce chapitre contient les sections suivantes :

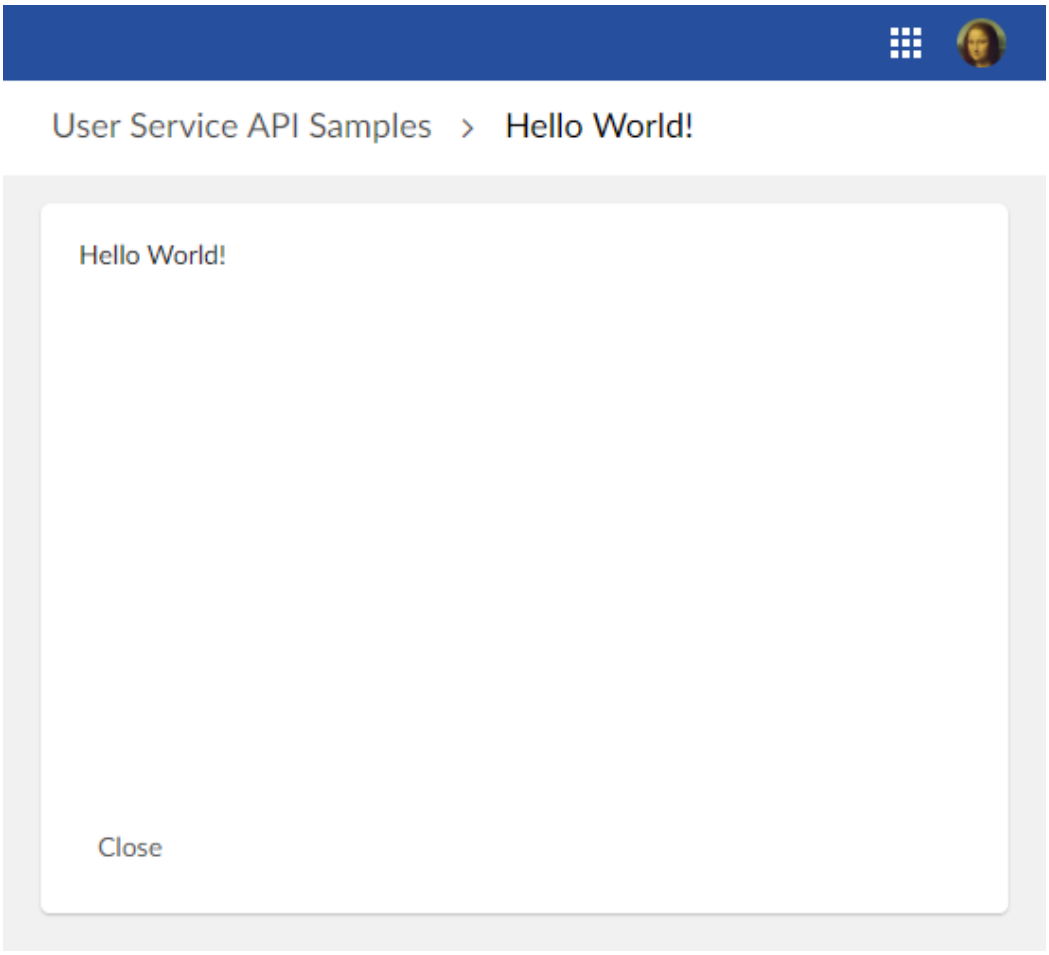
1. Main classes
2. Hello world

91.1 Main classes

The minimum requirement is to implement two classes, one for the service declaration and one for the implementation itself.

91.2 **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 HelloWordService implements UserService<DatasetEntitySelection>
{
 public HelloWordService()
 {
 }

 @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 HelloWordService();
 }

 @Override
 public void defineActivation(ActivationContextOnDataset aContext)
 {
  // The service is activated for all datasets instanciated 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 `SchemaExtensions`[API].

CHAPITRE 92

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

92.1 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>`

92.2 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.createUserService`[API] 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.

92.3 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.setupObjectContext`[API].

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** `Adaptation`[API], **path** `Path`[API] 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 final static ObjectKey _PersonObjectKey = ObjectKey.forName("person");
 private final static ObjectKey _PartnerObjectKey = ObjectKey.forName("partner");

 // Define a path for each property:
 private final static Path _FirstName = Path.parse("firstName");
 private final static Path _LastName = Path.parse("lastName");
 private final static 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 familly 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);
  }

  ...
 }
```

92.4 **Display setup**

The display is set up by the user service's implementation of the method `UserService.setupDisplay`[API].

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.setupDisplay`[API].

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

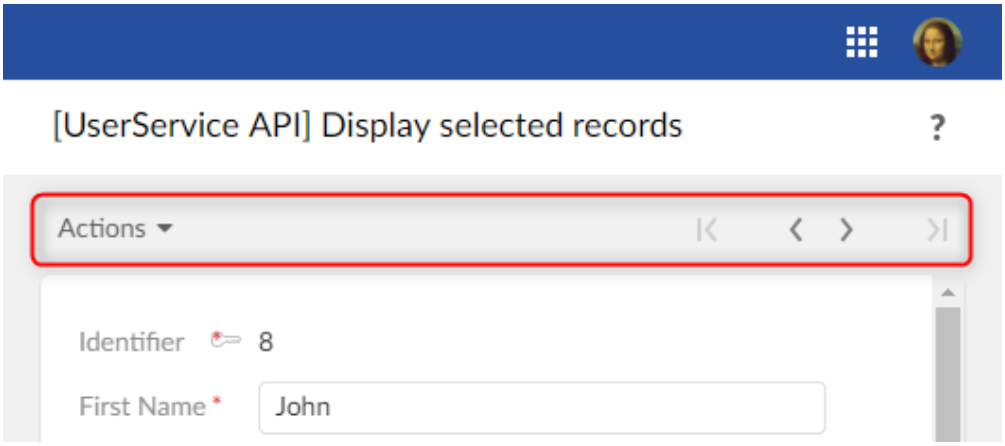
This callback usually implements the interface `UserServicePane`[API] to render a plain EBX® form. The callback can also be an instance of `UserServiceRootTabbedPane`[API] to render an EBX® form with tabs.

For specific cases, the callback can implement `UserServiceRawPane`[API]. This interface has restrictions but is useful when one wants to implement an HTML form that is not managed by EBX®.

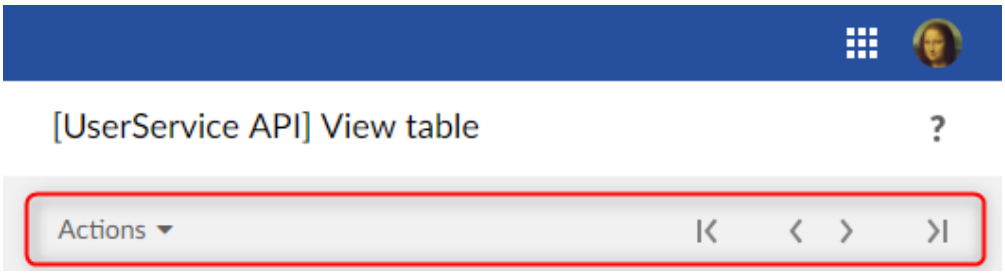
Toolbars

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 final static 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 succeded.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 ULR 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 HelloWordService 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;
 }
}
```

92.5 **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 final static ObjectKey _Person1_ObjectKey = ObjectKey.forName("person1");
 private final static 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;
}
```

92.6 **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>");
 }
}
```

92.7 **REST data services**

A user service can access REST data services through HTTP requests.

The client should use the URL generated by: `UIResourceLocator.getURLForRest`[API]. This URL includes required information for the user authentication.

For more information on REST data services see the Built-in RESTful services [p 683].

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-dataservices/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("}");
 }
}
```

92.8 **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 final static ObjectKey _File_ObjectKey = ObjectKey.forName("file");

 // Paths for the "file" object.
 public final static Path _Title = Path.parse("title");
 public final static 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 `UIWidgetFileUpload`[API].

92.9 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.getDateTimeInstance(
   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();
 }
}
```

92.10 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** `UserServiceExtended`[API] and method `UserServiceExtended.initialize`[API].

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 **93**

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

93.1 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`

93.2 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.

93.3 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 486] 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].

93.4 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].

93.5 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 dataspaces except the "Reference" branch.
  aContext.includeAllDataspaces(DataspaceType.BRANCH);
  aContext.excludeDataspacesMatching(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.defineActivation`[API].

For more information about the resolution of the user service availability, see Résolution de permissions pour les services [p 313].

93.6 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.declareWebComponent`[API] 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 Utiliser TIBCO EBX® comme composant web [p 211] 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 486] 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].

93.7 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 614] provided by EBX®,
- Custom User Service Groups [p 615] declared in a module.

The link between groups and services is made upon service declaration. See Associating a service to a group [p 615].

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 **94**

Development recommendations

Ce chapitre contient les sections suivantes :

1. HTML
2. CSS
3. JavaScript

94.1 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 `UIHttpManagerComponent`[API]. 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 `UIComponentWriter`[API], uses a `UIHttpManagerComponent`[API] 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("\";");
```

94.2 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 486].

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;
}
```

94.3 JavaScript

Public JS functions

The catalog of JavaScript functions `JavaScriptCatalog`[API] 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 `UIJavaScriptWriter`[API].

This JavaScript is executed once the whole page is loaded. It is possible to instantly manage the HTML elements written with `UIBodyWriter.add`[API]. 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 486]. 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 `UIDependencyRegisterer`[API] 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 **95**

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

95.1 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 641], for an integration use case, a WSDL [p 633] 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 dataspaces 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 630].

95.2 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 343] for more information.

In case of specific deployment, for example using reverse-proxy mode, see URLs computing [p 382] 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 381] and Using JMS [p 624] for more information.

95.3 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.getTrackingInfo`[API] 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.getInputParameterValue`[API] 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 381]. 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`.

95.4 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 QWxpYmFiYTpvcGVuIHNlc2FtZQ==
```

- '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 636] section.

For more information on this authentication scheme, see `Directory.authenticateUserFromLoginPassword`[API].

- 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 641].

- '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.authenticateUserFromHttpRequest`[API] for more information.
- The 'SOAP Specific Header Authentication Scheme'.

  For more information, see Overriding the SOAP security header [p 625].

Global permissions

Global access permissions can be independently defined for the SOAP and WSDL connector accesses. For more information see Global permissions [p 409].

Overriding the SOAP security header

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

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

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

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

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

The corresponding XML Schema header declaration would be as follows:

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

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

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

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

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

> **Note**
>
> Only available for SOAP operations [p 641].

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 625] • The SOAP request is received over the JMS protocol. • The SOAP header content must contains a `Security` element. SOAP Specific Header [p 625] • The SOAP request is received over the JMS protocol. • The SOAP header content must not contain a `Security` element.
SOAP / HTTP	Basic [p 624] • 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 624] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. SOAP Security Header [p 625] • 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 625] • The HTTP request must not satisfy the conditions of the previous authentication methods. SOAP Specific Header [p 625] • 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 624] • 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 624] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. Specific [p 625] • 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`.

95.5 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 377]*

TIBCO EBX® main configuration file *[p 371]*

95.6 **SOAP and REST comparative**

Operations	**SOAP**	**REST**
Data		
Select or count records (with filter and/or view publication)	X	X
Selector for possible enumeration values (with filter)		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 dataspaces or snapshots	X	
Refresh a replication unit	X	
Get credentials for records	X	
Dataspaces		
Create, close, merge a dataspace	X	
Create, close a snapshot	X	
Validate a dataspace or a snapshot	X	
Validate a dataset	X	
Locking a dataspace	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

Operations	SOAP	REST
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 747] for more information.

95.7 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 **96**

WSDL generation

Ce chapitre contient les sections suivantes :

1. Supported standard
2. Operation types
3. WSDL download methods
4. HTTP examples

96.1 Supported standard

TIBCO EBX® generates a WSDL that complies with the W3C Web Services Description Language 1.1 standard.

96.2 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 637] or `operations` [p 637] 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.

96.3 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 202] 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 634].
- `{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 634]. 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` *(OK)*	The WSDL content was successfully generated and is returned by the request (optionally in an attachment [p 638]).
`400` *(Bad request)*	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 572] for more information.
`401` *(Unauthorized)*	Request requires an authenticated user.
`403` *(Forbidden)*	Request is not allowed for the authenticated user.
`405` *(Method not allowed)*	Request is not allowed in this configuration.
`500` *(Internal error)*	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`.

96.4 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 97

SOAP operations

Ce chapitre contient les sections suivantes :

1. Operations generated from a data model
2. Operations on datasets and dataspaces
3. Operations on data workflows
4. Administrative services

97.1 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 301] 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 566]*

Permissions *[p 301]*

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 249] 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 235].	Only required for the insert and update operations
viewPublication	This parameter can be combined with the predicate [p 643] parameter as a logical `AND` operation. The behavior of this parameter is described in the section EBX® as a Web Component [p 215]. It cannot be used if the 'viewId' parameter is used, and cannot be used on hierarchical views.	No
viewId	*Deprecated since version 5.2.3*. This parameter has been replaced by the parameter 'viewPublication'. While it remains available for backward compatibility, it will eventually be removed in a future version. This parameter cannot be used if the 'viewPublication' parameter is used.	No
blockingConstraintsDisabled	This property is available for all table updates data service operations. If `true`, the validation process disables blocking constraints defined in the data model. If this parameter is not present, the default is `false`. See Blocking and non-blocking constraints [p 548] 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 643] 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 380] 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 643].	
version	See the description under Common parameters [p 643].	
instance	See the description under Common parameters [p 643].	
predicate	See the description under Common parameters [p 643]. This parameter can be combined with the viewPublication [p 643] parameter as a logical `AND` operation.	
viewPublication	See the description under Common parameters [p 643].	
includesTechnicalData	The response will contain technical data if `true`. See also the optimistic locking [p 660] section. Each returned record will contain additional attributes for this technical information, for instance: `...<table ebxd:lastTime="2010-06-28T10:10:31.046" ebxd:lastUser="Uadmin" ebxd:uuid="9E7D0530-828C-11DF-B733-0012D01B6E76">... .`	No
exportCredentials	If `true` the select will also return the credentials for each record.	No
pagination	Enables pagination, see child elements below.	No
pageSize (nested under the `pagination` element)	When pagination is enabled, defines the number of records to retrieve.	When pagination is enabled, yes
previousPageLastRecordPredicate (nested under the `pagination` element)	When pagination is enabled, XPath predicate that defines the record after which the page must fetched, this value is provided by the previous response, as the element `lastRecordPredicate`. If the passed record is not found, the first page will be returned.	No
disableRedirectionToLastBroadcast	See the description under Common parameters [p 644].	

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 `previousPageLastRecordPredicated`.

See also the optimistic locking [p 660] 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 643].	
version	See the description under Common parameters [p 643].	
instance	See the description under Common parameters [p 643].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 644].	

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 643].	
instance	See the description under Common parameters [p 643].	
predicate	See the description under Common parameters [p 643].	
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	*Deprecated since version 5.7.0* 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 660] section.	No
blockingConstraintsDisabled	See the description under Common parameters [p 643].	
details	See the description under Common parameters [p 643].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 644].	

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 643].
version	See the description under Common parameters [p 643].
instance	See the description under Common parameters [p 643].
predicate	See the description under Common parameters [p 643].
disableRedirectionToLastBroadcast	See the description under Common parameters [p 644].

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 643].	
instance	See the description under Common parameters [p 643].	
updateOrInsert	If `true` and the record does not currently exist, the operation creates the record. `boolean` type, the default value is `false`.	No
byDelta	If `true` and an element does not currently exist in the incoming message, the target value is not changed. If `false` and node is declared `hiddenInDataServices`, the target value is not changed. The complete behavior is described in the sections Opérations d'insertion et de mise à jour [p 236].	No
blockingConstraintsDisabled	See the description under Common parameters [p 643].	
details	See the description under Common parameters [p 643].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 644].	
data	See the description under Common parameters [p 643].	

Voir aussi *Optimistic locking [p 660]*

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 643].	
instance	See the description under Common parameters [p 643].	
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 236].	No
blockingConstraintsDisabled	See the description under Common parameters [p 643].	
details	See the description under Common parameters [p 643].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 644].	
data	See the description under Common parameters [p 643].	

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 641].

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 643].	
version	See the description under Common parameters [p 643].	
instance	See the description under Common parameters [p 643].	
compareWithBranch	The identifier of the dataspace with which to compare.	One of either this parameter or the 'compareWithVersion [p 653]' parameter must be defined.
compareWithVersion	The identifier of the snapshot with which to compare.	One of either this parameter or the 'compareWithBranch [p 653]' parameter must be defined.
compareWithInstance	The identifier of the dataset with which to compare. If it is undefined, instance [p 653] parameter is used.	No
resolvedMode	Defines whether or not the difference is calculated in resolved mode. Default is `true`. See **Resolved mode** `DifferenceHelper.resolvedMode`[API] 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 `getChanges` and `getChangesOnDataSet`. Allows client to define pagination context size. Each page contains a collection of inserted, updated and/or deleted records of tables according to the maximum size. Get changes persisted context is built at first call according to the page size parameter in request. The pagination context is persisted on the server file system [p 406] and allows invoking the next page until last page or when a timeout is reached. For creation: Defines `pageSize` parameter. For next: Defines `context` element with `identifier` from previous response. Enables pagination, see child elements below.	No
pageSize (nested under `pagination` element)	Defines maximum number of records in each page. Minimal size is `50`.	No (Only for creation)
context (nested under `pagination` element)	Defines content of pagination context.	No (Only for next)

Element	Description	Required
identifier (nested under `context` element)	Pagination context identifier.	Yes
disableRedirectionToLastBroadcast	See the description under Common parameters [p 644].	

> **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 `compareWithBranch` or `compareWithVersion`. 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 `identifier`. Is the last page: Defines `context` element without `identifier`. Enables pagination, see child elements below.	No
context (nested under `pagination` element)	Defines content of pagination context.	Yes (Only for next and last)
identifier (nested under `context` element)	Pagination context identifier. Not defined at last returned page.	No
pageNumber (nested under `context` element)	Current page number in pagination context.	Yes
totalPages (nested under `context` element)	Total pages in pagination context.	Yes

Get changes operation with pagination enabled

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

For creation:

Extract of request:

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

Extract of response:

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

For next:

Extract of request:

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

Extract of response:

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

For last:

Extract of request:

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

Extract of response:

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

Get credentials operation

Get credentials request

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

with:

Element	Description	Required
branch	See the description under Common parameters [p 643].	
version	See the description under Common parameters [p 643].	
instance	See the description under Common parameters [p 643].	
predicate	See the description under Common parameters [p 643].	
viewPublication	See the description under Common parameters [p 643].	

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 and isolation levels [p 492].

```
<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 643].	
version	See the description under Common parameters [p 643].	
instance	See the description under Common parameters [p 643].	
blockingConstraintsDisabled	See the description under Common parameters [p 643].	
details	See the description under Common parameters [p 643].	
disableRedirectionToLastBroadcast	See the description under Common parameters [p 644].	
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`).
- The `select` operation cannot use the pagination parameter.

Voir aussi

Procedure[API]

Repository[API]

Multiple operations response

See each response operation for details.

```
<ns1:multi_Response xmlns:ns1="urn:ebx-schemas:dataservices_1.0">
 <response id="id1">
  <{operation}_{TableName}Response>
  ...
  </{operation}_{TableName}Response>
 </response>
 <response id="id2">
  <{operation}_{TableName}Response>
  ...
  </{operation}_{TableName}Response>
 </response>
</ns1:multi_Response>
```

with:

Element	Description
response	This element contains the response of one operation. It is be repeated multiple times for additional operations. Each response is identified by an 'id' attribute set in the request or automatically generated. The content of the element corresponds to the response of a single operation, such as `count`, `select`, `getChanges`, `getCredentials`, `insert`, `delete` or `update`.

Optimistic locking

To prevent an update or a delete operation 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.

97.2 **Operations on datasets and dataspaces**

Parameters for operations on dataspaces and snapshots are as follows:

Element	*Description*	*Required*
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 `severityThreshold` defines the lowest severity level of message to return. The possible values are, in descending order of severity, 'fatal', 'error', 'warning', or 'info'. For example, setting the value to 'error' will return error and fatal validation messages. If this attribute is not defined, all levels of messages are returned by default. The optional attribute `locale` (default 'en-US') defines the language in which the validation messages are to be returned.	No. If not specified, no details are returned.
owner	Defines the owner. Must respect the inner format as returned by `Profile.format`[API].	No
branchToCopyPermissionFrom	Defines the identifier of the dataspace from which to copy the permissions.	No
documentation	Documentation for a dedicated language.	No

Element	*Description*	*Required*
	Multiple `documentation` elements may be used for several languages.	
locale (nested under the `documentation` element)	Locale of the documentation.	Only required when the `documentation` element is used
label (nested under the `documentation` element)	Label for the language.	No
description (nested under the `documentation` element)	Description for the language.	No

Validate a 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>
```

```
     <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 `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 380]. See Deleting data and history [p 404] 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 `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 380]. See Deleting data and history [p 404] 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 `ebx.dataservices.dataDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 380]. See Deleting data and history [p 404] 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 `ebx.dataservices.historyDeletionOnCloseOrMerge.default` in the EBX® main configuration file [p 380]. See Deleting data and history [p 404] 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 662].	Yes
instance	See the description under Common parameters [p 662].	Yes
unitName	Name of the replication unit. **Voir aussi** *Replication refresh information [p 282]*	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.

97.3 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 623] for more information.

Element	Description	Required
parameters	*Deprecated since version 5.7.0* 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 662].	No
parameters	*Deprecated since version 5.7.0* See the description under Common parameters [p 669].	No

Sample response:

```
<m:workflowProcessInstanceStart_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
```

```
 <processInstanceKey>String</processInstanceKey>
</m:workflowProcessInstanceStart_Response>
```

with:

Element	Description	Required
processInstanceId	*Deprecated since version 5.6.1* This parameter has been replaced by the '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	*Deprecated since version 5.7.0* See the description under Common parameters [p 669].	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	*Deprecated since version 5.6.1* Due to a limitation this parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No
processInstanceId	*Deprecated since version 5.6.1* Due to a limitation this parameter has been replaced by the 'processInstanceKey' parameter. While it remains available for backward compatibility, it will eventually be removed in a future major version.	No

Sample response:

```
<m:workflowProcessInstanceEnd_Response xmlns:m="urn:ebx-schemas:dataservices_1.0">
 <status>String</status>
</m:workflowProcessInstanceEnd_Response>
```

with:

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

97.4 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 641] 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 651] for more information.

User interface operations

See Application locking [p 414] 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 `locale` (default: EBX® default `locale`) defines the language in which the system item messages must be returned.	No, but if specified, the `locale` 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 98

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

98.1 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 683].

Built-in REST data services allow to perform operations such as:

- Selecting, inserting, updating, deleting, or counting records
- Selecting or counting history records
- Selecting, updating, or counting dataset values
- Administrative operations to manage access to the UI or to system information

> **Note**
>
> See SOAP and REST comparative [p 681].

98.2 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 343] for more information.

In case of specific deployment, for example using reverse-proxy mode, see URLs computing [p 382] for more information.

Currently only protocol HTTP(S) is supported.

98.3 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 JSON request contains:

```
{
 "procedureContext": // JSON Object (optional)
 {
    "trackingInformation": "String" // JSON String (optional)
 }, ...
}
```

For more information, see `Session.getTrackingInfo`[API] 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 JSON 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.getInputParameterValue`[API] 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": [
   {
    "severity": "...",     // JSON String, severity (optional)
    "rowIndex": 999,       // JSON Number, request row index (optional)
    "userCode": "...",     // JSON String, user code (optional)
    "message": "...",      // JSON String, message
    "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 of each error is specified by a character, with one of the possible values (`F`: fatal, `E`: error, `W`: warning, `I`: information).

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 689]*

Voir aussi *`Severity.toParsableString`*[API]

98.4 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 692].

  Voir aussi *HTTP Authorization header policy* *[p 382]*

- '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 QWxpYmFiYTpvcGVuIHNlc2FtZQ==
  ```

  > **Note**
  >
  > The `WWW-Authenticate` `[p 688]` header can be valued with this method.

  Voir aussi *HTTP Authorization header policy* *[p 382]*

- '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 636] section.

  For more information on this authentication scheme, see `Directory.authenticateUserFromLoginPassword`[API].

- The 'REST Forward Authentication Scheme' is used only when calling a REST service from a user service [p 586], that reuses the current authenticated session.

  For more information, see Implementing a user service [p 597] making a call to REST data services [p 604].

- '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.authenticateUserFromHttpRequest`[API] for more information.

- 'Anonymous authentication Scheme' is used only to access the REST services handling the authentication operations. The credentials acquisition, password changes, etc. imply that the user cannot be known yet.

Global permissions

Global access permissions can be independently defined for the REST built-in services access. For more information see Global permissions [p 409].

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 678] • 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 678] • 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 678] • The HTTP request must not hold an `Authorization` header. • A `login` and a `password` are provided in the URL parameters. Rest forward [p 678] • The HTTP request must not contain an `Authorization` header. • No `login` is provided in the URL parameters. Specific [p 678] • The HTTP request must not satisfy the conditions of the previous authentication methods. Anonymous [p 679] • 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`.

98.5 **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 377]*

TIBCO EBX® main configuration file *[p 371]*

98.6 **SOAP and REST comparative**

Operations	**SOAP**	**REST**
Data		
Select or count records (with filter and/or view publication)	X	X
Selector for possible enumeration values (with filter)		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 dataspaces or snapshots	X	
Refresh a replication unit	X	
Get credentials for records	X	
Dataspaces		
Create, close, merge a dataspace	X	
Create, close a snapshot	X	
Validate a dataspace or a snapshot	X	
Validate a dataset	X	
Locking a dataspace	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

Operations	SOAP	REST
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 747] for more information.

98.7 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 **99**

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. Limitations

99.1 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 to experience an intuitive and straightforward navigation, which implies that the data details could be obtained through a link.

> **Note**
>
> All operations are stateless.

99.2 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 677]*

Security *[p 678]*

HTTP method

Considered HTTP methods for built-in RESTful services, are:

- `GET`: used to select master data defined in the URL (the URL size limit depends on the application server or on the browser, that must be lower than or equal to 2KB).
- `POST`: used to insert one or more 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 with the table URL and the record table 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 685].
- `{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 687] or dedicated operation parameters passed by the URL.

Operation category

It specializes the operation, it is added in the path of the URL in `{category}` and it takes one of the following values:

`admin`	Administration operations reserved to administrators. For more information, see: Administration operations [p 690].
`auth`	Manage token authentication method. For more information, see: Token authentication operations [p 692] and Token Authentication Scheme [p 678].
`data`	Lists dataset content, requests a table, a record or a field record content, including modified operations on dataset node, table, record and record field. For more information, see: Data operations [p 695].
`history`	Lists history dataset content, requests a history table, a history of a record or a history record. For more information, see: Data operations [p 695]. **Voir aussi** *Historique [p 273]*

Header fields

These header field definitions are used by TIBCO EBX®.

`Accept`	Used to specify content types by order of preference 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 the `ebx.dataservices.rest.request.checkAccept` property: • 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 380]*
`Accept-Language`	Used for specifying the preferred locale for the response. The supported locales are defined in the schema model. If none of the preferred locale are supported, the default locale for the current model is used.
`Authorization`	Used for 'Basic Authentication Scheme' and 'Token Authentication Scheme' methods, otherwise the request is rejected. **Voir aussi** *Authentication* *[p 678]*
`Content-Type`	Used to specify the request body media type. The supported types are `application/json` and `application/x-www-form-urlencoded`. The request value is checked and if it is not supported, then an HTTP error message is returned with the code `415` *(Unsupported media type)*. **Voir aussi** *Configuring data services* *[p 380]*
`X-Requested-With`	If present and in case of authentication failure, prevents the addition of the `WWW-Authenticate` header in the response. **Voir aussi** *Response header `WWW-Authenticate`* *[p 688]*

See RFC2616 for more information about HTTP Header Field Definitions.

Common parameters

These optional parameters are available for all data service operations.

Parameter	Description
`disableRedirectionToLastBroadcast`	This parameter only has impact on a D3 architecture. 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 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 `false` (redirection to a D3 master enabled), unless the configuration property ebx.dataservices.disableRedirectionToLastBroadcast.default [p 380] has been set. **Voir aussi** *Primary node* *[p 453]*
`indent`	Specifies if the response should be indented, to be easier to read for a human. `Boolean` type, default value is `false`.

Message body

It contains the request data using the JSON format, see JSON Request body [p 722].

> **Note**
>
> Requests may define a message body only when using `POST` or `PUT` HTTP methods.

99.3 **Response**

This chapter describes the responses returned by built-in RESTful services.

- See Exception handling [p 677] 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 *(Unauthorized)* 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 administration property ebx.dataservices.rest.auth.tryBasicAuthentication [p 380] must be set to `true`. **Voir aussi** *Request header `X-Requested-With`* *[p 686]* *Authentication* *[p 678]*

HTTP codes

HTTP code	Description
`200` *(OK)*	The request has been successfully handled.
`201` *(Created)*	A new record has been created, in this case, the header field `Location` is returned with its resource URL.
`204` *(No content)*	The request has been successfully handled but no response body is returned.
`400` *(Bad request)*	the request URL or body is not well-formed or contains invalid content.
`401` *(Unauthorized)*	Authentication has failed.
`403` *(Forbidden)*	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 679].
`404` *(Not found)*	The resource specified in the URL cannot be found.
`406` *(Not acceptable)*	Content type defined in the request's `Accept` parameter is not supported. This error can be returned only if the EBX® property `ebx.rest.request.checkAccept` is set to `true`.
`409` *(Conflict)*	A concurrent modification has occurred. **Voir aussi** *Optimistic locking [p 715]*
`415` *(Unsupported media type)*	The request content is not supported, the request header value `Content-Type` is not supported by the operation.
`422` *(Unprocessable entity)*	The new resource's content cannot be accepted for semantic reasons.
`500` *(Internal error)*	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 (JSON [p 725] 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 677] for details on the format.

99.4 Administration operations

Administration operations are related to:

- the administration category.
- the administration dataspaces accessible through the `data` category.

> **Note**
>
> administration category and administration dataspaces 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 425].

> **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 695] and update [p 708] 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 695] and update [p 708] 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 695], update [p 708], insert [p 703] and delete [p 710] 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 695], update [p 708], insert [p 703] and delete [p 710] 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 414]*

Retrieve user interface state

User interface status and the unavailability message are accessible like dataset nodes.

See Select operation [p 695] and the JSON [p 732] 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 708] and the JSON [p 725] 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 398]

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 726] example of the flat representation.

See the JSON [p 726] example of the hierarchical representation.

99.5 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 380] (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 677].

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.authenticateUserFromLoginPassword`[API] 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.authenticateUserFromHttpRequest`[API] for more information.

See the JSON [p 722] 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 694]. • `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 727] 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 678]*

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 692] 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 722] example of a password change and token creation request.

HTTP codes

HTTP code	Description
`204` *(No content)*	The password has been changed.
`400` *(Bad request)*	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 `login` and the user's session one mismatch • the operation is not supported.
`401` *(Unauthorized)*	For the following reason: • The `login` and/or `password` is/are incorrect.
`422` *(Unprocessable entity)*	For one of the following reasons: • `PasswordChangeAbort`: `passwordNew` is empty. • `PasswordChangeAbort`: `passwordNew` is equal to `password`. • `PasswordChangeAbort`: `password` is incorrect. • `RestrictedAccess`: 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` *(No content)*	The token has been revoked successfully.
`400` *(Bad request)*	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` *(Unauthorized)*	Authentication has failed.

99.6 Data operations

Operations from the `data` operation category concern the datasets, the dataset fields, tables, records or record fields.

Operations from the `history` category and concern historized content from datasets, tables, records or the record fields.

Select operation

Select operation may use one of the following methods:

- `GET` HTTP method,
- `POST` HTTP method without message body or
- `POST` HTTP method with message body and optionally session parameters [p 677].

URL formats are:

- **Dataset tree**, depending on operation category [p 685]:

  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}
```

> **Note**
>
> Terminal nodes and sub-nodes are not included.

- **Dataset node**: the `data` category returns the terminal nodes contained in the selected node.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
  {pathInDataset}
  ```

  > **Note**
  >
  > Not applicable with the `history` category.

- **Table**, depending on operation category [p 685]:

  the `data` 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}
  ```

  Voir aussi *Count operation [p 712]*

- **Record**, depending on operation category [p 685]:

  the `data` 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}
  ```

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/{category}/v1/{dataspace}/
  {dataset}/{pathInDataset}?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 685]:

  the `data` 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}
```

> **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 685].
- `{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 *`RESTEncodingHelper`*[API]

- `{xpathExpression}` corresponds to the record primary key, using the XPath expression.
- `{pathInRecord}` corresponds to the path starting from the table node.

Parameters

The following parameters are applicable to the select operation.

Parameter	Description
`includeContent`	Includes the `content` field with the content corresponding to the selection. `Boolean` type, default value is `true`.
`includeDetails`	Includes the `details` field in the metadata and the response, for each indirect reachable resource. The returned value corresponds to its URL resource. Type `Boolean`, default value is `true`. **Voir aussi** *includeMeta* [p 698]
`includeHistory`	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 698] parameter. The returned value corresponds to its URL resource. `Boolean` type, default value is `false`. **Note** The `includeHistory` parameter is ignored in the `history` category, the default value is `true`. **Voir aussi** *includeMeta* [p 698] *includeDetails* [p 698] *`AdaptationTable.getHistory`*[API]
`includeLabel`	Includes the `label` field associated with each simple type `content`. Possible values are: • `yes`: the `label` is included for the foreign key, enumeration, record and selector values. • `all`: the `label` field is included, as for the `yes` value and also for the Content of simple type [p 739]. **Voir aussi** *`SchemaNode.displayOccurrence`*[API] • `no`: the `label` field is not included (integration use case). `String` type, default value is `yes`. **Note** The `label` field is not included if it is equal to the `content` field.
`includeMeta`	Includes the `meta` field corresponding to the description of the structure returned in the `content` field. `Boolean` type, default value is `false`.

Parameter	Description
	Voir aussi *includeHistory* [p 698] *includeDetails* [p 698]
`includeMergeInfo`	Includes the `merge_info` corresponding to a field of the technical data of an history transaction and has a potentially high access cost. `Boolean` type, default value is `true`. **Note** This parameter is ignored with the `data` category. **Voir aussi** *Accès REST aux tables historisées* [p 329]
`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 702]
`includeSortCriteria`	Includes the `sortCriteria` field corresponding to the list of sort criteria applied. The sort criteria parameters are added by using: • sort [p 701] • sortOnLabel [p 701] • viewPublication [p 702] `Boolean` type, default value is `false`. Example JSON [p 735]
`includeTechnicals`	Includes the internal technical data. `Boolean` type, default value is `false`. **Voir aussi** *Technical data* [p 744] **Note** This parameter is ignored with the `history` category.
`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 698]

Table parameters

The following parameters are applicable to tables, associations and selection nodes.

Parameter	Description
`filter`	XPath predicate [p 249] 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.
`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 dataspaces The default value is `CurrentDataSpaceOnly`. **Voir aussi** *`history` [p 685]* **Note** This parameter is ignored with the `data` category.
`includeOcculting`	Includes the records in occulting mode. `Boolean` type, default value is `false`.
`primaryKey`	Search a record by a primary key, using the XPath expression. The XPath predicate [p 249] 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 700], viewPublication [p 702], sort [p 701], etc.) `String` type value.
`pageFirstRecordFilter`	*Deprecated since version 5.9.0*, replaced by `pageRecordFilter`
`pageRecordFilter`	Specifies the record XPath predicate [p 249] expression filter of the page. `String` type value.

Parameter	Description
`pageAction`	Specifies the pagination action to perform from page defined by `pageRecordFilter`. `String` type, possible values are: • `first` • `previous` • `next` • `last` **Voir aussi** *`RequestPagination`*[API]
`pageSize`	Specifies the number of records per page. `Integer` type, default value is based on the user preferences. `String` type, the `unbounded` value can be defined to return all records. Only for this case, no pagination context is returned.

`sort`	Specifies that the operation result will be sorted according to the specified criteria. The criteria are composed of one or more criteria, 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 701] parameter as a new criteria added after the `sort`.
	2. the viewPublication [p 702] 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
	The history code operation value is usable with the `ebx-operationCode` path field from the `meta` section associated with this field.
	Voir aussi *`Request.setSortCriteria`*[API]

`sortOnLabel`	Specifies that the operation result will be sorted according to the record label. This parameter can be combined with: 1. the sort [p 701] parameter as a new criteria added before the `sortOnLabel`. 2. the viewPublication [p 702] parameter as a new criteria added after the `sortOnLabel`. The value structure is as follows: `<order>`

Parameter	Description

	Where:
	• `<order>` corresponds to the sorting order, with one of the following values:
	  • `lasc`: ascending order on label,
	  • `ldesc`: descending order on label.
	The behavior of this parameter is described in the section defaultLabel [p 521].
	`String` type value.
	Voir aussi *Limitation [p 718]*

`viewPublication`	Specifies the name of the published view. This parameter can be combined with: 1. the filter [p 700] parameter as the logical `and` operation. 2. the sort [p 701] parameter as a new criteria added before the `viewPublication`. 3. the sortOnLabel [p 701] parameter as new criteria added before the `viewPublication`. The behavior of this parameter is described in the section EBX® as a Web Component [p 215]. `String` type value.

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or `osd:resource` (Example JSON [p 742]). By default, a pagination mechanism is enabled.

Parameter	Description
`selector`	Specifies whether: • `true`: returns all possible values (includes their labels) • `false`: returns the current values for the current field. `Boolean` type, default value is `false`. **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. `String` type, the `unbounded` value can be defined to return all values. Only for this case, no pagination context is returned.
`selectorFilter`	Specifies the filter of the selector. `String` type value, the syntax complies with the Recherche textuelle [p 126].

HTTP codes

HTTP code	Description
`200` *(OK)*	The selected resource is successfully retrieved.
`400` *(Bad request)*	The request is incorrect. This occurs when: • The selected field in a record or a dataset is sub-terminal, • The XPath predicate of the `filter` parameter is malformed or contains unfilterable nodes. • The XPath predicate of the `primaryKey` parameter is malformed or is not a record primary key. • The sort criteria of the `sort` parameter have an invalid syntax or contain unsortable nodes. • `pageAction` parameter value is not included in allowed values, or `pageRecordFilter` is malformed or non-existent when selecting next or previous page. • `pageSize` parameter value is below 2, or is a string different from `unbounded`. • The table view for the `viewPublication` parameter is either hierarchical, non-existent or non-published. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` 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.

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.

Example: JSON [p 727].

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/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.

Parameters

The following parameters are applicable with the insert operation.

Parameter	Description
`includeDetails`	Includes the `details` field in the answer for access to the details of the data. The returned value corresponds to its URL resources. Type `Boolean`, the default value is `false`. **Note** Only applicable on the **record table**.
`includeForeignKey`	Includes the `foreignKey` field in the answer for each record. The returned value corresponds to the value of a foreign key field that was referencing this record. `Boolean` type, the default value is `false`. **Note** Only applicable on the **record table**.
`includeLabel`	Includes the `label` field in the answer for each record. Possible values are: • `yes`: the `label` field is included. • `no`: the `label` field is not included (use case: integration). `String` type, the default value is `no`. **Note** Only applicable on the **record table**.
`updateOrInsert`	Specifies the behavior when the record to insert already exists: • If `true`: the existing record is updated with new data. For a request on a **record table**, the `code` field is added to the report in order to specify if this is an insert `201` or an update `204`. • If `false` (default value): a client error is returned and the operation is aborted. `Boolean` type value.
`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 744] section. `Boolean` type, the default value is `true`. **Note** Applicable on record in update mode and if the updateOrInsert [p 705] parameter is `true`.
`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 `false`.

Parameter	Description
	See Blocking and non-blocking constraints [p 548] 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 JSON [p 723]).
- **Record table**: Similar to the select operation on a table but without the pagination information (example JSON [p 724]).

Voir aussi *Inheritance [p 716]*

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 `Location` is returned with its resource URL.
`204` *(No content)*	If the request relates to a **record**. Only available if `updateOrInsert` is `true`, an existing record has been successfully updated, in this case, the header field `Location` 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 706].
`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 `updateOrInsert` is `true`, the Optimistic locking [p 715] 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 `blockingConstraintsDisabled` is `false`). • The record cannot be inserted because a record with the same primary key already exists (only available if `updateOrInsert` 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 742]). 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 716]*

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/data/v1/{dataspace}/{dataset}/
  {pathInDataset}
  ```

- **Record**: modifies the content of selected record.

  > **Note**
  >
  > Also available for `POST` HTTP methods. In this case, the URL must correspond to the table by setting the parameter `updateOrInsert` to `true`.

- **Field**: modifies the field content.

  ```
  http[s]://<host>[:<port>]/<ebx-dataservices>/rest/data/v1/{dataspace}/{dataset}/
  {pathInDataset}/{encodedPrimaryKey}/{pathInRecord}
  ```

  > **Note**
  >
  > The field must be either a terminal node or above.

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 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 *`RESTEncodingHelper`*[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 `false`. See Blocking and non-blocking constraints [p 548] 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 744] 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 715] section. `DateTime` type value.

Message body

The request must define a message body.

The structure is the same as for:

- the dataset node (sample JSON [p 722]),
- the record (sample JSON [p 723]),
- the record fields (sample JSON [p 723]),

depending on the updated scope, by only keeping the `content` entry.

Voir aussi *Inheritance [p 716]*

HTTP codes

HTTP code	Description
`204` *(No content)*	The record, field or dataset node has been successfully updated.
`400` *(Bad request)*	The request is incorrect. This occurs when the body request structure does not comply.
`403` *(Forbidden)*	Authenticated user is not allowed to update the specified resource or the request body contains a read-only field.
`404` *(Not found)*	The selected resource is not found.
`409` *(Conflict)*	Concurrent modification, the Optimistic locking [p 715] is activated and the content has changed in the meantime, it must be reloaded before the update.
`422` *(Unprocessable entity)*	The request cannot be processed. This occurs when: • A blocking validation error occurs (only available if `blockingConstraintsDisabled` is `false`). • 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 *`RESTEncodingHelper`*[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 716]*

Parameters

Here are the following parameters applicable with delete operation.

Parameter	Description
`includeOcculting`	Includes occulted records. `Boolean` type, the default value is `false`.
`inheritIfInOccultingMode`	*Deprecated since version 5.8.1* 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 `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 715] section. `DateTime` type value.
`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 `false`. See Blocking and non-blocking constraints [p 548] 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]

  Example JSON [p 724].

HTTP codes

HTTP code	Description
`200` *(OK)*	The operation has been executed successfully. A report is returned in the response body.
`400`*(Bad request)*	The request is incorrect. This occurs when: • the structure of the message body does not comply with Message body [p 711]. • the message body contains a record table while the URL specifies a record.
`403` *(Forbidden)*	Authenticated user is not allowed to delete or occult the specified record.
`404` *(Not found)*	The selected record is not found. In the child dataset context, it should be necessary to use the `includeOcculting` parameter.
`409` *(Conflict)*	Concurrent modification, The Optimistic locking [p 715] is activated and the content has changed in the meantime, it must be reloaded before deleting the record. The parameter value `checkNotChangedSinceLastUpdateDate` exists but does not correspond to the actual last update date of the record.
`422` *(Unprocessable entity)*	Only available if `blockingConstraintsDisabled` is `false`, 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 743].

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/data/v1/{dataspace}/{dataset}/
  {pathInDataset}?count=true
  ```

  > **Note**
  >
  > Not applicable with the `history` category.

- **Table** depending on the operation category [p 685]:

  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=true
```

- **Field** depending on the operation category [p 685]:

  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=true
  ```

  > **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 685].
- `{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 *`RESTEncodingHelper`*[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 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 249] 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 dataspaces The default value is `CurrentDataSpaceOnly`. **Voir aussi** *`history` [p 685]* **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 714] parameter as the logical `and` operation. The behavior of this parameter is described in the section EBX® as a Web Component [p 215].

Selector parameters

The following parameters are only applicable to fields that return an enumeration, foreign key or `osd:resource`.

Parameter	Description
`selector`	Specifies whether: • `true`: returns the number of all possible values • `false`: returns the number of possible values for the current field. `Boolean` type, default value is `false`. **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 textuelle [p 126].

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 `filter` parameter is malformed or contains unfilterable nodes. • The table view for the `viewPublication` parameter is either hierarchical, non-existent or non-published. • The `selector` parameter is used for a non-enumerated node, or the `firstElementIndex` 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 744] 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 723] 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.

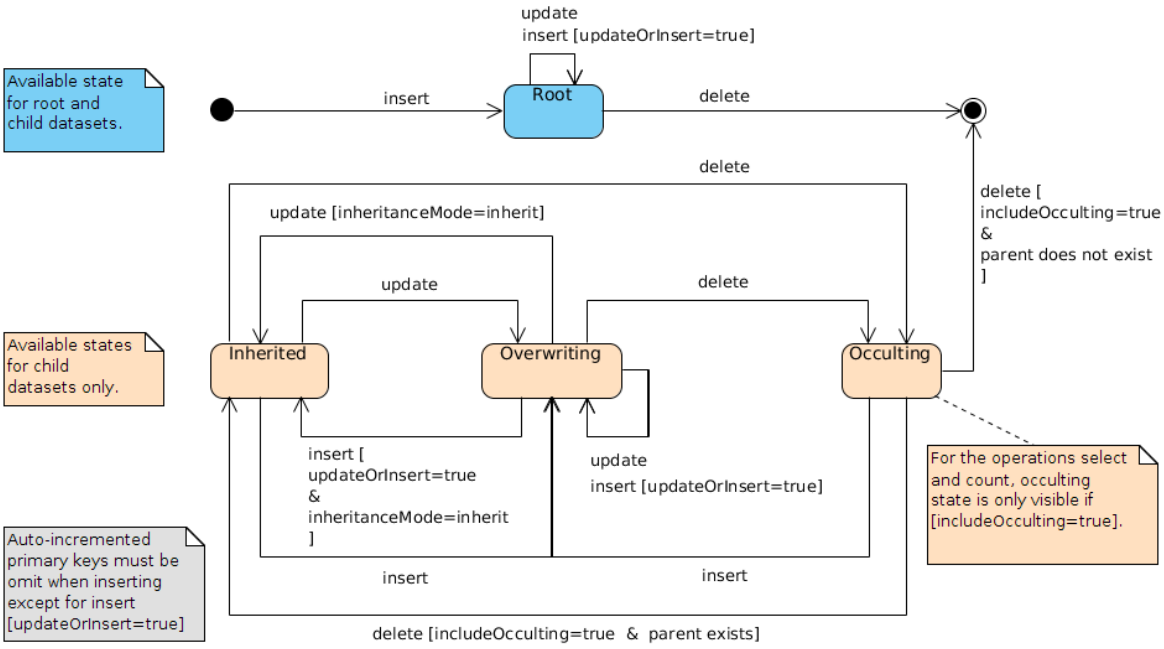
> **Note**
>
> The `checkNotChangedSinceLastUpdateDate` parameter may be used more than once but only on the same record. This implies that if the request URL returns more than one record, the request will fail.

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 *Héritage et résolution de valeur [p 296]*

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 733]
`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 734]
`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 Mécanisme de résolution d'enregistrement [p 298].
	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 Héritage et résolution de valeur [p 296].

Property	Location	Description
`inheritedFieldMode`	inherited field	Specifies the inherited field's inheritance state. To set a field's inheritance to `inherit`, its `inheritedFieldMode` value must be 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 Mécanisme de résolution de valeur [p 299].

99.7 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 563] for more information about terminal nodes.

Read operations

- Within the `selector`, the pagination context is limited to the `nextPage` property.
- 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 691] for more information.

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 548] for more information.

Directory operations

- Changing or resetting a password for a user is not supported.

CHAPITRE **100**

JSON format

Ce chapitre contient les sections suivantes :

1. Introduction
2. Global structure
3. Meta-data
4. Sort criteria information
5. Validation
6. Content
7. Update modes
8. Known limitations

100.1 Introduction

The JSON (JavaScript Object Notation) is the data-interchange format used by TIBCO EBX® RESTful operations [p 683].

This format is lightweight, self-describing and can be used to design UIs or to integrate EBX® in a company's information system.

- The data context is exhaustive, except for association fields and selection nodes that are not directly returned. However, these fields are included in the response with a URL link named `details` included by default. It can be indirectly used to get the fields content.
- The volume of data is limited by a pagination mechanism activated by default, it can be configured or disabled.

URL formatted links allow retrieving:

- Tables, records, dataset non-terminal nodes, foreign keys, resource fields (property `details`).
- Possible values for foreign keys or enumerations (`selector` parameter).

Voir aussi *Activation and configuration [p 676]*

> **Note**
>
> JSON data are always encoded in UTF-8.

100.2 Global structure

JSON Request body

Request body is represented by a JSON `Object` whose content varies according to the operation and its 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 692]*

- **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 694]*

Data category

The request body contains at least a `content` property where master data values will be defined.

- **Dataset node**

  Specifies the target values of terminal nodes under the specified node. This request is used on the dataset node update operation.

```
{
   "content": {
      "nodeName1": {
         "content": true
      },
      "nodeName2": {
         "content": 2
      },
      "nodeName3": {
         "content": "Hello"
      }
   }
}
```

Voir aussi *Update operation* *[p 708]*

- **Record**

  Specifies the target record content by setting the value for each field. For missing fields, the behavior depends on the request parameter `byDelta`. This structure is used on table record insert or on record update.

  Voir aussi *Inheritance* *[p 716]*

  Some technical data can be added beside the `content` property such as `lastUpdateDate`.

  Voir aussi *Optimistic locking* *[p 715]*

```
{
   ...
   "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 703]*

  Update operation *[p 708]*

- **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 716]*

```
{
   "content": [
      {
         "content": "CM"
      },
      {
         "content": "PI"
      }
   ]
}
```

Voir aussi *Update operation [p 708]*

- **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 of 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 703]

  Update operation [p 708]

- **Record table to be 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 710]*

- **Field**

  Specifies the target field content. This request is used on the field update.

  The request has the same structure as defined in node value [p 737] by only keeping the `content` entry. Other entries are simply ignored.

  Voir aussi *Update operation [p 708]*

- **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 691]*

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 its category.

Admin category

The selection operation for this category only provides the values corresponding to the request 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 691]*

Auth category

The response body contains several properties directly placed in its root JSON `object`.

- **Token creation**

  Contains the token value and its type.

```
{
  "accessToken": "...",            // JSON String
  "tokenType": "..."               // JSON String
}
```

Voir aussi *Create token operation [p 692]*

Data category

The selection operation contains two different parts.

The first one named `meta` contains the exhaustive structure of the response.

The other, 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",
         "content": {
            "settings": {
               "details": "http://.../rootName/settings",
               "content": {
                  "weekTimeSheet": {
                     "details": "http://.../rootName/settings/settingA"
                  },
                  "vacationRequest": {
                     "details": "http://.../rootName/settings/settingB"
                  }
               }
            },
            "table1": {
               "details": "http://.../rootName/table1"
            },
            "table2": {
               "details": "http://.../rootName/table2"
            }
         }
      }
   }
}
```


The `meta` and `validation` properties are optional.

Voir aussi

Meta-data [p 732]

Validation [p 736]

Select operation [p 695]

- **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 695]*

- **Table**

  JSON `Object` containing the properties:

  - (Optional) The table meta data [p 732],
  - (Optional) The sort criteria applied,
  - (Optional) The table validation report,
  - `rows` containing a table of selected records. Each record is represented by an object, if no record is selected then the table is empty.
  - (Optional) `pagination` containing pagination [p 741] information if activated.

```
{
   "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 695]*

- **Record**

  JSON `object` containing:

  - The label,
  - (Optional) The record URL,
  - (Optional) The technical data [p 744],
  - (Optional) The table metadata [p 732],
  - (Optional) The record validation report,
  - (Optional) The inheritance mode of the record is: `root`, `inherit`, `overwrite` or `occult`. This value is available for a child dataset,

    Voir aussi

    Mécanisme de résolution d'enregistrement *[p 298]*

    Inheritance *[p 716]*
  - 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": "Identifier",
            "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,
         "validation":
         [
            {
               "level": "error",
               "message": "Field 'Parent' is mandatory."
            }
         ]
      }
   },
   "validation": {
      ...
   }
}
```


Voir aussi *Select operation [p 695]*

- **Fields**

  For association or selection nodes, contains the target table with associated records if, and only if, the `includeDetails` parameter is set to `true`.

  For other kinds of nodes, contains the current node value [p 737], by adding `meta` entry if enabled.

  Voir aussi *Select operation [p 695]*

- **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 691]*

> **Note**
>
> Node, records and field in `meta`, `rows` and `content` may be hidden depending on their resolved permissions *(see permissions [p 301])*.

100.3 **Meta-data**

This section can be activated on demand with the `includeMeta` [p 698] 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 695]*

Structure of table

Table meta-data is represented by a JSON `object` with the following properties:

JSON property	JSON format	Description	Required
`name`	`String`	Name of the table defined in schema.	Yes
`label`	`String`	Table label. If undefined, the name of the schema node is returned.	Yes
`description`	`String`	Table description.	No
`type`	`String`	Always equal to: `table`.	Yes
`minOccurs`	`Number`	Number of minimum authorized record(s).	Yes
`maxOccurs`	`Number` or `String`	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. **Voir aussi** *Historique* [p 273]	No
`primaryKeyFields`	`Array`	Array of the paths corresponding to the primary key.	Yes
`inheritance`	`Boolean`	Specifies whether the dataset inheritance is activated for the table. Its value is `true` if inheritance is activated, `false` otherwise. **Voir aussi** *Héritage et résolution de valeur* [p 296]	No
`fields`	`Array`	Array of fields, that are direct children of the record node. Each field may also recursively contain sub-fields.	Yes

Structure of field

Each authorized field is represented by a JSON `object` with the following properties:

JSON property	JSON format	Description	Required
`name`	`String`	Name of the current authorized schema node.	Yes
`label`	`String`	Node label. If undefined, the name of the schema node is returned.	Yes
`description`	`String`	Node description.	No
`type`	`String`	Node type: simple type [p 739], `group`, `table`, `foreignKey`, etc.	Yes
`minOccurs`	`Number`	Number of minimum authorized occurrence(s).	Yes
`maxOccurs`	`Number` or `String`	Number of maximum authorized occurrence(s) or `unbounded`.	Yes
`inheritedField`	`Object`	Holds information related to the inherited field's value source. `"inheritedField": { "sourceRecord": "/path/to/record", // (optional) "sourceNode": "./path/to/Node" }` **Voir aussi** *Héritage et résolution de valeur [p 296]*	No
`foreignKey`	`Object`	Contains information related to the target table. `{ "dataspace": "BAuthors", "dataset": "Authors", "tablePath": "/root/Authors", "details": "http://.../BAuthors/Authors/root/Authors" }`	No (*)
`dataspace`	`String`	Target dataspace or snapshot identifier. This property is placed under the `foreignKey` property.	No (*)
`dataset`	`String`	Target dataset identifier. This property is placed under the `foreignKey` property.	No (*)
`tablePath`	`String`	Target table path. This property is placed under the `foreignKey` property.	Yes
`details`	`String`	Target table URL. This property is placed under the `foreignKey` property and is included by default.	No
`enumeration`	`String`	Specifies if the field is an enumeration value. Possible values are: • `foreignKey`	No

JSON property	JSON format	Description	Required
		• `static` • `dynamic` • `programmatic` • `nomenclature` • `resource` **Voir aussi** *`SchemaFacetEnumeration`*[API] Specifies whether the field can be used for retrieving possible values by using the `selector` request parameter.	
`valueFunction`	`Boolean`	Specifies if the field is a computed value. **Voir aussi** *Computed values* [p 553]	No
`pathInDataset`	`String`	Relative field path starting from the schema node.	No (**)
`pathInRecord`	`String`	Relative field path starting from the table node.	No (*)
`filterable`	`Boolean`	Specifies whether the field can be used for filtering record using `filter` request parameter.	No (*)
`sortable`	`Boolean`	Specifies whether the field can be used in sort criteria using `sort` request parameter.	No (*)
`fields`	`Array` of `Object` elements	Contains the structure and typing of each group field.	No

(*) Only available for table, record and record field operations.

(**) Only available for dataset tree operations.

100.4 Sort criteria information

The sort criteria applied to the request can be returned on demand, by using the `includeSortCriteria` [p 699] parameter (deactivated by default). If it is activated, a `sortCriteria` property is directly added to the response root node.

A `sortCriteria` property contains a JSON `Array` that contains ordered sort criteria, and for each sort criterion, a JSON `Object` is added with the following properties:

JSON property	JSON format	Description	Required
`path`	`String`	Field path.	Yes
`order`	`String`	Possible values are: `asc`, `lasc`, `desc` or `ldesc`.	Yes

100.5 **Validation**

The validation can be activated on demand with the `includeValidation` [p 699] parameter (deactivated by default). If it is activated, `validation` properties are directly added on target nodes with one or several messages. For messages without a target node path, a `validation` property is added on the root node.

A `validation` property contains a JSON `Array` and for each message, corresponding to a validation item, a JSON `Object` with properties:

JSON property	JSON format	Description	Required
`level`	`String`	Severity of the validation item, the possible values are: `info`, `warn`, `error`.	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 the table and dataset scopes, if associated resources exist and if it is included. **Voir aussi** *`includeDetails` parameter* [p 698]	No

100.6 **Content**

This section can be deactivated on demand with the `includeContent` [p 698] 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 739] Content of group and list [p 741]	Contains the node value. Available for all nodes except `association` and `selection`. However, their content can be retrieved by invoking the URL provided in `details`.	`No`
`details`	`String` corresponding to an absolute URL.	By invocation, the node details are returned. Response type after invocation depending on the meta type. • `foreignKey`: target record (available on table, record and field operation). • `resource`: target resource [p 544] (available on dataset node, table, record and field operations). • `association`: target table containing associated records (available on table and record operations). • `selection`: target table containing 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`
`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. `inheritedFieldMode` and `inheritanceMode` properties cannot be both defined on the same node. **Voir aussi** *`inheritedFieldMode` [p 737]* *Mécanisme de résolution d'enregistrement [p 298]*	`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. `inheritedFieldMode` and `inheritanceMode` properties cannot be both defined on the same node. **Voir aussi** *`inheritanceMode` [p 737]* *Mécanisme de résolution de valeur [p 299]*	`No`
`selector`	`String`	Correspond to the URL for selector [p 742] operation.	`No`

JSON property	JSON format	Description	Required
	corresponding to an absolute URL.	Example: `http://.../BReference/dataset/root/table/pk/enumField?selector=true`	
`validation`	`Array`	Contains the validation report that concerns the current node context.	`No`

Content of simple type

A simple field value is stored in a JSON object and the content is the value of the `content` property.

XML Schema	JSON format	Examples	Meta type

`xs:string`
`xs:Name`
`osd:html`
`osd:email`
`osd:text`
	`String` (Unicode characters, cf. RFC4627)	"A text"
		"The escape of \"special character\" is preceded by a backslash."
		" An HTML tag can thus be written without trouble "
		"employee@mycompany.com"
		null
			`string`
			`name`
			`html`
			`email`
			`text`

`osd:locale`	`String` (Language tag, cf. RFC1766)	"en-US"	`locale`
`xs:string` (Foreign key)	`String` contains the value of the formatted foreign key.	`"0"` `"true|99"`	`foreignKey`
`xs:boolean`	`Boolean`	true false null	`boolean`
`xs:decimal`	`Number` or `null`	-10.5 20.001 15 -1e-13	`decimal`
`xs:date`	`String` with format: "yyyy-MM-dd"	"2015-04-13"	`date`
`xs:time`	`String` with format: • "HH:mm:ss" • "HH:mm:ss.SSS"	"11:55:00" "11:55:00.000"	`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`
`xs:anyURI`	`String` (Uniform Resource Identifier, cf. RFC3986)	"https://fr.wikipedia.org/wiki/René_Descartes"	`anyURI`
`xs:int` `xs:integer`	`Number` or `null`	1596	`int`
`osd:resource`	`String`	"ebx-tutorial:ext-images:frontpages/Ajax for Dummies.jpg"	`resource`

XML Schema	JSON format	Examples	Meta type
	contains the resource formatted value.		
`osd:color`	`String` with format: "#[A-Fa-f0-9]{6}" contains the formatted value for the color.	"#F6E0E0"	`color`
`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`
`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`

Content of group and list

XML Schema	JSON format	Examples	Meta type
Group `xs:complexType`	`Object` Contains a property per sub-node.	Example for a simple-occurrence group. `{ "road" : {"content" : "11 rue scribe"}, "zipcode" : {"content" : "75009"}, "country" : {"content" : "France"} }`	`group`

List
`maxOccurs > 1`
	`Array`
	Contains an array of all field occurrences represented by a JSON `Object`.
	Each object is represented as a node value [p 737].
		Example for a multi-occurrence field of the `xs:int` type.
		`[ {"content": 0}, {"content": 1}, {"content": 2}, {"content": 3} ]`
		Example for a multi-occurrence group.
		`[ { "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"} } } ]`
			Meta of simple type [p 739],
			or
			`group`

Pagination

This feature allows returning a limited and parameterizable number of data. Pagination can be applied to data of the following types: records, association values, selection node values and selectors. A context named `pagination` is returned only if it has been activated. This context allows browsing data similarly to the UI.

Pagination is activated by default.

Voir aussi *Select operation [p 695]*

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.

Selector

By invoking the URL represented by the property `selector` on a field that provides an enumeration, this returns a JSON `Object` containing the properties:

- `rows` containing an `Array` of JSON `Object` where each one contains two entries, such as the returned `content` that can be persisted and the corresponding `label`. The list of possible items is established depending on the current context.
- (Optional) `pagination` containing pagination [p 741] information (activated by default).

```
{
   "rows": [
      {
         "content": "F",
         "label": "feminine"
      },
      {
         "content": "M",
         "label": "masculine"
      }
   ],
   "pagination": {
      "nextPage": null
   }
}
```

Voir aussi *includeSelector [p 699]*

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 `ObjectArray`, where each element corresponds to the result of a request element.
- `code` contains an `int` of the 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 the 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 the 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 the 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 703]*

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 the JSON `Number` type, corresponds to the number of deleted records.
- `occultedCount` containing an integer of the JSON `Number` type, corresponds to the number of occulted records.
- `inheritedCount` containing an integer of the JSON `Number` type, corresponds to the number of inherited records.

```
{
 "deletedCount": 1,
 "occultedCount": 0,
 "inheritedCount": 0
}
```

Voir aussi *Delete operation* *[p 710]*

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 identifier.	Yes
`lastUpdateDate`	`String`	Last update date.	Yes
`lastUpdateUser`	`String`	Last update user identifier.	Yes

```
{
   ...
   "creationDate": "2015-12-24T19:00:53.158",
   "creationUser": "admin",
   "lastUpdateDate": "2015-12-25T00:00:00.001",
   "lastUpdateUser": "admin",
   ...
}
```

100.7 **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 708] and insert [p 703], as well as `ImportSpec.setByDelta`[API] 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 `null`. • 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 `null` from an empty string, the target field value is set to `null`. 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 566] **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 `null` (for example, `"content": null`)	The target field is always set to `null` except for lists, in which case it is not supported.

100.8 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.

CHAPITRE 101

REST Toolkit

Ce chapitre contient les sections suivantes :

1. Introduction
2. Application definitions
3. Service and operation definitions
4. Authentication and lookup mechanism
5. REST authentication and permissions
6. URI builders
7. Exception handling
8. Monitoring
9. Packaging and registration

101.1 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 749].

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 built URIs without worrying about the configuration (for example reverse-proxy [p 676] or REST forward [p 604] 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 ans JSON-B annotations are also supported.

- logging and debugging utilities.

Voir aussi *JAX-RS: JavaTM API for RESTful Web Services 2.1*

101.2 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()));
   }
}
```

101.3 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()
   .parallelStream()
   .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;
 }
}
```

101.4 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 `AnonymousAccessEnabled`[API]. See REST authentication and permissions [p 752] for more information.

Voir aussi

Authentication [p 678]

Lookup mechanism [p 679]

101.5 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()
 {
  ...
 }
}
```

101.6 **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].

101.7 **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 `UnprocessableEntityException`[API] 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" }
  ]
}
```

101.8 **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 679]*

Configuring the EBX® logs *[p 377]*

Some additional properties are available to configure the log messages. See Configuring REST toolkit services [p 382] for further information.

101.9 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 343] for more information.

The registration of a REST Toolkit application is integrated into the EBX® module registration process. The registration class must extend `ModuleRegistrationListener`[API], declare the Servlet 3.0 annotation `@WebListener` and override the `handleContextInitialized` method.

See Module registration [p 486] 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);
 }
}
```