HP Universal CMDB

Version du logiciel : 10.10

Manuel de référence du développeur

Date de publication du document : Novembre 2013

Date de lancement du logiciel : Novembre 2013

Mentions légales

Garantie

Les seules garanties applicables aux produits et services HP sont celles figurant dans les déclarations de garantie expresse accompagnant les dits produits et services. Aucun terme de ce document ne peut être interprété comme constituant une garantie supplémentaire. HP ne peut en aucun cas être tenu pour responsable des erreurs ou omissions techniques ou rédactionnelles du présent document.

Les informations contenues dans le présent document sont susceptibles d'être modifiées sans préavis.

Légende de restriction des droits

Logiciel confidentiel. Licence HP valide requise pour la détention, l'utilisation ou la copie. En accord avec les articles FAR 12.211 et 12.212, les logiciels informatiques, la documentation des logiciels et les informations techniques commerciales sont concédés au gouvernement américain sous licence commerciale standard du fournisseur.

Copyright

© Copyright 2002 - 2013 Hewlett-Packard Development Company, L.P.

Marques

Adobe® est une marque déposée de Adobe Systems Incorporated.

Microsoft® et Windows® sont des marques déposées de Microsoft Corporation aux États-Unis.

UNIX® est une marque déposée de The Open Group.

Crédits

Ce produit inclut un logiciel développé par Apache Software Foundation (http://www.apache.org/).

Ce produit inclut un logiciel développé par OpenSSL Project destiné à être utilisé dans le kit de ressources OpenSSL (http://www.openssl.org/)

Ce produit inclut un logiciel de chiffrement développé par Eric Young (eay@cryptsoft.com)

Ce produit inclut un logiciel développé par Tim Hudson (tjh@cryptsoft.com)

Mises à jour de la documentation

La page de titre du présent document contient les informations d'identifications suivantes :

- le numéro de version du logiciel ;
- la date de publication du document, qui change à chaque mise à jour de ce dernier ;
- la date de lancement du logiciel.

Pour obtenir les dernières mises à jour ou vérifier que vous disposez de l'édition la plus récente d'un document, accédez à la page : **http://h20230.www2.hp.com/selfsolve/manuals**

Pour accéder à ce site, vous devez créer un compte HP Passport et vous connecter comme tel. Pour obtenir un identifiant HP Passport, accédez à l'adresse : **http://h20229.www2.hp.com/passport-registration.html**

Vous pouvez également cliquer sur le lien **New users - please register** dans la page de connexion de HP Passport.

En vous abonnant au service d'assistance du produit approprié, vous recevrez en outre les dernières mises à jour ou les nouvelles éditions. Pour plus d'informations, contactez votre revendeur HP.

Assistance

Visitez le site d'assistance HP Software à l'adresse : **http://www.hp.com/go/hpsoftwaresupport**

Ce site fournit les informations de contact et les détails sur les offres de produits, de services et d'assistance HP Software.

L'assistance en ligne de HP Software propose des fonctions de résolution autonome. Le site constitue un moyen efficace d'accéder aux outils interactifs d'assistance technique nécessaires à la gestion de votre activité. En tant que client privilégié de l'assistance, vous pouvez depuis ce site :

- rechercher des documents de connaissances présentant un réel intérêt ;
- soumettre et suivre des demandes d'assistance et des demandes d'améliorations ;
- télécharger des correctifs logiciels ;
- gérer des contrats d'assistance ;
- rechercher des contacts de l'assistance HP ;
- consulter les informations sur les services disponibles ;
- participer à des discussions avec d'autres utilisateurs d'un même logiciel ;
- rechercher des cours de formation sur les logiciels et vous y inscrire.

Pour accéder à la plupart des offres d'assistance, vous devez vous enregistrer en tant qu'utilisateur disposant d'un compte HP Passport et vous identifier comme tel. De nombreuses offres nécessitent en outre un contrat d'assistance. Pour obtenir un identifiant HP Passport, accédez à l'adresse suivante :

http://h20229.www2.hp.com/passport-registration.html

Les informations relatives aux niveaux d'accès sont détaillées à l'adresse suivante :

http://h20230.www2.hp.com/new_access_levels.jsp

HP Software Solutions Now accède au site Web du portail HPSW Solution and Integration. Ce site vous permet d'explorer les pages de HP Product Solutions qui comprendent une liste complète des intégrations entre produits HP, ainsi qu'une liste des processus ITIL. L'URL de ce site Web est **http://h20230.www2.hp.com/sc/solutions/index.jsp**

Informations sur la version PDF de l'aide en ligne

Ce document est une version PDF de l'aide en ligne. Il facilite l'impression de plusieurs rubriques issues de l'aide en ligne ou leur lecture au format PDF. Ce contenu a été initialement créé pour être considérée comme une aide en ligne dans un navigateur web, c'est pourquoi que certains sujets peuvent être mal formatés. Certaines rubriques interactives ne sont peut-être pas présents dans cette version PDF. Il est néanmoins possible de les imprimer normalement depuis l'aide en ligne.

Table des matières

Table des matières 4
Création d'adaptateurs d'intégration et de découverte 11
Chapitre 1 : Écriture et développement d'adaptateurs 12
Écriture et développement d'adaptateurs - Présentation 12
Création d'un contenu 12
Cycle de développement d'un adaptateur 13
Gestion des flux de données et intégration 16
Association d'une valeur métier au développement de découverte 17
Recherche d'exigences d'intégration 18
Développement d'un contenu d'intégration 21
Développement d'un contenu de découverte 23
Adaptateurs de découverte et composants associés 23
Séparation d'adaptateurs 24
Implémentation d'un adaptateur de découverte 26
Étape 1 : Création d'un adaptateur 28
Étape 2 : Affectation d'un travail à l'adaptateur 35
Étape 3 : Création d'un code Jython 37
Configurer l'exécution d'un processus distant 37
Chapitre 2 : Développement d'adaptateurs Jython 39
Référence des API de gestion des flux de données HP 39
Création d'un code Jython 39
Utilisation de fichiers JAR Java externes dans Jython 40
Exécution du code 40
Modification de scripts prêts à l'emploi 40
Structure du fichier Jython 41
Importations 42
Fonction principale – DiscoveryMain 42
Définitions de fonctions 43
Génération de résultats par le script Jython 44

Syntaxe ObjectStateHolder 44
Envoi de grandes quantités de données 46
Instance Framework 47
Recherche des informations d'identification correctes (pour les adaptateurs de connexion) 50
Traitement des exceptions Java 53
Résolution des problèmes de migration de Jython à partir de la version 2.1 vers la version 2.5.3 53
Prise en charge des paramètres régionaux dans les adaptateurs Jython 55
Ajout de la prise en charge d'une nouvelle langue 55
Changement de la langue par défaut 57
Détermination du jeu de caractères pour le codage 57
Définition d'un nouveau travail fonctionnant avec des données localisées 58
Décodage de commandes sans mot-clé 59
Utilisation de groupes de ressources 59
Référence des API 60
Enregistrement du code GFD 63
Bibliothèques et utilitaires Jython 64
Chapitre 3 : Messages d'erreur 68
Messages d'erreur - Présentation 68
Conventions d'écriture des erreurs 68
Niveaux de gravité des erreurs 71
Chapitre 4 : Développement d'adaptateurs de base de données génériques 73
Adaptateur de base de données générique - Présentation 74
Requêtes TQL pour l'adaptateur de base de données générique 74
Rapprochement 75
Hibernate comme fournisseur JPA 76
Préparation de la création d'un adaptateur 78
Préparation du package de l'adaptateur 83
Configuration de l'adaptateur - Méthode minimale 86
Configuration du fichier adapter.conf 87

Exemple : Remplissage d'un nœud et d'une adresse IP à l'aide de la méthode simplifiée 87
Configuration de l'adaptateur - Méthode avancée 91
Implémentation d'un plug-in 96
Déploiement de l'adaptateur 99
Modification de l'adaptateur 99
Créer un point d'intégration 99
Création d'une vue 100
Calcul des résultats 100
Affichage des résultats 100
Affichage des rapports 100
Activation des fichiers journaux 101
Utilisation d'Eclipse pour mapper des attributs de type de CI sur des tables de base de données 101
Fichiers de configuration de l'adaptateur 109
Fichier adapter.conf 110
Fichier simplifiedConfiguration.xml 112
Fichier orm.xml 114
Fichier reconciliation_types.txt 128
Fichier reconciliation_rules.txt (pour compatibilité rétroactive) 128
Fichier transformations.txt 130
Fichier discriminator.properties 131
Fichier replication_config.txt 132
Fichier fixed_values.txt 133
Fichier persistence.xml 133
Connexion à une base de données à l'aide de l'authentification NT 134
Convertisseurs prêts à l'emploi 135
Plug-in 140
Exemples de configuration 140
Fichiers journaux de l'adaptateur 149
Références externes 151
Résolution des problèmes et limitations 151

Chapitre 5 : Développement d'adaptateurs Java 153
Présentation de l'infrastructure de fédération 153
Adaptateur et interaction de mappage avec l'infrastructure de fédération 158
Infrastructure de fédération pour requêtes TQL fédérées 159
Interactions entre l'infrastructure de fédération, le serveur, l'adaptateur et le moteur de mappage 160
Flux d'infrastructure de fédération pour le remplissage 170
Interfaces d'adaptateur 171
Ressources d'adaptateur et débogage 173
Ajout d'un adaptateur pour une nouvelle source de données externe 173
Création d'un exemple d'adaptateur 182
Propriétés et balises de configuration XML 183
Interface DataAdapterEnvironment 185
OutputStream openResourceForWriting(String resourceName) throws FileNotFoundException; 185
InputStream openResourceForReading(String resourceName) throws FileNotFoundException; 186
Properties openResourceAsProperties(String propertiesFile) throws IOException; 186
String openResourceAsString(String resourceName) throws IOException; 187
public void saveResourceFromString(String relativeFileName, String value) throws IOException; 188
boolean resourceExists(String resourceName); 188
boolean deleteResource(String resourceName); 188
Collection<String> listResourcesInPath(String path); 189
DataAdapterLogger getLogger(); 189
DestinationConfig getDestinationConfig(); 189
int getChunkSize(); 189
int getPushChunkSize(); 189
ClassModel getLocalClassModel(); 190
CustomerInformation getLocalCustomerInformation(); 190
Object getSettingValue(String name); 190
Map<String, Object> getAllSettings(); 190
boolean isMTEnabled(); 190

String getUcmdbServerHostName(); 190
Chapitre 6 : Développement d'adaptateurs d'émission (push) 192
Développement et déploiement d'adaptateurs d'émission de données 192
Création d'un package d'adaptateur 193
Résolution des problèmes 196
Meilleures pratiques TQL recommandées pour les adaptateurs d'émission de données 197
Création de mappages 197
Création d'un fichier de mappage 197
Préparation des fichiers de mappage 198
Écriture de scripts Jython 201
Prise en charge de la synchronisation différentielle 205
Requêtes SQL de l'adaptateur d'émission de données XML générique 207
Adaptateur d'émission de données de service Web générique 207
Fichier de mappage de référence 227
Schéma du fichier de mappage 229
Schéma des résultats de mappage 241
Personnalisation 245
Chapitre 7 : Développement d'adaptateurs d'émission (push) génériques avancés 247
Présentation 247
Fichier de mappage 247
Voyageur Groovy 250
Écriture de scripts Groovy 253
Implémentation de l'interface PushConnector 254
Création d'un package d'adaptateur 255
Schéma du fichier de mappage 256
Utilisation d'API 263
Chapitre 8 : Introduction aux API 264
Présentation des API 264
Chapitre 9 : API HP Universal CMDB 265
Conventions 265
Utilisation de l'API HP Universal CMDB 265

Structure générale d'une application 266
Placement du fichier Jar de l'API dans le classpath 269
Création d'un utilisateur d'intégration 269
Cas d'utilisation de l'API UCMDB 271
Exemples 273
Chapitre 10 : API de service Web HP Universal CMDB 274
Conventions 274
Présentation de l'API de service Web HP Universal CMDB 275
Appel du service Web 278
Interrogation de CMDB 278
Mise à jour d'UCMDB 282
Interrogation du modèle de classe UCMDB 283
getClassAncestors 283
getAllClassesHierarchy 284
getCmdbClassDefinition 284
Requête d'analyse d'impact 285
Paramètres généraux UCMDB 285
Paramètres de sortie UCMDB 288
Méthodes de requête UCMDB 289
executeTopologyQueryByNameWithParameters 289
executeTopologyQueryWithParameters 290
getChangedCIs 291
getCINeighbours 292
getCIsByID 292
getCIsByType 293
getFilteredCIsByType 293
getQueryNameOfView 297
getTopologyQueryExistingResultByName 297
getTopologyQueryResultCountByName 297
pullTopologyMapChunks 298
releaseChunks 300

Méthodes de mise à jour UCMDB 300
addCIsAndRelations 300
addCustomer 301
deleteCIsAndRelations 301
removeCustomer 302
updateCIsAndRelations 302
Méthodes d'analyse d'impact UCMDB 303
calculateImpact 303
getImpactPath 303
getImpactRulesByNamePrefix 304
API de service Web d'état réel 305
Cas d'utilisation de l'API de service Web UCMDB 306
Exemples 307
Chapitre 11 : API Java de gestion des flux de données 309
Utilisation de l'API Java de gestion des flux de données 309
Chapitre 12 : API de service Web de la gestion des flux de données 311
API de service Web de la gestion des flux de données - Présentation 311
Conventions 312
Appel du service Web 312
Structures de données et méthodes de gestion des flux de données 312
Structures de données 313
Méthodes de gestion des travaux de découverte 313
Méthodes de gestion de déclencheurs 315
Méthodes de données de domaine et de sonde 317
Méthodes de données d'informations d'identification 319
Méthodes d'actualisation de données 321
Exemple de code 323
Exemple d'ajout d'informations d'identification 326
Vos commentaires sont toujours les bienvenus. 330

Création d'adaptateurs d'intégration et de découverte

Chapitre 1 : Écriture et développement d'adaptateurs

Contenu de ce chapitre :

Écriture et développement d'adaptateurs - Présentation 12

Création d'un contenu 12

Développement d'un contenu d'intégration 21

Développement d'un contenu de découverte 23

Implémentation d'un adaptateur de découverte 26

Étape 1 : Création d'un adaptateur 28

Étape 2 : Affectation d'un travail à l'adaptateur 35

Étape 3 : Création d'un code Jython 37

Configurer l'exécution d'un processus distant 37

Écriture et développement d'adaptateurs - Présentation

Avant d'entamer la planification proprement dite du développement d'adaptateurs, il est important de comprendre les processus et les interactions communément associés à ce développement.

Les sections qui suivent vous aident à comprendre ce que vous devez savoir et faire en vue de réussir la gestion et l'exécution d'un projet de développement de découverte.

Ce chapitre :

- Supposent que vous connaissez le fonctionnement de HP Universal CMDB et que les éléments du système vous sont familiers. Vise à vous aider au cours du processus d'apprentissage et ne saurait constituer un guide exhaustif.
- Couvre les étapes de la planification, de la recherche et de l'implémentation d'un nouveau contenu de découverte pour HP Universal CMDB, ainsi que certaines directives et considérations dont vous devez tenir compte.
- Fournit des informations sur les API essentielles de l'infrastructure de la gestion des flux de données. Pour une documentation complète sur les API disponibles, voir *HP Universal CMDB Data Flow Management API Reference*. (D'autres API non formelles existent ; toutefois, même utilisées sur des adaptateurs prêts à l'emploi, elles peuvent être sujettes à modification.)

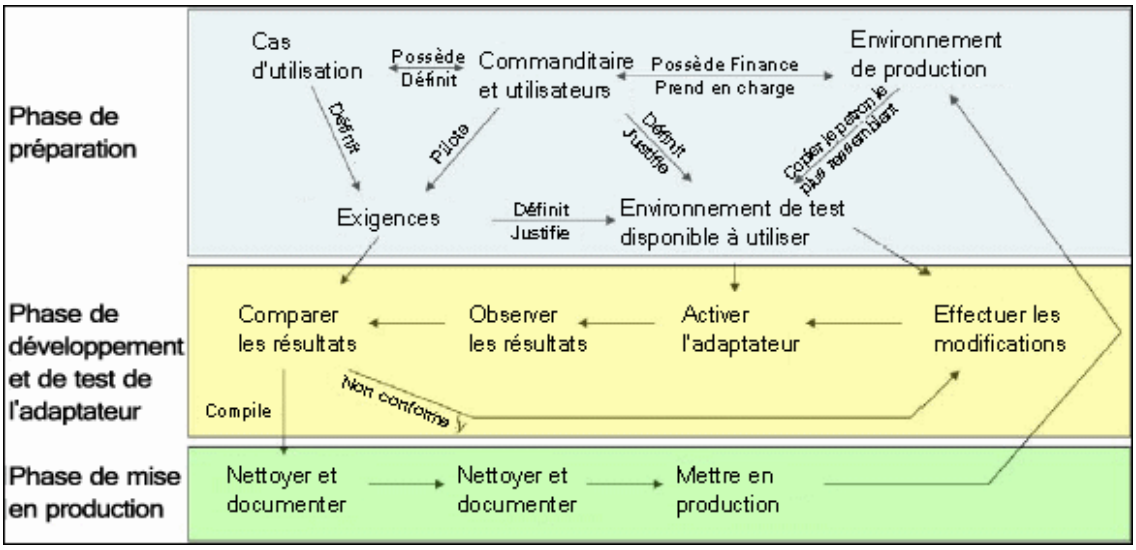
Création d'un contenu

Contenu de cette section :

- " Cycle de développement d'un adaptateur " , ci-dessous
- " Gestion des flux de données et intégration " , page 16
- " Association d'une valeur métier au développement de découverte " , page 17
- " Recherche d'exigences d'intégration " , page 18

Cycle de développement d'un adaptateur

L'illustration ci-après représente un organigramme d'écriture d'adaptateur. La majeure partie du temps est consacrée à la section médiane, qui correspond à la boucle itérative du développement et des tests.



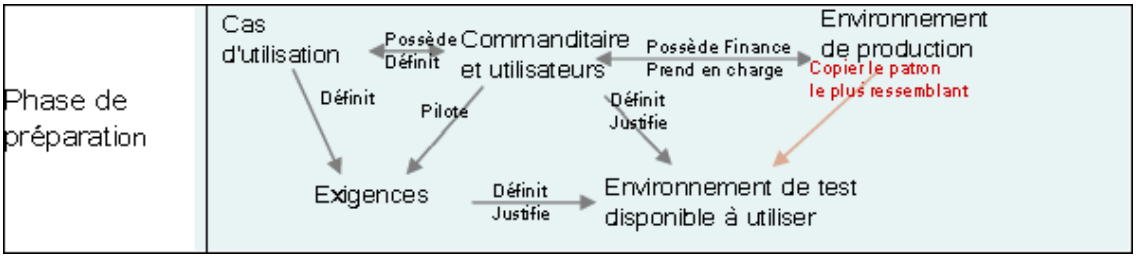
Chaque phase du développement d'un adaptateur repose sur celle qui la précède.

Une fois satisfait de l'aspect et du fonctionnement de l'adaptateur, vous voici prêt à le compresser. Utilisez le Gestionnaire des packages UCMDB ou effectuez une exportation manuelle des composants pour créer un fichier *.zip compressé. Pour une pratique optimale, déployez et testez ce package sur un autre système UCMDB avant de le mettre à disposition en production. Vous vous assurez ainsi que tous les composants sont pris en compte et que leur compression est réussie. Pour plus d'informations sur le conditionnement, voir "Package Manager" dans le *Manuel d'administration HP Universal CMDB*.

Les sections suivantes examinent chacune des phases qui comprennent les étapes les plus essentielles et les pratiques optimales :

- " Phase de préparation et de recherche " , page suivante
- " Développement et test de l'adaptateur " , page suivante
- " Compression et mise en production de l'adaptateur " , page 15

Phase de préparation et de recherche



La phase de préparation et recherche comprend les besoins métier et les cas d'utilisation qui motivent le développement. Elle prend également en charge la sécurisation des installations nécessaires au développement et au test de l'adaptateur.

1. Lorsque vous envisagez de modifier un adaptateur, la première étape technique consiste à faire une sauvegarde de cet adaptateur et à vous assurer que vous pouvez rétablir son état antérieur intact. Si vous envisagez de créer un adaptateur, copiez l'adaptateur le plus similaire et sauvegardez-le sous un nom adapté. Pour plus d'informations, voir « Volet Ressources » dans le *Manuel de gestion des flux de données HP Universal CMDB*.

2. Recherchez la méthode que doit utiliser l'adaptateur pour collecter les données :

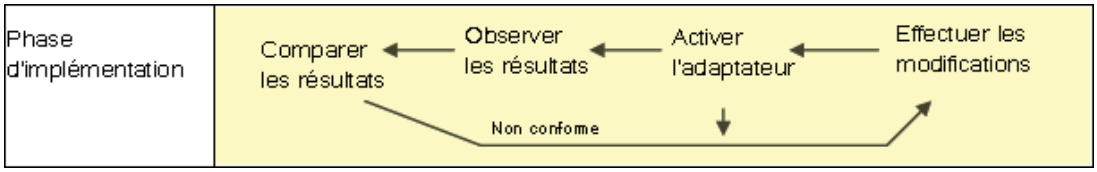
   - Utilisez des outils/protocoles externes pour obtenir les données.
   - Développez la manière dont l'adaptateur devra créer des éléments de configuration (CI) en fonction des données.
   - Vous savez à présent à quoi doit ressembler un adaptateur similaire.

3. Déterminez les adaptateurs les plus similaires selon les critères suivants :

   - création de CI identiques ;
   - utilisation de protocoles identiques (SNMP) ;
   - mêmes types de cibles (par type de SE, versions, etc.).

4. Copiez l'ensemble du package.

5. Décompressez le contenu du package dans l'espace de travail, puis renommez les fichiers de l'adaptateur (XML) et Jython (.py).



Développement et test de l'adaptateur

La phase de développement et test de l'adaptateur est un processus particulièrement itératif. À mesure que l'adaptateur prend forme, entamez les tests en fonction des cas d'utilisation finale,

effectuez des modifications, testez à nouveau, et répétez ce processus jusqu'à ce que l'adaptateur soit conforme aux exigences.

Démarrage et préparation d'une copie

- Modifiez les différentes parties XML de l'adaptateur : nom (identifiant) en ligne 1, types de CI créés, et nom du script Jython appelé.
- Générez une copie qui produit les mêmes résultats que l'adaptateur d'origine.
- Mettez en commentaire la majeure partie du code, particulièrement le code de production de résultats essentiel.

Développement et test

- Utilisez d'autres échantillons de code pour développer des modifications.
- Testez l'adaptateur en l'exécutant.
- Utilisez une vue dédiée pour valider des résultats complexes, cherchez à valider des résultats simples.

Compression et mise en production de l'adaptateur

La phase Compression et mise en production de l'adaptateur est la dernière étape du processus de développement. Dans le cadre d'une pratique optimale, une passe finale doit permettre de nettoyer les restes d'un débogage, les documents et les commentaires, et d'examiner les considérations liées à la sécurité, etc., avant de procéder à l'emballage. Vous devez toujours disposer au moins d'un document de type Lisez-moi qui explique les mécanismes de fonctionnement internes de l'adaptateur. Une tierce personne (voire vous-même) peut être amenée à examiner cet adaptateur ultérieurement. Même la documentation la plus restreinte lui sera d'une aide considérable à ce moment.

Nettoyage et documentation

- Éliminez le débogage.
- Commentez toutes les fonctions et ajoutez des commentaires d'introduction dans la section principale.
- Créez un échantillon de code TQL et une vue qui permettra à l'utilisateur de le tester.

Création d'un package

- Exportez les ressources, notamment les adaptateurs et le code TQL, au moyen du Gestionnaire des packages. Pour plus d'informations, voir "Package Manager" dans le *Manuel d'administration HP Universal CMDB*.
- Vérifiez toute dépendance éventuelle de votre package vis-à-vis d'autres ; par exemple, si les CI générés par ces packages constituent des CI d'entrée pour votre adaptateur.

- Utilisez le Gestionnaire des packages pour compresser votre package au format .zip. Pour plus d'informations, voir "Package Manager" dans le *Manuel d'administration HP Universal CMDB*.
- Testez le déploiement en éliminant des parties du nouveau contenu puis en le redéployant, ou en le déployant sur un autre système de test.

Gestion des flux de données et intégration

Les adaptateurs de gestion de flux de données (GFD) sont en mesure de s'intégrer à d'autres produits. Tenez compte des définitions suivantes :

- La fonction GFD collecte des contenus auprès de nombreuses cibles.
- L'intégration collecte de nombreux types de contenus auprès d'un système unique.

Remarquez que ces définitions ne font pas de distinction entre les méthodes de collecte. La technologie GFD non plus. Le processus de développement d'un nouvel adaptateur reste le même lors du développement d'une nouvelle intégration. Vous effectuez la même recherche et les mêmes choix d'adaptateurs nouveaux ou existants, écrivez les adaptateurs de la même manière, etc. Seuls quelques éléments changent :

- La planification de l'adaptateur final. Les adaptateurs d'intégration peuvent s'exécuter plus fréquemment que les adaptateurs de découverte, mais cela dépend des cas d'utilisation.
- Les CI d'entrée :
  - Intégration : déclencheur non CI à exécuter sans entrée : un nom de fichier ou une source est transmis par le biais du paramètre de l'adaptateur.
  - Découverte : utilise des CI CMDB standard en guise d'entrée.

Dans le cadre de projets d'intégration, vous devez presque systématiquement réutiliser un adaptateur existant. Le sens de l'intégration (intégration de HP Universal CMDB à un autre produit, ou d'un autre produit à HP Universal CMDB) peut influer sur votre approche du développement. Des packages pratiques sont disponibles. Vous pouvez les copier pour votre propre usage au moyen de techniques éprouvées.

Pour intégrer HP Universal CMDB à un autre projet :

- Créez un code TQL qui produit les CI et les relations à exporter.
- Utilisez un adaptateur enveloppeur générique qui exécutera le code TQL et écrira les résultats dans un fichier XML que lira le produit externe.

> **Remarque :** Pour obtenir des exemples de packages pratiques, contactez Assistance HP Software.

Pour intégrer un autre produit à HP Universal CMDB, le mode d'action de l'adaptateur d'intégration varie selon la manière dont l'autre produit expose ses données :

Type d'intégration	Exemple de référence à réutiliser
Accès direct à la base de données du produit	HP ED
Lecture dans un fichier csv ou xml produit par une exportation	HP ServiceCenter
Accès à l'API d'un produit	BMC Atrium/Remedy

Association d'une valeur métier au développement de découverte

En pratique, le développement d'un contenu de découverte doit être motivé par un plan et un dossier d'affaire en vue d'induire une valeur métier. En d'autres termes, l'objectif du mappage des composants système et des CI, ainsi que de leur ajout à la base CMDB consiste à produire une valeur métier.

Le contenu peut ne pas être systématiquement utilisé pour le mappage d'applications, même si cette opération constitue une étape intermédiaire répandue dans de nombreux cas d'utilisation. Quelle que soit l'utilisation finale du contenu, votre plan doit répondre aux questions suivantes :

- Qui est le consommateur ? Comment le consommateur doit-il agir sur les informations mises à disposition par les CI (et les relations entre ceux-ci) ? Dans quel contexte métier les CI et les relations seront-ils visualisés ? Le consommateur de ces CI est-il une personne ou un produit ? Ou les deux ?
- Une fois la combinaison parfaite des CI et des relations en place dans la base CMDB, comment planifier leur utilisation pour produire une valeur métier ?
- À quoi devrait ressembler le mappage parfait ?
  - Quel terme décrirait le plus pertinemment les relations entre les différents CI ?
  - Quels sont les types de CI les plus importants à inclure ?
  - Quel est l'usage final et l'utilisateur final du mappage ?
- Quelle serait l'organisation de rapport parfaite ?

Une fois la justification métier établie, l'étape suivante consiste à donner corps à la valeur métier dans un document. Cette étape signifie représenter le mappage parfait au moyen d'un outil de dessin, et comprendre les éléments suivants : l'interdépendance des CI et leur incidence ; les rapports ; le mode de suivi des modifications ; la nature des changements importants ; la surveillance ; la conformité, ainsi que la valeur métier supplémentaire, selon les exigences des cas d'utilisation.

Ce dessin (ou modèle) est appelé **plan de projet**.

Par exemple, s'il est essentiel que l'application soit informée de la modification d'un fichier de configuration donné, ce fichier doit être mappé et lié au CI approprié (auquel il est associé) sur le plan dessiné.

Travaillez avec un expert du domaine, ou SME (Subject Matter Expert), qui sera également utilisateur final du contenu développé. Cet expert identifiera les entités essentielles (les CI avec attributs et relations) qui doivent exister dans la base CMDB pour fournir une valeur métier.

Fournir un questionnaire au propriétaire de l'application (également au SME dans le cas présent) constitue une méthode adaptée. Le propriétaire doit être en mesure de spécifier les objectifs et le plan de projet ci-dessus. Le propriétaire doit fournir au moins une architecture courante de l'application.

Vous devez mapper uniquement les données critiques et non les données superflues : vous pourrez toujours améliorer l'adaptateur par la suite. L'objectif consiste à mettre en œuvre une découverte restreinte qui fonctionne et fournit une valeur. Le mappage de grandes quantités de données produit des mappages plus impressionnants, mais le développement peut alors se révéler déroutant et plus long.

Une fois le modèle et la valeur métier clarifiés, passez à l'étape suivante. Vous pourrez revenir sur cette étape par la suite, dès lors que les étapes qui la suivent mettent à disposition des informations plus concrètes.

Recherche d'exigences d'intégration

Les exigences préalables à cette étape correspondent à un **plan de projet** décrivant les CI et les relations que le processus GFD doit découvrir, notamment les attributs à découvrir. Pour plus d'informations, voir " Écriture et développement d'adaptateurs - Présentation " , page 12.

Contenu de cette section :

- " Modification d'un adaptateur existant " , ci-dessous
- " Écriture d'un nouvel adaptateur " , page suivante
- " Recherche de modèle " , page suivante
- " Recherche de technologie " , page suivante
- " Directives pour la sélection de modes d'accès aux données " , page 20
- " Récapitulatif " , page 21

Modification d'un adaptateur existant

Lorsqu'il existe un adaptateur de champ ou prêt à l'emploi, vous modifiez tout de même un adaptateur existant si les conditions suivantes s'appliquent :

- l'adaptateur ne découvre pas les attributs spécifiques nécessaires ;
- un type de cible spécifique (SE) n'est pas découvert ou l'est de manière inappropriée ;

- une relation spécifique n'est ni découverte, ni créée.

Si un adaptateur existant ne remplit que partiellement la tâche, votre première approche consiste à évaluer les adaptateurs existants et à vérifier si l'un d'eux effectue la quasi-totalité des tâches requises ; dans l'affirmative, vous pouvez modifier cet adaptateur.

Vous devez également déterminer si un adaptateur pratique est disponible. Les adaptateurs de champ sont des adaptateurs de découverte disponibles mais non prêts à l'emploi. Contactez l'Assistance HP Software pour obtenir la liste à jour des adaptateurs de champ.

Écriture d'un nouvel adaptateur

Un nouvel adaptateur doit être développé dans les circonstances suivantes :

- lorsqu'il est plus rapide d'écrire un adaptateur que d'insérer les informations manuellement dans la base CMDB (généralement, entre 50 et 100 CI et relations) ou lorsque l'effort correspondant n'est pas ponctuel ;
- lorsque le besoin justifie l'effort ;
- si aucun adaptateur de champ ou prêt à l'emploi n'est disponible ;
- si les résultats peuvent être réutilisés ;
- lorsque l'environnement cible ou ses données sont disponibles (vous ne pouvez découvrir ce que vous ne voyez pas).

Recherche de modèle

- Parcourez le modèle de classe UCMDB (Gestionnaire des types de CI) et associez les entités et les relations issues de votre **plan de projet** aux types d'élément de configuration (CI) existants. Il est fortement recommandé de se conformer au modèle courant afin d'éviter toute complication potentielle au cours de la mise à jour d'une version. Si vous êtes amené à étendre le modèle, vous devrez créer des type de CI. En effet, une mise à jour est susceptible d'écraser les types de CI prêts à l'emploi.
- Si certains attributs, entités ou relations sont manquants dans le modèle courant, vous devrez les créer. Mieux vaut créer un package avec ces types de CI (qui parallèlement renfermeront plus tard l'ensemble de la découverte, des vues et des autres éléments associés au package) étant donné que vous devez être en mesure de déployer ces types de CI sur chaque installation de HP Universal CMDB.

Recherche de technologie

Après avoir vérifié que la base CMDB contient les CI pertinents, l'étape suivante consiste à décider de la manière d'extraire ces données des systèmes appropriés.

L'extraction des données implique généralement l'utilisation d'un protocole pour accéder à une partie gestion de l'application, aux données de l'application proprement dites, ou aux bases de données ou fichiers de configuration associés à l'application. Toute source de données susceptible de fournir des informations sur un système est précieuse. La recherche d'une technologie requiert une connaissance étendue du système en question et, parfois, une certaine créativité.

Pour les applications maison, il peut s'avérer utile de fournir un questionnaire au propriétaire de l'application. Sur ce formulaire, le propriétaire répertoriera toutes les sections de l'application susceptibles de fournir des informations nécessaires dans le cadre du plan de projet et des valeurs métier. Ces informations devront inclure (sans toutefois s'y limiter) les bases de données et interfaces de gestion, fichiers de configuration, fichiers journaux, programmes d'administration, services Web, ou encore messages et événements envoyés.

Dans le cas de produits prêts à l'emploi, vous devez vous focaliser sur les documentations, forums ou ressources d'assistance du produit. Consultez entre autres les manuels d'administration, les manuels de gestion et ceux relatifs aux intégrations et aux plug-in. Si des données manquent encore au niveau des interfaces de gestion, consultez les fichiers de configuration de l'application, les entrées de registre, les journaux d'événements NT ainsi que tout élément de l'application contrôlant son bon fonctionnement.

Directives pour la sélection de modes d'accès aux données

Pertinence : Sélectionnez les sources ou la combinaison de sources qui fournissent le plus de données. Si une source unique fournit la majeure partie des informations alors que le reste est éparpillé ou difficile d'accès, tentez d'évaluer la valeur des informations restantes en la comparant au risque ou à l'effort lié à leur obtention. Parfois, vous pouvez être amené à décider de réduire le plan de projet dès lors que la valeur ou le coût ne garantit pas l'effort investi.

Réutilisation : Si HP Universal CMDB intègre déjà la prise en charge d'un protocole de connexion spécifique, mieux vaut l'exploiter. En d'autres termes, l'infrastructure GFD est alors en mesure de fournir un client et une configuration de connexion prêts à l'emploi. Dans la négative, vous pouvez être amené à investir dans un développement d'infrastructure. Pour afficher les protocoles de connexion de HP Universal CMDB pris en charge, sélectionnez **Gestion des flux de données > Configuration de Data Flow Probe > volet Domaines et sondes**. Pour plus d'informations sur chaque protocole, reportez-vous à la section décrivant les protocoles pris en charge dans le *Manuel du Content Pack d'intégration et de découverte HP UCMDB*.

Vous pouvez ajouter de nouveaux protocoles en ajoutant de nouveaux CI au modèle. Pour plus d'informations, contactez l'Assistance HP Software.

Remarque : Pour accéder aux données du Registre de Windows, utilisez WMI ou NTCmd.

Sécurité : L'accès aux informations requiert généralement des données d'identification (nom d'utilisateur, mot de passe) qui sont entrées dans le CMDB et sont conservées en lieu sûr dans le produit. Le cas échéant, et si l'ajout d'une sécurité n'entre pas en conflit avec d'autres principes que vous avez définis, sélectionnez le protocole ou les informations d'identification les moins sensibles qui répondent toutefois aux besoins d'accès. Par exemple, si des informations sont disponibles à la fois par le biais de JMX (interface d'administration standard, restreinte) et de Telnet, mieux vaut utiliser JMX sachant que cette technologie offre de manière inhérente un accès limité, voire (généralement) nul, à la plate-forme sous-jacente.

Confort : Certaines interfaces de gestion intègrent des caractéristiques plus avancées. Par exemple, il peut s'avérer plus facile d'émettre des requêtes (SQL, WMI) que de naviguer au sein d'arborescences d'informations ou de développer des expressions régulières à des fins d'analyse.

Public des développeurs : Les personnes qui développeront réellement les adaptateurs peuvent avoir une préférence pour une technologie donnée. Cette situation peut également être prise en

compte lorsque deux technologies fournissent pratiquement les mêmes informations pour un coût égal selon d'autres facteurs.

Récapitulatif

Cette étape a pour résultat un document qui décrit les méthodes d'accès et les informations pertinentes que chacune d'elles peut extraire. Le document doit également contenir un mappage de chaque source sur chacune des données de plan de projet pertinentes.

Chaque méthode d'accès doit être marquée conformément aux instructions ci-dessus. Enfin, vous devriez à présent disposer d'un plan identifiant les différentes sources à découvrir et les informations à extraire de chacune d'elles pour enrichir le modèle du plan de projet (qui devrait être, à présent, mappé sur le modèle UCMDB correspondant).

Développement d'un contenu d'intégration

Avant de créer une intégration, vous devez en comprendre les exigences :

- L'intégration doit-elle copier des données dans la base CMDB ? Les données doivent-elles être suivies selon un historique ? La source est-elle non fiable ?

  Si vous répondez oui à ces questions, un **remplissage** est nécessaire.

- L'intégration doit-elle fédérer des données à la volée pour les vues et les requêtes TQL ? La précision des modifications apportées aux données est-elle essentielle ? La quantité de données est-elle trop importante pour une copie dans la base CMDB alors même que la quantité de données requise est généralement faible ?

  Si vous répondez oui à ces questions, une **fédération** est nécessaire.

- L'intégration doit-elle émettre des données vers des sources de données distantes ?

  Si vous répondez oui à ces questions, une **émission de données** est nécessaire.

> **Remarque :** Pour une flexibilité maximale, les flux de fédération et de remplissage peuvent être configurés pour la même intégration.

Pour plus d'informations sur les différents types d'intégration, voir Integration Studio dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Cinq options distinctes sont disponibles pour créer des adaptateurs d'intégration :

1. Adaptateur Jython :
   - Patron de découverte classique
   - Écrit en Jython

- Utilisé pour le remplissage

Pour plus d'informations, voir " Développement d'adaptateurs Jython " , page 39.

2. Adaptateur Java :

   - Adaptateur qui implémente une des interfaces d'adaptateur dans l'infrastructure du SDK de fédération (Federation SDK Framework).
   - Peut être utilisé pour une ou plusieurs fonctions de fédération, remplissage ou émission de données (selon l'implémentation requise).
   - Intégralement écrit en Java. Permet ainsi d'écrire un code qui se connectera à toute source ou cible possible.
   - Convient aux travaux qui se connectent à une seule cible ou source de données.

   Pour plus d'informations, voir " Développement d'adaptateurs Java " , page 153.

3. Adaptateur de base de données générique :

   - Adaptateur abstrait fondé sur l'adaptateur Java qui utilise l'infrastructure du SDK de fédération (Federation SDK Framework).
   - Permet la création d'adaptateurs qui se connectent à des référentiels de données externes.
   - Prend en charge la fédération et le remplissage (au moyen d'un plug-in Java implémenté pour la prise en charge des changements).
   - Relativement facile à définir car il repose essentiellement sur des fichiers XML et de configuration de propriétés.
   - La configuration principale repose sur un fichier **orm.xml** qui mappe les classes UCMDB sur des colonnes de base de données.
   - Convient aux travaux qui se connectent à une seule source de données.

   Pour plus d'informations, voir " Développement d'adaptateurs de base de données génériques " , page 73.

4. Adaptateur d'émission générique :

   - Adaptateur abstrait fondé sur l'adaptateur Java (l'infrastructure du SDK de fédération) et l'adaptateur Jython.
   - Permet la création d'adaptateurs qui émettent des données vers des cibles distantes.
   - Relativement facile à définir car il vous suffit de définir le mappage entre le code XML et les classes UCMDB, ainsi qu'un script Jython qui émet les données vers la cible.

- Convient aux travaux qui se connectent à une seule cible de données.
- Utilisé pour l'émission de données.

Pour plus d'informations, voir " Développement d'adaptateurs d'émission (push) " , page 192.

5. Adaptateur d'émission générique avancé :
   - Intègre toutes les fonctions de l'adaptateur d'émission de données décrites ci-dessus.
   - Adaptateur basé sur un élément racine
   - Mappe une structure de données de l'arborescence UCMDB sur une structure de données d'une arborescence cible.

   Pour plus d'informations, voir " Développement d'adaptateurs d'émission (push) génériques avancés " , page 247.

Le tableau suivant présente les capacités de chaque adaptateur :

Flux/Adaptateur	Adaptateur Jython	Adaptateur Java	Adaptateur de base de données générique	Adaptateur d'émission générique	Adaptateur d'émission générique avancé
Remplissage	✓	✓	✓	×	×
Fédération	×	✓	✓	×	×
Émission de données	×	✓	×	✓	✓

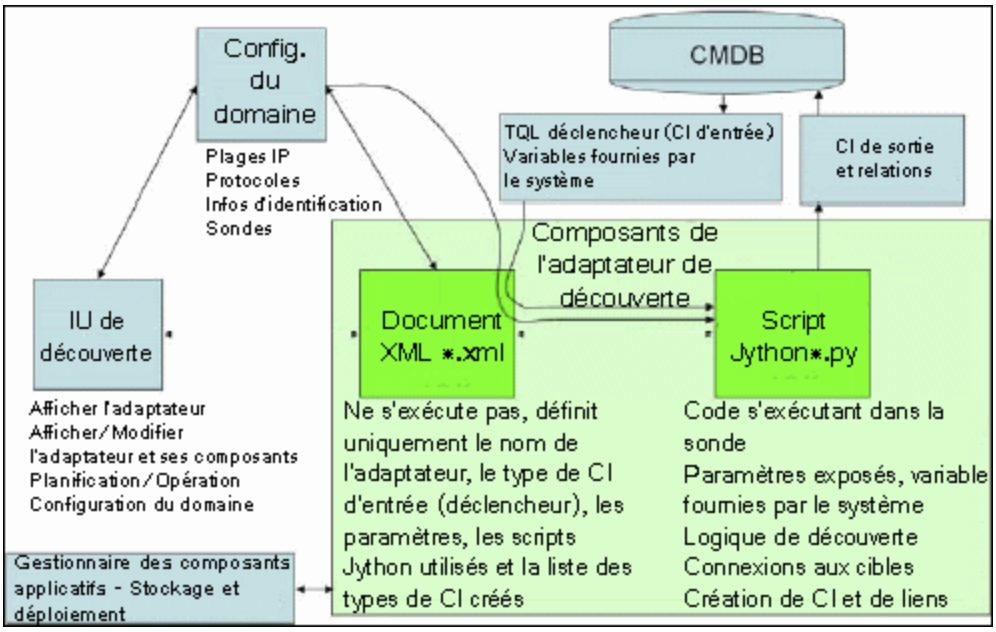
Développement d'un contenu de découverte

Contenu de cette section :

- " Adaptateurs de découverte et composants associés " , ci-dessous
- " Séparation d'adaptateurs " , page suivante

Adaptateurs de découverte et composants associés

Le diagramme ci-après présente les composants d'un adaptateur et les composants avec lesquels ils interagissent pour exécuter la découverte. Les composants présentés en vert constituent les adaptateurs proprement dits. Les composants présentés en bleu sont ceux qui interagissent avec les adaptateurs.



Remarquez que la notion minimale d'adaptateur correspond à deux fichiers : un document XML et un script Jython. L'infrastructure de découverte, notamment les CI d'entrées, les informations d'identification et les bibliothèques fournies par l'utilisateur, est exposée à l'adaptateur au moment de l'exécution. Les deux composants de l'adaptateur de découverte sont administrés par le biais de la gestion de flux de données. Ils sont enregistrés sous forme opérationnelle dans la base CMDB proprement dite ; si le package externe reste présent, il n'est pas référencé dans le cadre d'une exploitation. Le Gestionnaire des packages active le maintien de la nouvelle capacité de contenu de découverte et d'intégration.

Les CI d'entrée vers l'adaptateur sont fournis au moyen d'un code TQL et sont exposés au script de l'adaptateur dans des variables fournies par le système. Des paramètres d'adaptateur sont également fournis en tant que données de destination. Vous pouvez ainsi configurer le fonctionnement de l'adaptateur selon une fonction spécifique à celui-ci.

L'application GFD permet de créer et de tester les nouveaux adaptateurs. Pour écrire l'adaptateur, vous utilisez les pages Universal Discovery, Gestion de l'adaptateur et Configuration de Data Flow Probe.

Les adaptateurs sont enregistrés et transportés sous forme de packages. L'application Gestionnaire des packages et la console JMX permettent de créer des packages à partir d'adaptateurs nouvellement créés et de déployer des adaptateurs sur de nouveaux systèmes.

Séparation d'adaptateurs

Une découverte entière pourrait être définie dans un seul adaptateur. Toutefois, une conception dans les règles de l'art exige qu'un système complexe soit séparé en composants plus simples, plus faciles à gérer.

Ci-après figurent les directives et les pratiques optimales qui président à la division du processus d'adaptateur :

- La découverte doit s'effectuer par étapes. Chaque étape doit être représentée par un adaptateur qui doit mapper une section ou un niveau du système. Pour poursuivre la découverte du système, les adaptateurs doivent reposer sur l'étape ou le niveau précédent à découvrir. Par exemple, un adaptateur A est déclenché par le résultat TQL d'un serveur d'applications et mappe le niveau du serveur d'applications. Dans le cadre de ce mappage, un composant de connexion JDBC est mappé. L'adaptateur B enregistre un composant de connexion JDBC en tant que code TQL déclencheur et utilise les résultats produits par l'adaptateur A pour accéder au niveau base de données (par exemple, par le biais de l'attribut URL de la connexion JDBC) et mappe le niveau base de données.

- **Paradigme de connexion en deux phases** : La plupart des systèmes requièrent des informations d'identification pour accéder à leurs données. En d'autres termes, une combinaison utilisateur/mot de passe doit être soumise à ces systèmes. L'administrateur GFD fournit les informations d'identification de manière sécurisée au système. Il peut en fournir plusieurs pour des connexions gérées selon des priorités. Ces ressources sont désignées sous le nom de **dictionnaire de protocoles**. Si le système n'est pas accessible (pour quelque raison que ce soit), il est inutile de procéder à une découverte plus avancée. Si la connexion aboutit, il convient d'indiquer les informations d'identification utilisées afin d'accéder ultérieurement à la découverte.

  Ces deux phases conduisent à une séparation des deux adaptateurs dans les cas suivants :

  - **Adaptateur de connexion** : Il s'agit d'un adaptateur qui accepte un déclencheur initial et recherche l'existence d'un agent distant sur ce déclencheur. Pour ce faire, il essaie toutes les entrées du dictionnaire de protocoles qui correspondent au type de cet agent. Si cette démarche aboutit, cet adaptateur produit comme résultat un CI d'agent distant (SNMP, WMI, etc.) qui pointe également sur l'entrée appropriée dans le dictionnaire de protocoles, à des fins de connexion ultérieure. Ce CI d'agent fait alors partie intégrante d'un déclencheur pour l'adaptateur de contenu.

  - **Adaptateur de contenu** : La condition préalable au fonctionnement de cet adaptateur est la connexion réussie de l'adaptateur précédent (conditions préalables spécifiées par les codes TQL). Ces types d'adaptateur n'ont plus besoin de passer en revue tout le dictionnaire de protocoles, étant donné qu'ils disposent d'un moyen d'obtenir les informations d'identification appropriées auprès du CI de l'agent distant et de les utiliser pour se connecter au système qui fait l'objet de la découverte.

- Différentes considérations de planification peuvent également avoir une incidence sur la division de la découverte. Par exemple, un système peut n'accepter les requêtes qu'au cours des heures non ouvrées. Aussi, même s'il est raisonnable de joindre l'adaptateur au même adaptateur découvrant un autre système, les planifications différentes vous contraindront à créer deux adaptateurs au final.

- L'utilisation de différentes interfaces ou technologies de gestion en vue de découvrir un même système doit être répartie sur des adaptateurs distincts. Ainsi, vous pouvez activer la méthode d'accès appropriée pour chaque système ou entreprise. Par exemple, certaines entreprises sont dotées d'un accès WMI aux machines, mais ces dernières ne disposent d'aucun agent SNMP installé.

Implémentation d'un adaptateur de découverte

Une tâche GFD a pour objectif d'accéder à des systèmes distants (ou locaux), en modélisant des données extraites en tant que CI et en enregistrant les CI dans CMDB. Cette tâche comprend les étapes suivantes :

1. **Créer un adaptateur.**

   Vous configurez un fichier d'adaptateur qui renferme le contexte, les paramètres et les types de résultats, en sélectionnant les scripts qui feront partie intégrante de l'adaptateur. Pour plus d'informations, voir " Étape 1 : Création d'un adaptateur " , page 28.

2. **Créer un travail de découverte.**

   Vous configurez un travail au moyen d'informations de planification et d'une requête déclencheur. Pour plus d'informations, voir " Étape 2 : Affectation d'un travail à l'adaptateur " , page 35.

3. **Modifier le code de découverte.**

   Vous pouvez modifier le code Jython ou Java que renferment les fichiers d'adaptateur et qui référence l'infrastructure GFD. Pour plus d'informations, voir " Étape 3 : Création d'un code Jython " , page 37.

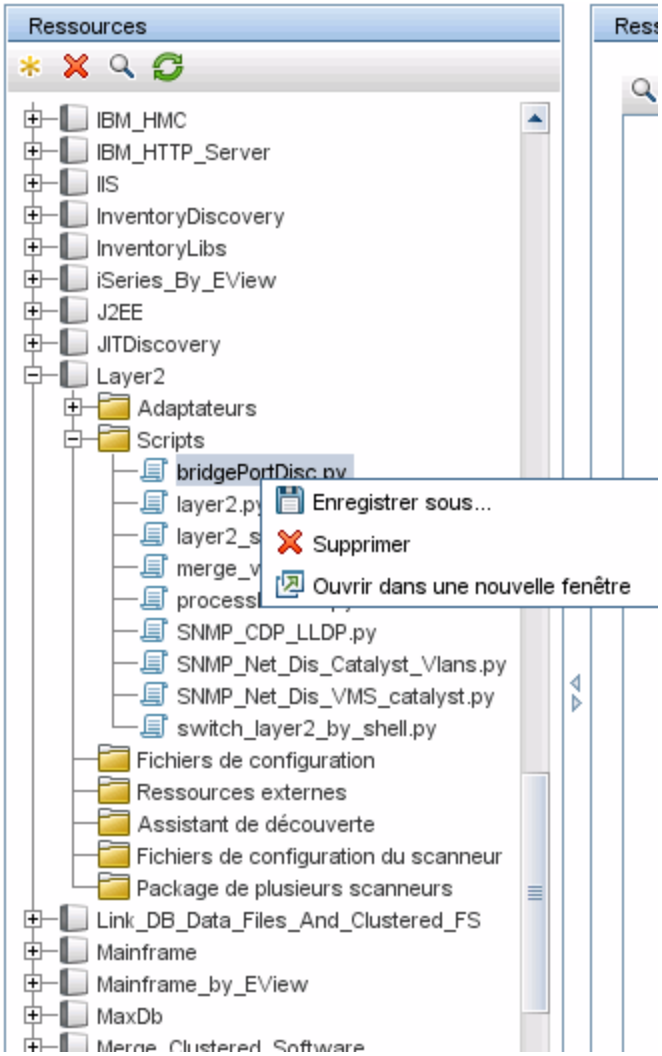
Pour écrire de nouveaux adaptateurs, vous créez chacun des composants ci-dessus, chacun d'eux étant automatiquement lié au composant de l'étape précédente. Par exemple, une fois que vous avez créé un travail et sélectionné l'adaptateur pertinent, le fichier d'adaptateur est lié à ce travail.

Code d'adaptateur

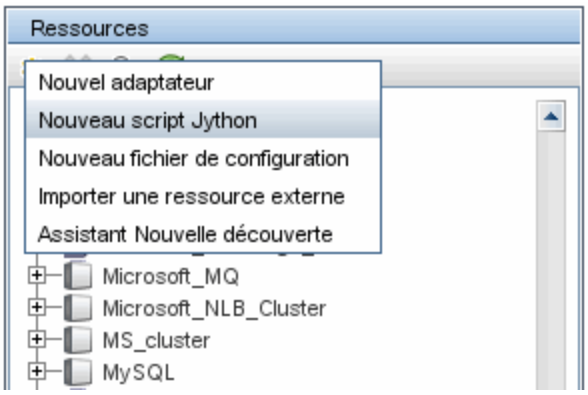
L'implémentation proprement dite de la connexion au système distant, de l'interrogation de ses données et de leur mappage en tant que données CMDB s'effectue au moyen du code Jython. Par exemple, le code contient la logique nécessaire à la connexion à une base de données et à l'extraction des données de celle-ci. Dans ce cas, le code s'attend à recevoir une URL JDBC, un nom d'utilisateur, un mot de passe, un port, etc. Ces paramètres sont spécifiques à chaque instance de la base de données qui répond à la requête TQL. Vous définissez ces variables dans l'adaptateur (dans les données du CI déclencheur). Lorsque le travail s'exécute, ces détails spécifiques sont transmis au code à des fins d'exécution.

L'adaptateur peut faire référence à ce code au moyen d'un nom de classe Java ou d'un nom de script Jython. Dans cette section, nous examinons l'écriture d'un code GFD sous forme de scripts Jython.

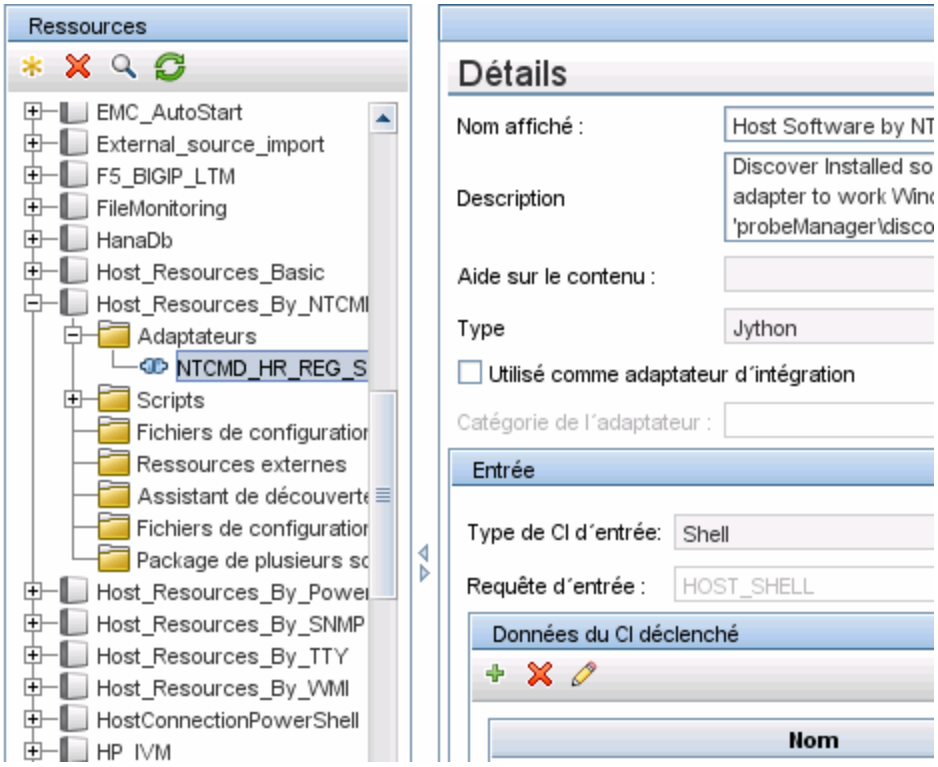
Un adaptateur peut contenir une liste de scripts utilisés lors de l'exécution de la découverte. Lors de la création d'un adaptateur, vous créez généralement un script que vous affectez à l'adaptateur. Un nouveau script comprend des modèles de base, mais vous pouvez utiliser un des autres scripts en tant que modèle en cliquant avec le bouton droit de la souris, puis en sélectionnant **Enregistrer sous** :



Pour plus d'informations sur l'écriture de scripts Jython, voir " Étape 3 : Création d'un code Jython " , page 37. Vous ajoutez des scripts au moyen du volet Ressources :



Les scripts répertoriés s'exécutent l'un après l'autre, dans l'ordre dans lequel ils sont définis dans l'adaptateur :



Remarque : Un script doit être spécifié même s'il est utilisé uniquement en tant que bibliothèque par un autre. Dans ce cas, le script de bibliothèque doit être défini avant le script qui l'utilise. Dans cet exemple, le script `processdbutils.py` est une bibliothèque utilisée par le dernier script `host_processes.py`. Les bibliothèques se distinguent des scripts exécutables standard par l'absence de la fonction `DiscoveryMain()`.

Étape 1 : Création d'un adaptateur

Un adaptateur peut être considéré comme la définition d'une fonction. Cette fonction établit une définition d'entrée, exécute une logique sur l'entrée, définit la sortie et fournit un résultat.

Chaque adaptateur spécifie une entrée et une sortie : l'entrée comme la sortie sont des CI déclencheurs spécifiquement définis dans l'adaptateur. L'adaptateur extrait des données du CI déclencheur d'entrée et les transmet au code sous forme de paramètres. Les données issues de CI associés sont parfois également transmises au code. Pour plus d'informations, voir Fenêtre CI associés dans le *Manuel de gestion des flux de données HP Universal CMDB*. Le code d'un adaptateur est générique, à l'exception de ces paramètres de CI déclencheur d'entrée spécifiques transmis au code.

Pour plus d'informations sur les composants d'entrée, voir Concepts de gestion des flux de données dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Contenu de cette section :

- " Définition d'une entrée d'adaptateur (type de CI déclencheur et requête d'entrée) " , ci-dessous
- " Définition de la sortie de l'adaptateur " , page 31
- " Remplacer les paramètres de l'adaptateur " , page 33
- " Remplacer la sélection de la sonde - Facultatif " , page 34
- " Configurer un classpath pour un processus distant - Facultatif " , page 35

1. **Définition d'une entrée d'adaptateur (type de CI déclencheur et requête d'entrée)**

   Vous pouvez utiliser les composants type de CI déclencheur et requête d'entrée pour définir des CI spécifiques en tant qu'entrée d'adaptateur :

   - Le type de CI déclencheur définit le type de CI utilisé en tant qu'entrée pour l'adaptateur. Par exemple, pour un adaptateur chargé de découvrir des adresses IP, le type de CI d'entrée est Network.

   - La requête d'entrée est une requête modifiable standard. Elle définit la requête en fonction de la base CMDB. La requête d'entrée définit des contraintes supplémentaires qui s'appliquent au type de CI (par exemple, si la tâche requiert un attribut `hostID` ou `application_ip`), et si elle peut définir davantage de données CI, si celles-ci étaient nécessaires à l'adaptateur.

     Si l'adaptateur requiert des informations supplémentaires de la part des CI associés au CI déclencheur, vous pouvez ajouter des nœuds supplémentaires au code TQL d'entrée. Pour plus d'informations, voir Ajout de nœuds de requête et de relations à une requête TQL dans le *Manuel de modélisation HP Universal CMDB*.

   - Les données du CI déclencheur contiennent toutes les informations nécessaires sur ce dernier, ainsi que, le cas échéant, des informations issues des autres nœuds spécifiés dans le code TQL d'entrée. La gestion des flux de données utilise des variables pour extraire des données des CI. Lorsque la tâche est téléchargée vers la sonde, les variables des données du CI déclencheur sont remplacées par les véritables valeurs présentes dans les attributs des vraies instances du CI.

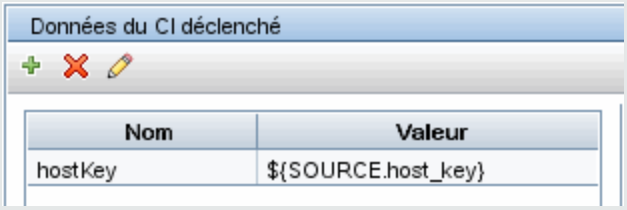
   - Si la valeur des données de destination est une liste, vous pouvez définir le nombre d'éléments de la liste à envoyer à la sonde. Pour cela, ajoutez un signe deux-points après la valeur par défaut, suivi du nombre d'éléments. Entrez deux signes deux-points s'il n'existe aucune valeur par défaut pour les données de destination.

     Par exemple, si vous saisissez les données de destination suivantes : `name=portId, value= ${PHYSICALPORT.root_id:NA:1}` ou `name=portId, value= ${PHYSICALPORT.root_id::1}`, seul le premier port de la liste des ports est envoyé à la sonde.

Exemple de remplacement de variables par des données réelles :

Dans cet exemple, les variables remplacent les données du CI **IpAddress** par des valeurs réelles qui existent sur de vraies instances du CI **IpAddress** dans votre système.

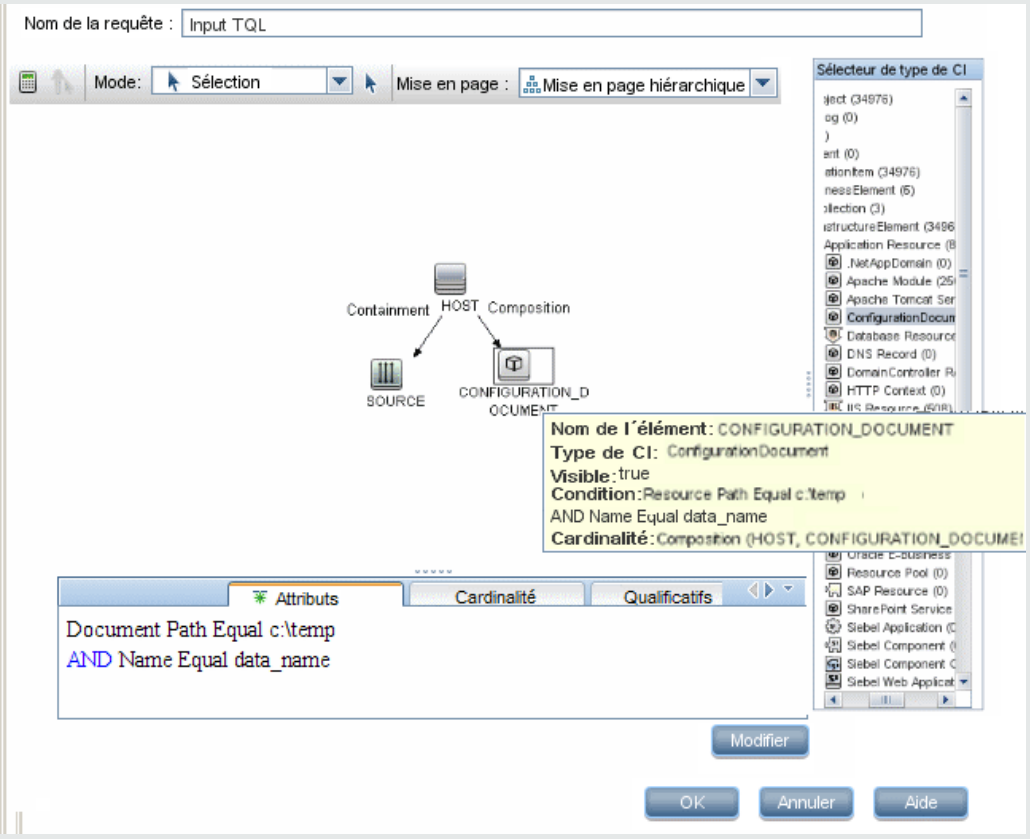
Les données du CI déclenché pour le CI **IpAddress** comprennent une variable `fileName`. Cette variable permet le remplacement du nœud **CONFIGURATION_DOCUMENT** dans le code TQL d'entrée par les valeurs réelles du fichier de configuration installé sur un hôte :



Les données du CI déclencheur sont téléchargées vers la sonde avec toutes les variables remplacées par de vraies valeurs. Le script de l'adaptateur comprend une commande qui permet d'utiliser l'DFM Framework pour extraire les valeurs réelles des variables définies :

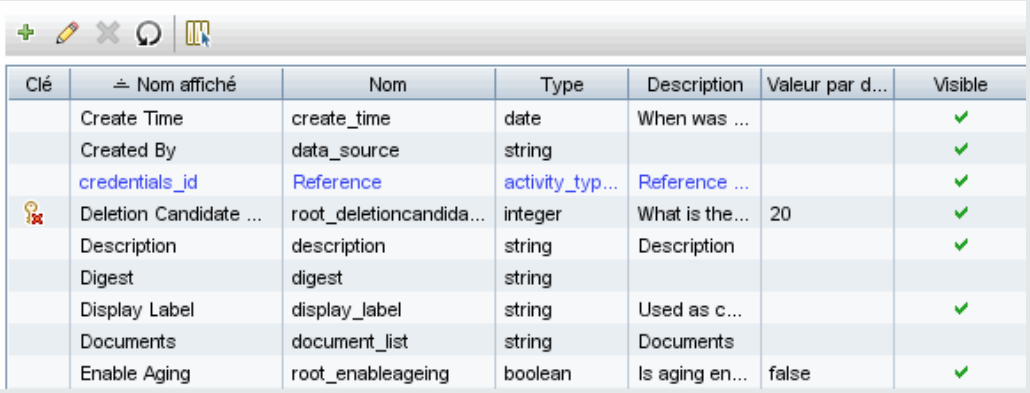
```
Framework.getTriggerCIData ('ip_address')
```

Les variables `fileName` et `path` utilisent les attributs `data_name` et `document_path` du nœud **CONFIGURATION_DOCUMENT** (défini dans l'Éditeur de requêtes d'entrée ; voir exemple précédent).



Cliquez sur la vignette pour afficher l'image en taille réelle.

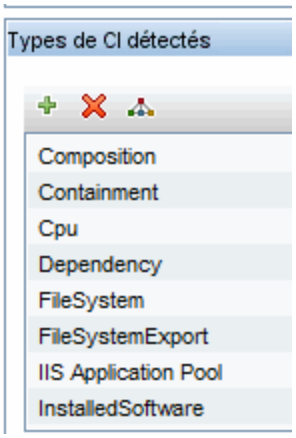
Les variables `Protocol`, `credentialsId` et `ip_address` utilisent les attributs `root_class`, `credentials_id` et `application_ip` :



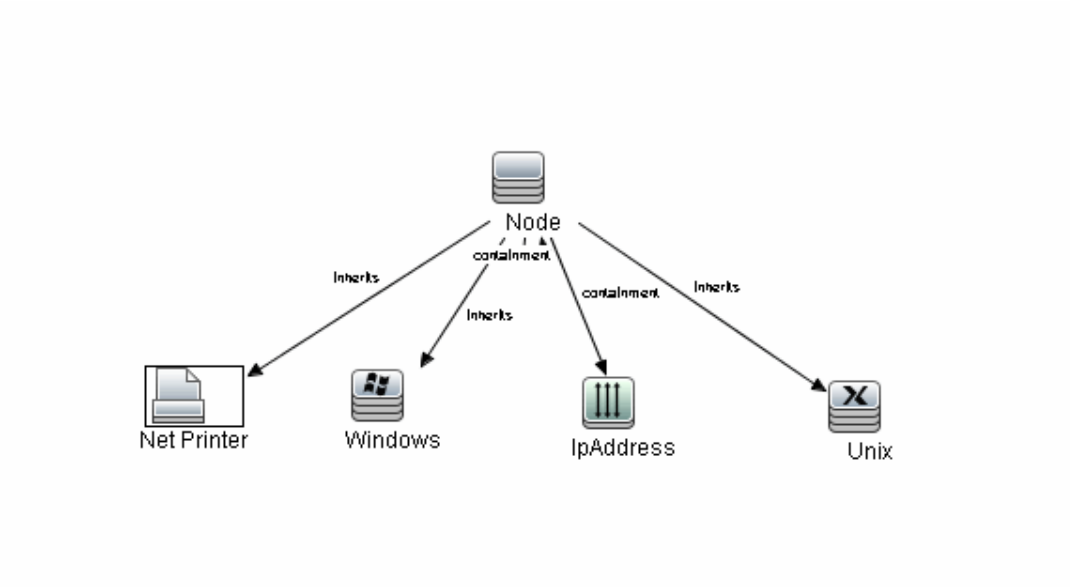
Clé	Nom affiché	Nom	Type	Description	Valeur par d...	Visible
	Create Time	create_time	date	When was ...		✓
	Created By	data_source	string			✓
	credentials_id	Reference	activity_typ...	Reference ...		✓
	Deletion Candidate ...	root_deletioncandida...	integer	What is the...	20	✓
	Description	description	string	Description		✓
	Digest	digest	string			
	Display Label	display_label	string	Used as c...		✓
	Documents	document_list	string	Documents		
	Enable Aging	root_enableageing	boolean	Is aging en...	false	✓

2. **Définition de la sortie de l'adaptateur**

La sortie de l'adaptateur comprend une liste des CI découverts (**Gestion des flux de données > Gestion de l'adaptateur > onglet Définition de l'adaptateur > Types de CI détectés**) et les liens qui les interconnectent :



Vous pouvez également afficher les types de CI sous la forme d'une carte topologique, c'est-à-dire selon une représentation des composants et des liens qui les interconnectent (cliquez sur le bouton **Afficher les types de CI détectés sous forme de carte**) :



Les CI découverts sont renvoyés par le code GFD (c'est-à-dire le script Jython) au format UCMDB`ObjectStateHolderVector`. Pour plus d'informations, voir " Génération de résultats par le script Jython " , page 44.

Exemple de sortie d'adaptateur :

Dans cet exemple, vous spécifiez les types de CI qui feront partie de la sortie du CI IP.

a. Accédez à **Gestion des flux de données > Gestion de l'adaptateur**.

b. Dans le volet Ressources, sélectionnez **Network > Adaptateurs > NSLOOKUP_on_Probe**.

c. Dans l'onglet Définition de l'adaptateur, accédez au volet Types de CI détectés.

d. Les types de CI qui font partie de la sortie de l'adaptateur y sont répertoriés. Ajoutez des types de CI à la liste ou supprimez-en de celle-ci. Pour plus d'informations, voir Onglet Définition de l'adaptateur dans le *Manuel de gestion des flux de données HP Universal CMDB*.

3. **Remplacer les paramètres de l'adaptateur**

Pour configurer un adaptateur pour plusieurs travaux, vous pouvez remplacer ses paramètres. Par exemple, l'adaptateur `SQL_NET_Dis_Connection` est utilisé à la fois par les travaux `MSSQL Connection by SQL` et `Oracle Connection by SQL`.

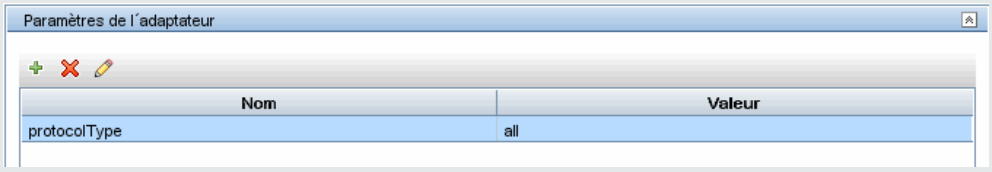
Exemple de remplacement d'un paramètre d'adaptateur :

Cet exemple illustre le remplacement d'un paramètre d'adaptateur afin qu'un même adaptateur puisse découvrir à la fois des bases de données Microsoft SQL Server et Oracle.

a. Accédez à **Gestion des flux de données > Gestion de l'adaptateur**.

b. Dans le volet Ressources, sélectionnez **Database_Basic > Adaptateurs > SQL_NET_Dis_Connection**.

c. Dans l'onglet Définition de l'adaptateur, recherchez le volet **Paramètres de l'adaptateur**. Le paramètre `protocolType` a pour valeur **all** :



d. Cliquez sur l'adaptateur **SQL_NET_Dis_Connection_MsSql** avec le bouton droit de la souris, puis sélectionnez **Aller au travail de découverte> MSSQL Connection by SQL**.

e. Affichez l'onglet **Propriétés**. Accédez au volet Paramètres :

Paramètres

Remplacer	Nom	Valeur
☑	protocolType	MicrosoftSQLServer

La valeur `all` est remplacée par la valeur `MicrosoftSQLServer`.

Remarque : Le travail **Oracle Connection by SQL** comprend le même paramètre mais la valeur est remplacée par une valeur Oracle.

Pour plus d'informations sur l'ajout, la suppression ou la modification de paramètres, voir Onglet Définition de l'adaptateur dans le *Manuel de gestion des flux de données HP Universal CMDB*.

La gestion des flux de données commence à rechercher des instances Microsoft SQL Server conformément à ce paramètre.

4. **Remplacer la sélection de la sonde - Facultatif**

Le serveur UCMDB comporte un mécanisme d'acheminement qui consiste à récupérer les CI déclencheurs reçus de UCMDB et à choisir automatiquement la sonde devant exécuter le travail pour chaque CI déclencheur selon l'une des options suivantes :

- **Pour le type de CI IP address :** choisir la sonde définie pour cette adresse IP.
- **Pour le type de CI running software :** utiliser les attributs **application_ip** et **application_ip_domain** et choisir la sonde définie pour l'adresse IP dans le domaine concerné.
- **Pour tous les autres types de CI :** choisir l'adresse IP du nœud selon le nœud associé du CI (le cas échéant).

La sélection automatique de la sonde est effectuée selon le nœud associé du CI. Après avoir obtenu le nœud associé du CI, le mécanisme d'acheminement choisit une des adresses IP du nœud, puis la sonde en fonction des définitions d'étendue réseau de celle-ci.

Dans les cas suivants, vous devez spécifier la sonde manuellement et faire abstraction du mécanisme d'acheminement automatique :

- Vous savez déjà quelle sonde exécuter pour l'adaptateur et vous n'avez pas besoin du mécanisme d'acheminement automatique pour sélectionner la sonde (par exemple, si le CI déclencheur correspond à la passerelle de la sonde).
- La sélection automatique de la sonde risque d'échouer. Cela peut se produire dans les cas suivants :
  - Un CI déclencheur ne dispose pas d'un nœud associé (tel que le type de CI network).

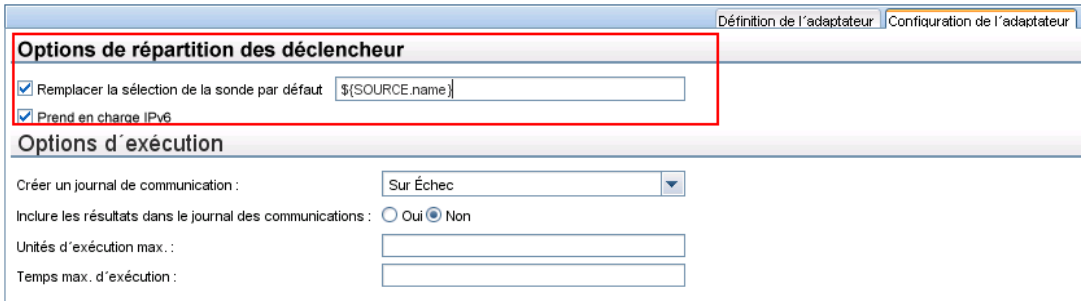
- Le nœud d'un CI déclencheur comporte plusieurs adresses IP appartenant chacune à une sonde différente.

Pour spécifier manuellement la sonde à utiliser avec l'adaptateur :

- Sélectionnez l'adaptateur et cliquez sur l'onglet **Configuration de l'adaptateur**.
- Sous **Options de répartition des déclencheur**, sélectionnez **Remplacer la sélection de la sonde par défaut**.
- Dans la zone de texte, entrez la sonde dans l'un des formats suivants :

Nom de la sonde	Nom de la sonde.
Adresse IP	Adresse IP de la sonde pouvant être définie au format IPv4 ou IPv6.
IP,Domaine	Format IPv4 : **16.59.63.86,DomaineParDéfaut** Format IPv6 : **2001:0:9d46:953c:34a9:1e6b:f2ff:fffe,DomainePersonnalisé**
Nom de domaine	Domaine dans lequel la sonde doit être sélectionnée.

Exemple :



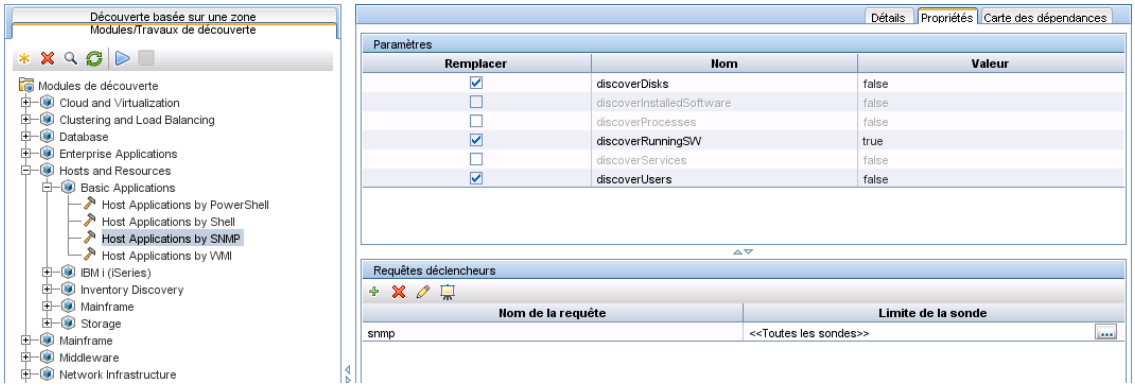
5. **Configurer un classpath pour un processus distant - Facultatif**

   Pour plus d'informations, voir " Configurer l'exécution d'un processus distant " , page 37.

Étape 2 : Affectation d'un travail à l'adaptateur

Chaque adaptateur est associé à un ou plusieurs travaux qui définissent la politique d'exécution. Les travaux permettent de planifier le même adaptateur de différentes manières selon différents jeux de CI déclenchés. Ils permettent également de fournir des paramètres distincts pour chaque jeu.

Les travaux s'affichent dans l'arborescence Modules de découverte. Il s'agit de l'entité que l'utilisateur active, comme illustré ci-après .



Choisir un TQL déclencheur

Chaque travail est associé à des codes TQL déclencheurs. Ces codes TQL déclencheurs publient les résultats utilisés en tant que CI déclencheurs d'entrée pour l'adaptateur du travail concerné.

Un code TQL déclencheur peut ajouter des contraintes à une requête TQL d'entrée. Par exemple, si les résultats d'une requête TQL d'entrée sont des adresses IP connectées à un objet SNMP, les résultats d'un code TQL déclencheur peuvent être des adresses IP connectées à un objet SNMP dans la plage allant de 195.0.0.0 à 195.0.0.10.

Remarque : Un code TQL déclencheur doit référencer les mêmes objets que le code TQL d'entrée. Par exemple, si un code TQL d'entrée recherche des adresses IP qui s'exécutent pour un objet SNMP, vous ne pouvez pas définir un code TQL déclencheur (pour le même travail) en vue de détecter des adresses IP connectées à un hôte. En effet, certaines des adresses IP détectées pourraient ne pas être connectées à un objet SNMP, contrairement aux exigences du code TQL d'entrée.

Définir les informations de planification

Les informations de planification destinées à la sonde spécifient à quel moment le code doit être exécuté sur les CI déclencheurs. Si la case à cocher **Appeler immédiatement les nouveaux CI déclenchés** est activée, le code s'exécute également une fois sur chaque CI déclencheur lorsqu'il atteint la sonde, et ce quels que soient les paramètres de planification ultérieurs.

Pour chaque occurrence planifiée pour chaque travail, la sonde exécute le code sur tous les CI déclencheurs accumulés pour ce travail. Pour plus d'informations, voir Discovery Scheduler Dialog Box dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Remplacer les paramètres de l'adaptateur

Lorsque vous configurez un travail, vous pouvez remplacer les paramètres d'adaptateur. Pour plus d'informations, voir " Remplacer les paramètres de l'adaptateur " , page 33.

Étape 3 : Création d'un code Jython

HP Universal CMDB utilise des scripts Jython pour l'écriture d'adaptateurs. Par exemple, l'adaptateur `SNMP_NET_Dis_Connection` se sert du script `SNMP_Connection.py` pour tenter de se connecter à des machines utilisant SNMP. Jython est un langage basé sur Python et optimisé par Java.

Pour plus d'informations sur l'utilisation de Jython, vous pouvez consulter les sites Web suivants :

- http://www.jython.org
- http://www.python.org

Pour plus d'informations, voir " Création d'un code Jython " , page 39.

Configurer l'exécution d'un processus distant

Vous pouvez exécuter une découverte pour un travail de découverte dans un processus distinct du processus de Data Flow Probe.

Par exemple, vous pouvez exécuter le travail dans un processus distant à part s'il utilise des bibliothèques **.jar** dont la version diffère de celles de la sonde ou qui sont incompatibles avec ces dernières.

Vous pouvez également exécuter le travail dans un processus distant à part s'il est potentiellement très gourmand en mémoire (découverte de nombreuses données) et que vous voulez isoler la sonde de problèmes éventuels liés à une **saturation de la mémoire**.

Pour configurer un travail à exécuter comme processus distant, définissez les paramètres suivants dans le fichier de configuration de son adaptateur :

Paramètre	Description
remoteJVMArgs	Paramètres JVM concernant le processus Java distant.
runInSeparateProcess	Lorsqu'il a la valeur **true**, le travail de découverte s'exécute dans un processus distant.

Paramètre	Description

remoteJVMClasspath	(Facultatif) Permet la personnalisation du classpath du processus distant, remplaçant le classpath de sonde par défaut. Ce paramètre est utile en cas d'éventuelle incompatibilité de version entre les fichiers jar de la sonde et les fichiers jar personnalisés requis pour la découverte définie par le client.
	
	Si le paramètre **remoteJVMClasspath** n'est pas défini, ou s'il est laissé vide, le classpath de sonde par défaut est utilisé.
	
	En cas de développement d'un nouveau travail de découverte, pour vous assurer que la bibliothèque jar de la sonde n'entre pas en conflit avec les bibliothèques jar du travail, vous devez utiliser au moins le classpath minimal requis pour exécuter une découverte de base. Le classpath minimal est défini par le paramètre **basic_discovery_minimal_classpath** dans le fichier **DataFlowProbe.properties**.
	
	Exemples de personnalisation du paramètre **remoteJVMClasspath** :
	
	• Pour ajouter (en préfixe ou suffixe) des fichiers jar personnalisés au classpath de sonde par défaut, il convient de personnaliser le paramètre **remoteJVMClasspath** comme suit :
	
	`custom1.jar;%classpath%;custom2.jar -`
	
	Dans cas, **custom1.jar** est placé avant le classpath de sonde par défaut et **custom2.jar**, après.
	
	• Pour utiliser le classpath minimal, personnalisez le paramètre **remoteJVMClasspath** comme suit :
	
	`custom1.jar;%minimal_classpath%;custom2.jar`

Chapitre 2 : Développement d'adaptateurs Jython

Contenu de ce chapitre :

Référence des API de gestion des flux de données HP 39

Création d'un code Jython 39

Prise en charge des paramètres régionaux dans les adaptateurs Jython 55

Enregistrement du code GFD 63

Bibliothèques et utilitaires Jython 64

Référence des API de gestion des flux de données HP

Pour plus d'informations sur les API disponibles, voir le manuel *HP Universal CMDB Data Flow Management API Reference*. Ces fichiers figurent dans le dossier suivant :

<répertoire d'installation UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_JavaDoc\index.html

Création d'un code Jython

HP Universal CMDB utilise des scripts Jython pour l'écriture d'adaptateurs. Par exemple, l'adaptateur **SNMP_NET_Dis_Connection** emploie le script **SNMP_Connection.py** pour tenter de se connecter à des ordinateurs utilisant SNMP. Jython est un langage basé sur Python et optimisé par Java.

Pour plus d'informations sur l'utilisation de Jython, vous pouvez consulter les sites Web suivants :

- http://www.jython.org
- http://www.python.org

La section suivante décrit l'écriture effective de code Jython à l'intérieur de l'infrastructure de la gestion des flux de données (GFD). Cette section concerne plus particulièrement les points de contact entre le script Jython et l'infrastructure qu'il appelle, et décrit les bibliothèques et utilitaires Jython qui doivent être utilisés dans la mesure du possible.

Remarque :

- Les scripts écrits pour Universal Discovery doivent être compatibles avec Jython version 2.5.3.
- Pour plus d'informations sur les API disponibles, voir le manuel *HP Universal CMDB Data Flow Management API Reference*.

Contenu de cette section :

- " Utilisation de fichiers JAR Java externes dans Jython " , ci-dessous
- " Exécution du code " , ci-dessous
- " Modification de scripts prêts à l'emploi " , ci-dessous
- " Structure du fichier Jython " , page suivante
- " Génération de résultats par le script Jython " , page 44
- " Instance Framework " , page 47
- " Recherche des informations d'identification correctes (pour les adaptateurs de connexion) " , page 50
- " Traitement des exceptions Java " , page 53

Utilisation de fichiers JAR Java externes dans Jython

Lors du développement de nouveaux scripts Jython, des bibliothèques Java externes (fichiers JAR) ou des fichiers exécutables tiers sont parfois nécessaires sous forme d'archives utilitaires Java, d'archives de connexion telles que des fichiers JAR du pilote JDBC Driver ou de fichiers exécutables (par exemple, **nmap.exe** est utilisé pour la découverte sans informations d'identification).

Ces ressources doivent être regroupées dans le package sous le dossier des **ressources externes**. Toute ressource placée dans ce dossier est automatiquement envoyée à une sonde qui se connecte à votre serveur HP Universal CMDB.

Par ailleurs, lorsque la découverte est lancée, les ressources de fichiers JAR sont chargées dans le classpath de Jython, rendant toutes les classes qu'elles contiennent disponibles pour importation et utilisation.

Exécution du code

Lorsqu'un travail est activé, une tâche comportant toutes les informations requises sont téléchargées sur la sonde.

La sonde commence à exécuter le code GFD à l'aide des informations spécifiées dans la tâche.

Le flux de code Jython commence à s'exécuter à partir d'une entrée principale dans le script, exécute le code permettant de découvrir des CI et fournit les résultats d'un vecteur de CI découverts.

Modification de scripts prêts à l'emploi

En cas de modification d'un script prêt à l'emploi, n'y apportez que des changements minimes et placez les méthodes nécessaires dans un script externe. Vous pouvez suivre plus efficacement les

modifications et, lors du passage à une version plus récente de HP Universal CMDB, votre code n'est pas écrasé.

Par exemple, la ligne unique de code suivante dans un script prêt à l'emploi appelle une méthode qui calcule un nom de serveur Web d'une façon propre à une application :

```
serverName = iplanet_cspecific.PlugInProcessing(serverName, transportHN, mam_uti
ls)
```

La logique plus complexe qui décide du mode de calcul de ce nom est contenue dans un script externe :

```
# implement customer specific processing for 'servername' attribute of httpplugi
n
#
def PlugInProcessing(servername, transportHN, mam_utils_handle):
        # support application-specific HTTP plug-in naming
        if servername == "appsrv_instance":
                # servername is supposed to match up with the j2ee server name,
however some groups do strange things with their
                # iPlanet plug-in files. this is the best work-around we could f
ind. this join can't be done with IP address:port
                # because multiple apps on a web server share the same IP:port f
or multiple websphere applications
                logger.debug('httpcontext_webapplicationserver attribute has bee
n changed from [' + servername + '] to [' + transportHN[:5] + '] to facilitate w
ebsphere enrichment')
                servername = transportHN[:5]
        return servername
```

Enregistrez le script externe dans le dossier des ressources externes. Pour plus d'informations, voir Resources Pane dans le *Manuel de gestion des flux de données HP Universal CMDB*. Si vous ajoutez ce script à un package, vous pourrez également l'utiliser pour d'autres travaux. Pour plus d'informations sur l'utilisation du Gestionnaire des packages, voir "Package Manager" dans le *Manuel d'administration HP Universal CMDB*.

Au cours de la mise à niveau, comme la modification que vous apportez à la ligne de code unique est écrasée par la nouvelle version du script prêt à l'emploi, vous devez remplacer cette ligne. Cependant, le script externe n'est pas écrasé.

Structure du fichier Jython

Le fichier Jython se compose de trois parties, dans un ordre précis :

1. Importations
2. Fonction principale - DiscoveryMain
3. Définitions de fonctions (facultatives)

Voici un exemple de script Jython :

```
#   imports section
from appilog.common.system.types import ObjectStateHolder
from appilog.common.system.types.vectors import ObjectStateHolderVector
# Function definition
def foo:
    # do something
#   Main Function
def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()
    ## Write implementation to return new result CIs here...
    return OSHVResult
```

Importations

Les classes Jython sont réparties dans des espaces de noms hiérarchiques. La version 7.0 ou ultérieure, contrairement aux versions précédentes, ne contient pas d'importations implicites ; par conséquent, chaque classe que vous utilisez doit être importée explicitement. (Cette modification a été effectuée pour des raisons de performances et pour permettre une meilleure compréhension du script Jython en ne masquant pas les détails nécessaires.)

- Pour importer un script Jython :

  ```
  import logger
  ```

- Pour importer une classe Java :

  ```
  from appilog.collectors.clients import ClientsConsts
  ```

Fonction principale – DiscoveryMain

Chaque fichier de script exécutable Jython contient une fonction principale : DiscoveryMain.

La fonction `DiscoveryMain` est le point d'entrée principal du script ; c'est la première fonction exécutée. La fonction principale peut appeler d'autres fonctions qui sont définies dans les scripts :

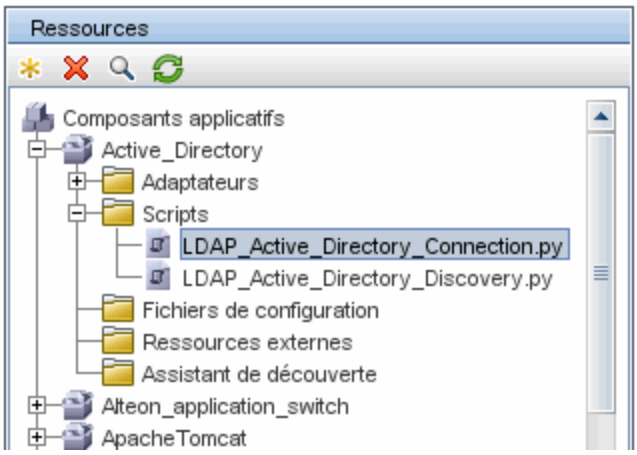
```
def DiscoveryMain(Framework):
```

L'argument Framework doit être spécifié dans la définition de la fonction principale. La fonction utilise cet argument afin d'extraire des informations requises pour exécuter les scripts (par exemple, des informations sur le CI déclencheur et les paramètres) et peut également l'utiliser pour signaler des erreurs survenues au cours de l'exécution d'un script.

Vous pouvez créer un script Jython sans méthode principale. Ce type de script est utilisé comme script de bibliothèque, appelé à partir d'autres scripts.

Définitions de fonctions

Chaque script peut contenir des fonctions supplémentaires appelées à partir du code principal. Chacune de ces fonctions peut en appeler une autre, qui existe dans le script en cours ou dans un autre script (utilisez l'argument `import`). Notez que pour utiliser un autre script, vous devez l'ajouter à la section Scripts du package :



Exemple de fonction appelant une autre fonction :

Dans l'exemple suivant, le code principal appelle la méthode `doQueryOSUsers(..)` qui appelle une méthode interne `doOSUserOSH(..)` :

```
def doOSUserOSH(name):
    sw_obj = ObjectStateHolder('winosuser')

    sw_obj.setAttribute('data_name', name)
    # return the object
    return sw_obj
def doQueryOSUsers(client, OSHVResult):
    _hostObj = modeling.createHostOSH(client.getIpAddress())
    data_name_mib = '1.3.6.1.4.1.77.1.2.25.1.1,1.3.6.1.4.1.77.1.2.25.1.2,str
ing'
    resultSet = client.executeQuery(data_name_mib)
    while resultSet.next():
        UserName = resultSet.getString(2)
        ########## send object ##############
        OSUserOSH = doOSUserOSH(UserName)
        OSUserOSH.setContainer(_hostObj)
        OSHVResult.add(OSUserOSH)
def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()
    try:
        client = Framework.createClient(Framework.getTriggerCIData(BaseClien
t.CREDENTIALS_ID))
```

```
    except:
        Framework.reportError('Connection failed')
    else:
        doQueryOSUsers(client, OSHVResult)
        client.close()
    return OSHVResult
```

Si ce script est une bibliothèque globale concernant de nombreux adaptateurs, vous pouvez l'ajouter à la liste des scripts dans le fichier de configuration `jythonGlobalLibs.xml`, au lieu de l'ajouter à chaque adaptateur (**Gestion de l'adaptateur > volet Ressources > AutoDiscoveryContent > Fichiers de configuration**).

Génération de résultats par le script Jython

Chaque script Jython est exécuté sur un CI déclencheur particulier et se termine par des résultats qui sont renvoyés par la valeur de retour de la fonction `DiscoveryMain`.

Le résultat du script est en fait un groupe de CI et de liens qui doivent être insérés ou mis à jour dans la base CMDB. Le script renvoie ce groupe de CI et de liens au format `ObjectStateHolderVector`.

La classe `ObjectStateHolder` est une façon de représenter un objet ou un lien défini dans la base CMDB. L'objet `ObjectStateHolder` contient le nom du type de CI et une liste d'attributs avec leurs valeurs. `ObjectStateHolderVector` est un vecteur d'instances `ObjectStateHolder`.

Syntaxe ObjectStateHolder

Cette section explique comment générer les résultats GFD dans un modèle UCMDB.

Exemple de définition d'attributs sur les CI :

La classe ObjectStateHolder décrit le graphique de résultat GFD. Chaque CI et lien (relation) est placé dans une instance de la classe `ObjectStateHolder`, comme dans l'exemple de code Jython suivant :
siebel application server 1 appServerOSH = ObjectStateHolder('siebelappserver') 2 appServerOSH.setStringAttribute('data_name', sblsvrName) 3 appServerOSH.setStringAttribute ('application_ip', ip) 4 appServerOSH.setContainer (appServerHostOSH)

- La ligne 1 crée un CI de type **siebelappserver**.
- La ligne 2 crée un attribut appelé **data_name** avec la valeur **sblsvrName** qui est une variable Jython définie avec la valeur découverte pour le nom du serveur.
- La ligne 3 définit un attribut non-clé qui est mis à jour dans la base CMDB.
- La ligne 4 est la création de la relation contenant-contenu (le résultat étant un graphique). Elle indique que ce serveur d'applications est contenu dans un hôte (une autre classe `ObjectStateHolder` dans l'étendue).

Remarque : Chaque CI signalé par le script Jython doit inclure des valeurs pour tous les attributs clés du type de CI.

Exemple de relations (liens) :

L'exemple de lien suivant explique comment le graphique est représenté :
1 linkOSH = ObjectStateHolder('route') 2 linkOSH.setAttribute('link_end1', gatewayOSH) 3 linkOSH.setAttribute('link_end2', appServerOSH)

- La ligne 1 crée le lien (qui est également de la classe `ObjectStateHolder`. La seule différence est que `route` est un type CI de lien).
- Les lignes 2 et 3 définissent les nœuds à l'extrémité de chaque lien. C'est ce que font les attributs **end1** et **end2** du lien qui doivent être spécifiés (parce qu'ils sont les attributs clés minimaux de chaque lien). Les valeurs des attributs sont des instances `ObjectStateHolder`. Pour plus d'informations sur End 1 et End 2, voir " Lien " on page 1 dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Attention : Un lien est directionnel. Vous devez vérifier que les nœuds End 1 et End 2 correspondent à des types de CI valides à chaque extrémité. Si les nœuds ne sont pas valides, la validation de l'objet de résultat échoue et ce dernier n'est pas rapporté correctement. Pour plus d'informations, voir " Relations des types de CI " on page 1 dans le *Manuel de modélisation HP Universal CMDB*.

Exemple de vecteur (regroupement de CI) :

Après avoir créé des objets avec des attributs, et des liens avec des objets à leurs extrémités, vous devez maintenant les regrouper. Pour ce faire, vous les ajoutez à une instance `ObjectStateHolderVector`, comme suit :

```
oshvMyResult = ObjectStateHolderVector()
oshvMyResult.add(appServerOSH)
oshvMyResult.add(linkOSH)
```

Pour plus d'informations sur la transmission de ce résultat composite à Framework afin qu'il puisse être envoyé au serveur CMDB, voir la méthode sendObjects.

Une fois que le graphique de résultat est assemblé dans une instance `ObjectStateHolderVector`, il convient de le renvoyer à l'infrastructure GFD pour l'insérer dans la base CMDB. C'est ce qui est fait en renvoyant l'instance `ObjectStateHolderVector` comme résultat de la fonction `DiscoveryMain()`.

Remarque : Pour plus d'informations sur la création d'**OSH** pour des types de CI courants, voir " modeling.py " , page 66.

Envoi de grandes quantités de données

L'envoi d'une grande quantité de données (généralement supérieure à 20 Ko) est difficile à traiter dans UCMDB. Au-delà de cette quantité, les données doivent être divisées en blocs de plus petite taille avant leur envoi vers UCMDB. Pour insérer correctement tous les blocs dans UCMDB, chacun d'eux doit contenir les informations d'identification requises pour les CI. Il s'agit d'un scénario courant lors du développement d'intégrations Jython. La méthode sendObjects est utilisée pour envoyer les résultats dans des blocs. Si le script Jython envoie un grand nombre de résultats (la valeur par défaut est 20 000, mais elle peut être configurée dans le fichier " Fichier DataFlowProbe.properties " à l'aide de la clé **appilog.agent.local.maxTaskResultSize**), il doit segmenter les résultats en fonction de leur topologie. Cette segmentation doit être effectuée en tenant compte des règles d'identification afin que les résultats soient introduits correctement dans UCMDB. Si le script Jython ne segmente pas les résultats, la sonde tente de les segmenter ; cependant, cette procédure risque de dégrader les performance si le jeu de résultats est de grande taille.

Remarque : La segmentation doit être utilisée pour les adaptateurs d'intégration Jython, mais pas pour les travaux de découverte standard. En effet, comme les travaux de découverte découvrent généralement des informations relatives à un déclencheur spécifique, ils n'envoient donc pas de grandes quantités d'informations. En revanche, avec les intégrations Jython, de grandes quantités de données sont découvertes sur le déclencheur unique de l'intégration.

Il est également possible d'utiliser la segmentation pour des résultats de petite taille. Dans ce cas, il existe une relation entre les CI des différents blocs, et le développeur du script Jython dispose de deux options :

- Envoyer à nouveau l'intégralité du CI et toutes ses informations d'identification dans chaque bloc contenant un lien vers ce CI.

- Utiliser l'ID UCMDB du CI. Pour cela, le script Jython doit attendre que chaque bloc soit traité dans le serveur UCMDB pour obtenir les ID UCMDB. Pour activer ce mode (appelé envoi de résultats en mode synchrone), ajoutez la balise SendJythonResultsSynchronously à l'adaptateur. Cette balise garantit qu'une fois le bloc envoyé, les ID UCMDB des CI inclus dans le bloc ont déjà été reçus par la sonde. Le développeur d'adaptateurs peut utiliser les ID UCMDB pour générer le bloc suivant. Pour ce faire, l'API d'infrastructure getIdMapping doit être utilisée.

Exemple d'utilisation de getIdMapping

Vous envoyez des nœuds dans le premier bloc, et des processus dans le deuxième bloc. Le conteneur racine du processus est un nœud. Au lieu d'envoyer l'intégralité de l'instance objectStateHolder du nœud dans l'attribut root_container du processus, vous pouvez obtenir l'ID UCMDB du nœud à l'aide de l'API getIdMapping et utiliser uniquement l'ID de nœud dans l'attribut root_container du processus pour rédure la taille du bloc.

Instance Framework

L'instance Framework est le seul argument qui est fourni dans la fonction principale du script Jython. Il s'agit d'une interface qui peut être utilisée pour extraire les informations requises pour exécuter le script (par exemple, des informations sur les CI déclencheurs et les paramètres d'adaptateur) et qui sert également à signaler les erreurs survenues au cours de l'exécution du script. Pour plus d'informations, voir " Référence des API de gestion des flux de données HP " , page 39.

L'utilisation correcte de l'instance Framework consiste à la transmettre comme argument à chaque méthode qui l'utilise.

Exemple :

```
def DiscoveryMain(Framework):
         OSHVResult = helperMethod (Framework)
            return OSHVResult
def helperMethod (Framework):
    ....
     probe_name    = Framework.getDestinationAttribute('probe_name')
     ...
     return result
```

Cette section décrit les utilisations les plus importantes de Framework :

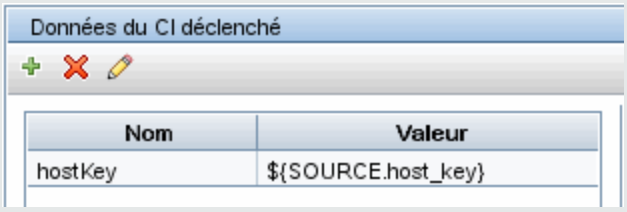
- " Framework.getTriggerCIData(String attributeName) " , page suivante
- " Framework.createClient(credentialsId, props) " , page suivante
- " Framework.getParameter (String parameterName) " , page 49
- " Framework.reportError(String message) et Framework.reportWarning(String message) " , page 50

Framework.getTriggerCIData(String attributeName)

Cette API fournit l'étape intermédiaire entre les données de CI déclencheur définies dans l'adaptateur et le script.

Exemple d'extraction d'informations d'identification :

Vous demandez des informations sur les données de CI déclencheur suivantes :



Pour extraire les informations d'identification de la tâche, utilisez l'API suivante :

```
credId = Framework.getTriggerCIData('credentialsId')
```

Framework.createClient(credentialsId, props)

Vous établissez une connexion à une machine distante en créant un objet client et en exécutant des commandes sur ce client. Pour créer un client, extrayez la classe `ClientFactory`. La méthode getClientFactory() reçoit le type du protocole client demandé. Les constantes du protocole sont définies dans la classe ClientsConsts. Pour plus de détails sur les informations d'identification et les protocoles pris en charge, voir le *Manuel du Content Pack d'intégration et de découverte HP UCMDB*.

Exemple de création d'une instance Client pour l'ID d'informations d'identification :

Pour créer une instance `Client` pour l'ID d'informations d'identification :

```
properties = Properties()
codePage = Framework.getCodePage()
properties.put( BaseAgent.ENCODING, codePage)
client = Framework.createClient(credentailsID ,properties)
```

Vous pouvez maintenant utiliser l'instance `Client` pour vous connecter à la machine ou à l'application concernée.

Exemple de création d'un client WMI et d'exécution d'une requête WMI :

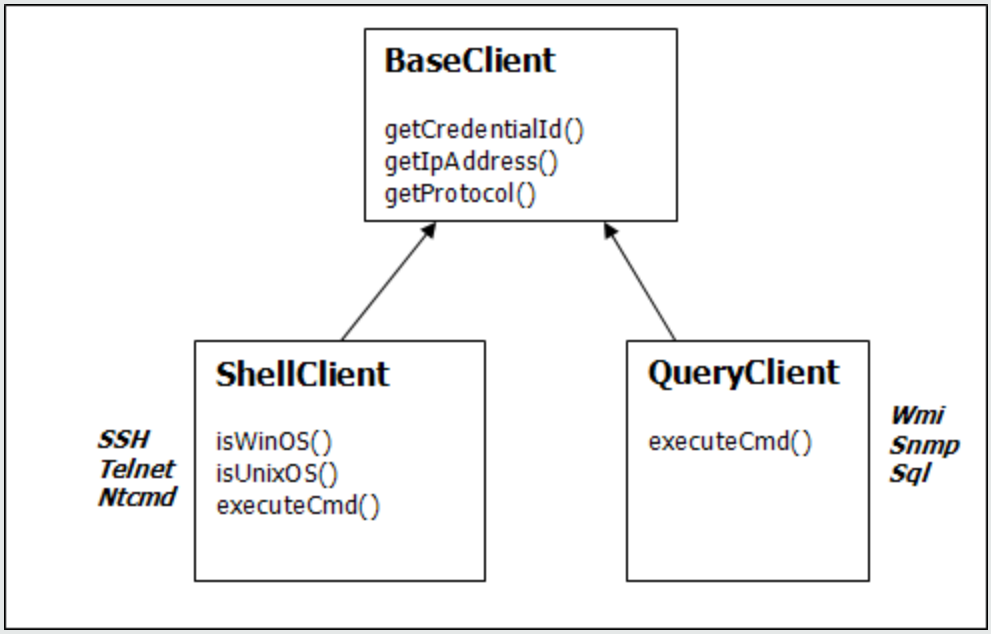
Pour créer un client WMI et exécuter une requête WMI à l'aide de ce client :

```
wmiClient = Framework.createClient(credential)
resultSet = wmiClient. executeQuery("SELECT TotalPhysicalMemory
                                                        FROM    Win32_Logical
MemoryConfiguration")
```

Remarque : Pour que l'API `createClient()` fonctionne, ajoutez le paramètre suivant aux paramètres de données de CI déclencheur : **credentialsId = ${SOURCE.credentials_id}**

dans le volet Données du CI déclenché. Vous pouvez également ajouter manuellement l'ID des informations d'identification en appelant la fonction :
wmiClient = clientFactory().createClient(credentials_id).

Le diagramme suivant illustre la hiérarchie des clients, avec leurs API couramment prises en charge :



Pour plus d'informations sur les clients et leurs API prises en charge, voir BaseClient, ShellClient et QueryClient dans le manuel DFM Framework. Ces fichiers figurent dans le dossier suivant :

<répertoire racine de UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

Framework.getParameter (String parameterName)

En plus d'extraire des informations sur le CI déclencheur, vous devez souvent extraire une valeur de paramètre d'adaptateur. Par exemple :

Paramètres

Remplacer	Nom	Valeur
☑	protocolType	MicrosoftSQLServer

Exemple d'extraction de la valeur du paramètre protocolType :

Pour extraire la valeur du paramètre `protocolType` depuis le script Jython, utilisez l'API suivante :

```
protocolType = Framework.getParameterValue('protocolType')
```

Framework.reportError(String message) et Framework.reportWarning (String message)

Certaines erreurs (par exemple, les échecs de connexion, problèmes de matériel, dépassements de délai) peuvent se produire au cours d'une exécution de script. Lorsque ces erreurs sont détectées, Framework peut signaler le problème. Le message rapporté atteint le serveur et est affiché à l'intention de l'utilisateur.

Exemple de signalement d'erreur et de message :

L'exemple suivant illustre l'utilisation de l'API `reportError(<Error Msg>)` :

```
try:
    client = Framework.createClient(Framework.getTriggerCIData(BaseClient.CR
EDENTIALS_ID))

except:
    strException = str(sys.exc_info()[1]).strip()
    Framework. reportError ('Connection failed: %s' % strException)
```

Vous pouvez utiliser indifféremment l'API `Framework.reportError(String message)` ou `Framework.reportWarning(String message)` pour signaler un problème. La différence entre les deux est que lorsqu'une erreur est signalée, la sonde enregistre dans le système de fichiers un fichier journal de communication avec les paramètres de toute la session. Ainsi, vous pouvez analyser la session pour comprendre l'erreur.

Pour plus d'informations sur les messages d'erreur, voir " Messages d'erreur " , page 68.

Recherche des informations d'identification correctes (pour les adaptateurs de connexion)

Un adaptateur tentant de se connecter à un système distant doit essayer toutes les informations d'identification possibles. L'ID des informations d'identification est l'un des paramètres nécessaires lors de la création d'un client. Le script de connexion obtient un accès aux jeux d'informations d'identification possibles et les essaie l'un après l'autre à l'aide de la méthode `Framework.getAvailableProtocols()`. Lorsque l'un des jeux d'informations d'identification aboutit, l'adaptateur signale un objet de connexion CI sur l'hôte de ce CI déclencheur (avec l'ID d'informations d'identification qui correspond à l'IP) à la base CMDB. Les adaptateurs suivants peuvent utiliser directement ce CI d'objet de connexion pour se connecter au jeu d'informations d'identification (autrement dit, les adaptateurs n'ont pas besoin d'essayer de nouveau toutes les informations d'identification possibles).

Remarque : L'accès aux données sensibles (mots de passe, clés privées, etc.) est bloqué pour les types de protocole suivants :

sshprotocol, ntadminprotocol, as400protocol, vmwareprotocol, wmiprotocol, vcloudprotocol, sapjmxprotocol, webspherepprotocol, siebelgtwyprotocol, sapprotocol, ldapprotocol, udaprotocol, ntcmdprotocol, snmpprotocol, jbossprotocol, telnetprotocol, powershellprotocol, sqlprotocol, weblogicprotocol

Ces types de protocole doivent être utilisés par le biais de clients dédiés.

L'exemple suivant montre comment obtenir toutes les entrées du protocole SNMP. Notez que dans ce cas, l'IP est obtenu à partir des données du CI déclencheur (`# Get the Trigger CI data values`).

Le script de connexion demande toutes les informations d'identification de protocole possibles (`# Go over all the protocol credentials`) et les essaie en boucle jusqu'à ce que l'une d'entre elles aboutisse (`resultVector`). Pour plus de détails, voir l'entrée **paradigme de connexion en deux phases** dans " Séparation d'adaptateurs " , page 24.

Exemple

```
import logger
import netutils
import sys
import errorcodes
import errorobject

# Java imports
from java.util import Properties
from com.hp.ucmdb.discovery.common import CollectorsConstants
from appilog.common.system.types.vectors import ObjectStateHolderVector
from com.hp.ucmdb.discovery.library.clients import ClientsConsts
from com.hp.ucmdb.discovery.library.scope import DomainScopeManager

TRUE = 1
FALSE = 0

def mainFunction(Framework, isClient, ip_address = None):
       _vector = ObjectStateHolderVector()
       errStr = ''
       ip_domain  = Framework.getDestinationAttribute('ip_domain')
       # Get the Trigger CI data values
       ip_address = Framework.getDestinationAttribute('ip_address')

          if (ip_domain == None):
             ip_domain = DomainScopeManager.getDomainByIp(ip_address, None)

          protocols = netutils.getAvailableProtocols(Framework, ClientsConst
          s.SNMP_PROTOCOL_NAME, ip_address, ip_domain)
          if len(protocols) == 0:
              errStr = 'No credentials defined for the triggered ip'
              logger.debug(errStr)
              errObj = errorobject.createError(errorcodes.NO_CREDENTIALS_FOR_
          TRIGGERED_IP, [ClientsConsts.SNMP_PROTOCOL_NAME], errStr)
              return (_vector, errObj)

          connected = 0
          # Go over all the protocol credentials
          for protocol in protocols:
```

```
        client = None
        try:
            try:
                logger.debug('try to get snmp agent for: %s:%s' % (ip_
address, ip_domain))
                if (isClient == TRUE):
                    properties = Properties()
                    properties.setProperty(CollectorsConstants.DESTINA
TION_DATA_IP_ADDRESS, ip_address)
                    properties.setProperty(CollectorsConstants.DESTINA
TION_DATA_IP_DOMAIN, ip_domain)
                    client = Framework.createClient(protocol, properti
es)
                else:
                    properties = Properties()
                    properties.setProperty(CollectorsConstants.DESTINA
TION_DATA_IP_ADDRESS, ip_address)
                    client = Framework.createClient(protocol, properti
es)
                logger.debug('Running test connection queries')
                testConnection(client)
                Framework.saveState(protocol)
                logger.debug('got snmp agent for: %s:%s' % (ip_address,
ip_domain))
                isMultiOid = client.supportMultiOid()
                logger.debug('snmp server isMultiOid state=%s' %isMult
iOid)

             client.close()
             client = None

            except:
                if client != None:
                    client.close()
                    client = None
                logger.debugException('Unexpected SNMP_AGENT Exceptio
n:')
                lastExceptionStr = str(sys.exc_info()[1]).strip()
        finally:
            if client != None:
                client.close()
                client = None

    return (_vector, error)
```

Traitement des exceptions Java

Certaines classes Java lèvent une exception en cas d'échec. Il est recommandé d'intercepter l'exception et de la traiter, sinon elle provoque un arrêt inattendu de l'adaptateur.

Lors de l'interception d'une exception connue, il est recommandé dans la plupart des cas d'imprimer l'arborescence des appels de procédure dans le journal et d'émettre un message approprié pour l'interface utilisateur.

> **Remarque :** Il est très important d'importer la classe d'exception de base Java, comme indiqué dans l'exemple suivant, en raison de la présence de la classe d'exception de base portant le même nom dans Python.

```
from java.lang import Exception as JException
try:
       client = Framework.createClient(Framework.getTriggerCIData(BaseClient.CREDENTIA
LS_ID))
except JException, ex:
       # process java exceptions only
       Framework.reportError('Connection failed')
       logger.debugException(str(ex))
        return
```

Si l'exception n'est pas fatale et que le script peut continuer, vous devez omettre l'appel de la méthode `reportError()` et permettre au script de se poursuivre.

Résolution des problèmes de migration de Jython à partir de la version 2.1 vers la version 2.5.3

Universal Discovery utilise à présent la version 2.5.3 de Jython. Tous les scripts prêts à l'emploi ont été migrés correctement. Si vous avez développé vos propres scripts Jython utilisés par Discovery avant cette mise à niveau, vous risquez de rencontrer les problèmes décrits ci-dessous avec la correction à apporter.

> **Remarque :** Vous devez être un développeur Jython expérimenté pour effectuer ces modifications.

Formatage de chaîne

- **Message d'erreur :** `TypeError: int argument required`
- **Cause possible :** formatage appliqué à un nombre entier décimal d'une variable de chaîne contenant des données de type integer.
- **Code Jython 2.1 problématique :**

```
variable = "43"
print "%d" % variable
```

- **Code Jython 2.5.3 correct :**

```
variable = "43"
print "%s" % variable
```

ou

```
variable = "43"
print "%d" % int(variable)
```

Vérification du type de chaîne

Le code suivant risque de ne pas fonctionner correctement si l'entrée contient des chaînes unicode :

- **Code Jython 2.1 problématique :** `isinstance(unicodeStringVariable,'')`
- **Code Jython 2.5.3 correct :** `isinstance(unicodeStringVariable,basestring)`

  La comparaison doit être effectuée avec `basestring` pour tester si un objet est une instance de `str` ou unicode.

Caractère non-ASCII dans un fichier

- **Message d'erreur :**
  `SyntaxError: Non-ASCII character in file 'x', , but no encoding declared; Pour plus d'informations, voir http://www.python.org/peps/pep-0263.html`
- **Code Jython 2.5.3 correct :** (à ajouter dans la première ligne du fichier)

```
# coding: utf-8
```

Importation de sous-packages

- **Message d'erreur :**
  `AttributeError: 'module' object has no attribute 'sub_package_name'`
- **Cause possible :** importation d'un sous-package sans indication explicite du nom dans l'instruction d'importation.
- **Code Jython 2.1 problématique :**

```
import a
print dir(a.b)
```

Le sous-package n'est pas explicitement importé.

- **Code Jython 2.5.3 correct :**

  ```
  import a.b
  ```

  ou

  ```
  from a import b
  ```

Changement d'itérateur

À partir de la version Jython 2.2, la méthode **__iter__** est utilisée pour effectuer une boucle sur une collection dans le champ d'application d'un bloc **for-in**. L'itérateur doit implémenter la méthode **next** renvoyant un élément approprié ou lever l'erreur **StopIteration** s'il a atteint la fin de la collection. Si la méthode **__iter__** n'est pas implémentée, elle est remplacée par la méthode **getitem**.

Levage d'exceptions

- **La méthode de levage d'exceptions de Jython 2.1 est obsolète :**
  ```
  raise Exception, 'Failed getting contents of file'
  ```

- **Méthode de Jython 2.5.3 recommandée pour le levage d'exceptions :**
  ```
  raise Exception('Failed getting contents of file')
  ```

Prise en charge des paramètres régionaux dans les adaptateurs Jython

La fonction de paramètres régionaux multilingues permet à la gestion des flux de données (GFD) de fonctionner sur des systèmes d'exploitation (SE) en différentes langues et d'activer les personnalisations appropriées lors de l'exécution.

Contenu de cette section :

- " Ajout de la prise en charge d'une nouvelle langue " , ci-dessous
- " Changement de la langue par défaut " , page 57
- " Détermination du jeu de caractères pour le codage " , page 57
- " Définition d'un nouveau travail fonctionnant avec des données localisées " , page 58
- " Décodage de commandes sans mot-clé " , page 59
- " Utilisation de groupes de ressources " , page 59
- " Référence des API " , page 60

Ajout de la prise en charge d'une nouvelle langue

Cette tâche décrit comment ajouter la prise en charge d'une nouvelle langue.

Cette tâche comprend les étapes suivantes :

- "Add a Resource Bundle (*.properties Files)"
- " Déclaration et enregistrement de l'objet Language " , ci-dessous

1. **Ajout d'un groupe de ressources (fichiers *.properties)**

   Ajoutez un groupe de ressources en fonction du travail à exécuter. Le tableau suivant répertorie les travaux GFD et le groupe de ressources qui est utilisé pour chaque travail :

Travail	Nom de base du groupe de ressources
Moniteur de fichiers par shell	langFileMonitoring
Ressources et applications hôtes par shell	langHost_Resources_By_TTY, langTCP
Hôtes par shell par l'intermédiaire de NSLOOKUP dans le serveur DNS	langNetwork
Connexion hôte par shell	langNetwork
Collecte de données réseau par shell ou SNMP	langTCP
Ressources et applications hôtes par SNMP	langTCP
Connexion Microsoft Exchange par NTCMD, Topologie Microsoft Exchange par NTCMD	msExchange
Cluster MS par NTCMD	langMsCluster

   Pour plus d'informations sur les paquets, voir " Utilisation de groupes de ressources " , page 59.

2. **Déclaration et enregistrement de l'objet Language**

   Pour définir une nouvelle langue, ajoutez les deux lignes suivantes de code au script **shellutils.py**, qui contient actuellement la liste de toutes les langues prises en charge. Le script est inclus dans le package `AutoDiscoveryContent`. Pour visualiser le script, accédez à la fenêtre Gestion de l'adaptateur. Pour plus d'informations, voir Adapter Management Window dans le *Manuel de gestion des flux de données HP Universal CMDB*.

   a. Déclarez la langue comme suit :

   ```
   LANG_RUSSIAN = Language(LOCALE_RUSSIAN, 'rus', ('Cp866', 'Cp1251'), (104
   9,), 866)
   ```

   Pour plus d'informations sur la langue des classes, voir " Référence des API " , page 60. Pour plus d'informations sur l'objet Class Locale, voir http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html. Vous pouvez utiliser des paramètres régionaux existants ou en définir de nouveaux.

b. Enregistrez la langue en l'ajoutant à la collection suivante :

```
LANGUAGES = (LANG_ENGLISH, LANG_GERMAN, LANG_SPANISH, LANG_RUSSIAN, LANG_
JAPANESE)
```

Changement de la langue par défaut

S'il est impossible de déterminer la langue du SE, la langue par défaut est utilisée. La langue par défaut est spécifiée dans le fichier **shellutils.py**.

```
#default language for fallback
DEFAULT_LANGUAGE = LANG_ENGLISH
```

Pour changer de langue par défaut, vous initialisez la variable DEFAULT_LANGUAGE avec une autre langue. Pour plus d'informations, voir " Ajout de la prise en charge d'une nouvelle langue " , page 55.

Détermination du jeu de caractères pour le codage

Le jeu de caractères adapté pour le décodage des résultats de commande est déterminé lors de l'exécution. La solution multilingue est basée sur les faits et hypothèses suivants :

1. Il est possible de déterminer la langue du SE indépendamment des paramètres régionaux, par exemple en exécutant la commande **chcp** sous Windows ou la commande **locale** sous Linux.

2. Le codage de langue de relations est bien connu et peut être défini statiquement. Ainsi, la langue russe possède deux des codages les plus répandus : `Cp866` et `Windows-1251`.

3. Il est préférable d'utiliser un jeu de caractères par langue ; par exemple, le jeu de caractères préférable pour le russe est `Cp866`. Cela signifie que la plupart des commandes produisent des sorties dans ce codage.

4. Le codage dans lequel est générée la sortie de la commande suivante est imprévisible, mais c'est l'un des codages possibles pour une langue donnée. Par exemple, sur une machine Windows avec des paramètres régionaux russes, le système fournit la sortie de commande **ver** dans le codage Cp866, tandis que la commande **ipconfig** est fournie dans Windows-1251.

5. Une commande connue produit des mots-clés connus dans sa sortie. Ainsi, la commande **ipconfig** contient la forme traduite de la chaîne **IP-Address**. Par conséquent, la sortie de la commande **ipconfig** contient **IP-Address** pour le SE anglais, IP-Адрес pour le SE russe, **IP-Adresse** pour le SE allemand, etc.

Une fois que la langue de génération de la sortie de commande est découverte (# 1), les jeux de caractères possibles sont limités à un ou deux (# 2). De plus, les mots-clés contenus dans cette sortie sont également connus (# 5).

La solution est donc de décoder la sortie de commande avec l'un des codages possibles en recherchant un mot-clé dans le résultat. Si le mot-clé est trouvé, le jeu de caractères actuel est considéré comme le bon.

Définition d'un nouveau travail fonctionnant avec des données localisées

Cette tâche décrit comment écrire un nouveau travail qui peut fonctionner avec les données localisées.

Les scripts Jython exécutent généralement des commandes et analysent leur sortie. Pour recevoir cette sortie de commande correctement décodée, utilisez l'API pour la classe **ShellUtils**. Pour plus d'informations, voir " Présentation de l'API de service Web HP Universal CMDB " , page 275.

Ce code prend généralement la forme suivante :

```
client = Framework.createClient(protocol, properties)
shellUtils = shellutils.ShellUtils(client)
languageBundle = shellutils.getLanguageBundle ('langNetwork', shellUtils.osLangu
age, Framework)
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_addres
s')
ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig /all', strWindowsI
PAddress)
#Do work with output here
```

1. Créez un client :

   ```
   client = Framework.createClient(protocol, properties)
   ```

2. Créez une instance de la classe **ShellUtils** et ajoutez-y la langue du système d'exploitation. Si la langue n'est pas ajoutée, la langue par défaut est utilisée (généralement l'anglais) :

   ```
   shellUtils = shellutils.ShellUtils(client)
   ```

   Au cours de l'initialisation des objets, la gestion des flux de données détecte automatiquement la langue de la machine et définit le codage préférable à partir de l'objet `Language` prédéfini. Le codage préférable est la première instance qui apparaît dans la liste de codage.

3. Extrayez le groupe de ressources approprié de **shellclient** en utilisant la méthode **getLanguageBundle** :

   ```
   languageBundle = shellutils.getLanguageBundle ('langNetwork', shellUtils.osL
   anguage, Framework)
   ```

4. Extrayez du groupe de ressources un mot-clé qui convient à une commande particulière :

   ```
   strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_addr
   ess')
   ```

5. Appelez la méthode **executeCommandAndDecode** et transmettez-lui le mot-clé sur l'objet **ShellUtils** :

```
ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig /all', strWind
owsIPAddress)
```

L'ShellUtils object est également nécessaire pour renvoyer l'utilisateur au document de référence des API (où cette méthode est décrite de manière détaillée).

6. Faites une analyse syntaxique de la sortie comme d'habitude.

Décodage de commandes sans mot-clé

L'approche actuelle en matière de localisation consiste à faire appel à un mot-clé pour décoder toutes les sorties de commande. Pour plus d'informations, voir l'étape relative à l'extraction d'un mot-clé du groupe de ressources à la section " Définition d'un nouveau travail fonctionnant avec des données localisées " , page précédente.

Cependant, une autre approche consiste à utiliser un mot-clé pour décoder uniquement la première sortie de commande, puis à décoder les commandes suivantes avec le jeu de caractères utilisé pour décoder la première commande. Pour ce faire, vous vous servez des méthodes **getCharsetName** et **useCharset** de l'objet **ShellUtils**.

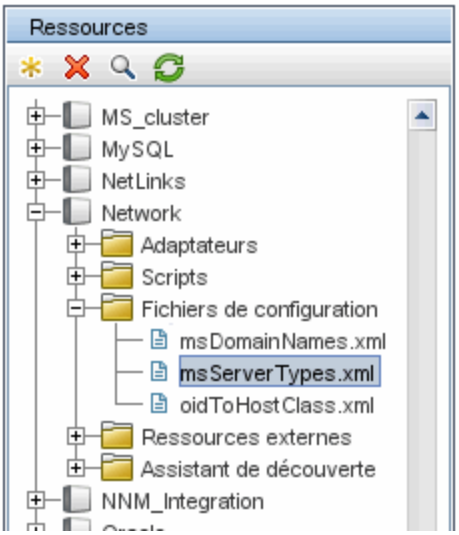
Le cas d'utilisation standard fonctionne comme suit :

1. Appelez la méthode **executeCommandAndDecode** une fois.
2. Obtenez le nom du jeu de caractères utilisé en dernier, par le biais de la méthode **getCharsetName**.
3. Faites en sorte que **shellUtils** utilise ce jeu de caractères par défaut, en appelant la méthode **useCharset** sur l'objet **ShellUtils**.
4. Appelez la méthode **execCmd** de **ShellUtils** une ou plusieurs fois. La sortie est renvoyée avec le jeu de caractères spécifié à l'étape précédente. Aucune autre opération de décodage n'est effectuée.

Utilisation de groupes de ressources

Un groupe de ressources est un fichier doté d'une extension properties (***.properties**). Un fichier de propriétés peut être considéré comme un dictionnaire qui stocke les données sous la forme `clé = valeur`. Chaque ligne d'un fichier de propriétés contient une association `clé = valeur`. La principale fonction d'un groupe de ressources est de renvoyer une valeur en fonction de sa clé.

Les groupes de ressources se trouvent sur la machine de la sonde : **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles**. Ils sont téléchargés à partir du serveur UCMDB comme tous les autres fichiers de configuration. Ils peuvent être modifiés, ajoutés ou supprimés dans la fenêtre Ressources. Pour plus d'informations, voir Configuration File Pane dans le *Manuel de gestion des flux de données HP Universal CMDB*.



Lors de la découverte d'une destination, la gestion des flux de données doit généralement réaliser une analyse syntaxique de la sortie de commande ou du contenu du fichier. Cette analyse est souvent basée sur une expression régulière. Différentes langues exigent l'utilisation d'expressions régulières différentes pour l'analyse syntaxique. Pour écrire le code une seule fois pour toutes les langues, il convient d'extraire toutes les données de langue dans des groupes de ressources. Il existe un groupe de ressources pour chaque langue. (Bien qu'un groupe de ressources puisse contenir des données de langues différentes, dans la gestion des flux de données, chaque groupe de ressources contient les données d'une seule langue.)

Le script Jython lui-même ne comprend pas de données de langue figées dans le code (par exemple, des expressions régulières propres à une langue). Le script détermine la langue du système distant, charge le groupe de ressources approprié et obtient toutes les données de langue selon une clé spécifique.

Dans la gestion des flux de données, les groupes de ressources revêtent un format de nom particulier : `<nom_base>_<identifiant_langue>.properties`, par exemple, `langNetwork_spa.properties`. (Le groupe de ressources par défaut a le format suivant : `<nom_base>.properties`, par exemple, `langNetwork.properties`.)

Le format `nom_base` reflète la fonction prévue de ce groupe. Par exemple, **langMsCluster** signifie que le groupe de ressources contient des ressources de langue utilisées par les travaux du cluster MS.

Le format de l'`identifiant_langue` est un acronyme de 3 lettres utilisé pour identifier la langue. Par exemple, `rus` désigne le russe et `ger` l'allemand. Cet identifiant de langue est inclus dans la déclaration de l'objet `Language`.

Référence des API

Contenu de cette section :

- " Classe Language " , ci-dessous
- " Méthode executeCommandAndDecode " , ci-dessous
- " Méthode getCharsetName " , page suivante
- " Méthode useCharset " , page suivante
- " Méthode getLanguageBundle " , page suivante
- " Champ osLanguage " , page suivante

Classe Language

Cette classe encapsule des informations sur la langue, telles que le suffixe du groupe de ressources, le codage possible, etc.

Champs

Nom	Description
locale	Objet Java qui représente les paramètres régionaux.
bundlePostfix	Suffixe du groupe de ressources. Ce suffixe est utilisé dans les noms de fichier du groupe de ressources pour identifier la langue. Par exemple, le groupe **langNetwork_ger.properties** comprend un suffixe de groupe **ger**.
charsets	Jeux de caractères utilisés pour coder cette langue. Chaque langue peut comporter plusieurs jeux de caractères. Par exemple, le russe est couramment codé avec les codages `Cp866` et `Windows-1251`.
wmiCodes	Liste des codes WMI utilisés par le SE Microsoft Windows pour identifier la langue. Tous les codes possibles sont répertoriés dans http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (section OSLanguage). L'une des méthodes permettant d'identifier la langue du SE consiste à interroger le SE de la classe WMI pour la propriété `OSLanguage`.
codepage	Page de codes utilisée avec une langue particulière. Par exemple, `866` est utilisé pour les machines russes et `437` pour les machines anglaises. L'une des méthodes permettant d'identifier la langue du SE consiste à extraire sa page de codes par défaut (par exemple, par la commande `chcp`).

Méthode executeCommandAndDecode

Cette méthode est conçue pour être utilisée par les scripts Jython de logique métier. Elle encapsule l'opération de décodage et renvoie une sortie de commande décodée.

Arguments

Nom	Description
cmd	Commande effective à exécuter.
keyword	Mot-clé à utiliser pour l'opération de décodage.
framework	Objet Framework transmis à tous les scripts Jython exécutables dans la gestion des flux de données.
timeout	Délai d'attente de la commande.
waitForTimeout	Indique si le client doit attendre lorsque le délai d'attente est dépassé.
useSudo	Indique si `sudo` doit être utilisé (ne concerne que les clients UNIX).
language	Permet de spécifier la langue directement au lieu de la détecter automatiquement.

Méthode getCharsetName

Cette méthode renvoie le nom du jeu de caractères utilisé en dernier.

Méthode useCharset

Cette méthode définit le jeu de caractères sur l'instance `ShellUtils`, qui utilise ce jeu de caractères pour le décodage de données initial.

Arguments

Nom	Description
charsetName	Nom du jeu de caractères, par exemple, `windows-1251` ou `UTF-8`.

Voir aussi " Méthode getCharsetName " , ci dessus.

Méthode getLanguageBundle

Cette méthode doit être utilisée pour obtenir le groupe de ressources correct. Elle remplace l'API suivante :

```
Framework.getEnvironmentInformation().getBundle(...)
```

Arguments

Nom	Description
baseName	Nom du groupe sans le suffixe de langue, par exemple `langNetwork`.
language	Objet de langue. `ShellUtils.osLanguage` doit être transmis ici.
framework	Objet commun Framework qui est transmis à tous les scripts Jython exécutables dans la gestion des flux de données.

Champ osLanguage

Ce champ contient un objet qui représente la langue.

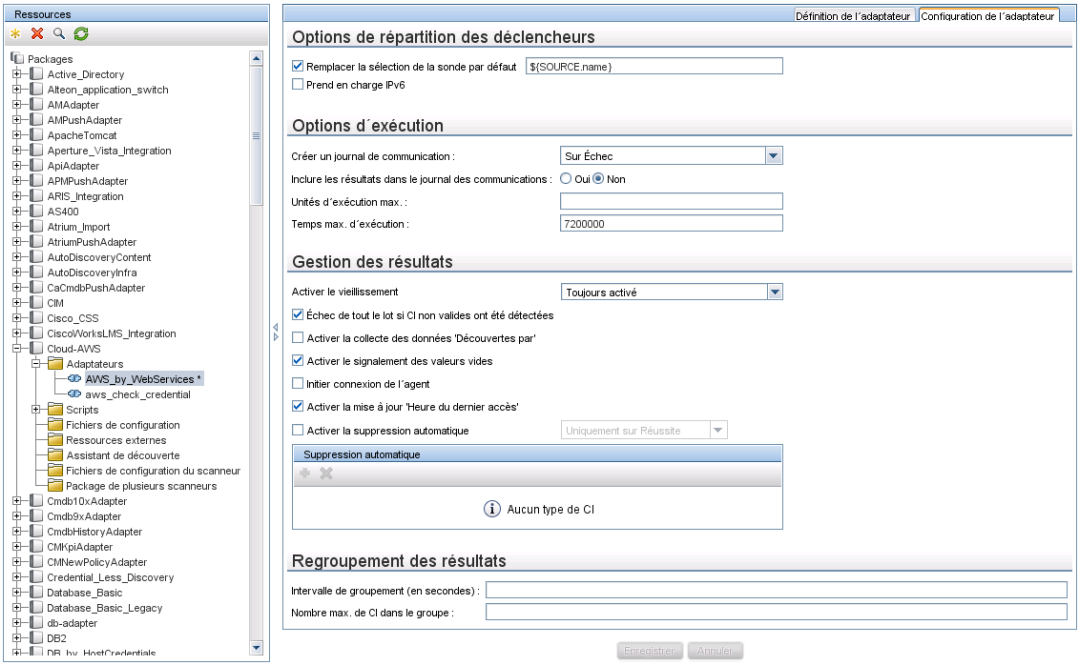
Enregistrement du code GFD

Il peut être très utile d'enregistrer une exécution entière, avec tous les paramètres, par exemple lors du débogage et du test d'un code. Cette tâche décrit comment enregistrer une exécution entière avec toutes les variables pertinentes. De plus, vous pouvez consulter des informations de débogage supplémentaires qui ne sont généralement pas imprimées dans les fichiers journaux, même au niveau du débogage.

Pour enregistrer du code GFD :

1. Accédez à **Gestion des flux de données > Universal Discovery**. Cliquez avec le bouton droit de la souris sur le travail dont l'exécution doit être consignée et sélectionnez **Aller à l'adaptateur** pour ouvrir l'application Gestion de l'adaptateur.

2. Repérez le volet **Options d'exécution** dans l'onglet **Configuration de l'adaptateur** (voir ci-dessous).



3. Dans la zone **Créer un journal de communication**, sélectionnez **Toujours**. Pour plus d'informations sur la définition des options de journalisation, voir "Execution Options Pane" dans le manuel *Manuel de gestion des flux de données HP Universal CMDB*.

   L'exemple suivant représente le fichier journal XML créé lors de l'exécution du travail **Host Connection by Shell** (Connexion hôte par shell), avec la valeur **Toujours** ou **Sur échec** attribuée à la zone **Créer un journal de communication** :

Nom du travail Données du CI déclencheur

```
- <execution jobId="Host Connection by Shell" destinationid="0e9787433d65e4a68839bfa8b224c92d">
  - <destination>
      <destinationData name="ip_domain">DefaultDomain</destinationData>
      <destinationData name="hostId" />
      <destinationData name="ip_address">16.59.63.34</destinationData>
      <destinationData name="id">0e9787433d65e4a68839bfa8b224c92d</destinationData>
    </destination>
```

L'exemple suivant contient les paramètres message et stacktrace :

Stacktrace

```
- <exec start="18:41:55" duration="2062" type="ssh" credentialsId="f464999bdfe5a1e1407b479b6f730d5b">
    <cmd>[CDATA: client_connect]</cmd>
    <result IS_NULL="Y" />
  - <error class="com.hp.ucmdb.discovery.probe.services.dynamic.agents.SSHAgentException">
      <message>[CDATA: Failed to connect: Error connecting: Connection refused: connect]</message>
    - <stacktrace>
        <frame class="com.hp.ucmdb.discovery.probe.services.dynamic.agents.SSHAgent" method="connect" file
        <frame class="com.hp.ucmdb.discovery.probe.clients.shell.SSHClient" method="createWrapper" file="SSH
        <frame class="com.hp.ucmdb.discovery.probe.clients.BaseClient" method="initPrivate" file="BaseClient.ja
```

Bibliothèques et utilitaires Jython

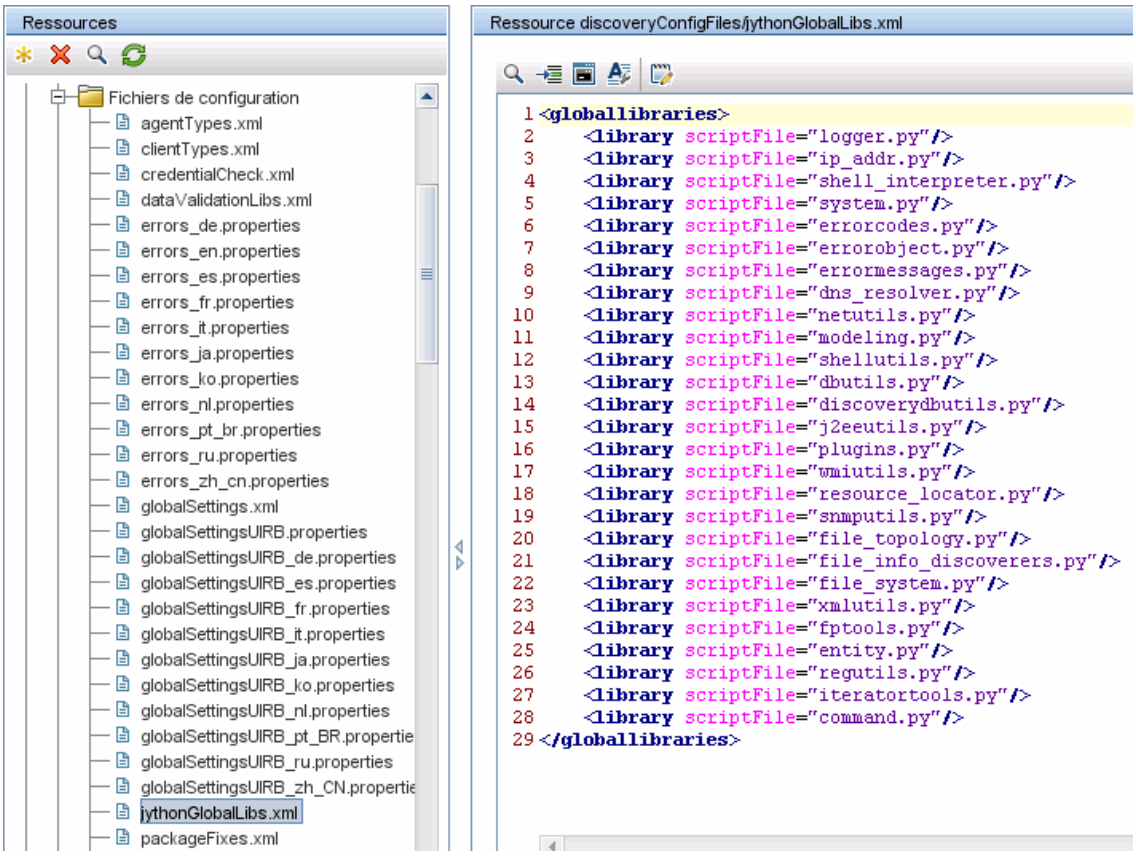
Plusieur scripts utilitaires sont couramment employés dans les adaptateurs. Ces scripts font partie du package `AutoDiscovery` et résident sous : **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** avec les autres scripts qui sont téléchargés sur la sonde.

> **Remarque :** Le dossier `discoveryScript` est créé dynamiquement lorsque la sonde commence à fonctionner.

Pour utiliser l'un des scripts utilitaires, ajoutez la ligne d'importation suivante dans la section d'importation du script :

```
import <nom du script>
```

La bibliothèque Python AutoDiscovery contient des scripts utilitaires Jython. Ces scripts de bibliothèque sont considérés comme la bibliothèque externe de la gestion des flux de données. Ils sont définis dans le fichier `jythonGlobalLibs.xml` (situé dans le dossier **Fichiers de configuration**).



Chaque script qui figure dans le fichier `jythonGlobalLibs.xml` étant chargé par défaut au démarrage de la sonde, il n'est pas nécessaire de les utiliser explicitement dans la définition d'adaptateur.

Contenu de cette section :

- " logger.py " , ci-dessous
- " modeling.py " , page suivante
- " netutils.py " , page suivante
- " shellutils.py " , page 67

logger.py

Le script **logger.py** contient des utilitaires de consignation et des fonctions d'assistance pour le signalement d'erreurs. Vous pouvez appeler ses API de débogage, d'information et d'erreur pour écrire dans les fichiers journaux. Les messages de journal sont enregistrés dans **C:\hp\UCMDB\DataFlowProbe\runtime\log**.

Les messages sont entrés dans le fichier journal en fonction du niveau de débogage défini pour l'appender PATTERNS_DEBUG dans le fichier **C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties**. (Par défaut, le niveau est `DEBUG`.) Pour plus d'informations, voir " Niveaux de gravité des erreurs " , page 71.

```
#####################################################################
################         PATTERNS_DEBUG log                 ###############
#####
#####################################################################
log4j.category.PATTERNS_DEBUG=DEBUG, PATTERNS_DEBUG
log4j.appender.PATTERNS_DEBUG=org.apache.log4j.RollingFileAppender
log4j.appender.PATTERNS_DEBUG.File=C:\hp\UCMDB\DataFlowProbe\runtime\log/pro
beMgr-patternsDebug.log
log4j.appender.PATTERNS_DEBUG.Append=true
log4j.appender.PATTERNS_DEBUG.MaxFileSize=15MB
log4j.appender.PATTERNS_DEBUG.Threshold=DEBUG
log4j.appender.PATTERNS_DEBUG.MaxBackupIndex=10
log4j.appender.PATTERNS_DEBUG.layout=org.apache.log4j.PatternLayout
log4j.appender.PATTERNS_DEBUG.layout.ConversionPattern=<%d> [%-5p] [%t] - %m
%n
log4j.appender.PATTERNS_DEBUG.encoding=UTF-8
```

Les messages d'information et d'erreur apparaissent également dans la console d'invite de commande.

On distingue deux jeux d'API :

- `logger.<debug/info/warn/error>`
- `logger.<debugException/infoException/warnException/errorException>`

Le premier jeu sort la concaténation de tous ses arguments de type chaîne au niveau de consignation approprié, tandis que le second sort cette concaténation ainsi que l'arborescence des appels de procédure de la dernière exception levée, afin de fournir plus d'informations. Exemple :

```
logger.debug('found the result')
logger.errorException('Error in discovery')
```

modeling.py

Le script **modeling.py** contient des API pour la création d'hôtes, d'IP, de CI de processus, etc. Ces API permettent la création d'objets communs et rendent le code plus lisible. Par exemple :

```
ipOSH= modeling.createIpOSH(ip)
host = modeling.createHostOSH(ip_address)
member1 = modeling.createLinkOSH('member', ipOSH, networkOSH)
```

netutils.py

La bibliothèque **netutils.py** sert à extraire des informations sur le réseau et TCP, par exemple extraire des noms de système d'exploitation, vérifier si une adresse MAC ou IP est valide, etc. Par exemple :

```
dnsName = netutils.getHostName(ip, ip)
isValidIp = netutils.isValidIp(ip_address)
address = netutils.getHostAddress(hostName)
```

shellutils.py

La bibliothèque **shellutils.py** fournit une API pour l'exécution de commandes shell et l'extraction du statut final d'une commande exécutée, et permet d'exécuter plusieurs commandes basées sur ce statut final. La bibliothèque est initialisée avec un client shell et utilise ce client pour exécuter des commandes et extraire des résultats. Par exemple :

```
ttyClient = Framework.createClient(Framework.getTriggerCIData(BaseClient.CREDENT
IALS_ID), Props)
clientShUtils = shellutils.ShellUtils(ttyClient)
if (clientShUtils.isWinOs()):
        logger.debug ('discovering Windows..')
```

Chapitre 3 : Messages d'erreur

Contenu de ce chapitre :

Messages d'erreur - Présentation 68

Conventions d'écriture des erreurs 68

Niveaux de gravité des erreurs 71

Messages d'erreur - Présentation

Au cours d'une découverte, de nombreuses erreurs peuvent survenir telles que des échecs de connexion, des problèmes matériels, des exceptions, des dépassements de délai, etc. Ces erreurs s'affichent dans la fenêtre Universal Discovery chaque fois que le flux de découverte normal n'aboutit pas. Vous pouvez explorer le CI déclencheur à l'origine du problème pour afficher le message d'erreur.

Le processus GFD fait la différence entre les erreurs qui peuvent parfois être ignorées (par exemple, un hôte injoignable) et celles qui doivent être traitées (par exemple, les problèmes portant sur les informations d'identification ou sur des fichiers de configuration ou DLL manquants). Par ailleurs, le processus GFD signale les erreurs une seule fois, même si elles se reproduisent lors d'exécutions successives et les signale même si elles se produisent une seule fois.

Lors de la création d'un package, vous pouvez ajouter des messages appropriés comme ressources de ce package. Au déploiement du package, les messages sont également déployés à l'emplacement correct. Les messages doivent être conformes aux conventions décrites à la section " Conventions d'écriture des erreurs " , ci-dessous.

Le processus GFD prend en charge les messages d'erreur en plusieurs langues. Vous pouvez localiser les messages que vous rédigez afin qu'ils apparaissent dans la langue locale.

Pour plus d'informations sur la recherche d'erreurs, voir « Avancement et résultats des découvertes » dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Pour plus d'informations sur la définition de journaux de communication, voir « Volet Options d'exécution » dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Conventions d'écriture des erreurs

- Chaque erreur est identifiée par un code de message d'erreur et une matrice d'arguments (**int, String[]**). C'est la combinaison d'un code de message et d'une matrice d'arguments qui définit une erreur particulière. La matrice de paramètres peut être nulle.

- Chaque code d'erreur est mappé sur un **message court** qui est une chaîne fixe et un **message détaillé** qui est une chaîne modèle contenant ou non des arguments. On suppose que le nombre d'arguments du modèle et le nombre réel de paramètres concordent.

Exemple de code de message d'erreur :

10234 peut représenter une erreur comportant le message court suivant :

```
Connection Error
```

et le message détaillé suivant :

```
Could not connect via {0} protocol due to timeout of {1} msec
```

où

{0} = premier argument : un nom de protocole

{1} = second argument : la longueur du dépassement de délai en millisecondes

Cette section comprend aussi les rubriques suivantes:

- " Contenu du fichier de propriétés " , ci-dessous
- " Fichier de propriétés des messages d'erreur " , ci-dessous
- " Conventions de dénomination des paramètres régionaux " , page suivante
- " Codes de messages d'erreur " , page suivante
- " Erreur de contenu inconnu " , page 71
- " Modifications de l'infrastructure " , page 71

Contenu du fichier de propriétés

Un fichier de propriétés doit contenir deux clés pour chaque code de message d'erreur. Par exemple, pour l'erreur 45 :

- **DDM_ERROR_MESSAGE_SHORT_45**. Description courte de l'erreur.
- **DDM_ERROR_MESSAGE_LONG_45**. Description longue de l'erreur (peut contenir des paramètres, par exemple, {**0**},{**1**}).

Fichier de propriétés des messages d'erreur

Un fichier de propriétés établit une correspondance entre un code de message d'erreur et deux messages (un court et un détaillé).

Lorsqu'un fichier de propriétés est déployé, ses données sont fusionnées avec les données existantes, c'est-à-dire que de nouveaux codes de message sont ajoutés en remplacement des anciens.

Les fichiers de propriétés d'infrastructure font partie du package **AutoDiscoveryInfra**.

Conventions de dénomination des paramètres régionaux

- Pour les paramètres régionaux par défaut : **<nom de fichier>.properties.errors**
- Pour des paramètres régionaux particuliers : **<nom de fichier>_xx.properties.errors**

  où **xx** désigne les paramètres régionaux (par exemple, **infraerr_fr.properties.errors** ou **infraerr_en_us.properties.errors**).

Codes de messages d'erreur

Les codes d'erreur suivants sont inclus par défaut dans HP Universal CMDB. Vous pouvez ajouter vos propres messages à cette liste.

Nom de l'erreur	Code de l'erreur	Description
Interne	100-199	Généralement résolue à partir des exceptions levées au cours des exécutions de script Jython
Connexion	200-299	Échec de la connexion, pas d'agent sur la machine cible, destination injoignable, etc.
Informations d'identification	300-399	Autorisation refusée, tentative de connexion bloquée par manque d'informations d'identification
Dépassement de délai	400-499	Dépassement de délai lors de la connexion/d'une commande
Comportement inattendu ou incorrect	500-599	Fichiers de configuration manquants, interruptions inattendues, etc.
Récupération d'informations	600-699	Informations manquantes sur les machines cibles, échec d'interrogation de l'agent, etc.
Ressources	700-799	Erreurs liées à une mémoire insuffisante ou à des clients non correctement distribués
Analyse syntaxique	800-899	Erreur d'analyse syntaxique du texte
Codage	900	Erreur d'entrée, codage non pris en charge
SQL	901-903, 924	Erreurs consécutives à des opérations SQL
HTTP	904-909	Erreurs générées lors des connexions HTTP, analysées à partir des codes d'erreur HTTP.

Nom de l'erreur	Code de l'erreur	Description
Application spécifique	910-923	Erreur signalée en raison de problèmes propres à une application, par exemple, version de LSOF erronée, aucun gestionnaire de file d'attente trouvé, etc.

Erreur de contenu inconnu

Pour prendre en charge un contenu ancien sans provoquer de régression, l'application et les méthodes correspondantes du SDK traitent les erreurs portant le code 100 (c'est-à-dire les erreurs de script inconnu) différemment.

Ces erreurs ne sont pas regroupées (c'est-à-dire qu'elles ne sont pas considérées comme des erreurs du même type) selon le code, mais selon le contenu du message. Autrement dit, si un script signale une erreur par les méthodes anciennes obsolètes (avec une chaîne de message et sans code d'erreur), tous les messages reçoivent le même code d'erreur, mais dans l'application ou les méthodes correspondantes du SDK, des messages différents sont affichés comme erreurs différentes.

Modifications de l'infrastructure

(com.hp.ucmdb.discovery.library.execution.BaseFramework)

Les méthodes suivantes sont ajoutées à l'interface :

- `void reportError(int msgCode, String[] params);`
- `void reportWarning(int msgCode, String[] params);`
- `void reportFatal(int msgCode, String[] params);`

Les anciennes méthodes suivantes sont toujours prises en charge dans un souci de compatibilité rétroactive, mais sont marquées comme obsolètes :

- `void reportError (String message);`
- `void reportWarning (String message);`
- `void reportFatal (String message);`

Niveaux de gravité des erreurs

Lorsqu'un adaptateur finit d'être exécuté par rapport à un CI déclencheur, il renvoie un statut. S'il ne donne lieu à aucune erreur ni aucun avertissement, le statut est **Réussite**.

Les niveaux de gravité sont répertoriés ici selon leur étendue, de la plus étroite à la plus large :

Erreurs fatales

Ce niveau rapporte les erreurs graves, telles qu'un problème d'infrastructure, des fichiers DLL manquants ou des exceptions :

- Échec de génération de la tâche (sonde introuvable, variables introuvables, etc.).
- Il est impossible d'exécuter le script.
- Le traitement des résultats échoue sur le serveur et les données ne sont pas écrites dans la base CMDB.

Erreurs

Ce niveau rapporte les problèmes qui empêchent la gestion des flux de données de récupérer des données. Examinez ces erreurs, car elles exigent généralement une intervention (par exemple, une augmentation du délai, la modification d'une plage ou d'un paramètre, l'ajout d'autres informations d'identification de l'utilisateur, etc.).

- Lorsqu'une intervention de l'utilisateur peut être bénéfique, une erreur est signalée, un problème d'informations d'identification ou de réseau exigeant des recherches plus poussées. (Il ne s'agit pas d'erreurs de découverte, mais de configuration.)
- Échec interne, généralement dû à un comportement inattendu de la machine ou de l'application découverte, par exemple, fichiers de configuration manquants, etc.

Avertissements

Lorsqu'une exécution aboutit mais qu'il existe des problèmes bénins dont vous devez être informé, la gestion des flux de données indique la gravité **Avertissement**. Examinez ces CI pour savoir quelles données manquent avant d'entamer une session de débogage plus détaillée. Le niveau de gravité **Avertissement** peut comporter des messages signalant qu'il manque un agent installé sur un hôte distant, ou qu'un attribut n'a pas pu être calculé correctement en raison de données erronées.

- Agent de connexion manquant (SNMP, WMI).
- La découverte aboutit, mais toutes les informations disponibles ne sont pas détectées.

Chapitre 4 : Développement d'adaptateurs de base de données génériques

Contenu de ce chapitre :

Adaptateur de base de données générique - Présentation 74

Requêtes TQL pour l'adaptateur de base de données générique 74

Rapprochement 75

Hibernate comme fournisseur JPA 76

Préparation de la création d'un adaptateur 78

Préparation du package de l'adaptateur 83

Configuration de l'adaptateur - Méthode minimale 86

Configuration de l'adaptateur - Méthode avancée 91

Implémentation d'un plug-in 96

Déploiement de l'adaptateur 99

Modification de l'adaptateur 99

Créer un point d'intégration 99

Création d'une vue 100

Calcul des résultats 100

Affichage des résultats 100

Affichage des rapports 100

Activation des fichiers journaux 101

Utilisation d'Eclipse pour mapper des attributs de type de CI sur des tables de base de données 101

Fichiers de configuration de l'adaptateur 109

Convertisseurs prêts à l'emploi 135

Plug-in 140

Exemples de configuration 140

Fichiers journaux de l'adaptateur 149

Références externes 151

Résolution des problèmes et limitations 151

Adaptateur de base de données générique - Présentation

La plate-forme d'adaptateurs de base de données génériques a pour fonction la création d'adaptateurs capables de s'intégrer à des systèmes de gestion de base de données relationnelle (SGBDR) et d'exécuter des requêtes TQL et des travaux de remplissage sur la base de données. Les SGBDR pris en charge par l'adaptateur de base de données générique sont Oracle, Microsoft SQL Server et MySQL.

Cette version de l'implémentation d'un adaptateur de base de données repose sur une norme JPA (API de persistance Java) avec la bibliothèque Hibernate ORM comme fournisseur de persistance.

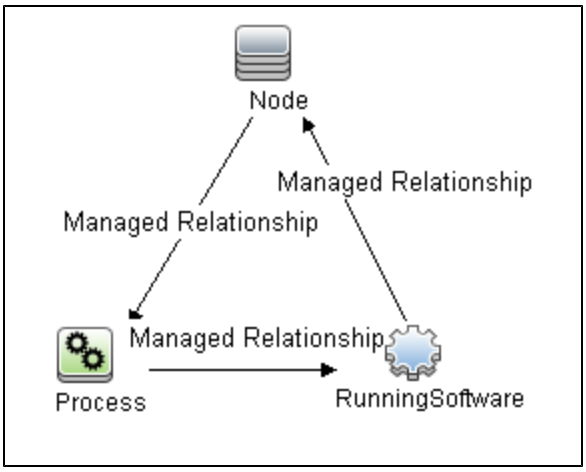
Requêtes TQL pour l'adaptateur de base de données générique

Dans le cas de travaux de remplissage, chaque mise en page requise d'un CI doit être définie dans la boîte de dialogue Paramètres de mise en page du Studio de modélisation. Pour plus d'informations, voir Query Node/Relationship Properties Dialog Box dans le *Manuel de modélisation HP Universal CMDB*. Sachez qu'un CI peut nécessiter l'identification d'un attribut sans lequel il ne peut être ajouté à UCMDB.

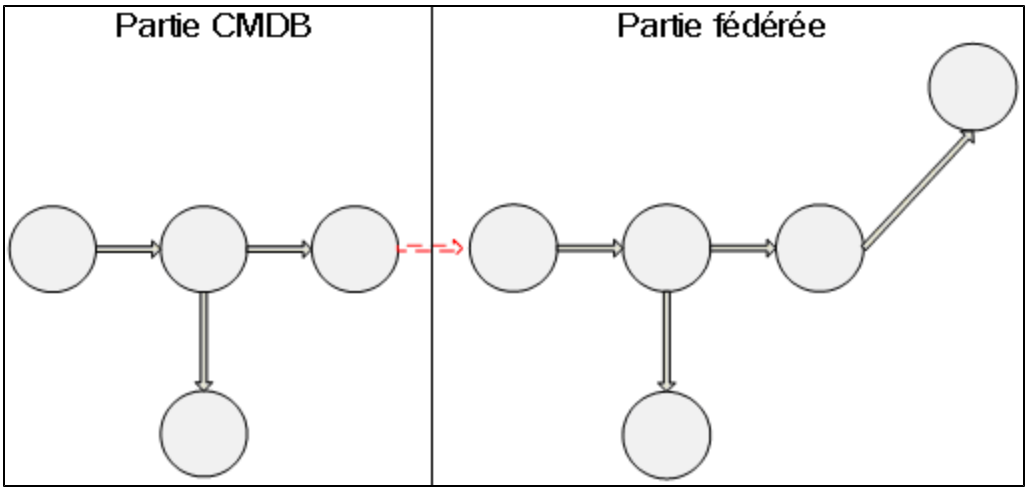
Les limites suivantes sont imposées sur les requêtes TQL calculées par l'adaptateur de base de données générique uniquement :

- les sous-graphiques ne sont pas pris en charge ;
- les relations composées ne sont pas prises en charge ;
- les cycles ou parties de cycle ne sont pas pris en charge.

  La requête TQL suivante est un exemple de cycle.



- la mise en page de fonctions n'est pas prise en charge ;
- la cardinalité 0..0 n'est pas prise en charge ;
- la relation de `jointure` n'est pas prise en charge ;
- les conditions qualificatives ne sont pas prises en charge ;
- pour la connexion entre deux CI, une relation sous la forme d'une table ou d'une clé étrangère doit exister dans la source de base de données externe.



Rapprochement

Le rapprochement est effectué dans le cadre du calcul TQL côté adaptateur. Pour que le rapprochement ait lieu, le côté CMDB est mappé sur une entité fédérée appelée type de CI de rapprochement.

Mappage. Chaque attribut de la base CMDB est mappé sur une colonne dans la source de données.

Bien que le mappage soit effectué directement, les fonctions de transformation sur les données de mappage sont également prises en charge. Vous pouvez ajouter de nouvelles fonctions par l'intermédiaire du code Java (par exemple, minuscules, majuscules). Ces fonctions ont pour objet de permettre les conversions de valeurs (valeurs stockées dans la base CMDB sous un format et dans la base fédérée sous un autre format).

Remarque :

- Pour connecter la base CMDB et la source de base de données externe, il doit exister une association appropriée dans la base de données. Pour plus d'informations, voir " Conditions préalables " , page 79.

- Le rapprochement avec l'ID CMDB est également pris en charge.
- Le rapprochement avec l'ID global est également pris en charge.

Hibernate comme fournisseur JPA

Hibernate est un outil de mappage relationnel-objet (RO), qui permet de mapper des classes Java à des tables sur plusieurs types de bases de données relationnelles (par exemple, Oracle et Microsoft SQL Server). Pour plus d'informations, voir " Limites fonctionnelles " , page 152.

Dans un mappage élémentaire, chaque classe Java est mappée sur une table unique. Un mappage plus élaboré permet un mappage d'héritage (comme celui qui peut avoir lieu dans la base de données CMDB).

Les autres fonctions prises en charge comprennent le mappage d'une classe sur plusieurs tables, la prise en charge des collections et les associations de type un à un, un à plusieurs et plusieurs à un. Pour plus d'informations, voir " Associations " , page 78ci-dessous.

Dans notre optique, il n'est pas nécessaire de créer des classes Java. Le mappage est défini entre les types de CI du modèle de classe CMDB et les tables de base de données.

Cette section comprend aussi les rubriques suivantes:

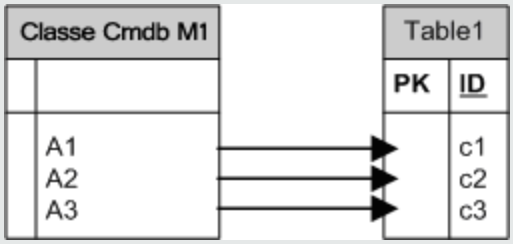
- " Exemples de mappage relationnel-objet " , ci-dessous
- " Associations " , page 78
- " Utilisation " , page 78

Exemples de mappage relationnel-objet

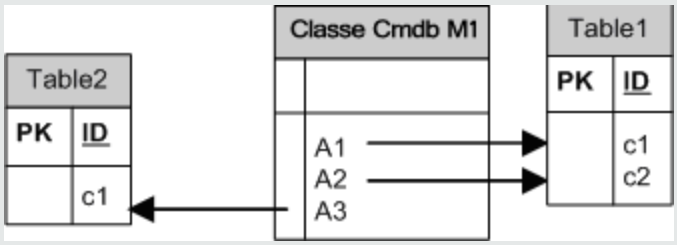
Les exemples suivants décrivent un mappage relationnel-objet :

Exemple d'une classe CMDB mappée sur une table de base de données :

La classe `M1`, avec les attributs `A1`, `A2` et `A3`, est mappée sur les colonnes `c1`, `c2` et `c3` de la table `1`. Cela signifie qu'une instance `M1` quelconque possède une ligne correspondante dans la table `1`.

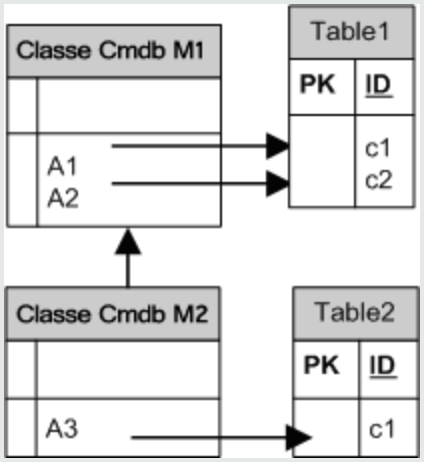


Exemple d'une classe CMDB mappée sur deux tables de base de données :



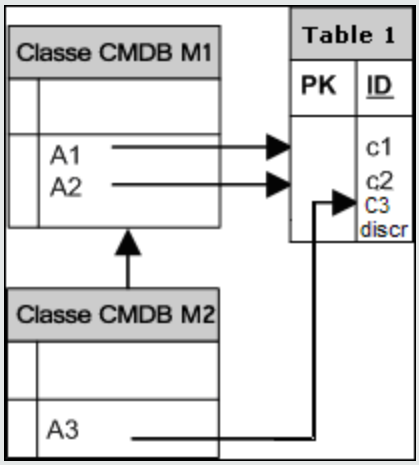
Exemple d'héritage :

Ce cas est utilisé dans la base CMDB, où chaque classe possède sa propre table de base de données.



Exemple d'héritage de table unique avec discriminateur :

Une hiérarchie entière de classes est mappée sur une table de base de données unique, dont les colonnes comprennent un sur-ensemble de tous les attributs des classes mappées. La table contient également une colonne supplémentaire (`Discriminator`), dont la valeur indique quelle classe spécifique doit être mappée sur cette entrée.



Associations

Il existe trois types d'associations : de un à plusieurs, de plusieurs à un et de plusieurs à plusieurs. Pour établir une connexion entre les différents objets de base de données, il convient de définir l'une de ces associations en utilisant une colonne de clé étrangère (pour le scénario de un à plusieurs) ou une table de mappage (pour le scénario de plusieurs à plusieurs).

Utilisation

Le schéma JPA étant très exhaustif, un fichier XML simplifié est fourni pour faciliter la définition d'associations.

Le cas d'utilisation de ce fichier XML est le suivant : les données fédérées sont modélisées en une classe fédérée. Cette classe comporte des relations de plusieurs à un avec une classe CMDB non fédérée. En outre, il n'existe qu'un seul type de relation possible entre la classe fédérée et la classe non fédérée.

Préparation de la création d'un adaptateur

Cette tâche décrit les préparatifs nécessaires pour la création d'un adaptateur.

Remarque : Vous pouvez consulter des échantillons d'adaptateur de base de données générique dans l'API UCMDB. En particulier, l'adaptateur DDMi contient un fichier **orm.xml** complexe, ainsi que les implémentations de certaines interfaces de plug-in.

Cette tâche comprend les étapes suivantes :

- " Conditions préalables " , page suivante
- " Création d'un type de CI " , page 80
- " Création d'une relation " , page 81

1. **Conditions préalables**

   Pour confirmer que vous pouvez utiliser l'adaptateur avec votre base de données, vérifiez les points suivants :

   - Les classes de rapprochement et leurs attributs (également appelés multinœuds) existent dans la base de données. Par exemple, si le rapprochement est effectué par nom de nœud, vérifiez qu'il existe une table contenant une colonne avec des noms de nœud. Si le rapprochement est effectué selon le nœud `cmdb_id`, vérifiez qu'il existe une colonne contenant des ID CMDB qui correspondent aux ID CMDB des nœuds dans la base CMDB. Pour plus d'informations sur le rapprochement, voir " Rapprochement " , page 75.

ID	NOM	IP_ADDRESS
31	BABA	16.59.33.60
33	ext3.devlab.ad	16.59.59.116
46	LABM1MAM15	16.59.58.188
72	cert-3-j2ee	16.59.57.100
102	labm1sun03.devlab.ad	16.59.58.45
114	LABM2PCOE73	16.59.66.79
116	CUT	16.59.41.214
117	labm1hp4.devlab.ad	16.59.60.182

   - Pour corréler deux types de CI avec une relation, des données de corrélation doivent exister entre les tables de types de CI. La corrélation peut s'effectuer soit par une colonne de clé étrangère, soit par une table de mappage. Par exemple, pour établir une corrélation entre un nœud et une alerte, la table d'alertes doit comporter une colonne contenant l'ID de nœud, la table de nœuds doit comporter une colonne contenant l'ID d'alerte associée ou il doit y avoir une table de mappage dans laquelle `end1` est l'ID de nœud et `end2` est l'ID d'alerte. Pour plus d'informations sur les données de corrélation, voir " Hibernate comme fournisseur JPA " , page 76.

     Le tableau suivant représente la colonne `NODE_ID` de la clé étrangère :

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	Contrôleur de bus série	Contrôleur d'hôte universel USB Intel 82801EB

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
3581	2	Système	LPC pour famille de circuits microprogrammés Intel 631xESB/6321ESB/3100
3581	3	Écran	ATI ES1000
3581	4	Périphérique système de base	Fonction de prise en charge héritée HP ProLiant iLO 2

- Chaque type de CI peut être mappé sur une ou plusieurs tables. Pour mapper un type de CI sur plusieurs tables, vérifiez qu'il y a une table principale dont la clé principale existe dans les autres tables et est une colonne à valeur unique.

  Par exemple, une alerte est mappée sur deux tables : `ticket1` et `ticket2`. La première table comporte les colonnes `c1` et `c2` et la seconde, les colonnes `c3` et `c4`. Pour qu'elles soient considérées comme une seule table, elles doivent toutes deux avoir la même clé principale. Sinon, la clé principale de la première table peut être une colonne de la seconde.

  Dans l'exemple suivant, les tables partagent la même clé principale, appelée `CARD_ID` :

CARD_ID	CARD_TYPE	CARD_NAME
1	Contrôleur de bus série	Contrôleur d'hôte universel USB Intel 82801EB
2	Système	LPC pour famille de circuits microprogrammés Intel 631xESB/6321ESB/3100
3	Écran	ATI ES1000
4	Périphérique système de base	Fonction de prise en charge héritée HP ProLiant iLO 2

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(Contrôleur hôte USB standard)
3	Hewlett-Packard Company
4	(Périphériques système standard)
5	Hewlett-Packard Company

2. **Création d'un type de CI**

Dans cette étape, vous créez un type de CI qui représente les données dans le SGBDR (la source de données externe).

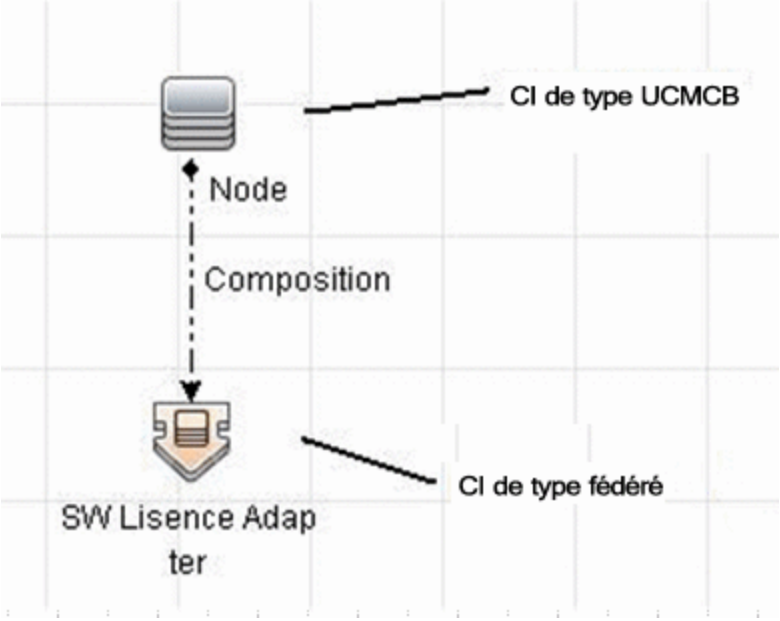
a. Dans UCMDB, accédez au Gestionnaire des types de CI et créez un type de CI. Pour plus d'informations, voir Create a CI Type dans le *Manuel de modélisation HP Universal CMDB*.

b. Ajoutez les attributs nécessaires au type de CI, tels que l'heure du dernier accès, le fournisseur, etc. Il s'agit des attributs que l'adaptateur va extraire de la source de données externe et importer dans des vues CMDB.

3. **Création d'une relation**

Au cours de cette étape, vous ajoutez une relation entre le type de CI UCMDB et le nouveau type de CI qui représente les données à fédérer à partir de la source de données externe.

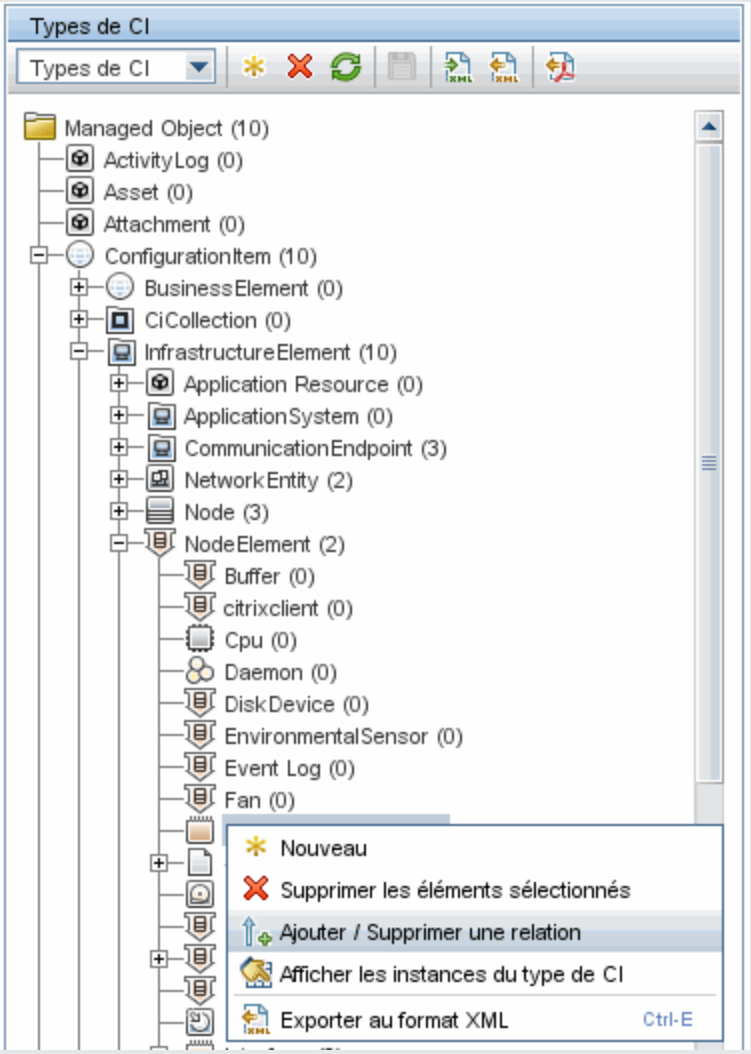
Ajoutez des relations appropriées valides au nouveau type de CI. Pour plus d'informations, voir Add/Remove Relationship Dialog Box dans le *Manuel de modélisation HP Universal CMDB*.



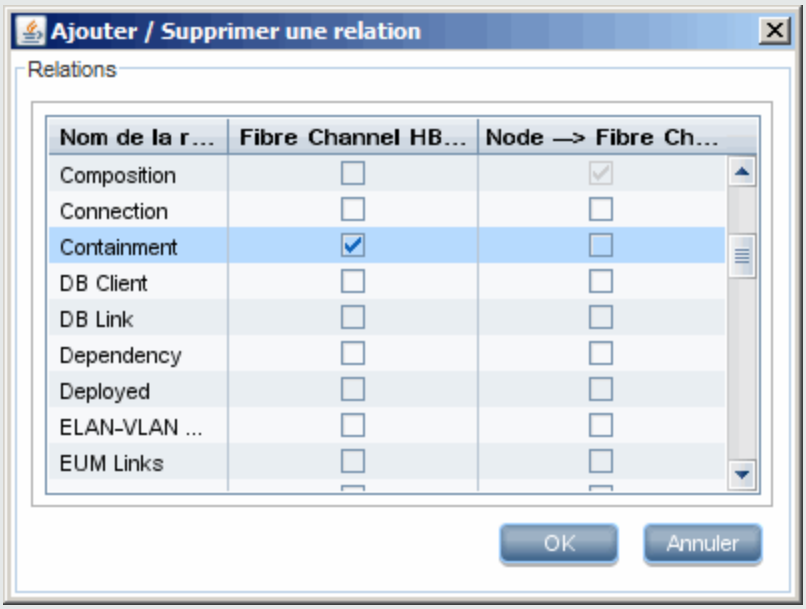
Remarque : À ce stade, vous ne pouvez pas encore visualiser les données fédérées ni remplir les données externes, puisque vous n'avez pas encore défini la méthode d'importation des données.

Exemple de création d'une relation contenant-contenu :

a. Dans le Gestionnaire des types de CI, sélectionnez les deux types de CI :



b. Créez une relation **Containment** entre les deux types de CI :



Préparation du package de l'adaptateur

Au cours de cette étape, vous recherchez et configurez le package de l'adaptateur de base de données générique.

1. Recherchez le package **db-adapter.zip** dans le dossier **C:\hp\UCMDB\UCMDBServer\content\adapters**.

2. Extrayez le package dans un répertoire temporaire local.

3. Modifiez le fichier XML de l'adaptateur :

   - Ouvrez le fichier **discoveryPatterns\db_adapter.xml** dans un éditeur de texte.

   - Recherchez l'attribut **adapter id** et remplacez le nom :

```
<pattern id="MyAdapter" displayLabel="My Adapter" xsi:noNamespaceSchem
aLocation="../../Patterns.xsd" description="Discovery Pattern Descript
ion"
         schemaVersion="9.0" xmlns:xsi="http://www.w3.org/2001/XMLSche
ma-instance" displayName="UCMDB API Population">
```

   Si l'adaptateur prend en charge le remplissage, la fonctionnalité suivante doit être ajoutée à l'élément **<adapter-capabilities>** :

```
<support-replication-data>
    <source>
        <changes-source>
    </source>
</support-replication-data>
```

Le nom affiché ou l'ID apparaît dans la liste des adaptateurs du volet Point d'intégration dans HP Universal CMDB.

Lors de la création d'un adaptateur de base de données générique, il n'est pas nécessaire de modifier la balise **changes-source** dans la balise **support-replicatioin-data**. Si le plug-in **FcmdbPluginForSyncGetChangesTopology** est implémentée, la topologie modifiée depuis la dernière exécution est renvoyée. En revanche, si l'implémentation du plug-in n'est pas effectuée, la topologie complète est renvoyée et la suppression automatique est exécutée selon les CI retournés.

Pour plus d'informations sur le remplissage de données dans le CMDB, voir « Page Studio d'intégration » dans le *Manuel de gestion des flux de données HP Universal CMDB*.

- Si l'adaptateur utilise le moteur de mappage de la version 8.x (c'est-à-dire qu'il n'utilise pas le nouveau moteur de mappage de rapprochement), remplacez l'élément suivant :

  ```
  <default-mapping-engine>
  ```

  par :

  ```
  <default-mapping-engine>com.hp.ucmdb.federation.
  mappingEngine.AdapterMappingEngine</default-mapping-engine>
  ```

  Pour rétablir le nouveau moteur de mappage, redonnez la valeur suivante à l'élément :

  ```
  <default-mapping-engine>
  ```

- Recherchez la définition **category** :

  ```
  <category>Generic</category>
  ```

  Remplacez le nom de catégorie **Generic** par celui de la catégorie de votre choix.

  > **Remarque :** Les adaptateurs dont les catégories sont désignées comme **Generic** ne sont pas répertoriés dans le Studio d'intégration lorsque vous créez un nouveau point d'intégration.

- La connexion à la base de données peut être décrite à l'aide d'un nom d'utilisateur (schéma), mot de passe, type de base de données, nom de l'ordinateur hôte dans la base de données et nom ou SID de la base de données.

Pour ce type de connexion, les paramètres disposent des éléments suivants dans la section **parameter** du fichier XML de l'adaptateur :

```
<parameters>
    <!--The description attribute may be written in simple text or HTM
L.-->
    <!--The host attribute is treated as a special case by UCMDB-->
    <!--and will automatically select the probe name (if possible)-->
    <!--according to this attribute's value.-->
    <!--Display name and description may be overwritten by I18N values-
->
        <parameter name="host" display-name="Hostname/IP" type="string"
description="The host name or IP address of the remote machine" mandat
ory="false" order-index="10" />
        <parameter name="port" display-name="Port" type="integer" desc
ription="The remote machine's connection port" mandatory="false" order-
index="11" />
        <parameter name="dbtype" display-name="DB Type" type="string"
description="The type of database" valid-values="Oracle;SQLServer;MySQ
L;BO" mandatory="false" order-index="13">Oracle</parameter>
        <parameter name="dbname" display-name="DB Name/SID" type="stri
ng" description="The name of the database or its SID (in case of Oracl
e)"  mandatory="false" order-index="13" />
        <parameter name="credentialsId" display-name="Credentials ID"
type="integer" description="The credentials to be used" mandatory="tru
e" order-index="12" />
</parameters>
```

Remarque : Il s'agit de la configuration par défaut. Par conséquent, le fichier **db_adapter.xml** contient déjà cette définition.

Il arrive que ce mode de configuration de la connexion à la base de données ne soit pas possible. C'est par exemple le cas pour la connexion à Oracle RAC ou la connexion à l'aide d'un pilote de base de données autre que celui fourni avec CMDB.

Il s'agit alors de décrire la connexion à l'aide du nom d'utilisateur (schéma), du mot de passe et d'une chaîne d'URL de connexion.

Pour définir ces éléments, modifiez la section réservée aux paramètres XML de l'adaptateur comme suit :

```
<parameters>
    <!--The description attribute may be written in simple text or HTML.-->
    <!--The host attribute is treated as a special case by CMDBRTSM-->
    <!--and will automatically select the probe name (if possible)-->
```

```
    <!--according to this attribute's value.-->
    <!--Display name and description may be overwritten by I18N values-->
        <parameter name="url" display-name="Connection String" type="strin
g" description="The connection string to connect to the database" mandatory
="true" order-index="10" />
        <parameter name="credentialsId" display-name="Credentials ID" type=
"integer" description="The credentials to be used" mandatory="true" order-i
ndex="12" />
</parameters>
```

Voici un exemple d'URL de connexion à un RAC Oracle à l'aide du pilote Data Direct prêt à l'emploi : **jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true**.

4. Dans le répertoire temporaire, ouvrez le dossier **adapterCode** et attribuez à **GenericDBAdapter** la valeur d'**adapter id** utilisée à l'étape précédente.

   Ce dossier contient la configuration de l'adaptateur, par exemple, le nom de l'adaptateur, les requêtes et les classes dans la base CMDB, ainsi que les champs du SGBDR que l'adaptateur prend en charge.

5. Configurez l'adaptateur comme nécessaire. Pour plus d'informations, voir " Configuration de l'adaptateur - Méthode minimale " , ci-dessous.

6. Créez un fichier `*.zip` portant le même nom que celui donné à l'attribut **adapter id** à l'étape" Préparation du package de l'adaptateur " , page 83.

   Remarque : Le fichier **descriptor.xml** est un fichier par défaut qui figure dans tous les packages.

7. Enregistrez le nouveau package que vous avez créé à l'étape précédente. Le répertoire par défaut des adaptateurs est le suivant : **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Configuration de l'adaptateur - Méthode minimale

La méthode simplifiée (minimale) consiste à créer le fichier de mappage **simplifiedConfiguration.xml** utilisé par l'adaptateur. Cette méthode autorise une fédération ou un remplissage de base d'un seul type de CI.

Les instructions fournies dans cette section décrivent une méthode de mappage du modèle de classe pour certains types de CI du CMDB sur un SGBDR.

Tous les fichiers de configuration mentionnés dans cette section se trouvent dans le package **db-adapter.zip**, dans le dossier **C:\hp\UCMDB\UCMDBServer\content\adapters** que vous avez extrait à la section " Préparation du package de l'adaptateur " , page 83.

Remarque : Le fichier **orm.xml** qui est généré automatiquement lors de l'exécution de cette

méthode est un bon exemple que vous pouvez utiliser avec la méthode avancée.

Cette méthode minimale est recommandée pour :

- fédérer/remplir un nœud unique tel qu'un attribut de nœud ;
- démontrer les capacités de l'adaptateur de base de données générique.

Cette méthode :

- prend en charge la fédération/le remplissage d'un seul nœud uniquement,
- prend uniquement en charge les relations virtuelles de plusieurs à un.

Configuration du fichier adapter.conf

Pour modifier les paramètres du fichier `adapter.conf` afin que l'adaptateur utilise la méthode de configuration simplifiée :

1. Ouvrez le fichier **adapter.conf** dans un éditeur de texte.
2. Recherchez la ligne suivante : **use.simplified.xml.config=<true/false>**.
3. Remplacez-la par **use.simplified.xml.config=true**.

Exemple : Remplissage d'un nœud et d'une adresse IP à l'aide de la méthode simplifiée

Cet exemple montre comment remplir un **nœud** associé par un lien d'imbrication à l'**adresse IP** dans la base UCMDB. Le SGBDR dispose d'une table nommée **simpleNode** qui contient des données relatives au nom, au nœud et à l'adresse IP de l'ordinateur.

Le contenu de la table **simpleNode** est indiqué ci-après :

host_id	host_name	note	ip_address
1	Comp1	Test Computer 1	12.33.211.52
2	Comp2	Test Computer 2	12.33.211.53
3	Comp3	Test Computer 3	12.33.211.54
4	Comp4	Test Computer 4	12.33.211.55

Le remplissage s'opère en trois étapes :

1. " Création du fichier simplifiedConfiguration.xml " , page suivante
2. " Création du code TQL " , page 89
3. " Création d'un point d'intégration " , page 91

Création du fichier simplifiedConfiguration.xml

Créez le fichier **simplifiedConfiguration.xml** comme suit :

1. Créez l'entité **cmdb-class** comme suit :

   ```
   <cmdb-class cmdb-class-name="node" default-table-name="simpleNode">
   ```

   Le type de CI est **node** et le nom de la table du SGBDR est **simpleNode**.

2. Définissez la clé principale de la table comme suit :

   ```
   <primary-key column-name="host_id"/>
   ```

   La clé principale correspond à l'ID d'entité dans le fichier **orm.xml**.

3. Définissez la règle **reconciliation-by-two-nodes** comme suit :

   ```
   <reconciliation-by-two-nodes connected-node-cmdb-class-name="ip_address" cmd
   b-link-type="containment">
   ```

   Cette balise définit la relation entre les types de CI **Node** et **IpAddress**. Le type de relation est un lien d'imbrication (containment). Le rapprochement est effectué par deux types de CI associés. Le mappage d'attributs du nœud connecté (dans ce cas IpAddress) est défini dans l'attribut **connected-node**.

4. Ajoutez la condition **or** entre les attributs de rapprochement comme suit :

   ```
   <or is-ordered="true">
   ```

   Cette balise définit une relation OR (OU) entre les attributs de rapprochement : le premier attribut de rapprochement ayant la valeur **true** définit l'intégralité de la règle de rapprochement sur **true**.

5. Ajoutez les attributs suivants :

   ```
   <attribute cmdb-attribute-name="name" column-name="host_name" ignore-case="t
   rue"/>
   ```

   Cette balise définit un mappage entre **node.name** dans la base UCMDB et la colonne **host_name** dans la table **simpleNode**.

   Procédez de même avec l'attribut **data_note**.

   ```
   <attribute cmdb-attribute-name="data_note" column-name="note" ignore-case="t
       rue"/>
   ```

   Ajoutez l'attribut de nœud connecté :

   ```
   <connected-node-attribute cmdb-attribute-name="name" column-name="ip_addres
   s"/>
   ```

   Cette balise définit un mappage entre **ip_address.name** et la colonne **ip_address** dans la table **simpleNode**.

6. Fermez la balise ouverte dans l'ordre suivant :

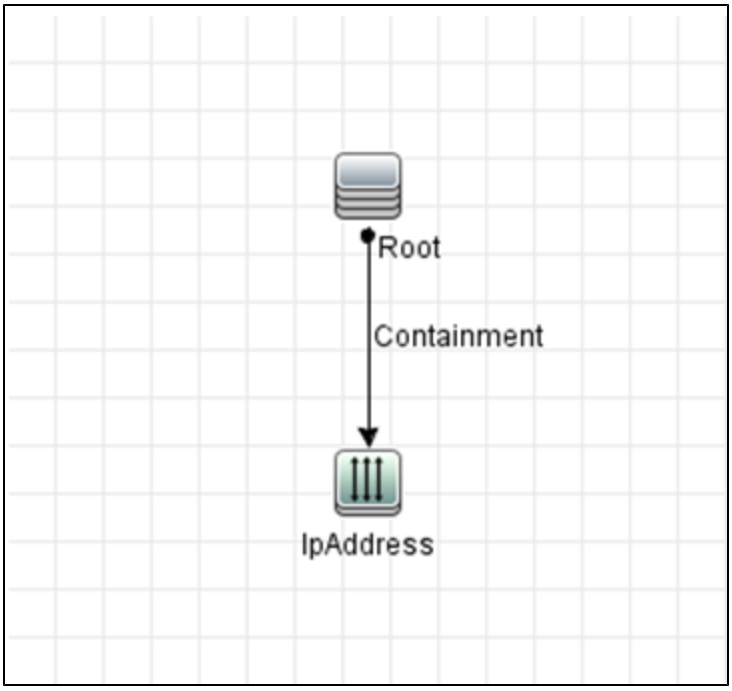
```
</or>

</reconciliation-by-two-nodes>

</cmdb-class>
```

Le contenu du fichier simplifiedConfiguration.xml est à présent le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-db-adapter-config xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfig
uration.xsd">
   <cmdb-class cmdb-class-name="node" default-table-name="simpleNode">
       <primary-key column-name="host_id"/>
   <reconciliation-by-two-nodes connected-node-cmdb-class-name="ip_address" cmd
b-link-type="containment">
       <or is-ordered="true">
       <attribute cmdb-attribute-name="name" column-name="host_name" ignore-
case="true"/>
       <attribute cmdb-attribute-name="data_note" column-name="note" ignore-
case="true"/>
          <connected-node-attribute cmdb-attribute-name="name" column-name="ip_a
ddress"/>
       </or>
       </reconciliation-by-two-nodes>
   </cmdb-class>
</generic-db-adapter-config>
```

Création du code TQL

Le TQL est un CI de type **node** connecté par un lien d'imbrication (containment) à **ip_address**. Le nœud doit être marqué comme **racine** (voir ci-dessous).



Pour créer le TQL :

1. Accédez à **Modélisation > Studio de modélisation**.
2. Cliquez sur le bouton **Nouveau** et créez une requête.
3. Accédez à l'onglet Types de CI et faites glisser les types de CI Node et IpAddress vers l'écran TQL.
4. Connectez **Node** et **IpAddress** avec une relation Containment.
5. Cliquez avec le bouton droit sur l'élément **Node**, puis sélectionnez Propriétés du nœud de requête.
6. Modifiez le **Nom de l'élément** par **Racine**.
7. Accédez à l'onglet **Mise en page de l'élément**. Sélectionnez **Attributs spécifiques** comme condition des attributs. Sélectionnez **Name** et **Note** dans la fenêtre Attributs disponibles et déplacez-les vers la fenêtre Attributs spécifiques.
8. Cliquez avec le bouton droit sur l'élément **IpAddress**, puis sélectionnez Propriétés du nœud de requête.
9. Accédez à l'onglet **Mise en page de l'élément**. Sélectionnez **Attributs spécifiques** comme condition des attributs. Sélectionnez **Nom** dans la fenêtre Attributs disponibles et déplacez-le vers la fenêtre Attributs spécifiques.
10. Enregistrez le TQL.

Création d'un point d'intégration

Créez un point d'intégration comme suit :

1. Sélectionnez **Gestionnaire des flux de données > Studio d'intégration**, puis cliquez sur le bouton **Nouveau point d'intégration**.
2. Insérez les détails du point d'intégration et cliquez sur **OK**.
3. Dans l'onglet Remplissage, sélectionnez le bouton **Nouveau travail d'intégration** et ajoutez le code TQL créé précédemment.
4. Enregistrez le point d'intégration et cliquez sur le bouton **Run Full Synchronization**.

Configuration de l'adaptateur - Méthode avancée

Ces fichiers de configuration sont inclus dans le package **db-adapter.zip**, dans le dossier **C:\hp\UCMDB\UCMDBServer\content\adapters** que vous avez extrait lors de la préparation du package de l'adaptateur. Pour plus d'informations, voir " Préparation du package de l'adaptateur " , page 83.

Cette tâche comprend les étapes suivantes :

- " Configuration du fichier orm.xml " , ci-dessous
- " Configuration du fichier reconciliation_rules.txt " , page 95

Configuration du fichier orm.xml

Au cours de cette étape, vous mappez les types de CI et relations de la base CMDB sur les tables du SGBDR.

1. Ouvrez le fichier **orm.xml** dans un éditeur de texte.

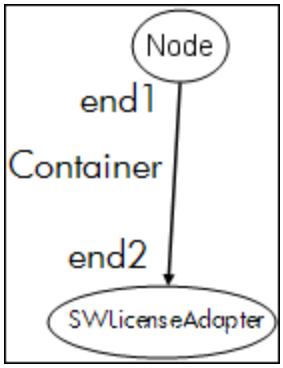
   Par défaut, ce fichier contient un modèle que vous utilisez pour mapper autant de types de CI et de relations que nécessaire.

   > **Remarque :** N'éditez le fichier **orm.xml** dans aucune version du Bloc-notes de Microsoft Corporation. Utilisez Notepad++, UltraEdit ou un autre éditeur de texte tiers.

2. Apportez des modifications au fichier en fonction des entités de données à mapper. Pour plus d'informations, voir les exemples suivants.

   Les types suivants de relations peuvent être mappés dans le fichier **orm.xml** :

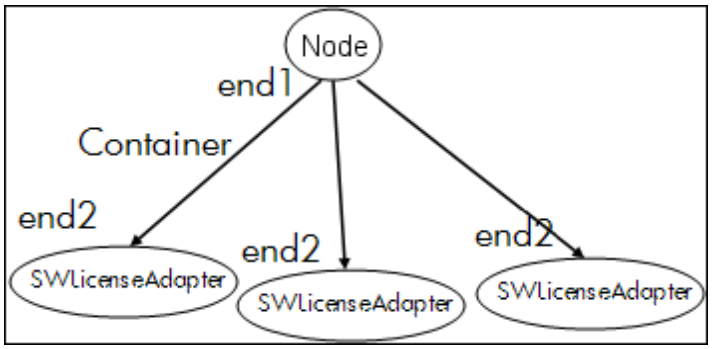
- De un à un :



Le code de ce type de relation est le suivant :

```
<one-to-one name="end1" target-entity="node">
        <join-column name="Device_ID" >
</one-to-one>
<one-to-one name="end2" target-entity="sw_sub_component">
        <join-column name="Device_ID" >
        <join-column name="Version_ID" >
</one-to-one>
```

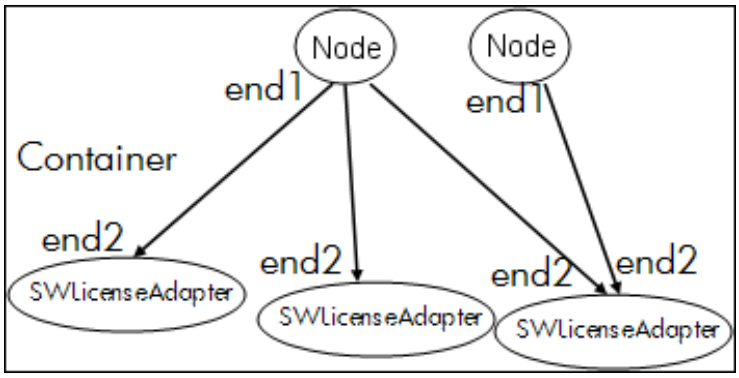
- De plusieurs à un :



Le code de ce type de relation est le suivant :

```
<many-to-one name="end1" target-entity="node">
        <join-column name="Device_ID" >
</many-to-one>
<one-to-one name="end2" target-entity="sw_sub_component">
        <join-column name="Device_ID" >
        <join-column name="Version_ID" >
</one-to-one>
```

- De plusieurs à plusieurs :



Le code de ce type de relation est le suivant :

```
<many-to-one name="end1" target-entity="node">
        <join-column name="Device_ID" >
</many-to-one>
<many-to-one name="end2" target-entity="sw_sub_component">
        <join-column name="Device_ID" >
        <join-column name="Version_ID" >
</many-to-one>
```

Pour plus d'informations sur les conventions de dénomination, voir " Conventions de dénomination " , page 118.

Exemple de mappage d'entité entre le modèle de données et le SGBDR :

Remarque : Les attributs qu'il n'est pas nécessaire de configurer sont omis des exemples suivants.

- Classe du type de CI CMDB :

```
<entity class="generic_db_adapter.node">
```

- Nom de la table dans le SGBDR :

```
<table name="Device"/>
```

- Nom de colonne de l'identifiant unique dans la table du SGBDR :

```
<column name="Device ID"/>
```

- Nom de l'attribut dans le type de CI CMDB :

```
<basic name="name">
```

- Nom du champ de table dans la source de données externe :

```
<column name="Device_Name"/>
```

- Nom du nouveau type de CI que vous avez créé dans " Création d'un type de CI " , page 80 :

```
<entity class="generic_db_adapter.MyAdapter">
```

- Nom de la table correspondante dans le SGBDR :

```
<table name="SW_License"/>
```

- Identifiant unique dans le SGBDR :

- Nom de l'attribut dans le type de CI CMDB et nom de l'attribut correspondant dans le SGBDR :

Exemple de mappage de relation entre le modèle de données et le SGBDR :

- Classe de la relation CMDB :

```
<entity class="generic_db_adapter.node_containment_MyAdapter">
```

- Nom de la table du SGBDR dans laquelle la relation est établie :

```
<table name="MyAdapter"/>
```

- ID unique dans le SGBDR :

```
<id name="id1">
        <column updatable="false" insertable="false"
name="Device_ID">
        <generated-value strategy="TABLE"/>
</id>
<id name="id2">
        <column updatable="false" insertable="false"
name="Version_ID">
        <generated-value strategy="TABLE"/>
</id>
```

- Type de relation et type de CI CMDB :

```
<many-to-one target-entity="node" name="end1">
```

- Champs de clé principale et de clé étrangère dans le SGBDR :

```
<join-column updatable="false" insertable="false"
referenced-column-name="[column_name]" name="Device_ID" />
```

Configuration du fichier reconciliation_rules.txt

Au cours de cette étape, vous définissez les règles par lesquelles l'adaptateur rapproche la base CMDB et le SGBDR (uniquement si le moteur de mappage est utilisé, à des fins de compatibilité rétroactive avec la version 8.x) :

1. Ouvrez **META-INF\reconciliation_rules.txt** dans un éditeur de texte.

2. Apportez des modifications au fichier en fonction du type de CI que vous mappez. Par exemple, pour mapper un type de CI node, utilisez l'expression suivante :

```
multinode[node] ordered expression[^name]
```

Remarque :

- Si les données de la base de données sont sensibles à la casse, ne supprimez pas le caractère de contrôle (**^**).
- Vérifiez que chaque crochet ouvrant correspond à un crochet fermant.

Pour plus d'informations, voir " Fichier reconciliation_rules.txt (pour compatibilité rétroactive) " , page 128.

Implémentation d'un plug-in

Cette tâche explique comment implémenter et déployer un adaptateur de base de données générique avec des plug-in.

Remarque : Avant d'écrire un plug-in pour un adaptateur, assurez-vous d'avoir exécuté toutes les étapes nécessaires décrites à la section " Préparation du package de l'adaptateur " , page 83.

1. Option 1 – Écriture d'un plug-in de type Java

   a. Copiez les fichiers jar suivants depuis le répertoire d'installation du serveur UCMDB vers le classpath de votre développement :

      - Copiez les fichiers **db-interfaces.jar** et **db-interfaces-javadoc.jar** depuis le dossier **tools\adapter-dev-kit\db-adapter-framework**.
      - Copiez les fichiers **federation-api.jar** et **federation-api-javadoc.jar** depuis le dossier **\tools\adapter-dev-kit\SampleAdapters\production-lib**.

      Remarque : Pour plus d'informations sur le développement d'un plug-in, voir les fichiers **db-interfaces-javadoc.jar** et **federation-api-javadoc.jar** ainsi que la documentation en ligne disponible sous :

      - **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DBAdapterFramework_JavaAPI\index.html**
      - **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\Federation_JavaAPI\index.html**

   b. Écrivez une classe Java implémentant l'interface Java du plug-in. Les interfaces sont définies dans le fichier **db-interfaces.jar**. Le tableau ci-après indique l'interface à implémenter pour chaque plug-in :

Type de plug-in	Nom de l'interface	Méthode
Synchronisation de la topologie complète	FcmdbPluginForSyncGetFullTopology	getFullTopology
Synchronisation des modifications	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Synchronisation de la mise en page	FcmdbPluginForSyncGetLayout	getLayout
Extraction des requêtes prises en charge	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Modification de la définition de requête TQL et des résultats	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Modification de la demande de mise en page de CI	FcmdbPluginGetCIsLayout	getCisLayout
Modification de la demande de mise en page de liens	FcmdbPluginGetRelationsLayout	getRelationsLayout
ID renvoyés	FcmdbPluginPushBackIds	getPushBackIdsSQL

La classe du plug-in doit comporter un constructeur par défaut public. Par ailleurs, toutes les interfaces présentent une méthode appelée initPlugin. Cette méthode est assurée d'être appelée avant toutes les autres et permet d'initialiser l'adaptateur avec l'objet d'environnement de l'adaptateur qui le contient.

Si **FcmdbPluginForSyncGetChangesTopology** est implémenté, il existe deux façons différentes de signaler les changements :

- **Signaler l'intégralité de la topologie racine à tout moment.** Selon la topologie, la fonction de suppression automatique recherche les CI à supprimer. Dans ce cas, il convient d'activer cette fonction en respectant la syntaxe ci-après :

```
<autoDeleteCITs isEnabled="true">
                <CIT>link</CIT>
                <CIT>object</CIT>
            </autoDeleteCITs>
```

- **Signaler chaque instance de CI supprimée/mise à jour.** Dans ce cas, il convient de

désactiver le mécanisme de suppression automatique selon la syntaxe ci-après :

```
<autoDeleteCITs isEnabled="false">
                <CIT>link</CIT>
                <CIT>object</CIT>
            </autoDeleteCITs>
```

c. Vérifiez que le fichier JAR du SDK de fédération et les fichiers JAR de l'adaptateur de BD générique figurent dans votre classpath avant de compiler votre code Java. Le SDK de fédération est le fichier **federation_api.jar**, qui figure dans le répertoire **C:\hp\UCMDB\UCMDBServer\lib**.

d. Compressez votre classe dans un fichier jar et placez-la sous le dossier adapterCode\<Nom de votre adaptateur> dans le package de l'adaptateur, avant de la déployer.

2. Option 2 – Écriture d'un plug-in de type Groovy

   a. Créez un fichier de code Groovy (MyPlugin.groovy) dans le menu Gestion de l'adaptateur, sous les fichiers de configuration du package de l'adaptateur.

   b. Dans la classe Groovy, implémentez les interfaces appropriées. Les interfaces sont définies dans le fichier db-interfaces.jar (voir le tableau ci-dessus).

3. Les plug-in sont configurés à l'aide du fichier **plugins.txt** inclus dans le dossier **\META-INF** de l'adaptateur.

   Voici un exemple du fichier de l'adaptateur DDMi :

```
# mandatory plugin to sync full topology
[getFullTopology]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin
# mandatory plugin to sync changes in topology
[getChangesTopology]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin
# mandatory plugin to sync layout
[getLayout]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin
# plugin to get supported queries in sync. If not defined return all tqls
names
[getSupportedQueries]
# internal not mandatory plugin to change tql definition and tql result
[getTopologyCmdbFormat]
# internal not mandatory plugin to change layout request and CIs result
[getCisLayout]
# internal not mandatory plugin to change layout request and relations
```

```
result
[getRelationsLayout]
# internal not mandatory plugin to change action on pushBackIds
[pushBackIds]
```

Légende :

- Ligne de commentaire.

[<Type d'adaptateur>] – Début de la section de définition pour un type d'adaptateur spécifique.

Chaque [<Type d'adaptateur>] peut être suivi d'une ligne vide, ce qui signifie qu'aucune classe de plug-in ne lui est associée, ou du nom complet de votre classe de plug-in.

4. Compressez votre adaptateur avec le nouveau fichier jar et le fichier **plugins.xml** mis à jour. Les autres fichiers du package doivent être les mêmes que dans tout adaptateur basé sur l'adaptateur de BD générique.

Déploiement de l'adaptateur

1. Dans UCMDB, accédez au Gestionnaire des packages. Pour plus d'informations, voir "Package Manager Page" dans le *Manuel d'administration HP Universal CMDB*.

2. Cliquez sur l'icône **Déployer les packages sur le serveur (à partir du disque local)** et accédez au package de votre adaptateur. Sélectionnez le package et cliquez sur **Ouvrir**, puis cliquez sur **Déployer** pour l'afficher dans le Gestionnaire des packages.

3. Sélectionnez votre package dans la liste et cliquez sur l'icône **Afficher les ressources du package** pour vérifier que le contenu du package est reconnu par le Gestionnaire des packages.

Modification de l'adaptateur

Après avoir créé et déployé l'adaptateur, vous pouvez le modifier dans UCMDB. Pour plus d'informations, voir « Configuration de l'adaptateur » dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Créer un point d'intégration

Cette étape permet de vérifier le fonctionnement de la fédération, c'est-à-dire, la validité de la connexion et du fichier XML. Cependant, ce contrôle ne vérifie pas que le code XML est mappé sur les champs corrects dans le SGBDR.

1. Dans UCMDB, accédez au Studio d'intégration (**Gestion des flux de données > Studio d'intégration**).

2. Créez un point d'intégration. Pour plus d'informations, voir New Integration Point/Edit Integration Point Dialog Box dans le *Manuel de gestion des flux de données HP Universal CMDB*.

   L'onglet Fédération affiche tous les types de CI qui peuvent être fédérés à l'aide de ce point d'intégration. Pour plus d'informations, voir Federation Tab dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Création d'une vue

Au cours de cette étape, vous créez une vue permettant d'afficher des instances du type de CI.

1. Dans UCMDB, accédez au Studio de modélisation (**Modélisation > Studio de modélisation**).

2. Créez une vue. Pour plus d'informations, voir Create a Pattern View dans le *Manuel de modélisation HP Universal CMDB*.

Calcul des résultats

Cette étape permet de vérifier les résultats.

1. Dans UCMDB, accédez au Studio de modélisation (**Modélisation > Studio de modélisation**).

2. Ouvrez une vue.

3. Calculez les résultats en cliquant sur le bouton **Calculer le total des résultats de la requête** .

4. Cliquez sur le bouton **Aperçu** pour afficher les CI dans la vue.

Affichage des résultats

Au cours de cette étape, vous affichez les résultats et déboguez les problèmes survenus dans la procédure. Par exemple, si rien n'apparaît dans la vue, vérifiez les définitions dans le fichier **orm.xml** ; supprimez les attributs de relation et rechargez l'adaptateur.

1. Dans UCMDB, accédez au Gestionnaire de l'Univers IT (**Modélisation > Gestionnaire de l'Univers IT**).

2. Sélectionnez un CI. L'onglet Propriétés affiche les résultats de la fédération.

Affichage des rapports

Cette étape consiste à afficher les rapports topologiques. Pour plus d'informations, voir Topology Reports Overview dans le *Manuel de modélisation HP Universal CMDB*.

Activation des fichiers journaux

Pour comprendre les flux de calcul, suivre le cycle de vie de l'adaptateur et afficher des informations de débogage, vous pouvez consulter les fichiers journaux. Pour plus d'informations, voir " Fichiers journaux de l'adaptateur " , page 149.

Utilisation d'Eclipse pour mapper des attributs de type de CI sur des tables de base de données

Attention : Cette procédure s'adresse aux utilisateurs possédant une bonne maîtrise du développement de contenu. Pour toute question, contactez le Assistance HP Software.

Cette tâche explique comment installer et utiliser le plug-in JPA, fourni avec l'édition J2EE d'Eclipse, pour effectuer les opérations suivantes :

- Permettre un mappage graphique entre des attributs de classe CMDB et des colonnes de table de base de données.
- Permettre l'édition manuelle du fichier de mappage (`orm.xml`), tout en assurant son exactitude. Le contrôle d'exactitude comprend une vérification de la syntaxe et contrôle que les attributs de classe et les colonnes mappées de la table de base de données sont correctement cités.
- Permettre le déploiement du fichier de mappage sur le serveur CMDB et l'affichage des erreurs, en guise de contrôle d'exactitude supplémentaire.
- Définir un exemple de requête sur le serveur CMDB et l'exécuter directement à partir d'Eclipse, pour tester le fichier de mappage.

La version 1.1 du plug-in est compatible avec UCMDB version 9.01 ou ultérieure et Eclipse IDE for Java EE Developers version 1.2.2.20100217-2310 ou ultérieure.

Cette tâche comprend les étapes suivantes :

- " Conditions préalables " , page suivante
- " Installation " , page suivante
- " Préparation de l'environnement de travail " , page suivante
- " Création d'un adaptateur " , page 103
- " Configuration du plug-in CMDB " , page 103
- " Importation du modèle de classe UCMDB " , page 104
- " Création du fichier ORM – Mappage des classes UCMDB sur des tables de base de données " , page 104

- " Mappage d'ID " , page 105
- " Mappage d'attributs " , page 105
- " Mappage d'un lien valide " , page 105
- " Création du fichier ORM – Utilisation de tables secondaires " , page 106
- " Définition d'une table secondaire " , page 106
- " Mappage d'un attribut sur une table secondaire " , page 107
- " Utilisation d'un fichier ORM existant comme base " , page 107
- " Importation d'un fichier ORM existant à partir d'un adaptateur " , page 107
- " Vérification de l'exactitude du fichier orm.xml – Contrôle d'exactitude intégré " , page 108
- " Création d'un point d'intégration " , page 108
- " Déploiement du fichier ORM dans la base CMDB " , page 108
- " Exécution d'un exemple de requête TQL " , page 108

1. ## Conditions préalables

   Installez la dernière mise à jour de **Java Runtime Environment (JRE) 6** sur l'ordinateur sur lequel vous allez exécuter Eclipse, à partir du site suivant : **http://java.sun.com/javase/downloads/index.jsp**.

2. ## Installation

   a. Téléchargez et extrayez **Eclipse IDE for Java EE Developers** de **http://www.eclipse.org/downloads**vers un dossier local, par exemple, **C:\Program Files\eclipse**.

   b. Copiez **com.hp.plugin.import_cmdb_model_1.0.jar** à partir de **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\bin** dans **C:\Program Files\Eclipse\plugins**.

   c. Lancez **C:\Program Files\Eclipse\eclipse.exe**. Si un message indiquant que la machine virtuelle Java est introuvable apparaît, lancez **eclipse.exe** avec la ligne de commande suivante :

   ```
   "C:\Program Files\eclipse\eclipse.exe" -vm "<dossier d'installation JRE>\
   bin"
   ```

3. ## Préparation de l'environnement de travail

   Cette étape consiste à configurer l'espace de travail, la base de données, les connexions et les propriétés des pilotes.

a. Extrayez le fichier **workspaces_gdb.zip** de **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** vers **C:\Documents and Settings\All Users.**

> **Remarque :** Vous devez utiliser le chemin exact. Si vous dézippez le fichier dans un chemin incorrect ou laissez le fichier zippé, la procédure ne fonctionnera pas.

b. Dans Eclipse, choisissez **File > Switch Workspace > Other** :

Si vous travaillez avec :

- SQL Server, sélectionnez le dossier suivant : **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.
- MySQL, sélectionnez le dossier suivant : **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
- Oracle, sélectionnez le dossier suivant : **C:\Documents and Settings\All Users\workspace_gdb_oracle**.

c. Cliquez sur **OK**.

d. Dans Eclipse, affichez la vue Project Explorer et sélectionnez **<Active project> > JPA Content > persistence.xml > <nom du projet actif> > orm.xml**.

e. Dans la vue Data Source Explorer (volet inférieur gauche), cliquez avec le bouton droit sur la connexion de base de données et sélectionnez le menu **Propriétés**.

f. Dans la boîte de dialogue **Properties for <nom de connexion>**, sélectionnez **Common** puis cochez la case **Connect every time the workbench is started**. Sélectionnez **Driver Properties** et complétez les propriétés de connexion. Cliquez sur **Test Connection** et vérifiez que la connexion fonctionne. Cliquez sur **OK**.

g. Dans la vue Data Source Explorer, cliquez avec le bouton droit sur la connexion de base de données, puis cliquez sur **Connect**. Une arborescence contenant les schémas et tables de base de données est affichée sous l'icône de la connexion de base de données.

4. ## Création d'un adaptateur

Créez un adaptateur à l'aide des instructions de la section " Étape 1 : Création d'un adaptateur " , page 28.

5. ## Configuration du plug-in CMDB

a. Dans Eclipse, cliquez sur **UCMDB > Settings** pour ouvrir la boîte de dialogue **CMDB Settings**.

b. Si ce n'est déjà fait, sélectionnez le nouveau projet JPA comme projet actif.

c. Entrez le nom d'hôte CMDB, par exemple, **localhost** ou **labm1.itdep1**. Il n'est pas nécessaire d'inclure le numéro de port ni le préfixe `http://` dans l'adresse.

d. Indiquez le nom d'utilisateur et le mot de passe permettant d'accéder à l'API CMDB, généralement **admin**/**admin**.

e. Assurez-vous que le dossier **C:\hp** sur le serveur CMDB est mappé comme lecteur réseau.

f. Sélectionnez le dossier de base de l'adaptateur concerné sous **C:\hp**. Le dossier de base est celui qui contient le fichier **dbAdapter.jar** et le sous-dossier **META-INF**. Son chemin doit être **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<nom de l'adaptateur>**. Vérifiez qu'il n'y a pas de barre oblique inverse (\) à la fin.

6. **Importation du modèle de classe UCMDB**

Dans cette étape, vous sélectionnez les types de CI à mapper comme entités JPA.

a. Cliquez sur **UCMDB > Import CMDB Class Model** pour ouvrir la boîte de dialogue **CI Types Selection**.

b. Sélectionnez les types de CI que vous comptez mapper comme entités JPA. Cliquez sur **OK**. Les types de CI sont importés en tant que classes Java. Vérifiez qu'ils apparaissent sous le dossier **src** du projet actif.

7. **Création du fichier ORM – Mappage des classes UCMDB sur des tables de base de données**

Au cours de cette étape, vous mappez les classes Java (que vous avez importées à l'étape précédente) sur les tables de base de données.

a. Assurez-vous que la connexion à la base de données est établie. Cliquez avec le bouton droit sur le projet actif (appelé myProject par défaut) dans Project Explorer. Sélectionnez la vue JPA, cochez la case **Override default schema from connection** et sélectionnez le schéma de base de données concerné. Cliquez sur **OK**.

b. Mappez un type de CI : dans la vue JPA Structure, cliquez avec le bouton droit sur la branche **Entity Mappings** et sélectionnez **Add Class**. La boîte de dialogue **Add Persistent Class** s'ouvre. Ne modifiez pas le champ **Map as** (**Entity**).

c. Cliquez sur **Browse** et sélectionnez la classe UCMDB à mapper (toutes les classes UCMDB appartiennent au package **generic_db_adapter**).

d. Cliquez sur **OK** dans les deux boîtes de dialogue. La classe sélectionnée est affichée sous la branche **Entity Mappings** dans la vue JPA Structure.

> **Remarque :** Si l'entité apparaît sans arborescence d'attributs, cliquez avec le bouton droit sur le projet actif dans la vue Project Explorer. Choisissez **Close** puis **Open**.

e. Dans la vue JPA Details, sélectionnez la table de base de données principale à laquelle la classe UCMDB doit être mappée. Ne modifiez pas les autres champs.

8. **Mappage d'ID**

Selon les normes JPA, chaque classe persistante doit comporter au moins un attribut ID. Pour les classes UCMDB, vous pouvez mapper jusqu'à trois attributs comme ID. Les attributs ID potentiels sont appelés **id1**, **id2** et **id3**.

Pour mapper un attribut ID :

a. Développez la classe correspondante sous la branche **Entity Mappings** dans la vue JPA Structure, cliquez avec le bouton droit sur l'attribut concerné (par exemple, **id1**) et sélectionnez **Add Attribute to XML and Map…**.

b. La boîte de dialogue **Add Persistent Attribute** s'ouvre. Sélectionnez **Id** dans le champ **Map as** et cliquez sur **OK**.

c. Dans la vue JPA Details, sélectionnez la colonne de table de base de données sur laquelle le champ ID doit être mappé.

9. **Mappage d'attributs**

Cette étape consiste à mapper des attributs sur les colonnes de base de données.

a. Développez la classe correspondante sous la branche **Entity Mappings** dans la vue JPA Structure, cliquez avec le bouton droit sur l'attribut concerné (par exemple, **host_hostname**) et sélectionnez **Add Attribute to XML and Map…**.

b. La boîte de dialogue **Add Persistent Attribute** s'ouvre. Sélectionnez **Basic** dans le champ **Map as** et cliquez sur **OK**.

c. Dans la vue JPA Details, sélectionnez la colonne de table de base de données sur laquelle le champ d'attribut doit être mappé.

10. **Mappage d'un lien valide**

Effectuez les opérations décrites plus haut à l'étape" Création du fichier ORM – Mappage des classes UCMDB sur des tables de base de données " , page précédente pour mapper une classe UCMDB indiquant un lien valide. Le nom de chacune des classes de ce type possède la structure suivante : **<nom d'entité end1>_<nom de lien>_<nom d'entité end2>**. Par exemple, un lien **Contains** entre un hôte et un emplacement est dénoté par une classe Java portant le nom **generic_db_adapter.host_contains_location**. Pour plus d'informations, voir " Fichier reconciliation_rules.txt (pour compatibilité rétroactive) " , page 128.

a. Mappez les attributs d'ID de la classe de lien comme décrit à l'étape " Mappage d'ID " , ci dessus. Pour chaque attribut d'ID, développez le groupe de cases à cocher **Details** dans la vue JPA Details et désactivez les cases **Insertable** et **Updateable**.

b. Mappez les attributs **end1** et **end2** de la classe de lien comme suit : Pour chacun des attributs **end1** et **end2** de la classe de lien :

- Développez la classe correspondante sous la branche **Entity Mappings** dans la vue JPA Structure, cliquez avec le bouton droit sur l'attribut concerné (par exemple, **end1**) et sélectionnez **Add Attribute to XML and Map…**.
- Dans la boîte de dialogue **Add Persistent Attribute**, sélectionnez **Many to One** ou **One to One** dans le champ **Map as**.
- Sélectionnez **Many to One** si le CI **end1** ou **end2** spécifié peut avoir plusieurs liens de ce type. Sinon, sélectionnez **One to One**. Par exemple, pour un lien **host_contains_ip**, l'extrémité **host** doit être mappée comme **Many to One**, car un hôte peut posséder plusieurs adresses IP, tandis que l'extrémité **ip** doit être mappée comme **One to One**, car une adresse IP ne peut avoir qu'un seul hôte.
- Dans la vue JPA Details, sélectionnez **Target entity**, par exemple, **generic_db_adapter.host**.
- Dans la section **Join Columns** de la vue JPA Details, cochez **Override Default**. Cliquez sur **Edit**. Dans la boîte de dialogue **Edit Join Column**, sélectionnez la colonne de clé étrangère de la table de base de données de lien qui désigne une entrée dans la table de l'entité cible **end1**/**end2**. Si le nom de colonne référencé dans la table de l'entité cible **end1**/**end2** est mappée sur son attribut ID, ne modifiez pas le champ **Referenced Column Name**. Sinon, sélectionnez le nom de la colonne désignée par la colonne de clé étrangère. Désactivez les cases à cocher **Insertable** et **Updatable**, puis cliquez sur **OK**.
- Si l'entité cible **end1**/**end2** a plusieurs ID, cliquez sur le bouton **Add** pour ajouter des colonnes de jointure supplémentaires et les mapper de la même manière qu'à l'étape précédente.

11. ## Création du fichier ORM – Utilisation de tables secondaires

JPA permet de mapper une classe Java sur plusieurs tables de base de données. Par exemple, **Host** peut être mappé sur la table **Device** pour permettre la persistance de la plupart de ses attributs et sur la table **NetworkNames** pour permettre la persistance de **host_hostName**. Dans ce cas, **Device** est la table principale et **NetworkNames**, la table secondaire. Il est possible de définir autant de tables secondaires que nécessaire. La seule condition est qu'il doit exister une relation de un à un entre les entrées des tables principale et secondaire.

12. ## Définition d'une table secondaire

Sélectionnez la classe appropriée dans la vue JPA Structure. Dans la vue **JPA Details**, accédez à la section **Secondary Tables** et cliquez sur **Add**. Dans la boîte de dialogue **Add Secondary Table**, sélectionnez la table secondaire appropriée. Ne modifiez pas les autres champs.

Si les tables principale et secondaire n'ont pas les mêmes clés principales, configurez les colonnes de jointure dans la section **Primary Key Join Columns** de la vue **JPA Details**.

13. **Mappage d'un attribut sur une table secondaire**

    Pour mapper un attribut de classe sur un champ d'une table secondaire, procédez comme suit :

    a. Mappez l'attribut comme indiqué plus haut à l'étape " Mappage d'attributs " , page 105.

    b. Dans la section **Column** de la vue JPA Details, sélectionnez le nom de la table secondaire dans le champ **Table**, en remplacement de la valeur par défaut.

14. **Utilisation d'un fichier ORM existant comme base**

    Pour utiliser un fichier **orm.xml** existant comme base de celui que vous développez, procédez comme suit :

    a. Vérifiez que tous les types de CI mappés dans le fichier **orm.xml** existant sont importés dans le projet Eclipse actif.

    b. Sélectionnez et copiez tout ou partie des mappages d'entités à partir du fichier existant.

    c. Sélectionnez l'onglet **Source** du fichier **orm.xml** dans la perspective JPA Eclipse.

    d. Collez tous les mappages d'entités copiés sous la balise **<entity-mappings>** du fichier **orm.xml** édité, après la balise **<schema>**. Assurez-vous que la balise schema est configurée comme décrit plus haut à l'étape " Création du fichier ORM – Mappage des classes UCMDB sur des tables de base de données " , page 104. Toutes les entités collées apparaissent maintenant dans la vue JPA Structure. Désormais, il est possible d'éditer les mappages graphiquement et manuellement par le biais du code xml du fichier **orm.xml**.

    e. Cliquez sur **Save**.

15. **Importation d'un fichier ORM existant à partir d'un adaptateur**

    S'il existe déjà un adaptateur, il est possible d'utiliser le plug-in Eclipse pour modifier graphiquement son fichier ORM. Importez le fichier **orm.xml** dans Eclipse, éditez-le à l'aide du plug-in puis redéployez-le sur la machine UCMDB. Pour importer le fichier ORM, clique sur le bouton approprié dans la barre d'outils Eclipse. Une boîte de dialogue de confirmation apparaît. Cliquez sur **OK**. Le fichier ORM est copié depuis la machine UCMDB dans le projet Eclipse actif et toutes les classes concernées sont importées depuis le modèle de classe UCMDB.

    Si les classes concernées n'apparaissent pas dans la vue JPA Structure, cliquez avec le bouton droit sur le projet actif dans la vue Project Explorer, choisissez **Close** puis **Open**.

    Le fichier ORM peut désormais être édité graphiquement à l'aide d'Eclipse, puis redéployé sur la machine UCMDB comme décrit à l'étape " Déploiement du fichier ORM dans la base CMDB " , page suivante.

16. **Vérification de l'exactitude du fichier orm.xml – Contrôle d'exactitude intégré**

    Le plug-in JPA Eclipse recherche les erreurs éventuelles et les marque dans le fichier **orm.xml**. Les erreurs de syntaxe (par exemple, nom de balise incorrect, balise non fermée, ID manquant) et de mappage (par exemple, nom d'attribut ou de champ de table de base de données incorrect) sont vérifiées. S'il y a des erreurs, leur description apparaît dans la vue **Problems**.

17. **Création d'un point d'intégration**

    S'il n'existe aucun point d'intégration dans la base CMDB pour cet adaptateur, vous pouvez en créer un dans le Studio d'intégration. Pour plus d'informations, voir Integration Studio dans le *Manuel de gestion des flux de données HP Universal CMDB*.

    Indiquez le nom du point d'intégration dans la boîte de dialogue qui s'ouvre. Le fichier **orm.xml** est copié dans le dossier de l'adaptateur. Un point d'intégration est créé avec tous les types de CI importés définis en tant que classes prises en charge, à l'exception des types de CI multinode, s'ils sont configurés dans le fichier **reconciliation_rules.txt**. Pour plus d'informations, voir " Fichier reconciliation_rules.txt (pour compatibilité rétroactive) " , page 128.

18. **Déploiement du fichier ORM dans la base CMDB**

    Enregistrez le fichier **orm.xml** et déployez-le sur le serveur UCMDB en cliquant sur **UCMDB > Deploy ORM**. Le fichier **orm.xml** est copié dans le dossier de l'adaptateur et ce dernier est rechargé. Le résultat de l'opération est affiché dans une boîte de dialogue **Operation Result**. Si une erreur se produit au cours du processus de rechargement, l'arborescence des appels de procédure de l'exception Java est affichée dans la boîte de dialogue. Si aucun point d'intégration n'a encore été défini à l'aide de l'adaptateur, aucune erreur de mappage n'est détectée lors du déploiement.

19. **Exécution d'un exemple de requête TQL**

    a. Définissez une requête (pas une vue) dans le Studio de modélisation. Pour plus d'informations, voir Modeling Studio dans le *Manuel de modélisation HP Universal CMDB*.

    b. Créez un point d'intégration à l'aide de l'adaptateur que vous avez créé à l'étape " Création d'un point d'intégration " , ci dessus. Pour plus d'informations, voir New Integration Point/Edit Integration Point Dialog Box dans le *Manuel de gestion des flux de données HP Universal CMDB*.

    c. Au cours de la création de l'adaptateur, vérifiez que les types de CI qui doivent être inclus dans la requête sont pris en charge par ce point d'intégration.

    d. Lors de la configuration du plug-in CMDB, utilisez le nom de cet exemple de requête dans la boîte de dialogue Paramètres. Pour plus d'informations, voir plus haut l'étape " Configuration du plug-in CMDB " , page 103.

    e. Cliquez sur le bouton **Run TWL** pour exécuter un exemple de code TQL et vérifiez s'il renvoie les résultats requis à l'aide du nouveau fichier **orm.xml**.

Fichiers de configuration de l'adaptateur

Les fichiers présentés dans cette section figurent dans le package **db-adapter.zip** du dossier **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Cette section décrit les fichiers de configuration suivants :

- " Fichier adapter.conf " , page suivante
- " Fichier simplifiedConfiguration.xml " , page 112
- " Fichier orm.xml " , page 114
- " Fichier reconciliation_types.txt " , page 128
- " Fichier reconciliation_rules.txt (pour compatibilité rétroactive) " , page 128
- " Fichier transformations.txt " , page 130
- " Fichier discriminator.properties " , page 131
- " Fichier replication_config.txt " , page 132
- " Fichier fixed_values.txt " , page 133
- " Fichier persistence.xml " , page 133

Configuration générale

- **adapter.conf.** Fichier de configuration de l'adaptateur. Pour plus d'informations, voir " Fichier adapter.conf " , page suivante.

Configuration simple

- **simplifiedConfiguration.xml**. Fichier de configuration qui remplace **orm.xml**, **transformations.txt** et **reconciliation_rules.txt** par moins de fonctionnalités. Pour plus d'informations, voir " Fichier simplifiedConfiguration.xml " , page 112.

Configuration avancée

- **orm.xml**. Fichier de mappage relationnel-objet dans lequel vous mappez des types de CI CMDB sur des tables de base de données. Pour plus d'informations, voir " Fichier orm.xml " , page 114.
- **reconciliation_rules.txt**. Contient les règles de rapprochement. Pour plus d'informations, voir " Fichier reconciliation_rules.txt (pour compatibilité rétroactive) " , page 128.
- **transformations.txt.** Fichier de transformations dans lequel vous indiquez les convertisseurs à appliquer pour convertir la valeur CMDB en valeur de base de données et vice versa. Pour plus

d'informations, voir " Fichier transformations.txt " , page 130.

- **Discriminator.properties.** Ce fichier mappe chaque type de CI pris en charge sur une liste séparée par des virgules de valeurs correspondantes possibles. Pour plus d'informations, voir " Fichier discriminator.properties " , page 131.

- **Replication_config.txt.** Ce fichier contient une liste de types de CI et de relations séparés par une virgule dont les conditions de propriété sont prises en charge par le plug-in de réplication. Pour plus d'informations, voir " Fichier replication_config.txt " , page 132

- **Fixed_values.txt.** Ce fichier permet de configurer des valeurs fixes pour des attributs spécifiques de certains types de CI. Pour plus d'informations, voir " Fichier fixed_values.txt " , page 133.

Configuration d'Hibernate

- **persistence.xml.** Utilisé pour remplacer les configurations d'Hibernate prêtes à l'emploi. Pour plus d'informations, voir " Fichier persistence.xml " , page 133.

Activation de la prise en charge de tables temporaires pour l'adaptateur

L'activation de tables temporaires permet à l'adaptateur d'utiliser plus efficacement la base de données distante, ce qui réduit la contrainte sur la base de données et améliore les performances.

Pour activer la prise en charge de tables temporaires dans l'adaptateur de base de données générique, les conditions suivantes doivent être remplies :

- Les informations d'identification fournies pour la connexion à la base de données doivent inclure l'autorisation de créer, de modifier et de supprimer des tables temporaires.

- Définissez les paramètres suivants dans le fichier de configuration adapter.conf :

  temp.tables.enabled=true

  performance.enable.single.sql=true

> **Remarque :** Les tables temporaires sont prises en charge uniquement pour Microsoft SQL et Oracle.

Fichier adapter.conf

Ce fichier contient les paramètres suivants :

- **use.simplified.xml.config=false.true** : utilise simplifiedConfiguration.xml.

  > **Remarque :** L'utilisation de ce fichier implique que les fichiers `orm.xml`,

`transformations.txt` et `reconciliation_rules.txt` soient remplacés par un nombre inférieur de fonctionnalités.

- **dal.ids.chunk.size=300.** Ne modifiez pas cette valeur.
- **dal.use.persistence.xml=false**. **true** : l'adaptateur lit la configuration Hibernate dans le fichier `persistence.xml`.

  Remarque : Il n'est pas recommandé de remplacer la configuration Hibernate.

- **performance.memory.id.filtering=true**. Lorsque GDBA exécute TQLS, il peut arriver qu'un grand nombre d'ID soit récupérés et renvoyés à la base de données en SQL. Pour éviter cette charge de travail supplémentaire et améliorer les performances, GDBA tente de lire la vue/table dans son intégralité et filtre les résultats en mémoire.
- **id.reconciliation.cmdb.id.type=string/bytes**. Lors du mappage de l'adaptateur de base de données générique à l'aide du rapprochement d'ID, vous pouvez mapper **cmdb_id** sur une colonne du type **string** ou **bytes/raw** en modifiant la propriété **META-INF/ adapter.conf**.
- **performance.enable.single.sql=true**. Ce paramètre est facultatif. S'il ne figure pas dans le fichier, sa valeur par défaut est **true**. Lorsque la valeur est **true**, l'adaptateur de base de données générique tente de générer une instruction SQL unique pour chaque requête exécutée (soit pour le remplissage, soit une requête fédérée). L'utilisation d'une instruction SQL unique améliore les performances et la consommation de mémoire de l'adaptateur de base de données générique. Lorsque la valeur est **false**, l'adaptateur de base de données générique génère plusieurs instructions SQL, ce qui implique un temps de traitement et une consommation de mémoire supérieurs. Dans les cas suivants, l'adaptateur ne génère pas d'instruction SQL unique même lorsque l'attribut a la valeur **true** :
  - La base de données à laquelle l'adaptateur se connecte n'est pas de type Oracle ou SQL Server.
  - Le code TQL en cours d'exécution contient une condition de cardinalité autre que 0..* et 1..* (par exemple, 2..* ou 0..2).
- **in.expression.size.limit=950** (par défaut). Ce paramètre divise l'expression « IN » du code SQL exécuté, lorsque la taille limite de la liste d'arguments est atteinte.
- **stringlist.delimiter.of.<Nom du type de CI>.<Nom de l'attribut>=<delimiter>**. Pour mapper un attribut de liste de type string sur une colonne de base de données dans l'adaptateur de base de données générique, l'attribut doit être mappé sur une colonne de type string contenant une liste de valeurs concaténées. Par exemple, pour mapper l'attribut **policy_ category** avec le type de CI **policy** alors que la colonne de type string contient la liste de valeurs valeur1##valeur2##valeur3 (définissant une liste de 3 valeurs, valeur1, valeur2, valeur3), utilisez le paramètre **stringlist.delimiter.of.policy.policy_category=##.**

- **temp.tables.enabled=true**. Permet d'utiliser des tables temporaires pour améliorer les performances. Ce paramètre n'est disponible que si **performance.enable.single.sql** est activé (non pris en charge dans Microsoft SQL et Oracle). Certaines autorisations peuvent être nécessaires dans le serveur de base de données.

- **temp.tables.min.value=50**. Définit le nombre de valeurs (ou ID) de condition qui sont nécessaires pour utiliser des tables temporaires.

Fichier simplifiedConfiguration.xml

Ce fichier est utilisé pour un mappage simple de classes UCMDB sur des tables de base de données. Pour accéder au modèle afin de modifier le fichier, sélectionnez **Gestion de l'adaptateur > db-adapter > Fichiers de configuration**.

Contenu de cette section :

- " Modèle de fichier simplifiedConfiguration.xml " , ci-dessous
- " Limitations " , page 114

Modèle de fichier simplifiedConfiguration.xml

- La propriété **CMDB-class-name** est le type multinode (nœud auquel les types de CI fédérés se connectent dans le code TQL) :

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
e" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="[table_name]">
        <primary-key column-name="[column_name]" />
```

- **reconciliation-by-two-nodes**. Le rapprochement peut être effectué avec un nœud ou deux nœuds. Dans cet exemple de cas, le rapprochement utilise deux nœuds.

- **connected-node-CMDB-class-name.** Deuxième type de classe nécessaire dans le code TQL de rapprochement.

- **CMDB-link-type.** Type de relation nécessaire dans le code TQL de rapprochement.

- **link-direction.** Sens de la relation dans le code TQL de rapprochement (de `node` à `ip_address` ou d'`ip_address` à `node`) :

```
<reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address" CMDB-
link-type="containment" link-direction="main-to-connected">
```

  L'expression de rapprochement se présente sous la forme de plusieurs opérateurs OR, dont chacun comprend des opérateurs AND.

- **is-ordered.** Détermine si le rapprochement est effectué sous la forme d'un ordre ou par une

comparaison `OR` normale.

```
<or is-ordered="true">
```

Si la propriété de rapprochement est extraite de la classe principale (multinode), utilisez la balise **attribute**, sinon utilisez la balise **connected-node-attribute**.

- **ignore-case.true** : lorsque les données du modèle de classe UCMDB sont comparées à celle du SGBDR, la casse n'a pas d'importance :

  ```
  <attribute CMDB-attribute-name="name" column-name="[column_name]" ignore-case=
  "true" />
  ```

  Le nom de colonne est le nom de la colonne de clé étrangère (celle dont les valeurs désignent la colonne de clé principale multinœud).

  Si la colonne de clé principale multinœud se compose de plusieurs colonnes, il doit y avoir plusieurs colonnes de clé étrangère, une pour chaque colonne de clé principale.

  ```
  <foreign-primary-key column-name="[column_name]" CMDB-class-primary-key-column
  ="[column_name]" />
  ```

  S'il y a peu de colonnes de clé principale, dupliquez cette colonne.

  ```
  <primary-key column-name="[column_name]" />
  ```

- Les propriétés **from-CMDB-converter** et **to-CMDB-converter** sont des classes Java qui implémentent les interfaces suivantes :

  - `com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB`
  - `com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB`

  Utilisez ces convertisseurs si les valeurs contenues dans CMDB et dans la base de données ne sont pas les mêmes.

  Dans cet exemple, `GenericEnumTransformer` permet de convertir l'énumérateur en fonction du fichier XML écrit entre parenthèses (**generic-enum-transformer-example.xml**) :

  ```
  <attribute CMDB-attribute-name="[CMDB_attribute_name]" column-name="[column_
  name]" from-CMDB-converter="com.mercury.topaz.fcmdb.
  adapters.dbAdapter.dal.transform.impl. GenericEnumTransformer
  (generic-enum-transformer-example.xml)" to-CMDB-onverter="com.
  mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
  GenericEnumTransformer(generic-enum-transformer-example.xml)" />
  ```

  ```
  <attribute CMDB-attribute-name="[CMDB_attribute_name]" column-name="[column_
  name]" />
  ```

```
<attribute CMDB-attribute-name="[CMDB_attribute_name]" column-name="[column_
name]" />

 </class>

</generic-DB-adapter-config>
```

Limitations

- Peut servir à mapper uniquement les requêtes TQL contenant un seul nœud (dans la source de la base de données). Par exemple, vous pouvez exécuter une requête TQL `node > ticket` et une requête TQL `ticket`. Pour extraire la hiérarchie des nœuds de la base de données, vous devez utiliser le fichier **orm.xml** avancé.
- Seules les relations un à plusieurs sont prises en charge. Par exemple, vous pouvez extraire une ou plusieurs alertes sur chaque nœud. Vous ne pouvez pas extraire des alertes qui appartiennent à plusieurs nœuds.
- Vous ne pouvez pas connecter la même classe à des types de CI CMDB différents. Par exemple, si vous définissez que `ticket` est connecté à `node`, il ne peut pas être connecté également à `application`.

Fichier orm.xml

Ce fichier permet de mapper des types de CI CMDB sur des tables de base de données.

Un modèle permettant de créer un fichier figure dans le répertoire **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF**.

Pour éditer le fichier XML pour un adaptateur déployé, accédez à **Gestion de l'adaptateur > db-adapter > Fichiers de configuration**.

Contenu de cette section :

- " Modèle de fichier orm.xml "
- " Plusieurs fichiers ORM " , page 118
- " Conventions de dénomination " , page 118
- " Utilisation d'instructions SQL en ligne au lieu de noms de table " , page 118
- " Schéma orm.xml " , page 119
- " Exemple de création du fichier orm.xml : " , page 124
- " Configuration d'un fichier orm.xml spécifique pour chaque version de produit distant " , page 128

Modèle de fichier orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xs
i="http://www.w3.org/2001/XMLSchema-instance" version="1.0" xsi:schemaLocati
on=
"http://java.sun.com/xml/ns/persistence/orm http://java.sun.com/xml/ns/persi
stence/orm_1_0.xsd">
    <description>Generic DB adapter orm</description>
```

Ne modifiez pas le nom du package.

```
    <package>generic_db_adapter</package>
```

entity. Nom du type de CI CMDB. Il s'agit de l'entité multinode.

Assurez-vous que **class** comprend un préfixe **generic_db_adapter.**.

```
    <entity class="generic_db_adapter.node">
        <table name="[table_name]" />
```

Utilisez une table secondaire si l'entité est mappée sur plusieurs tables.

```
        <secondary-table name="" />
        <attributes>
```

Pour un héritage de table unique avec discriminateur, utilisez le code suivant :

```
        <inheritance strategy="SINGLE_TABLE" />
        <discriminator-value>node</discriminator-value>
        <discriminator-column name="[column_name]" />
```

Les attributs avec la balise **id** sont les colonnes de clé principale. Assurez-vous que la convention de dénomination de ces colonnes de clé principale est **idX** (id1, id2, etc.), où **X** est l'index de colonne dans la clé principale.

```
            <id name="id1">
```

Ne modifiez que le nom de colonne de la clé principale.

```
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE"/>
            </id>
```

basic. Sert à déclarer les attributs CMDB. Veillez à ne modifier que les propriétés **name** et **column_name**.

```
            <basic name="name">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
            </basic>
```

Pour un héritage de table unique avec discriminateur, mappez les classes d'extension comme suit :

```
    <entity name="[cmdb_class_name]" class="generic_db_adapter.nt" name="nt">

        <discriminator-value>nt</discriminator-value>
        <attributes>
    </entity>
    <entity class="generic_db_adapter.unix" name="unix">
        <discriminator-value>unix</discriminator-value>
        <attributes>
    </entity>
    <entity name="[CMDB_class_name]" class="generic_db_adapter.[CMDB[cmdb_cl
ass_name]">
        <table name="[default_table_name]" />
        <secondary-table name="" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE"/>
            </id>
```

L'exemple suivant présente un nom d'attribut CMDB sans préfixe :

```
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
            </basic>
            <basic name="[CMDB_attribute_name]">
```

```
                <column updatable="false" insertable="false" name="[column_n
ame]" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
            </basic>
        </attributes>
    </entity>
```

Il s'agit d'une entité de relation. La convention de dénomination est **end1Type_linkType_ end2Type**. Dans cet exemple, **end1Type** a la valeur **node** et **linkType**, la valeur **composition**.

```
    <entity name="node_composition_[CMDB_class_name]" class="generic_db_adap
ter.node_composition_[CMDB_class_name]">
        <table name="[default_table_name]" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE"/>
            </id>
```

L'entité cible est l'entité désignée par cette propriété. Dans cet exemple, **end1** est mappé sur l'entité **node**.

many-to-one. De nombreuses relations peuvent être connectées à un nœud.

join-column. Colonne qui contient les ID **end1** (ID d'entité cible).

referenced-column-name. Nom de colonne dans l'entité cible (**node**) qui contient les ID utilisés dans la colonne de jointure.

```
            <many-to-one target-entity="node" name="end1">
                <join-column updatable="false" insertable="false" referenced-
column-name="[column_name]" name="[column_name]" />
            </many-to-one>
```

one-to-one. Une relation peut être connectée à un **[CMDB_class_name]**.

```
            <one-to-one target-entity="[CMDB_class_name]" name="end2">
                <join-column updatable="false" insertable="false" referenced-
column-name="" name="[column_name]" />
            </one-to-one>
        </attributes>
    </entity>
</entity-mappings>
```

node attribute. Cet exemple montre comment ajouter un attribut de nœud.

```
<entity class="generic_db_adapter.host_node">
  <discriminator-value>host_node</discriminator-value>
    <attributes/>
</entity>
<entity class="generic_db_adapter.nt">
  <discriminator-value>nt</discriminator-value>
    <attributes>
      <basic name="nt_servicepack">
        <column updatable="false" insertable="false" name="specific_type_value"/>
      </basic>
    </attributes>
</entity>
```

Plusieurs fichiers ORM

Plusieurs fichiers de mappage sont pris en charge. Le nom de chaque fichier de mappage doit se terminer par **orm.xml**. Tous les fichiers de mappage doivent être placés dans le dossier META-INF de l'adaptateur.

Conventions de dénomination

- Dans chaque entité, la propriété de classe doit correspondre à la propriété de nom portant le préfixe `generic_db_adapter`.
- Les colonnes de clé principale doivent porter des noms de la forme **idX**, où **X = 1**, **2**, ..., selon le nombre de clés principales contenues dans la table.
- Les noms d'attribut doivent correspondre aux noms d'attribut de classe, même en matière de casse.
- Le nom de relation prend la forme `end1Type_linkType_end2Type`.
- Les types de CI CMDB , qui sont aussi des mots réservés dans Java, doivent être préfixés par **gdba_**. Par exemple, pour le type de CI CMDB **goto**, l'entité ORM doit porter le nom **gdba_goto**.

Utilisation d'instructions SQL en ligne au lieu de noms de table

Vous pouvez mapper des entités sur des clauses `select` en ligne au lieu de tables de base de données. Cela équivaut à définir une vue dans la base de données et à mapper une entité sur cette vue. Par exemple :

```
    <entity class="generic_db_adapter.node">
        <table name="(select d.id as id1, d.name as name , d.os as host_os f
rom
Device d)" />
```

Dans cet exemple, les attributs de nœud doivent être mappés sur les colonnes id1, name et host_os, au lieu de id, name et os.

Les limites suivantes s'appliquent :

- L'instruction SQL en ligne n'est disponible que lorsque Hibernate est utilisé comme fournisseur JPA.
- Les parenthèses qui entourent la clause SQL en ligne select sont obligatoires.
- L'élément **<schema>** ne doit pas figurer dans le fichier **orm.xml**. Dans le cas de Microsoft SQL Server 2005, cela signifie que tous les noms de table doivent porter le préfixe `dbo.`, au lieu d'être définis globalement par `<schema>dbo</schema>`.

Schéma orm.xml

Le tableau suivant décrit les éléments communs du fichier **orm.xml**. Pour obtenir le schéma complet, voir http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. La liste n'est pas complète et explique principalement le comportement particulier de l'API de persistance Java standard pour l'adaptateur de base de données générique.

Nom et chemin d'accès de l'élément	Description	Attributs
entity-mappings	Élément racine pour le document de mappage d'entité. Cet élément doit être exactement identique à celui qui est indiqué dans les exemples de fichiers GDBA.	
description (entity-mappings)	Description en texte libre du document de mappage d'entité. (Facultatif)	

Nom et chemin d'accès de l'élément	Description	Attributs
package (entity-mappings)	Nom du package Java qui va contenir les classes de mappage. Doit toujours contenir le texte `generic_db_adapter`.	1. **Nom :** name **Description :** Nom du type de CI UCMDB sur lequel cette entité est mappée. Si cette entité est mappée sur un lien dans CMDB, son format doit être le suivant : `<end_1>_<link_name>_<end_2>`. Par exemple, `node_composition_cpu` définit une entité qui sera mappée sur le lien de composition entre un nœud et une UC. Si le nom du type de CI est le même que celui de la classe Java sans le préfixe du package, ce champ peut être omis. **Obligatoire ? :** Optional **Type :** String 2. **Nom :** class **Description :** Nom complet de la classe Java qui sera créée pour cette entité de base de données. Le nom du package de la classe Java doit être le même que le nom donné dans l'élément `package`. Vous ne pouvez pas utiliser de mots réservés Java, tels qu'interface ou switch, comme nom de classe. Ajoutez plutôt le préfixe `gdba_` au nom (ainsi, interface devient `generic_db_adapter.gdba_interface`. **Obligatoire ? :** Required **Type :** String

Nom et chemin d'accès de l'élément	Description	Attributs
table (entity-mappings>entity)	Cet élément définit la table principale de l'entité de BD. Ne peut apparaître qu'une seule fois. Obligatoire.	**Nom :** name **Description :** Nom de la table principale. Si le nom de la table ne contient pas le schéma auquel elle appartient, une recherche sur la table est effectuée uniquement dans le schéma de l'utilisateur ayant servi à créer le point d'intégration. Il peut également s'agir d'une instruction SELECT valide. S'il s'agit d'une instruction SELECT, elle doit être encapsulée entre parenthèses. **Obligatoire ? :** Required **Type :** String
secondary-table (entity-mappings > entity)	Cet élément peut être utilisé pour définir une table secondaire pour l'entité de BD. Cette table doit être connectée à la table principale par une relation de un à un. Vous pouvez définir plusieurs tables secondaires. Facultatif.	**Nom :** name **Description :** Nom de la table secondaire. Si le nom de la table ne contient pas le schéma auquel elle appartient, une recherche sur la table est effectuée uniquement dans le schéma de l'utilisateur ayant servi à créer le point d'intégration. Il peut également s'agir d'une instruction SELECT valide. S'il s'agit d'une instruction SELECT, elle doit être encapsulée entre parenthèses. **Obligatoire ? :** Required **Type :** String
primary-key-join-column (entity-mappings > entity > secondary-table)	Si la table secondaire et la table principale ne sont pas connectées par des champs portant le même nom, cet élément définit le nom du champ de clé principale dans la table secondaire qui doit être connecté au champ de clé principale de la table principale.	**Nom :** name **Description :** Nom du champ de clé principale dans la table secondaire. Si cet élément n'existe pas, on suppose que ce champ porte le même nom que le champ de clé principale de la table principale. **Obligatoire ? :** Optional **Type :** String

Nom et chemin d'accès de l'élément	Description	Attributs
inheritance (entity-mappings > entity)	Si l'entité actuelle est l'entité parente d'une famille d'entités de BD, utilisez cet élément pour la marquer comme telle. Facultatif.	**Nom :** strategy **Description :** Définit le mode d'implémentation de l'héritage dans votre base de données. **Obligatoire ? :** Required **Type :** L'une des valeurs suivantes : • SINGLE_TABLE : Cette entité et toutes les entités enfants existent dans la même table. • JOINED : Les entités enfants sont dans des tables jointes. • TABLE_PER_CLASS : Chaque entité est complètement définie par une table distincte.
discriminator-column (entity-mappings > entity)	Si l'héritage est de type SINGLE_TABLE, cet élément permet de définir le nom du champ servant à déterminer le type d'entité pour chaque ligne.	**Nom :** name **Description :** Nom de la colonne du discriminateur. **Obligatoire ? :** Required **Type :** String
discriminator-value (entity-mappings > entity)	Cet élément définit le type de l'entité spécifique dans l'arborescence d'héritage. Ce nom doit être identique à celui qui est défini dans le fichier **discriminator.properties** pour le groupe de valeurs de ce type d'entité spécifique.	
attributes (entity-mappings > entity)	Élément racine pour tous les mappages d'attributs d'une entité.	

Nom et chemin d'accès de l'élément	Description	Attributs
id (entity-mappings>entity attributes)	Cet élément définit le champ de clé pour l'entité. Au moins un champ d'ID doit être défini. S'il existe plusieurs éléments d'ID, leurs champs créent une clé composée pour l'entité. Essayez d'éviter les clés composées pour les entités CI (pas pour les liens).	**Nom :** name **Description :** Chaîne de type idX, où X est un chiffre compris entre 1 et 9. Le premier ID doit être marqué comme id1, le deuxième comme id2, etc. Il NE S'AGIT PAS du nom de l'attribut de clé dans UCMDB. **Obligatoire ? :** Required **Type :** String
basic (entity-mappings>entity attributes)	Cet élément définit un mappage entre un champ de la table, qui ne fait pas partie de la clé principale de la table, et un attribut UCMDB.	**Nom :** name **Description :** Nom de l'attribut UCMDB sur lequel le champ est mappé. Cet attribut doit exister dans le type de CI UCMDB sur lequel l'entité actuelle est mappée. **Obligatoire ? :** Required **Type :** String
column (entity-mappings>entity> attributes> id -OU- (entity-mappings > entity > attributes > basic)	Définit le nom de la colonne dans la table pour le mappage de base ou un champ d'ID.	1. **Nom :** name **Description :** Nom du champ. **Obligatoire ? :** Required **Type :** String 2. **Nom :** table **Description :** Nom de la table à laquelle le champ appartient. Il doit s'agir soit de la table principale, soit de l'une des tables secondaires définies pour l'entité. Si cet attribut est omis, on suppose que le champ appartient à la table principale. **Obligatoire ? :** Optional **Type :** String

Nom et chemin d'accès de l'élément	Description	Attributs
one-to-one (entity-mappings > entity > attributes)	Définit une colonne dont la valeur se trouve dans une autre table, les deux tables étant connectées par une relation de un à un. Cet élément n'est pris en charge que pour les mappages d'entités de lien, pas pour les autres types de CI. C'est la seule façon de définir un mappage entre une table et un lien UCMDB.	1. **Nom :** name **Description :** Indique laquelle des deux extrémités est représentée par ce champ. **Obligatoire ? :** Required **Type :** `end1` ou `end2` 2. **Nom :** target-entity **Description :** Nom de l'entité à laquelle l'extrémité se rapporte. **Obligatoire ? :** Required **Type :** L'un des noms d'entité définis dans le document de mappage d'entités.
join-column (entity-mappings > entity attributes > one-to-one)	Définit le mode de jointure entre le target-entity défini dans l'élément one-to-one parent et l'entité actuelle.	1. **Nom :** name **Description :** Nom du champ de la table actuelle qui sera utilisé pour effectuer la jointure de un à un. **Obligatoire ? :** Required **Type :** String 2. **Nom :** name **Description :** Nom d'un champ de l'entité jointe sur lequel effectuer la jointure. Si cet attribut est omis, on suppose que la table jointe possède une colonne portant le même nom que le champ défini dans l'attribut de nom. **Obligatoire ? :** Optional **Type :** String

Exemple de création du fichier orm.xml :

L'exemple ci-dessous montre comment créer le **fichier orm.xml**. Dans cet exemple, les tables SQL d'une base de donnée distante sont mappées sur les types de CI de l'UCMDB.

Compte tenu des tables du format suivant dans la base de données distante, il convient de remplir la table **Hosts** avec des nœuds et la table **IP_Addresses** avec des adresses IP, ainsi que de créer des liens entre les nœuds et les adresses IP comme suit :

Table Hosts

host_name	host_id
Test1	1
Test2	2
Test3	3

Table IP_Addresses

ip_address	ip_id
10.1.1.1	1
10.2.2.2	2
10.3.3.2	3
10.4.4.4	4

Table Host_IP_Link (liens entre les nœuds et les adresses IP)

host_id	ip_id
1	1
2	2
2	3
3	4

La clé primaire de la table **Hosts** correspond au champ **host_id**, tandis que dans la table **IP_ Addresses Table**, elle se rapporte au champ **ip_id**. Dans la table **Host_IP_Link**, **host_id** et **ip_id** sont des clés étrangères provenant des tables **Hosts** et **IP_Addresses Table**.

À partir des tables ci-dessus, créez le fichier **orm.xml** selon les étapes ci-après : Les entités utilisées dans cet exemple sont **node**, **ip_address** et **node_containment_ip_address**.

1. Créez l'entité **node** en mappant **host_id** à partir de la table **Hosts** comme suit :

```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
       /orm_1_0.xsd">
    <description>test_integration</description>
    <package>generic_db_adapter</package>
```

```
    <entity class="generic_db_adapter.node">
       <table name="Hosts"/>
       <attributes>
        <id name="id1">
                <column updatable="false" insertable="false" name=
                       "host_id"/>
                <generated-value strategy="TABLE"/>
        </id>
        <basic name="name">
                <column updatable="false" insertable="false" name=
                       "host_name"/>
        </basic>
       </attributes>
    </entity>
```

La **classe** d'entité doit être un type de CI déjà existant dans l'UCMDB. Le **nom** de la table correspond à la table au sein de la base de données qui contient à la fois les informations d'ID et d'hôte. L'attribut ID est obligatoire pour identifier les hôtes spécifiques et est utilisé ultérieurement lors du mappage. Dans cet exemple, l'attribut **name** de cette entité est rempli par la colonne **host_name** de la table hosts.

2. Pour l'entité suivante, mappez les adresses IP provenant de la table Interfaces :

```
<entity name="ip_address" class="generic_db_adapter.ip_address">
    <table name="IP_Addresses"/>
    <attributes>
       <id name="id1">
        <column insertable="false" updatable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
       </id>
       <basic name="name">
        <column updatable="false" insertable="false" name="ip_address"/>
       </basic>
    </attributes>
```

```
</entity>
```

3. Il s'agit ensuite de créer le lien entre le nœud et l'adresse IP via la table de mappage et de référencer le champ **ip_id** (bien qu'il puisse référencer les deux champs **host_id** et **ip_id** au besoin).

```
<entity name="node_containment_ip_address"
        class="generic_db_adapter.node_containment_ip_address">
    <table name="Host_IP_Link"/>
    <attributes>
       <id name="id1">
        <column updatable="false" insertable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
       </id>
       <many-to-one target-entity="node" name="end1">
        <join-column name="host_id"/>
       </many-to-one>
       <one-to-one target-entity="ip_address" name="end2">
        <join-column name="ip_id"/>
       </one-to-one>
    </attributes>
</entity>
```

Le nom d'entité du conteneur présente le format suivant : [end1 CIT]_[link CIT]_[end2 CIT]. Par conséquent, dans cet exemple, puisque le type de CI du lien est **containment**, le nom d'entité du conteneur est : **node_containment_ip_address** et la classe d'entité, **generic_db_adapter.node_containment_ip_address**. L'ID est obligatoire dans ce bloc de code. En outre, bien que cet exemple utilise un seul ID de la table Interface, les deux colonnes pourraient renvoyer à id1 et id2. Le code correspondant serait alors le suivant :

```
<id name=”id1”>
    <column updatable=”false” insertable=”false” name=”ip_id”/>
    <generated-value strategy=”TABLE”/>
</id>
```

```
<id name=”id2”>
    <column updatable=”false” insertable=”false” name=”host_id”/>
    <generated-value strategy=”TABLE”/>
</id>
```

Les deux extrémités du lien sont "many-to-one" et "one-to-one" : chaque adresse IP sera liée à un nœud, mais un nœud peut être lié à plusieurs adresses IP. Les colonnes comprises proviennent de la table Liens et renvoient aux tables Hosts et Interfaces.

Configuration d'un fichier orm.xml spécifique pour chaque version de produit distant

Il est possible de configurer un fichier **orm.xml** spécifique de sorte que l'adaptateur l'utilise pour une version donnée d'un produit distant. Par exemple, si le magasin de données distant comprend deux versions x et y d'un produit, le mappage des entités peut être différent pour chacune de ces versions.

Pour configurer un fichier orm.xml spécifique pour chaque version de produit distant :

1. Ajoutez au fichier **adapter.xml** un paramètre nommé **version** et spécifiez **valid-values** comme valeurs de version possibles.

2. Dans le package de l'adaptateur, sous le dossier META-INF, créez un dossier nommé **VersionOrm**.

3. Dans le dossier **VersionOrm**, créez un fichier **orm.xml** pour chaque version spécifique. Le nom de fichier doit contenir le préfixe de la version. Par exemple, si la version est appelée **x**, le nom de fichier doit être **x_orm.xml**.

Remarque : Le fichier **orm.xml** dans le dossier META-INF est chargé pour n'importe quelle version de produit distant, qu'un fichier **orm.xml** soit créé ou non pour une version de produit distant. Il peut comprendre des entités mappées de façon identique pour toutes les versions.

Fichier reconciliation_types.txt

À partir de UCMDB 10.00, le fichier **reconciliation_types.txt** n'est plus approprié. Les types de CI peuvent être utilisés pour le rapprochement. Le moteur de fédération exécute automatiquement le mappage.

Fichier reconciliation_rules.txt (pour compatibilité rétroactive)

Ce fichier permet de configurer les règles de rapprochement si vous souhaitez procéder à un rapprochement lorsque DBMappingEngine est configuré dans l'adaptateur. Si vous n'utilisez pas

DBMappingEngine, le mécanisme de rapprochement UCMDB générique est utilisé et il n'est pas nécessaire de configurer ce fichier.

Chaque ligne du fichier représente une règle. Par exemple :

```
multinode[node] expression[^node.name OR ip_address.name] end1_type[node]
end2_type[ip_address] link_type[containment]
```

Le multinœud est complété avec le nom du multinœud (type de CI CMDB qui est connecté au type de CI de la base de données fédérée dans la requête TQL).

Cette expression inclut la logique qui décide si deux multinœuds sont égaux (un multinœud dans CMDB et l'autre dans la source de base de données).

L'expression se compose d'opérateurs `OR` ou `AND`.

La convention relative aux noms d'attribut dans la partie expression est `[className].[attributeName]`. Par exemple, `attributeName` dans la classe `ip_address` est écrit sous la forme `ip_address.name`.

Pour une correspondance ordonnée (si la première sous-expression `OR` renvoie une réponse indiquant que les multinœuds ne sont pas égaux, la deuxième sous-expression OR n'est pas comparée), utilisez `ordered expression` au lieu d'`expression`.

Pour ignorer la casse au cours d'une comparaison, utilisez le signe de contrôle (**^**).

Les paramètres `end1_type`, `end2_type` et `link_type` ne sont utilisés que si la requête TQL de rapprochement contient deux nœuds au lieu d'un simple multinœud. Dans ce cas, la requête TQL de rapprochement est `end1_type > (link_type) > end2_type`.

Il n'est pas nécessaire d'ajouter la mise en page associée, car elle est extraite de l'expression.

Types de règles de rapprochement

Les règles de rapprochement prennent la forme de conditions OR et AND. Vous pouvez définir ces règles sur plusieurs nœuds différents (par exemple, le nœud est identifié par `name from nodeAND/ORname from ip_address`).

Les options suivantes recherchent une correspondance :

- **Correspondance ordonnée.** L'expression de rapprochement est lue de gauche à droite. Deux sous-expressions `OR` sont considérées comme égales si elles comportent des valeurs et sont égales. Deux sous-expressions `OR` sont considérées comme différentes si elles comportent des valeurs et ne sont pas égales. Dans tous les autres cas, aucune décision n'est prise et la sous-expression `OR` suivante fait l'objet d'un test d'égalité.

  name from node OR from ip_address. Si CMDB et la source de données contiennent `name` et sont égales, les nœuds sont considérés comme égaux. Si les deux contiennent `name` mais ne sont pas égales, les nœuds sont considérés comme différents sans que l'élément `name` de `ip_address` soit testé. S'il manque `name of node` dans CMDB ou la source de données, `name of ip_address` est vérifié.

- **Correspondance normale.** Si l'une des sous-expressions `OR` présente une égalité, CMDB et la source de données sont considérées comme égales.

name from node OR from ip_address. S'il n'y a aucune correspondance sur `name of node`, `name of ip_address` fait l'objet d'un contrôle d'égalité.

Pour les rapprochements complexes, dans lesquels l'entité de rapprochement est modélisée dans le modèle de classe sous la forme de plusieurs types de CI avec des relations (comme `node`), le mappage d'un nœud de sur-ensemble comprend tous les attributs associés de tous les types de CI modélisés.

Remarque : En conséquence, il existe une limite selon laquelle tous les attributs de rapprochement de la source de données doivent résider dans des tables qui partagent la même clé principale.

Une autre limite indique que la requête TQL de rapprochement ne doit pas comporter plus de deux nœuds. Par exemple, la requête TQL `node > ticket` comporte un nœud dans CMDB et une alerte dans la source de données.

Pour rapprocher les résultats, `name` doit être extrait du nœud et/ou d'`ip_address`.

Si l'élément `name` dans CMDB présente le format `*.m.com`, il est possible d'utiliser un convertisseur de CMDB vers la base de données fédérée, et vice versa, pour convertir ces valeurs.

La colonne `node_id` dans la table des alertes de la base de données permet une connexion entre les entités (l'association définie peut également être effectuée dans une table de nœuds) :

DB Node	
PK	node_id
	name

DB IP_Address	
PK	ip_id
	name

DB Ticket	
PK	ticket_id
	node_id

Remarque : Les trois tables doivent appartenir à la source du SGBDR fédéré et pas à la base de données CMDB.

Fichier transformations.txt

Ce fichier contient toutes les définitions de convertisseurs.

Son format est que chaque ligne contient une nouvelle définition.

Modèle de fichier transformations.txt

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]] to_DB_class[com.mercu
ry.topaz.fcmdb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

entity. Nom de l'entité tel qu'il apparaît dans le fichier `orm.xml`.

attribute. Nom de l'attribut tel qu'il apparaît dans le fichier `orm.xml`.

to_DB_class. Nom complet d'une classe qui implémente l'interface **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerTo. ExternalDB**. Les éléments entre parenthèses sont attribués à ce constructeur de classe. Utilisez ce convertisseur pour transformer des valeurs CMDB en valeurs de base de données, par exemple, pour ajouter le suffixe **.com** à la fin de chaque nom de nœud.

from_DB_class. Nom complet d'une classe qui implémente l'interface **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform. FcmdbDalTransformerFromExternalDB**. Les éléments entre parenthèses sont attribués à ce constructeur de classe. Utilisez ce convertisseur pour transformer des valeurs de base de données en valeurs CMDB, par exemple, pour ajouter le suffixe **.com** à la fin de chaque nom de nœud.

Pour plus d'informations, voir " Convertisseurs prêts à l'emploi " , page 135.

Fichier discriminator.properties

Ce fichier mappe chaque type de CI pris en charge (également utilisé comme valeur de discriminateur dans orm.xml) sur une liste séparée par des virgules des valeurs correspondantes possibles de la colonne de discriminateur, ou sur une condition permettant d'établir une correspondance avec les valeurs admises de la colonne de discriminateur.

Dans le cas de l'utilisation d'une condition, utilisez la syntaxe suivante : `like(condition)`, où `condition` désigne une chaîne pouvant contenir les caractères génériques suivants :

- **%** (signe pourcentage) - permet de rechercher une correspondance avec toute chaîne de toute longueur (y compris une chaîne vide)
- **_** (trait de soulignement) - permet de rechercher une correspondance avec un seul caractère

Par exemple, `like(%unix%)` correspond à unix, linux, unix-aix, etc. Les conditions like s'appliquent uniquement aux colonnes de chaîne.

Vous pouvez également mapper une valeur unique de discriminateur sur toute valeur qui n'appartient pas à un autre discriminateur grâce à l'instruction `'all-other'`.

Si l'adaptateur que vous créez utilise des fonctionnalités de discriminateur, vous devez définir toutes les valeurs de discriminateur dans le fichier **discriminator.properties**.

Exemple de mappage de discriminateur :

Par exemple, l'adaptateur prend en charge les types de CI `node`, `nt` et `unix`, tandis que la base de données contient une table unique nommée `t_nodes` dans laquelle figure une colonne **type**. Si le

type est 10001, la ligne représente un nœud ; si le type a la valeur 10004, la ligne correspond à un ordinateur Unix, etc. Le fichier **discriminator.properties** peut présenter l'aspect suivant :

```
node=10001, 10005
nt=10002,10003
unix=2%
mainframe=all-other
```

Le fichier **orm.xml** comprend le code suivant :

```
<entity class="generic_db_adapter.node" >
    <table name="t_nodes" />
    ...
    <inheritance strategy="SINGLE_TABLE" />
    <discriminator-value>node</discriminator-value>
    <discriminator-column name="type" />
    ...
</entity>
<entity class="generic_db_adapter.nt" name="nt">
    <discriminator-value>nt</discriminator-value>
    <attributes>
</entity>
<entity class="generic_db_adapter.unix" name="unix">
    <discriminator-value>unix</discriminator-value>
    <attributes>
</entity>
```

L'attribut discriminator_column est ensuite calculé comme suit :

- Si la colonne **type** contient `10002` ou `10003` pour une certaine entrée, celle-ci est mappée sur le type de CI **nt**.
- Si la colonne **type** contient `10001` ou `10005` pour une certaine entrée, celle-ci est mappée sur le type de CI **node**.
- Si la colonne **type** commence par `2` pour une certaine entrée, celle-ci est mappée sur le type de CI **unix**.
- Toute autre valeur contenue dans la colonne **type** est mappée sur le type de CI **mainframe**.

Remarque : Le type de CI **node** est également le parent de **nt** et **unix**.

Fichier replication_config.txt

Ce fichier contient une liste de types de CI et de relations séparés par une virgule dont les conditions de propriété sont prises en charge par le plug-in de réplication. Pour plus d'informations, voir " Plug-in " , page 140.

Fichier fixed_values.txt

Ce fichier permet de configurer des valeurs fixes pour des attributs spécifiques de certains types de CI. Ainsi, il est possible d'attribuer à chacun de ces attributs une valeur fixe qui n'est pas stockée dans la base de données.

Le fichier doit contenir zéro entrée ou plus au format suivant :

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

Par exemple :

```
entity[ip_address] attribute[ip_domain] value[DefaultDomain]
```

Le fichier prend également en charge une liste de constantes. Pour définir une liste de constantes, suivez la syntaxe ci-après :

```
entity[<entityName>] attribute[<attributeName>] value[{<Val1>, <Val2>, <Val3>, .
.. }]
```

Fichier persistence.xml

Ce fichier permet de remplacer les paramètres Hibernate par défaut et d'ajouter la prise en charge des types de base de données qui ne sont pas prêts à l'emploi (les types prêts à l'emploi étant Oracle Server, Microsoft SQL Server et MySQL).

Si vous devez prendre en charge un nouveau type de base de données, veillez à fournir un fournisseur de pool de connexions (par défaut, `c3p0`) et un pilote JDBC pour votre base de données (placez les fichiers `*.jar` dans le dossier de l'adaptateur).

Pour connaître toutes les valeurs Hibernate disponibles qu'il est possible de modifier, consultez la classe **org.hibernate.cfg.Environment** (pour plus d'informations, voir http://www.hibernate.org.)

Exemple de fichier persistence.xml :

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.
0">
    <!-- Don't change this value -->
    <persistence-unit name="GenericDBAdapter">
        <properties>
            <!-- Don't change this value -->
            <property name="hibernate.archive.autodetection" value="class,
               hbm" />
            <!--The driver class name"/-->
            <property name="hibernate.connection.driver_class" value="com.me
rcury.
               jdbc.MercOracleDriver" />
```

```
            <!--The connection url"/-->
            <property name="hibernate.connection.url" value="jdbc:mercury:or
acle:
               //artist:1521;sid=cmdb2" />
            <!--DB login credentials"/-->
            <property name="hibernate.connection.username" value="CMDB" />
            <property name="hibernate.connection.password" value="CMDB" />
            <!--connection pool properties"/-->
            <property name="hibernate.c3p0.min_size" value="5" />
            <property name="hibernate.c3p0.max_size" value="20" />
            <property name="hibernate.c3p0.timeout" value="300" />
            <property name="hibernate.c3p0.max_statements" value="50" />
            <property name="hibernate.c3p0.idle_test_period" value="3000" />
            <!--The dialect to use-->
            <property name="hibernate.dialect" value="org.hibernate.dialect.
               OracleDialect" />
        </properties>
    </persistence-unit>
</persistence>
```

Connexion à une base de données à l'aide de l'authentification NT

Il est possible de se connecter à un serveur MS SQL qui nécessite une authentification NT. Pour cela, vous devez disposer d'un pilote capable d'analyser un domaine (par exemple, le pilote jTDS JDBC).

L'authentification est effectuée en fonction des paramètres fournis (domaine, nom d'utilisateur, mot de passe) et non à l'aide des informations d'identification NT du processus d'exécution en cours.

1. Dans le fichier **persistence.xml**, modifiez les propriétés suivantes :

```
<!--The driver class name"/-->
<property name="hibernate.connection.driver_class"
value="net.sourceforge.jtds.jdbc.Driver"/>
<property name="hibernate.connection.url" value="jdbc:jtds:sqlserver://[host name]:
[port];DatabaseName=[database name];domain=[the domain]"/>
<!--DB login credentials"/-->
<property name="hibernate.connection.username" value="[username]"/>
<property name="hibernate.connection.password" value="[password]"/>
```

2. Placez le fichier du pilote JDBC sous : **<dossier d'installation de la sonde>\lib**.

3. Redémarrez la sonde.

Convertisseurs prêts à l'emploi

Vous pouvez utiliser les convertisseurs (transformateurs) suivants pour convertir des requêtes fédérées et des travaux de réplication vers et depuis des données de base de données.

Contenu de cette section :

- " Convertisseurs prêts à l'emploi " , ci dessus
- " Convertisseur SuffixTransformer " , page 138
- " Convertisseur PrefixTransformer " , page 138
- " Convertisseur BytesToStringTransformer " , page 139
- " Convertisseur StringDelimitedListTransformer " , page 139
- " Convertisseur personnalisé " , page 139

Convertisseur enum-transformer

Ce convertisseur utilise un fichier XML indiqué comme paramètre d'entrée.

Le fichier XML mappe des valeurs CMDB figées dans le code sur des valeurs de base de données (enums). Si l'une des valeurs n'existe pas, vous pouvez choisir de renvoyer la même valeur, de renvoyer la valeur null ou de lever une exception.

Le transformateur effectue une comparaison entre deux chaînes en respectant la casse ou non. Par défaut, la méthode utilisée est sensible à la casse. Pour la rendre non sensible à la casse, entrez : `case-sensitive="false"` dans l'élément enum-transformer.

Utilisez un fichier de mappage XML pour chaque attribut d'entité.

> **Remarque :** Ce convertisseur peut être utilisé pour les champs `to_DB_class` et `from_DB_class` dans le fichier **transformations.txt**.

Fichier XSD d'entrée :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="q
ualified" attributeFormDefault="unqualified">
  <xs:element name="enum-transformer">
    <xs:complexType>
        <xs:sequence>
           <xs:element ref="value" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="db-type" use="required">
```

```
          <xs:simpleType>
              <xs:restriction base="xs:string">
                  <xs:enumeration value="integer"/>
                  <xs:enumeration value="long"/>
                  <xs:enumeration value="float"/>
                  <xs:enumeration value="double"/>
                  <xs:enumeration value="boolean"/>
                  <xs:enumeration value="string"/>
                  <xs:enumeration value="date"/>
                  <xs:enumeration value="xml"/>
                  <xs:enumeration value="bytes"/>
             </xs:restriction>
          </xs:simpleType>
       </xs:attribute>
       <xs:attribute name="cmdb-type" use="required">
          <xs:simpleType>
             <xs:restriction base="xs:string">
                 <xs:enumeration value="integer"/>
                 <xs:enumeration value="long"/>
                 <xs:enumeration value="float"/>
                 <xs:enumeration value="double"/>
                 <xs:enumeration value="boolean"/>
                 <xs:enumeration value="string"/>
                 <xs:enumeration value="date"/>
                 <xs:enumeration value="xml"/>
                 <xs:enumeration value="bytes"/>
              </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="non-existing-value-action" use="required">
           <xs:simpleType>
```

```
                <xs:restriction base="xs:string">
                   <xs:enumeration value="return-null"/>
                   <xs:enumeration value="return-original"/>
                   <xs:enumeration value="throw-exception"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="case-sensitive" use="optional">
            <xs:simpleType>
                <xs:restriction base="xs:boolean">
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="value">
    <xs:complexType>
        <xs:attribute name="cmdb-value" type="xs:string" use="required"/>
        <xs:attribute name="external-db-value" type="xs:string" use="require
d"/>
        <xs:attribute name="is-cmdb-value-null" type="xs:boolean" use="optio
nal"/>
        <xs:attribute name="is-db-value-null" type="xs:boolean" use="optiona
l"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Exemple de conversion d'une valeur ’sys’ en valeur ’System’ :

Dans cet exemple, la valeur `sys` dans CMDB est transformée en valeur `System` dans la base de données fédérée et la valeur `System` dans la base de données fédérée est transformée en valeur `sys` dans CMDB.

Si la valeur n'existe pas dans le fichier XML (par exemple, la chaîne `demo`), le convertisseur renvoie la même valeur d'entrée qu'il a reçue.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-value-act
ion="return-original" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/generic-enum-transformer.xsd">
    <value CMDB-value="sys" external-DB-value="System" />
</enum-transformer>
```

Exemple de conversion d'une valeur externe ou CMDB en valeur NULL :

Dans cet exemple, une valeur **NNN** dans la base de données distante est transformée en valeur NULL dans la base de données CMDB.

```
<value cmdb-value="null" is-cmdb-value-null="true" external-db-value="NNN"/>
```

Dans cet exemple, une valeur **OOO** dans la base de données CMDB est transformée en valeur NULL dans la base de données distante.

```
<value cmdb-value="OOO" external-db-value="null" is-db-value-null="true"/>
```

Convertisseur SuffixTransformer

Ce convertisseur permet d'ajouter ou de supprimer des suffixes de la valeur CMDB ou de la source de base de données fédérée.

Il existe deux implémentations :

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddSuffixTransformer**. Ajoute le suffixe (donné en entrée) lors de la conversion de la valeur de la base de données fédérée en valeur CMDB et supprime le suffixe lors de la conversion de la valeur CMDB en valeur de la base de données fédérée.

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemoveSuffixTransformer**. Supprime le suffixe (donné en entrée) lors de la conversion de la valeur de la base de données fédérée en valeur CMDB et ajoute le suffixe lors de la conversion de la valeur CMDB en valeur de la base de données fédérée.

Convertisseur PrefixTransformer

Ce convertisseur permet d'ajouter ou de supprimer un préfixe de la valeur CMDB ou de la base de données fédérée.

Il existe deux implémentations :

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddPrefixTransformer**. Ajoute le préfixe (donné en entrée) lors de la conversion de la valeur de la base de données fédérée en valeur CMDB et supprime le préfixe lors de la conversion de la valeur CMDB en valeur de la base de données fédérée.

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemovePrefixTransformer**. Supprime le préfixe (donné en entrée) lors de la conversion de la valeur de la base de données fédérée en valeur CMDB et ajoute le préfixe lors de la conversion de la valeur CMDB en valeur de la base de données fédérée.

Convertisseur BytesToStringTransformer

Ce convertisseur permet de convertir des tableaux d'octets dans CMDB en leur représentation sous forme de chaîne dans la source de base de données fédérée.

Le convertisseur est : **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer**.

Convertisseur StringDelimitedListTransformer

Ce convertisseur permet de transformer un liste de chaînes unique en liste de nombres entiers/chaînes dans CMDB.

Le convertisseur est : **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.StringDelimitedListTransformer.**

Convertisseur personnalisé

Vous pouvez écrire entièrement votre propre convertisseur (transformateur) personnalisé de façon à l'adapter à vos besoins.

Il existe deux méthodes pour écrire un convertisseur personnalisé :

1. Écriture d'un convertisseur Java compilé
   a. Créez un projet Java dans un IDE Java (Eclipse, Intellij ou Netbeans).
   b. Ajoutez les fichiers federation-api.jar et db-interfaces.jar à votre classpath.
   c. Créez une classe Java qui implémente les interface suivantes (à partir de **db-interfaces.jar**) :
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   d. Compilez le projet et créez un fichier jar.
   e. Placez le fichier jar dans le package de l'adaptateur (sous adapterCode\<ID de l'adaptateur>)
   f. Déployez le package.
   g. Ajoutez le nom de la nouvelle classe de convertisseur dans le fichier **transformations.txt**.
2. Écriture d'un convertisseur Groovy (basé sur un script)

   Le package GDBA d'origine contient l'exemple **GroovyExampleTransformer.groovy**.

   a. Créez un fichier Groovy dans le package de l'adaptateur (sous adapterCode\<ID de l'adaptateur>) Vous pouvez le créer directement à partir du menu Gestion de l'adaptateur.

b. Créez une classe Groovy qui implémente les interface suivantes (à partir de **db-interfaces.jar**) :

   - FcmdbDalTransformerFromExternalDB
   - FcmdbDalTransformerValuesToExternalDB
   - FcmdbDalTransformerInit

c. Ajoutez le nom de la nouvelle classe de convertisseur Groovy dans le fichier **transformations.txt** en conséquence.

Remarque : Groovy est un langage de script qui étend Java. Le code Java standard est également un code Groovy valide.

Plug-in

L'adaptateur de base de données générique prend en charge les plug-in suivants :

- un plug-in facultatif pour la synchronisation complète de la topologie ;
- un plug-in facultatif pour la synchronisation des modifications de la topologie ; si aucun plug-in de synchronisation des modifications n'est implémenté, il est possible d'effectuer une synchronisation différentielle, mais cette dernière sera en fait complète ;
- un plug-in facultatif pour la synchronisation de la mise en page ;
- un plug-in facultatif pour extraire les requêtes prises en charge pour la synchronisation ; si ce plug-in n'est pas défini, tous les noms TQL sont renvoyés ;
- un plug-in facultatif interne pour modifier la définition et le résultat TQL ;
- un plug-in facultatif interne pour modifier une demande de mise en page et un résultat de CI ;
- un plug-in facultatif interne pour modifier une demande de mise en page et un résultat de relations ;
- un plug-in facultatif interne pour modifier l'action du renvoi des ID.

Pour plus d'informations sur l'implémentation et le déploiement de plug-in, voir " Implémentation d'un plug-in " , page 96.

Exemples de configuration

Cette section comprend des exemples de configuration.

Contenu de cette section :

- " Cas d'utilisation " , ci-dessous
- " Rapprochement de nœud unique " , ci-dessous
- " Rapprochement de deux nœuds " , page 144
- " Utilisation d'une clé principale contenant plusieurs colonnes " , page 146
- " Utilisation de transformations " , page 148

Cas d'utilisation

Une requête TQL est :

node > (composition) > card

où :

- **node** désigne l'entité CMDB
- **card** est l'entité de la source de base de données fédérée
- **composition** est la relation entre les deux

L'exemple est exécuté sur la base de données `ED`. Les `ED nodes` sont stockés dans la table `Device` et `card` est stocké dans la table `hwCards`. Dans les exemples suivants, `card` est toujours mappé de la même façon.

Rapprochement de nœud unique

Dans cet exemple, le rapprochement est exécuté sur la propriété `name`.

Définition simplifiée

Le rapprochement est effectué par `node` et est mis en évidence par la balise spéciale **CMDB-class**.

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-single-node>
            <or>
                <attribute CMDB-attribute-name="name" column-name="Device_Na
me" />
            </or>
        </reconciliation-by-single-node>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMD
B-class-name="node" link-class-name="composition">
```

```
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"
/>
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVend
or" />
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"
/>
    </class>
</generic-DB-adapter-config>
```

Définition avancée

Fichier orm.xml

Prêtez attention à l'ajout du mappage de relation. Pour plus d'informations, voir la section de définition dans " Fichier orm.xml " , page 114.

Exemple de fichier orm.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xs
i="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/
xml/ns/
persistence/orm http://java.sun.com/xml/ns/persistence/orm_1_0.xsd" version=
"1.0">
       <description>Generic DB adapter orm</description>
       <package>generic_db_adapter</package>
       <entity class="generic_db_adapter.node" >
               <table name="Device"/>
               <attributes>
                       <id name="id1">
                               <column name="Device_ID"
                                  insertable="false"
                                  updatable="false"/>
                               <generated-value strategy="TABLE"/>
                       </id>
                       <basic name="name">
                               <column name="Device_Name"/>
                       </basic>
               </attributes>
       </entity>
       <entity class="generic_db_adapter.card" >
               <table name="hwCards"/>
               <attributes>
                       <id name="id1">
```

```
                                <column name="hwCards_Seq" insertable="false"
                                    updatable="false"/>
                                <generated-value strategy="TABLE"/>
                        </id>
                        <basic name="card_class">
                                <column name="hwCardClass" insertable="false"
                                    updatable="false"/>
                        </basic>
                        <basic name="card_class">
                                <column name="hwCardVendor" insertable="false"
                                   updatable="false"/>
                        </basic>
                        <basic name="card_name">
                                <column name="hwCardName" insertable="false"
                                   updatable="false"/>
                        </basic>
                </attributes>
        </entity>
        <entity class="generic_db_adapter.node_composition_card" >
                <table name="hwCards"/>
                <attributes>
                        <id name="id1">
                                <column name="hwCards_Seq" insertable="false"
                                   updatable="false"/>
                                <generated-value strategy="TABLE"/>
                        </id>
                        <many-to-one name="end1" target-entity="node">
                                <join-column name="Device_ID" insertable="false"
                                   updatable="false"/>
                        </many-to-one>
                        <one-to-one name="end2" target-entity="card"
>                               <join-column name="hwCards_Seq"
                                referenced-column-name="hwCards_Seq" insertable=
                                "false" updatable="false"/>
                        </one-to-one>
                </attributes>
        </entity>
</entity-mappings>
```

Fichier reconciliation_rules.txt

Pour plus d'informations, voir " Fichier reconciliation_rules.txt (pour compatibilité rétroactive) " , page 128.

```
multinode[node] expression[node.name]
```

Fichier transformation.txt

Ce fichier reste vide car aucune valeur n'a besoin d'être convertie dans cet exemple.

Rapprochement de deux nœuds

Dans cet exemple, le rapprochement est calculé selon la propriété `name` de `node` et d'`ip_address` avec différentes variantes.

La requête TQL de rapprochement est **node > (containment) > ip_address**.

Définition simplifiée

Le rapprochement s'effectue sur `name` de `node` OU de `ip_address` :

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_addr
ess" CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-name="Device_Na
me" />
                <connected-node-attribute CMDB-attribute-name="name" column-
name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMD
B-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"
/>
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVend
or" />
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"
/>
    </class>
</generic-DB-adapter-config>
```

Le rapprochement s'effectue sur `name` de `node` ET de `ip_address` :

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
```

```
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_addr
ess" CMDB-link-type="containment">
            <and>
                <attribute CMDB-attribute-name="name" column-name="Device_Na
me" />
                <connected-node-attribute CMDB-attribute-name="name" column-
name="Device_PreferredIPAddress" />
            </and>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMD
B-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"
/>
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVend
or" />
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"
/>
    </class>
</generic-DB-adapter-config>
```

Le rapprochement s'effectue sur `name` de `ip_address` :

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_addr
ess" CMDB-link-type="containment">
            <or>
                <connected-node-attribute CMDB-attribute-name="name" column-
name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMD
B-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"
/>
```

```
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVend
or" />
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"
/>
    </class>
</generic-DB-adapter-config>
```

Définition avancée

Fichier orm.xml

Comme l'expression de rapprochement n'est pas définie dans ce fichier, la même version doit être utilisée pour toute expression de rapprochement.

Fichier reconciliation_rules.txt

Pour plus d'informations, voir " Fichier reconciliation_rules.txt (pour compatibilité rétroactive) " , page 128.

```
multinode[node] expression[ip_address.name OR node.name] end1_type[node] end
2_type[ip_address] link_type[containment]

multinode[node] expression[ip_address.name AND node.name] end1_type[node] en
d2_type[ip_address] link_type[containment]

multinode[node] expression[ip_address.name] end1_type[node] end2_type[ip_add
ress] link_type[containment]
```

Fichier transformation.txt

Ce fichier reste vide car aucune valeur n'a besoin d'être convertie dans cet exemple.

Utilisation d'une clé principale contenant plusieurs colonnes

Si la clé principale se compose de plusieurs colonnes, le code suivant est ajouté aux définitions XML :

Définition simplifiée

Il existe plusieurs balises de clé principale et une balise correspond à chaque colonne.

```
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMD
B-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID" />
        <primary-key column-name="Device_ID"/>
        <primary-key column-name="hwBusesSupported_Seq" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass"
/>
        <attribute CMDB-attribute-name="card_vendor" column-
```

```
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"
/>
    </class>
```

Définition avancée

Fichier orm.xml

Une nouvelle entité `id` mappée sur les colonnes de clé principale est ajoutée. Les entités qui utilisent cette entité `id` doivent ajouter une balise spéciale.

Si vous utilisez une clé étrangère (balise join-column) pour une clé principale de ce type, vous devez mapper chaque colonne de la clé étrangère sur une colonne de la clé principale.

Pour plus d'informations, voir " Fichier orm.xml " , page 114.

Exemple de fichier orm.xml :

```
<entity class="generic_db_adapter.card" >
        <table name="hwCards"/>
        <attributes>
            <id name="id1">
                <column name="Device_ID" insertable="false" updatable="fals
e"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false" updat
able="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column name="hwCards_Seq" insertable="false" updatable="fal
se"/>
                <generated-value strategy="TABLE"/>
            </id>

<entity class="generic_db_adapter.node_containment_card" >
        <table name="hwCards"/>
        <attributes>
            <id name="id1">
                <column name="Device_ID" insertable="false" updatable="fals
e"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false" updat
able="false"/>
                <generated-value strategy="TABLE"/>
```

```
            </id>
            <id name="id3">
                <column name="hwCards_Seq" insertable="false" updatable="fal
se"/>
                <generated-value strategy="TABLE"/>
            </id>
            <many-to-one name="end1" target-entity="node">
                <join-column name="Device_ID" insertable="false" updatable="
false"/>
            </many-to-one>
            <one-to-one name="end2" target-entity="card">
                <join-column name="Device_ID" referenced-column-name="Device_
ID" insertable="false" updatable="false"/>
                <join-column name="hwBusesSupported_Seq" referenced-column-n
ame="hwBusesSupported_Seq" insertable="false" updatable="false"/>
                <join-column name="hwCards_Seq" referenced-column-name="hwCa
rds_Seq" insertable="false" updatable="false"/>
            </one-to-one>
        </attributes>
    </entity>
</entity-mappings>
```

Utilisation de transformations

Dans l'exemple suivant, le transformateur **enum** générique est converti depuis les valeurs `1`, `2`, `3` dans les valeurs `a`, `b`, `c` respectivement dans la colonne `name`.

Le fichier de mappage est `generic-enum-transformer-example.xml`.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-value-action=
"return-original" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNa
mespaceSchemaLocation="../META-CONF/generic-enum-transformer.xsd">
    <value CMDB-value="1" external-DB-value="a" />
    <value CMDB-value="2" external-DB-value="b" />
    <value CMDB-value="3" external-DB-value="c" />
</enum-transformer>
```

Définition simplifiée

```
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
          CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"
                 from-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter
.dal.
                 transform.impl.GenericEnumTransformer(generic-enum-transformer-
example.
                 xml)" to-CMDB-
```

```
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
                 transform.impl.GenericEnumTransformer(generic-enum-transformer-
example.
                 xml)" />
                <connected-node-attribute CMDB-attribute-name="name"
                 column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
```

Définition avancée

Seul le fichier **transformation.txt** est modifié.

Fichier transformation.txt

Assurez-vous que les noms d'attribut et d'entité sont les mêmes que dans le fichier `orm.xml`.

```
entity[node] attribute[name]
to_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)] from_DB_class
[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

Fichiers journaux de l'adaptateur

Pour comprendre les flux de calcul et le cycle de vie de l'adaptateur, et pour afficher des informations de débogage, vous pouvez consulter les fichiers journaux suivants.

Contenu de cette section :

- " Niveaux de journalisation " , ci-dessous
- " Emplacements des journaux " , page suivante

Niveaux de journalisation

Vous pouvez configurer le niveau de journalisation de chacun des journaux.

Dans un éditeur de texte, ouvrez le fichier **C:\hp\UCMDB\UCMDBServer\conf\log\fcmdb.gdba.properties**
.

Le niveau de journalisation par défaut est **ERROR** :

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL
loglevel=ERROR
```

- Pour augmenter le niveau de journalisation de tous les fichiers journaux, remplacez **loglevel=ERROR** par **loglevel=DEBUG** ou **loglevel=INFO**.
- Pour modifier le niveau de journalisation d'un fichier particulier, modifiez en conséquence la ligne de catégorie **log4j** correspondante. Par exemple, pour modifier le niveau de journalisation de

fcmdb.gdba.dal.sql.log en **INFO**, remplacez :

```
log4j.category.fcmdb.gdba.dal.SQL=${loglevel},fcmdb.gdba.dal.SQL.appender
```

par :

```
log4j.category.fcmdb.gdba.dal.SQL=INFO,fcmdb.gdba.dal.SQL.appender
```

Emplacements des journaux

Les fichiers journaux se trouvent dans le répertoire **C:\hp\UCMDB\UCMDBServer\runtime\log**.

- **Fcmdb.gdba.log**

  Journal du cycle de vie de l'adaptateur. Donne des informations sur les dates de démarrage ou d'arrêt de l'adaptateur et les types de CI qu'il prend en charge.

  À consulter pour les erreurs d'initialisation (chargement/déchargement de l'adaptateur).

- **fcmdb.log**

  À consulter pour les exceptions.

- **cmdb.log**

  À consulter pour les exceptions.

- **Fcmdb.gdba.mapping.engine.log**

  Journal du moteur de mappage. Donne des informations sur la requête TQL de rapprochement utilisée par le moteur de mappage et les topologies de rapprochement qui sont comparées au cours de la phase de connexion.

  Consultez ce journal lorsqu'une requête TQL ne donne aucun résultat, même si vous savez que la base de données contient des CI pertinents, ou lorsque les résultats sont inattendus (vérifiez le rapprochement).

- **Fcmdb.gdba.TQL.log**

  Journal TQL. Donne des informations sur les requêtes TQL et leurs résultats.

  Consultez ce journal lorsqu'une requête TQL ne renvoie pas de résultats et que le moteur de mappage indique qu'il n'y a pas de résultats dans la source de données fédérée.

- **Fcmdb.gdba.dal.log**

  Journal du cycle de vie DAL. Donne des informations sur la génération des types de CI et la connexion à la base de données.

  Consultez ce journal lorsque vous ne parvenez pas à vous connecter à la base de donnés ou lorsque des types de CI ou des attributs ne sont pas pris en charge par la requête.

- **Fcmdb.gdba.dal.command.log**

  Journal des opérations DAL. Donne des informations sur les opérations DAL internes qui sont appelées. (Ce journal est identique à `cmdb.dal.command.log`).

- **Fcmdb.gdba.dal.SQL.log**

  Journal des requêtes SQL DAL. Donne des informations sur les JPAQL appelés (requêtes SQL orientées objet) et leurs résultats.

  Consultez ce journal lorsque vous ne parvenez pas à vous connecter à la base de donnés ou lorsque des types de CI ou des attributs ne sont pas pris en charge par la requête.

- **Fcmdb.gdba.hibrnate.log**

  Journal Hibernate. Donne des informations sur les requêtes SQL qui sont exécutées, l'analyse syntaxique de chaque JPAQL en SQL, les résultats des requêtes, les données relatives à la mise en cache Hibernate, etc. Pour plus d'informations sur Hibernate, voir " Hibernate comme fournisseur JPA " , page 76.

Références externes

Pour plus d'informations sur la norme JavaBeans 3.0, voir .

Résolution des problèmes et limitations

Cette section décrit la résolution des problèmes et les limites correspondant à l'adaptateur de base de données générique.

Limites générales

Lorsque vous mettez à jour un package d'adaptateur, utilisez Notepad++, UltraEdit, ou tout autre éditeur de texte tiers plutôt que le Bloc-notes (quelle que soit la version) de Microsoft Corporation pour modifier les modèles de fichiers. Cela empêche l'utilisation de symboles spéciaux, qui entraîne l'échec du déploiement du package préparé.

Limites JPA

- Toutes les tables doivent comporter une colonne de clé principale.
- Les noms d'attribut de classe CMDB doivent respecter la convention de dénomination JavaBeans (par exemple, ils doivent commencer par des lettres minuscules).
- Deux CI connectés par une relation dans le modèle de classe doivent présenter une association directe dans la base de données (par exemple, si `node` est connecté à `ticket`, ils doivent être connectés par une clé étrangère ou une table de liaison).
- Plusieurs tables mappées sur le même type de CI doivent partager la même table de clé principale.

Limites fonctionnelles

- Vous ne pouvez pas créer de relation manuelle entre CMDB et des types de CI fédérés. Pour pouvoir définir des relations virtuelles, il convient de définir une logique de relation spéciale (qui peut être basée sur les propriétés de la classe fédérée).
- Les types de CI fédérés ne peuvent pas être des types de CI déclencheurs dans une règle d'impact, mais ils peuvent être inclus dans une requête TQL d'analyse d'impact.
- Un type de CI fédéré peut faire partie d'un code TQL d'enrichissement, mais ne peut pas être utilisé comme nœud sur lequel l'enrichissement est effectué (vous ne pouvez pas ajouter, mettre à jour, ni supprimer le type de CI fédéré).
- L'utilisation d'un qualificatif de classe dans une condition n'est pas prise en charge.
- Les sous-graphiques ne sont pas pris en charge.
- Les relations composées ne sont pas prises en charge.
- L'`ID` CMDB du CI externe est composé à partir de sa clé principale mais pas de ses attributs de clé.
- Une colonne de type `bytes` ne peut pas être utilisée comme colonne de clé principale dans Microsoft SQL Server.
- Le calcul d'une requête TQL échoue si les noms des conditions d'attribut définies sur le nœud fédéré n'ont pas été mappés dans le fichier **orm.xml**.

Chapitre 5 : Développement d'adaptateurs Java

Contenu de ce chapitre :

Présentation de l'infrastructure de fédération 153

Adaptateur et interaction de mappage avec l'infrastructure de fédération 158

Infrastructure de fédération pour requêtes TQL fédérées 159

Interactions entre l'infrastructure de fédération, le serveur, l'adaptateur et le moteur de mappage 160

Flux d'infrastructure de fédération pour le remplissage 170

Interfaces d'adaptateur 171

Ressources d'adaptateur et débogage 173

Ajout d'un adaptateur pour une nouvelle source de données externe 173

Création d'un exemple d'adaptateur 182

Propriétés et balises de configuration XML 183

Interface DataAdapterEnvironment 185

Présentation de l'infrastructure de fédération

Remarque :

- Le terme **relation** est équivalent au terme **lien**.
- Le terme **CI** est équivalent au terme **objet**.
- Un graphique est un ensemble de nœuds et de liens.

La fonctionnalité d'infrastructure de fédération utilise une API pour extraire des informations de sources fédérées. L'infrastructure de fédération fournit trois capacités principales :

- **Fédération** à la volée. Toutes les requêtes s'exécutent sur des référentiels de données originaux et les résultats sont générés à la volée dans la base CMDB.
- **Remplissage.** Remplit la base CMDB avec des données (données topologiques et propriétés de CI) issues d'une source externe.
- **Émission de données.** Émet des données (données topologiques et propriétés de CI) depuis la base CMDB locale vers une source de données distante.

Tous les types d'action requièrent un adaptateur par référentiel de données. Cet adaptateur doit être en mesure de fournir des capacités spécifiques au référentiel associé et de récupérer et/ou mettre à

jour les données requises. Chaque requête adressée au référentiel de données s'effectue par le biais de son adaptateur.

Cette section comprend aussi les rubriques suivantes:

- " Fédération à la volée " , ci-dessous
- " Émission de données " , page suivante
- " Remplissage " , page 156

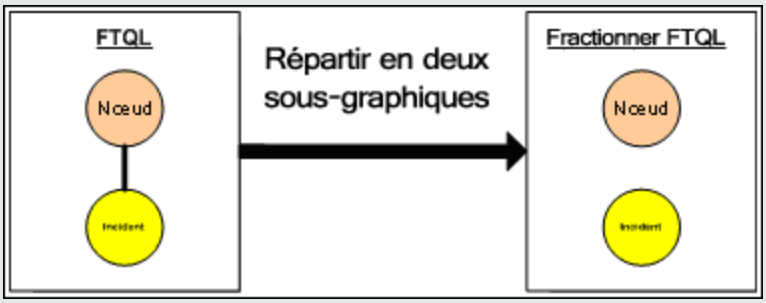
Fédération à la volée

Des requêtes TQL fédérées permettent l'extraction d'informations de n'importe quel référentiel sans réplication des données.

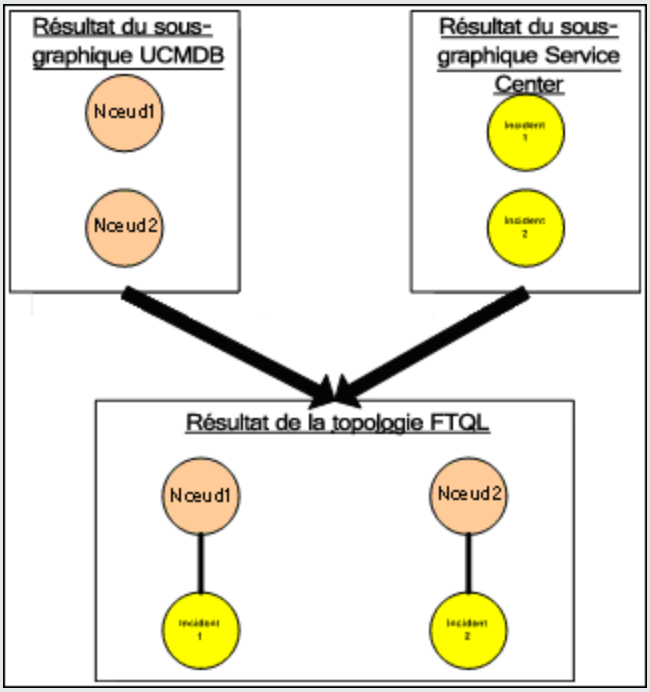
Une requête TQL fédérée exploite des adaptateurs qui représentent des référentiels de données externes. Elle permet d'établir des relations externes appropriées entre CI issus de référentiels de données externes distincts et CI UCMDB.

Exemple de flux de fédération à la volée :

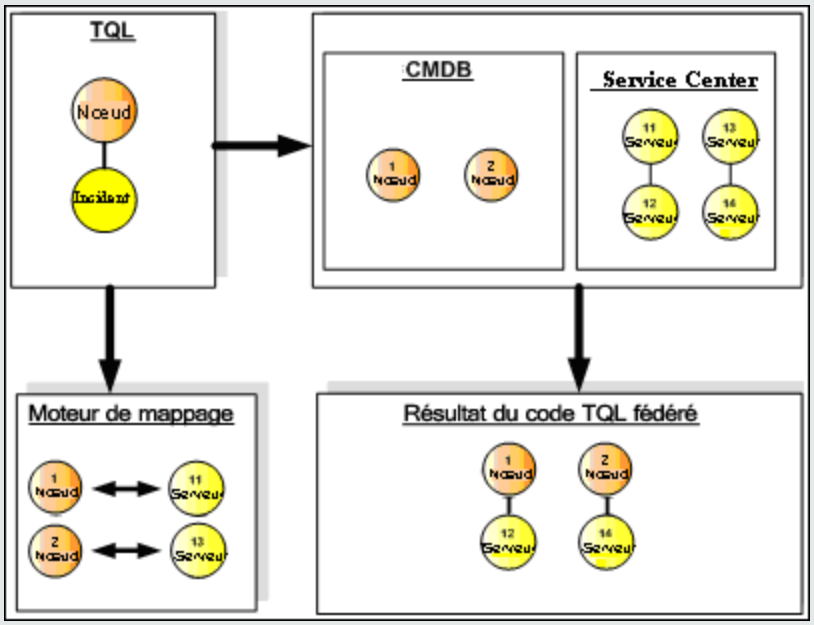
1. L'infrastructure de fédération divise une requête TQL fédérée en plusieurs sous-graphiques, dans lesquels tous les nœuds d'un sous-graphique référencent le même référentiel de données. Chaque sous-graphique est connecté aux autres sous-graphiques par une relation virtuelle (sans en contenir aucune lui-même).



2. Une fois la requête TQL fédérée répartie en sous-graphiques, l'infrastructure de fédération calcule la topologie de chaque sous-graphique et interconnecte deux sous-graphiques appropriés en établissant des relations virtuelles entre les nœuds concernés.



3. Une fois la topologie TQL fédérée calculée, l'infrastructure de fédération extrait une structure pour le résultat topologique.



Émission de données

Le flux d'émission de données permet de synchroniser des données depuis votre base CMDB locale courante vers un service distant ou un référentiel de données cible.

En émission de données, les référentiels de données se divisent en deux catégories : source (base CMDB locale) et cible. Les données sont extraites du référentiel de données source et mises à jour dans le référentiel de données cible. Le processus d'émission de données repose sur des noms de requête. En d'autres termes, les données sont synchronisées entre les référentiels de données source (base CMDB locale) et cible. Elles sont extraites de la base CMDB locale au moyen d'un nom de requête TQL.

Le flux d'émission de données comprend les étapes suivantes :

1. Extraction du résultat topologique avec signatures du référentiel de données source.
2. Comparaison des nouveaux résultats avec les résultats précédents.
3. Extraction d'une structure intégrale (c'est-à-dire, de toutes les propriétés de CI) des CI et des relations, pour les seuls résultats modifiés.
4. Mise à jour du référentiel de données cible avec la structure intégrale des CI et relations reçue. Si un CI ou une relation est supprimé dans le référentiel de données source, et si la requête est de nature exclusive, le processus de réplication élimine les CI et relations correspondants dans le référentiel de données cible.

La base CMDB présente deux sources de données cachées (**hiddenRMIDataSource** et **hiddenChangesDataSource**). Elles constituent systématiquement la source de données "source" dans les flux d'émission de données. Pour mettre en œuvre un nouvel adaptateur pour les flux d'émission de données, il vous suffit d'implémenter l'adaptateur "cible".

Remplissage

Le flux de remplissage vous permet de remplir la base CMDB avec des données issues de sources externes.

Le flux utilise systématiquement une source de données "source" particulière pour extraire les données et les émettre vers la sonde selon un processus similaire au flux d'un travail de découverte.

Pour implémenter un nouvel adaptateur pour les flux de remplissage, il vous suffit d'implémenter l'adaptateur source, étant donné que Data Flow Probe agit en tant que cible.

Dans le flux de remplissage, l'adaptateur s'exécute sur la sonde. Le débogage et la journalisation doivent s'effectuer au niveau de la sonde et non de la base CMDB.

Le flux de remplissage repose sur des noms de requête ; en d'autres termes, les données sont synchronisées entre le référentiel de données cible et Data Flow Probe, et sont extraites au moyen d'un nom de requête dans le référentiel de données source. Par exemple, dans la base UCMDB, le nom de requête correspond au nom de la requête TQL. Toutefois, dans un autre référentiel de données, le nom de requête peut correspondre au nom d'un code qui renvoie des données. L'adaptateur est conçu pour gérer correctement le nom de requête.

Chaque travail peut être défini en tant que travail exclusif. En d'autres termes, les CI et relations inclus dans les résultats du travail sont uniques dans la base CMDB locale, et aucune autre requête ne peut les acheminer vers la cible. L'adaptateur du référentiel de données source prend en charge des requêtes spécifiques et peut extraire les données de ce référentiel. L'adaptateur du référentiel de données cible permet la mise à jour des données extraites de ce référentiel.

Flux SourceDataAdapter

- Extrait le résultat topologique avec signatures du référentiel de données source.
- Compare des nouveaux résultats avec les résultats précédents.
- Extrait une structure intégrale (c'est-à-dire, toutes les propriétés de CI) des CI et des relations, pour les seuls résultats modifiés.
- Met à jour le référentiel de données cible avec la structure intégrale des CI et relations reçue. Si un CI ou une relation est supprimé dans le référentiel de données source, et si la requête est de nature exclusive, le processus de réplication élimine les CI et relations correspondants dans le référentiel de données cible.

Flux SourceChangesDataAdapter

- Extrait le résultat topologique qui s'est produit depuis la dernière date donnée.
- Extrait une structure intégrale (c'est-à-dire, toutes les propriétés de CI) des CI et des relations, pour les seuls résultats modifiés.
- Met à jour le référentiel de données cible avec la structure intégrale des CI et relations reçue. Si un CI ou une relation est supprimé dans le référentiel de données source, et si la requête est de nature exclusive, le processus de réplication élimine les CI et relations correspondants dans le référentiel de données cible.

Flux PopulateDataAdapter

- Extrait la topologie intégrale avec le résultat structurel demandé.
- Utilise le mécanisme de segmentation topologique pour extraire les données par segments.
- La sonde élimine par filtrage toute donnée déjà acheminée au cours d'exécutions précédentes.
- Met à jour le référentiel de données cible avec la structure des CI et relations reçue. Si un CI ou une relation est supprimé dans le référentiel de données source, et si la requête est de nature exclusive, le processus de réplication élimine les CI et relations correspondants dans le référentiel de données cible.

Flux PopulateChangesDataAdapter

- Extrait la topologie avec le résultat structurel requis qui présente des changements depuis la dernière exécution.
- Utilise le mécanisme de segmentation topologique pour extraire les données par segments.
- La sonde élimine par filtrage toute donnée déjà acheminée au cours d'exécutions précédentes (ce flux compris).
- Met à jour le référentiel de données cible avec la structure des CI et relations reçue. Si un CI ou une relation est supprimé dans le référentiel de données source, et si la requête est de nature

exclusive, le processus de réplication élimine les CI et relations correspondants dans le référentiel de données cible.

Flux de remplissage basé sur une instance

Si l'adaptateur est défini pour prendre en charge un flux basé sur une instance (au moyen de la balise **<instance-based-data>** comme décrit dans " Propriétés et balises de configuration XML " , page 183), le moteur de remplissage recherche automatiquement les CI supprimés dans l'instance et les supprime de UCMDB (en supposant que la suppression est autorisée pour le travail de remplissage spécifique). Chaque instance doit comporter un CI racine, marqué dans la définition TQL par le nom **Root**. Chaque fois qu'un CI racine est transmis, l'intégralité de son instance (tous les CI qui lui sont connectés) est comparée à la dernière qui a été envoyé à UCMDB, et tous les CI qui étaient alors connectés à la racine et qui ne le sont plus sont supprimés de UCMDB. Pour que l'adaptateur prenne en charge correctement le flux basé sur une instance, toute modification apportée à un CI ou à un attribut dans l'intégralité de l'instance doit déclencher le renvoi de celle-ci à UCMDB.

Adaptateur et interaction de mappage avec l'infrastructure de fédération

Dans la base UCMDB, un adaptateur est une entité qui représente des données externes (données non enregistrées dans la base UCMDB). Dans les flux fédérés, toutes les interactions avec des sources de données externes s'effectuent par le biais d'adaptateurs. Le flux d'interaction avec l'infrastructure de fédération et les interfaces d'adaptateurs sont différents pour la réplication et pour les requêtes TQL fédérées.

Cette section comprend aussi les rubriques suivantes:

- " Cycle de vie d'adaptateur " , ci-dessous
- " Méthodes assist de l'adaptateur " , ci-dessous

Cycle de vie d'adaptateur

Une instance d'adaptateur est créée pour chaque référentiel de données externe. L'adaptateur entame son cycle de vie par la première action qui s'applique à lui (telle qu'un `calcul de TQL` ou une `extraction/mise à jour de données`). Lorsque la méthode **start** est appelée, l'adaptateur reçoit des informations environnementales, notamment la configuration du référentiel de données ou un logger. Le cycle de vie de l'adaptateur s'achève lorsque le référentiel de données est supprimé de la configuration et lorsque la méthode **shutdown** est appelée. Ce qui signifie que l'adaptateur fonctionne selon un état et qu'il peut contenir la connexion vers le référentiel de données externe le cas échéant.

Méthodes assist de l'adaptateur

L'adaptateur dispose de plusieurs méthodes `assist` capables d'ajouter des configurations de référentiel de données externe. Ces méthodes ne font pas partie intégrante du cycle de vie de l'adaptateur. Elles créent un adaptateur à chaque fois qu'elles sont appelées.

- La première méthode teste la connexion au référentiel de données externe pour une configuration particulière. testConnection s'exécute soit sur le serveur UCMDB, soit sur Data Flow Probe, selon le type d'adaptateur.
- La deuxième méthode est pertinente uniquement pour l'adaptateur source et renvoie les requêtes prises en charge à des fins de réplication. (Cette méthode s'exécute uniquement sur la sonde.)
- La troisième méthode est pertinente uniquement pour les flux de fédération et de remplissage. Elle renvoie chaque classe externe prise en charge par le référentiel de données externe. (Cette méthode s'exécute uniquement sur le serveur UCMDB.)

Toutes ces méthodes sont utilisées lorsque vous créez ou visualisez des configurations d'intégration.

Infrastructure de fédération pour requêtes TQL fédérées

Contenu de cette section :

- " Définitions et termes " , ci-dessous
- " Moteur de mappage " , page suivante
- " Adaptateur fédéré " , page suivante

Voir " Interactions entre l'infrastructure de fédération, le serveur, l'adaptateur et le moteur de mappage " , page suivante pour visualiser les diagrammes illustrant les interactions entre l'infrastructure de fédération, le serveur UCMDB, l'adaptateur et le moteur de mappage.

Définitions et termes

Données de rapprochement. Règle de concordance des CI du type spécifié reçus de la base CMDB et du référentiel de données externe. La règle de rapprochement peut être d'un des trois types suivants :

- **Rapprochement d'identifiant (ID).** Ne s'utilise que si le référentiel de données externe contient l'identifiant (ID) CMDB des objets de rapprochement.
- **Rapprochement de propriété.** Utilisé lorsque la concordance peut s'effectuer au moyen des propriétés du type de CI de rapprochement uniquement.
- **Rapprochement topologique.** Utilisé lorsque vous voulez que les propriétés de type de CI supplémentaires (et non seulement du type de CI de rapprochement) appliquent une concordance à des CI de rapprochement. Par exemple, vous pouvez procéder au rapprochement du type de nœud par la propriété `name` qui appartient au type de CI `ip_address` .

Objet de rapprochement. L'objet est créé par l'adaptateur conformément aux données de rapprochement reçues. Cet objet doit référencer un CI externe et être utilisé par le moteur de mappage pour établir une connexion entre les CI externes et les CI CMDB.

Type de CI de rapprochement. Type de CI qui représente des objets de rapprochement. Ces CI doivent être enregistrés à la fois dans la base CMDB et dans les référentiels de données externes.

Moteur de mappage. Composant qui identifie les relations entre des CI issus de différents référentiels de données affichant eux-mêmes une relation virtuelle. L'identification s'effectue en rapprochant des objets de rapprochement CMDB et des objets de rapprochement de CI externes.

Moteur de mappage

L'infrastructure de fédération utilise le moteur de mappage pour calculer la requête TQL fédérée. Le moteur de mappage établit la connexion entre les CI reçus de différents référentiels de données connectés par des relations virtuels. Le moteur de mappage fournit également des données de rapprochement pour la relation virtuelle. L'une des extrémités de la relation virtuelle doit référencer la base CMDB. Cette extrémité est un type `reconciliation`. Pour le calcul des deux sous-graphes, une relation virtuelle peut partir de n'importe quel nœud d'extrémité.

Adaptateur fédéré

L'adaptateur fédéré récupère deux types de données auprès des référentiels de données externes : des données de CI externe et des objets de rapprochement qui appartiennent à des CI externes.

- **Données de CI externe.** Données externes qui n'existent pas dans la base CMDB. Il s'agit des données cible du référentiel de données externe.
- **Données d'objet de rapprochement.** Données auxiliaires utilisées par l'infrastructure de fédération pour connecter des CI CMDB et des données externes. Chaque objet de rapprochement doit référencer un CI externe. Le type d'objet de rapprochement est le type (ou sous-type) d'une des extrémités de la relation virtuelle dont les données sont extraites. Les objets de rapprochement doivent adapter l'adaptateur reçu aux données de rapprochement. L'objet de rapprochement peut être d'un des trois types suivants : `IdReconciliationObject`, `PropertyReconciliationObject` ou `TopologyReconciliationObject`.

Dans les interfaces fondées sur DataAdapter (DataAdapter, PopulateDataAdapter et PopulateChangesDataAdapter), le rapprochement est demandé dans le cadre de la définition de requête.

Interactions entre l'infrastructure de fédération, le serveur, l'adaptateur et le moteur de mappage

Les diagrammes suivants illustrent les interactions entre l'infrastructure de fédération, le serveur UCMDB, l'adaptateur et le moteur de mappage. Dans les diagrammes de l'exemple, la requête TQL fédérée ne présente qu'une seule relation virtuelle afin de n'impliquer que la base UCMDB et un référentiel de données externe.

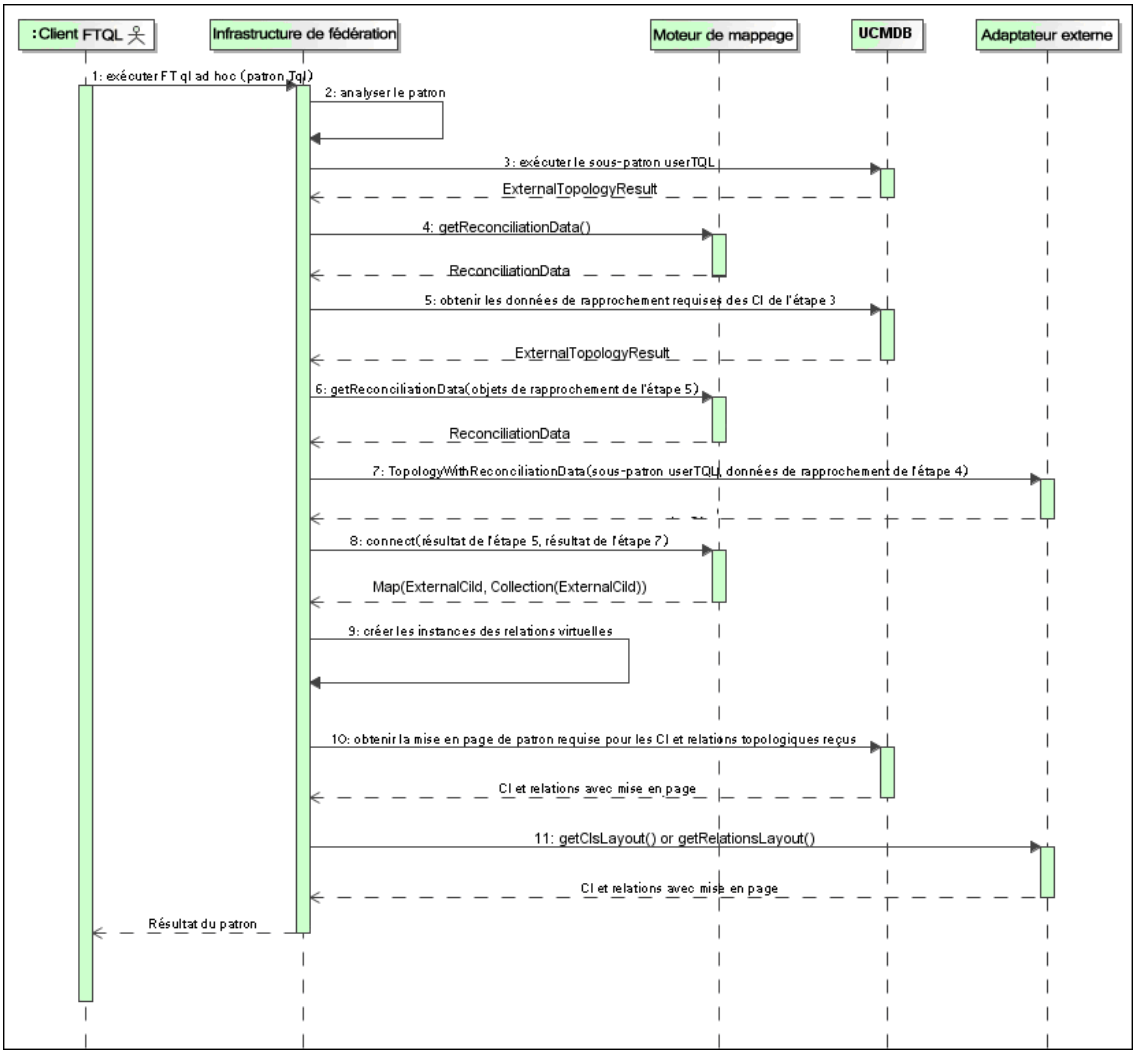
Contenu de cette section :

- " Le calcul démarre au niveau de l'extrémité constituée par le serveur " , page suivante
- " Le calcul débute au niveau de l'extrémité constituée par l'adaptateur externe " , page 163
- " Exemple de flux de l'infrastructure de fédération pour requêtes TQL fédérées " , page 165

Dans le premier diagramme, le calcul commence dans la base UCMDB, tandis que dans le deuxième, il commence au niveau de l'adaptateur externe. Chaque étape du diagramme comprend des références à l'appel de méthode approprié de l'adaptateur ou à l'interface du moteur de mappage.

Le calcul démarre au niveau de l'extrémité constituée par le serveur

Les diagrammes de séquence suivants illustrent l'interaction entre l'infrastructure de fédération, le serveur UCMDB, l'adaptateur et le moteur de mappage. Dans le diagramme de l'exemple, la requête TQL fédérée ne présente qu'une seule relation virtuelle afin de n'impliquer que la base UCMDB et un référentiel de données externe.



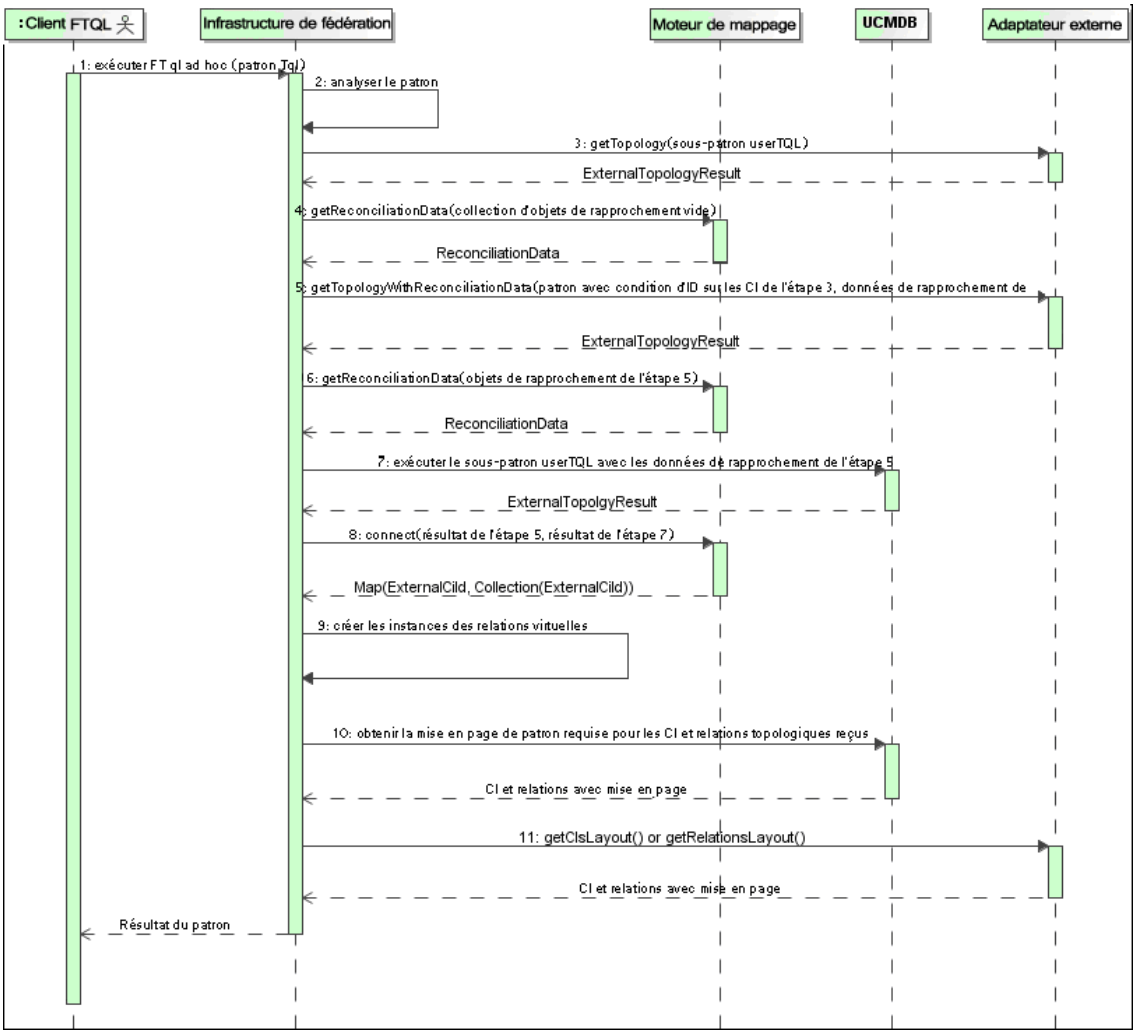
Les numéros qui figurent sur cette image sont expliqués ci-dessous :

Numéro	Explication
1	L'infrastructure de fédération reçoit un appel pour un calcul de requête TQL fédérée.

Numéro	Explication
2	L'infrastructure de fédération analyse l'adaptateur, identifie la relation virtuelle et scinde le code TQL original en deux sous-adaptateurs ; l'un pour la base UCMDB et l'autre pour le référentiel de données externe.
3	L'infrastructure de fédération demande la topologie du sous-code TQL issu de la base UCMDB.
4	Une fois les résultats topologiques reçus, l'infrastructure de fédération appelle le moteur de mappage approprié pour obtenir la relation virtuelle courante et sollicite des données de rapprochement. À ce stade, le paramètre `reconciliationObject` est vide ; en d'autres termes, aucune condition n'est ajoutée aux données de rapprochement dans cet appel. Les données de rapprochement renvoyées définissent les données nécessaires à la mise en concordance des CI de rapprochement présents dans la base UCMDB et du référentiel de données externe. Les données de rapprochement peuvent être d'un des trois types suivants : • **IdReconciliationData.** Les CI sont rapprochés en fonction de leur identifiant (ID). • **PropertyReconciliationData.** Les CI sont rapprochés en fonction des propriétés d'un des CI. • **TopologyReconciliationData.** Les CI sont rapprochés en fonction de la topologie (par exemple, pour rapprocher les CI d'un nœud, l'adresse IP de **IP** est également requise).
5	L'infrastructure de fédération sollicite des données de rapprochement pour les CI des extrémités de la relation virtuelle reçus à l'étape " 3 " , ci dessus de la base UCMDB.
6	L'infrastructure de fédération appelle le moteur de mappage pour extraire les données de rapprochement. Dans cet état (par contraste avec l'étape " 3 " , ci dessus), le moteur de mappage reçoit les objets de rapprochement issus de l'étape " 5 " , ci dessus en tant que paramètres. Le moteur de mappage traduit l'objet de rapprochement reçu en la condition appliquée aux données de rapprochement.
7	L'infrastructure de fédération demande la topologie du sous-code TQL issu du référentiel de données externe. L'adaptateur externe reçoit les données de rapprochement issues de l'étape " 6 " , ci dessus en tant que paramètre.
8	L'infrastructure de fédération appelle le moteur de mappage pour établir une connexion entre les résultats reçus. Le paramètre `firstResult` correspond au résultat topologique externe reçu de UCMDB à l'étape " 5 " , ci dessus, tandis que le paramètre `secondResult` constitue le résultat topologique externe reçu de l'adaptateur externe à l'étape " 7 " , ci dessus. Le moteur de mappage renvoie une carte dans laquelle l'ID de CI externe du premier référentiel de données (UCMDB dans ce cas) est mappé sur les ID de CI externes issus du second référentiel de données (externe).
9	Pour chaque mappage, l'infrastructure de fédération crée une relation virtuelle.

Numéro	Explication
10	Une fois les résultats de la requête TQL fédérée calculés (uniquement à l'étape topologique), l'infrastructure de fédération extrait la structure TQL originale pour les relations et les CI résultants issus des référentiels de données appropriés.

Le calcul débute au niveau de l'extrémité constituée par l'adaptateur externe



Les numéros qui figurent sur cette image sont expliqués ci-dessous :

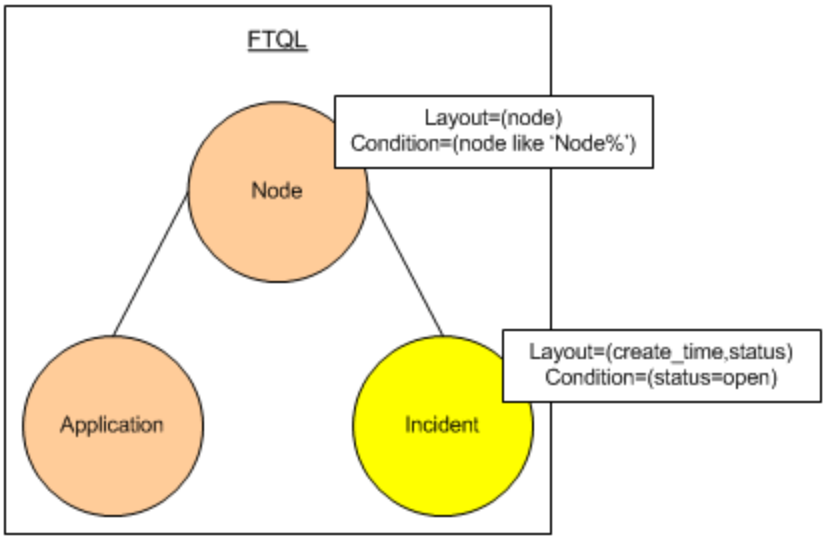
Numéro	Explication
1	L'infrastructure de fédération reçoit un appel pour un calcul de TQL fédérée.

Numéro	Explication
2	L'infrastructure de fédération analyse l'adaptateur, identifie la relation virtuelle et scinde le code TQL original en deux sous-adaptateurs ; l'un pour la base UCMDB et l'autre pour le référentiel de données externe.
3	L'infrastructure de fédération demande la topologie du sous-code TQL issu de l'adaptateur externe. Le paramètre `ExternalTopologyResult` renvoyé n'est pas censé contenir d'objet de rapprochement sachant que les données de rapprochement ne font pas partie de la requête.
4	Une fois les résultats topologiques reçus, l'infrastructure de fédération appelle le moteur de mappage approprié avec la relation virtuelle courante et sollicite des données de rapprochement. À ce stade, le paramètre `reconciliationObjects` est vide ; en d'autres termes, aucune condition n'est ajoutée aux données de rapprochement dans cet appel. Les données de rapprochement renvoyées définissent les données nécessaires à la mise en concordance des CI de rapprochement présents dans la base UCMDB et du référentiel de données externe. Les données de rapprochement peuvent être d'un des types suivants : • **IdReconciliationData.** Les CI sont rapprochés en fonction de leur identifiant (ID). • **PropertyReconciliationData.** Les CI sont rapprochés en fonction des propriétés d'un des CI. • **TopologyReconciliationData.** Les CI sont rapprochés en fonction de la topologie (par exemple, pour rapprocher les CI d'un nœud, l'adresse IP de **IP** est également requise).
5	L'infrastructure de fédération sollicite des objets de rapprochement pour les CI reçus du référentiel de données externe à l'étape 3. L'infrastructure de fédération appelle la méthode **getTopologyWithReconciliationData()** présente dans l'adaptateur externe, où la topologie sollicitée est une topologie à nœud unique accompagnée des CI reçus à l'étape 3 en tant que condition d'ID et des données de rapprochement issus de l'étape 4.
6	L'infrastructure de fédération appelle le moteur de mappage pour extraire les données de rapprochement. Dans cet état (par contraste avec l'étape 3), le moteur de mappage reçoit les objets de rapprochement issus de l'étape 5 en tant que paramètres. Le moteur de mappage traduit l'objet de rapprochement reçu en la condition appliquée aux données de rapprochement.
7	L'infrastructure de fédération demande la topologie du sous-code TQL avec les données de rapprochement issues, à l'étape 6, de la base UCMDB.

Numéro	Explication
8	L'infrastructure de fédération appelle le moteur de mappage pour établir une connexion entre les résultats reçus. Le paramètre `firstResult` correspond au résultat topologique externe issu de l'adaptateur externe à l'étape 5 tandis que le paramètre `secondResult` correspond au résultat topologique externe reçu de la base UCMDB à l'étape 7. Le moteur de mappage renvoie un plan où l'ID du CI externe issu du premier référentiel de données (le référentiel de données externe dans le cas présent) est mappé sur les ID des CI externes issus du deuxième référentiel de données (la base UCMDB).
9	Pour chaque mappage, l'infrastructure de fédération crée une relation virtuelle.
10	Une fois les résultats de la requête TQL fédérée calculés (uniquement à l'étape topologique), l'infrastructure de fédération extrait la structure TQL originale pour les relations et les CI résultants issus des référentiels de données appropriés.

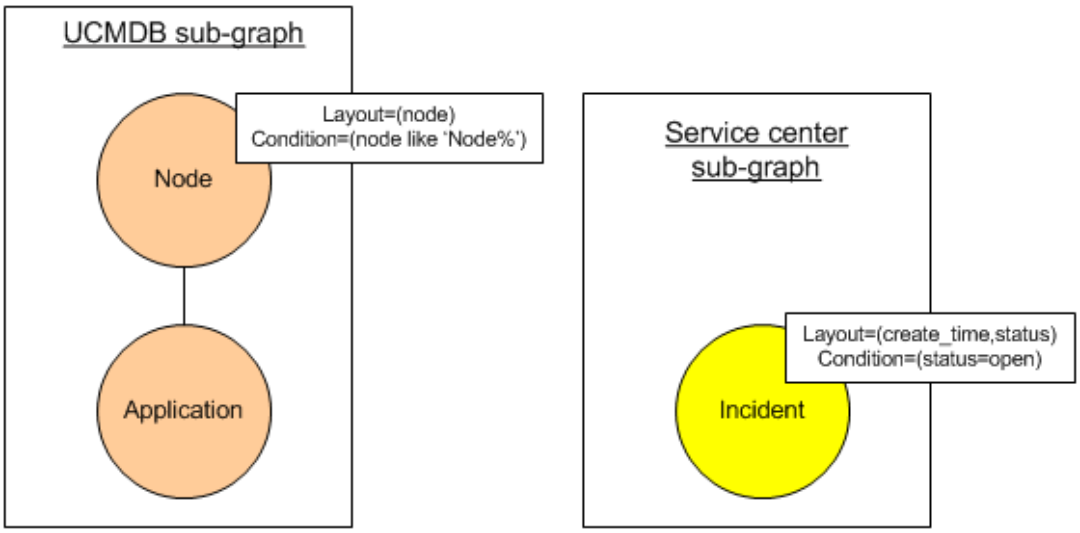
Exemple de flux de l'infrastructure de fédération pour requêtes TQL fédérées

Cet exemple explique comment afficher tous les incidents en cours sur des nœuds spécifiques. Le référentiel de données ServiceCenter est un référentiel de données externe. Les instances de nœud sont enregistrées dans la base UCMDB et les instances d'incident dans ServiceCenter. Pour connecter les instances d'incident au nœud approprié, les propriétés `node` et `ip_address` de l'hôte et l'adresse IP sont considérées comme nécessaires. Il s'agit des propriétés de rapprochement qui identifient les nœuds issus de ServiceCenter dans la base UCMDB.

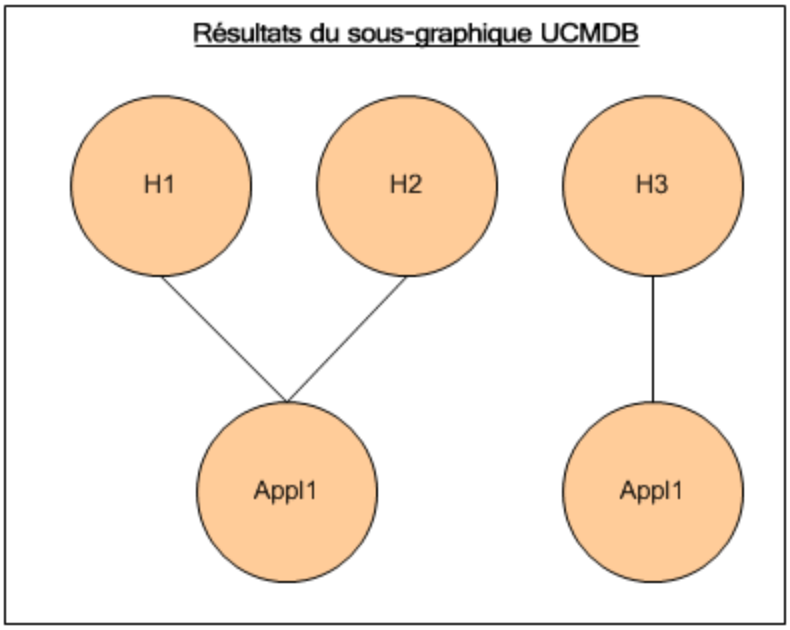


Remarque : Pour la fédération d'attributs, la méthode **getTopology** de l'adaptateur est appelée. Les données de rapprochement sont adaptées dans le code TQL utilisateur (dans le cas présent, l'élément CI).

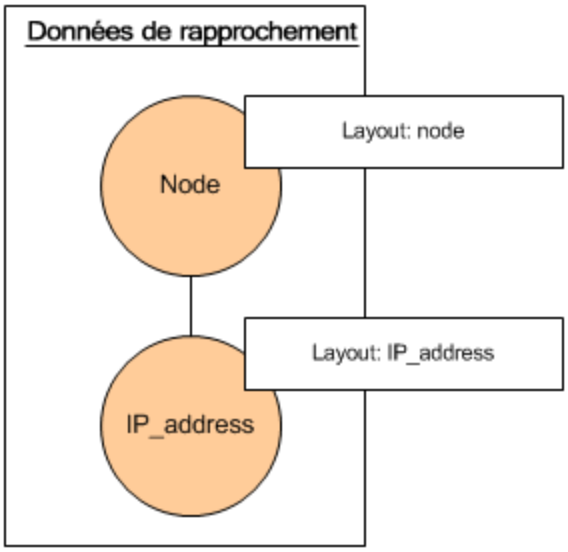
1. Une fois l'adaptateur analysé, l'infrastructure de fédération identifie la relation virtuelle entre les propriétés `Node` et `Incident`, et scinde la requête TQL fédérée en deux sous-graphiques :



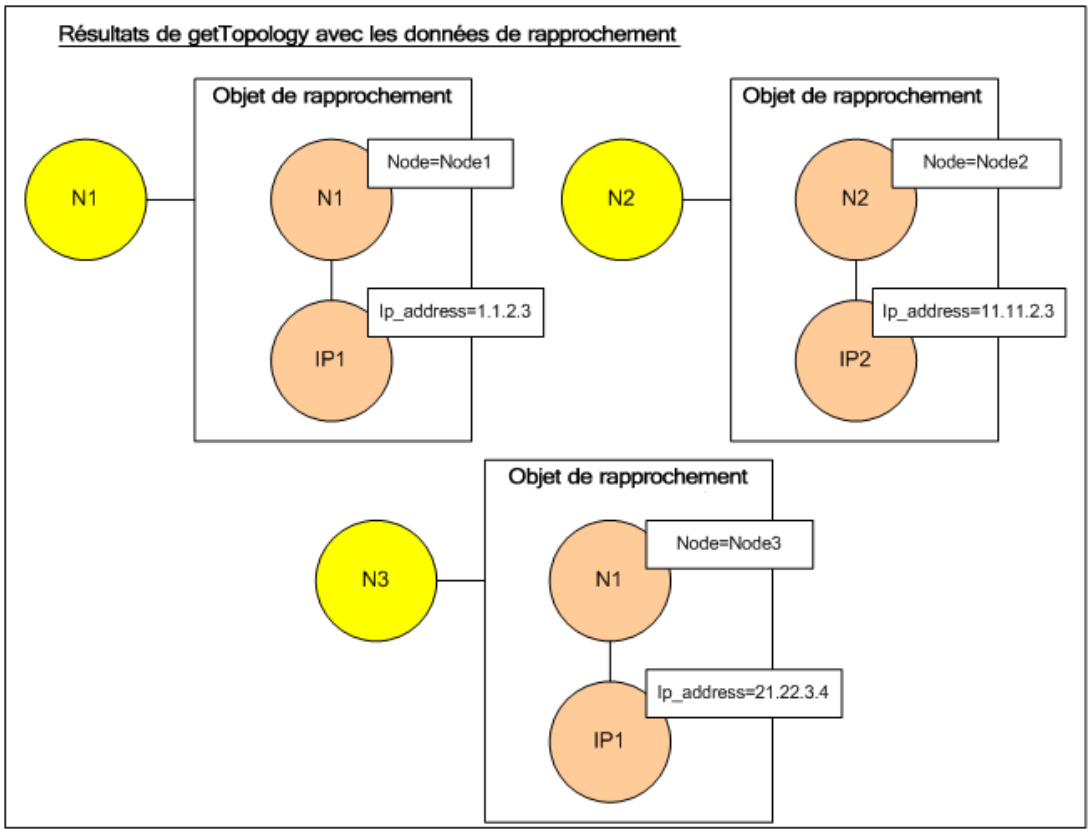
2. L'infrastructure de fédération exécute le sous-graphique UCMDB pour solliciter la topologie et reçoit les résultats suivants :



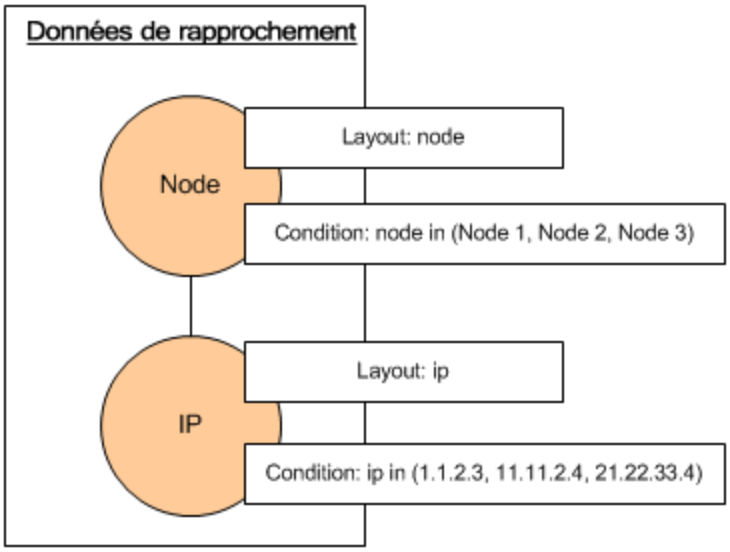
3. L'infrastructure de fédération sollicite auprès du moteur de mappage approprié les données de rapprochement pour le premier référentiel de données (la base UCMDB) qui contient les informations nécessaires pour établir une connexion entre les données reçues de deux référentiels. Dans ce cas, les données de rapprochement sont les suivantes :



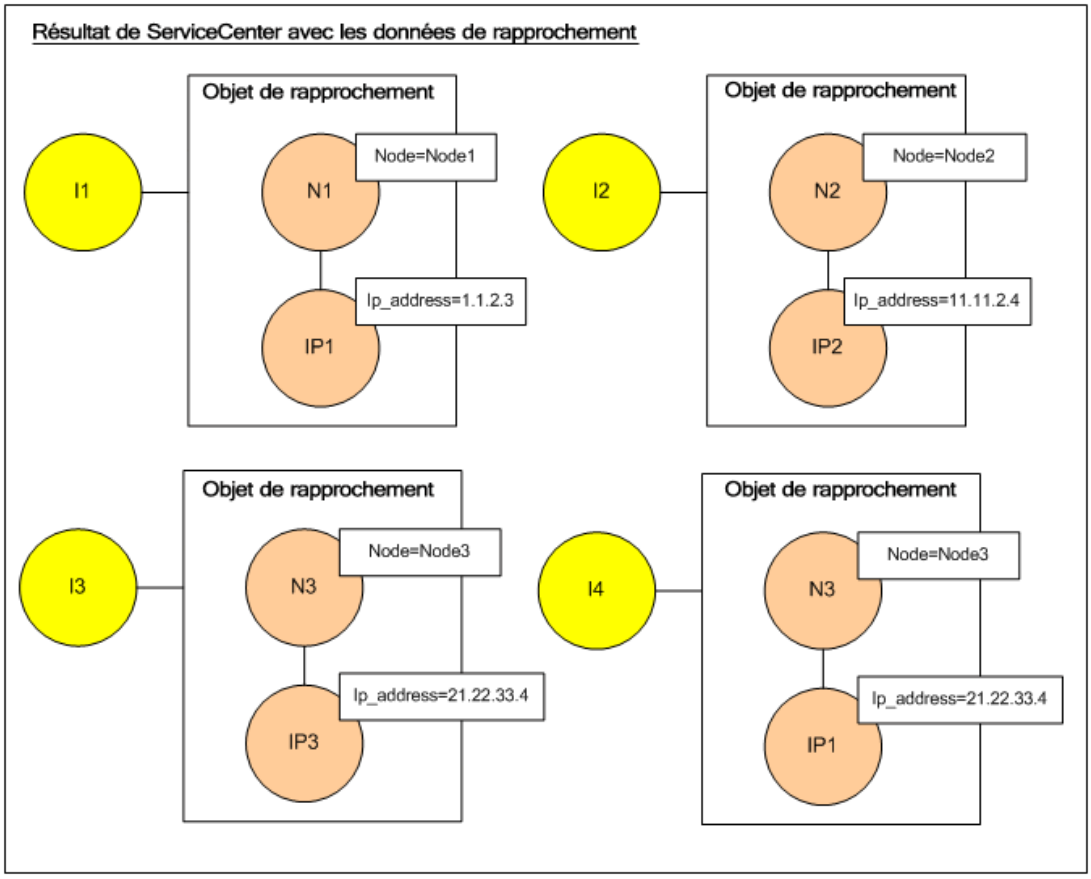
4. L'infrastructure de fédération crée une requête de topologie à un nœud intégrant les conditions Node et ID issues du résultat précédent (node en H1, H2, H3), puis applique cette requête avec les données de rapprochement requises à la base UCMDB. Le résultat comprend les CI de nœud pertinents vis-à-vis de la condition d'identifiant (ID), ainsi que l'objet de rapprochement approprié pour chaque CI :



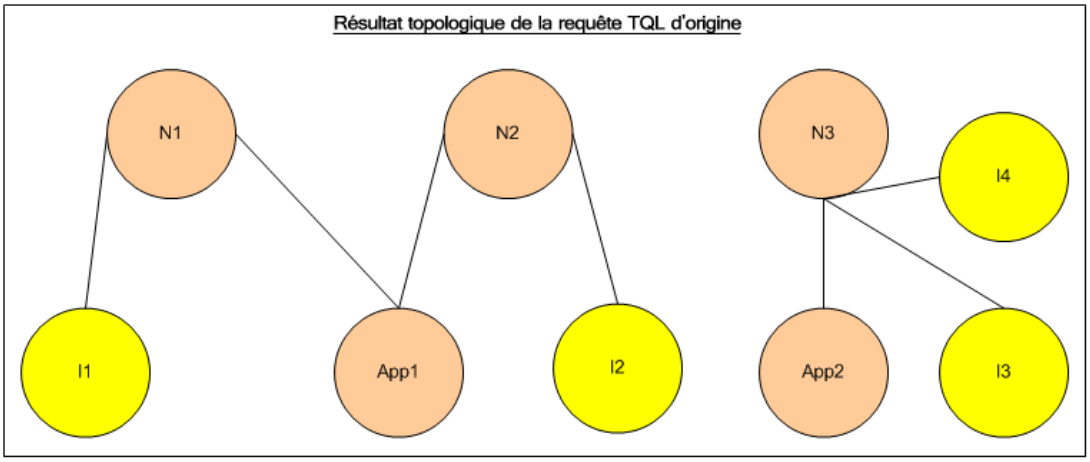
5. Les données de rapprochement destinées à ServiceCenter devraient contenir une condition pour les propriétés `node` et `ip` dérivées des objets de rapprochement reçus de la base UCMDB :



6. L'infrastructure de fédération exécute le sous-graphique de ServiceCenter avec les données de rapprochement pour solliciter la topologie et des objets de rapprochement adaptés. Elle reçoit les résultats suivants :



7. Le résultat après connexion dans le moteur de mappage et création des relations virtuelles est le suivant :



8. L'infrastructure de fédération sollicite la structure TQL d'origine pour les instances reçues de la base UCMDB et de ServiceCenter.

Flux d'infrastructure de fédération pour le remplissage

Contenu de cette section :

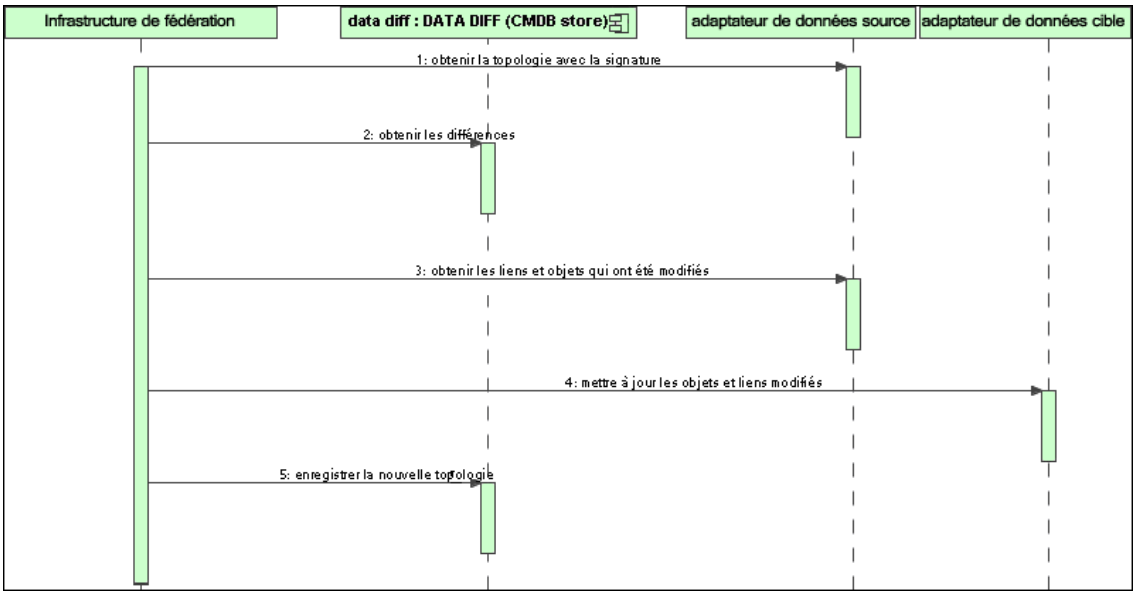
- " Définitions et termes " , ci-dessous
- " Diagramme de flux " , ci-dessous

Définitions et termes

Signature. Présente l'état des propriétés dans le CI. Si des changements sont apportés aux valeurs des propriétés d'un CI, la signature de ce CI doit également être modifiée. La signature du CI contribue à déterminer si un CI a changé sans extraire ni comparer toutes ses propriétés. Le CI et la signature du CI sont tous deux fournis par l'adaptateur approprié. L'adaptateur est responsable de la modification de la signature du CI en cas de modification des propriétés de ce dernier.

Diagramme de flux

Le diagramme de séquence suivant illustre l'interaction entre l'infrastructure de fédération et les adaptateurs source et cible dans un flux de remplissage :



1. L'infrastructure de fédération reçoit la topologie du résultat de requête issu de l'adaptateur source. L'adaptateur identifie la requête par son nom et l'exécute sur le référentiel de données externe. Le résultat topologique contient l'identifiant (ID) et la signature de chaque CI et relation présents dans le résultat. L'identifiant désigne l'identifiant logique qui définit le CI comme unique dans le référentiel de données externe. La signature doit être modifiée dès lors que le CI ou la relation est modifié.

2. L'infrastructure de fédération utilise des signatures pour comparer les résultats de requête topologique reçus avec ceux déjà enregistrés, et pour déterminer les CI qui ont changé.

3. Une fois que l'infrastructure de fédération a identifié les CI et les relations qui ont fait l'objet de

modifications, elle appelle l'adaptateur source, en utilisant les ID de ces CI et relations modifiés en tant que paramètre, pour extraire leur structure intégrale.

4. L'infrastructure de fédération envoie la mise à jour à l'adaptateur cible. L'adaptateur cible met à jour la source de données externe au moyen des données reçues.

5. Une fois la mise à jour terminée, l'infrastructure de fédération enregistre le résultat de la dernière requête.

Interfaces d'adaptateur

Contenu de cette section :

- " Définitions et termes " , ci-dessous

- " Interfaces d'adaptateur pour les requêtes TQL fédérées " , ci-dessous

Définitions et termes

Relation externe. Relation entre deux types de CI externes pris en charge par le même adaptateur.

Interfaces d'adaptateur pour les requêtes TQL fédérées

Utilisez l'interface appropriée à chaque adaptateur, comme indiqué ci-après.

- Utilisez une **interface topologique Single Node** lorsque l'adaptateur ne prend en charge aucune relation externe, c'est-à-dire, lorsqu'il n'est jamais censé recevoir de requête comportant plus d'un CI externe. Les données de rapprochement nécessaires à l'exécution de l'opération peuvent être considérées comme une requête complexe (voir SingleNodeFederationTopologyReconciliationAdapter ci-dessous).

  Toutes les interfaces SingleNode sont créées pour simplifier le flux de travail. Dans les cas où vous devez utiliser une requête plus étendue, utilisez l'interface **FederationTopologyAdapter**.

- Utilisez une interface **FederationTopologyAdapter** pour définir des adaptateurs qui prennent en charge des requêtes fédérées complexes. La requête de rapprochement intégrée dans ces adaptateurs fait partie du paramètre **QueryDefinition**.

  Le moteur de fédération utilise les données de rapprochement pour connecter les données fédérées aux CI locaux appropriés. Les données de fédération peuvent être récupérées dans plusieurs demandes (calculées de manière récursive en fonction des résultats). Dans ce cas, l'adaptateur reçoit une demande contenant uniquement les données de rapprochement.

Interfaces SingleNode

Les interfaces suivantes présentent différents types de données de rapprochement :

- **SingleNodeFederationIdReconciliationAdapter.** S'utilise si l'adaptateur prend en charge un **code TQL à nœud unique** et si le rapprochement entre des référentiels de données se calcule en fonction de l'identifiant (ID).

- **SingleNodeFederationPropertyReconciliationAdapter.** S'utilise si l'adaptateur prend en charge un **code TQL à nœud unique** et si le rapprochement entre des référentiels de données se calcule en fonction des propriétés d'un CI donné.

- **SingleNodeFederationTopologyReconciliationAdapter.** S'utilise si l'adaptateur prend en charge un **code TQL à nœud unique** et si le rapprochement entre des référentiels de données se calcule en fonction de la topologie. L'adaptateur doit prendre en charge le cas présentant un élément de requête vide et où seule la topologie de rapprochement est demandée.

Interfaces d'adaptateur de données

- **FederationTopologyAdapter.** Utilisez cet adaptateur pour prendre en charge des requêtes TQL fédérées complexes. Autorise la plus grande diversité. L'adaptateur doit prendre en charge le cas où la définition de requête décrit uniquement les données de rapprochement.

- **PopulateDataAdapter.** Utilisez cet adaptateur pour prendre en charge des requêtes TQL fédérées complexes et des flux de remplissage. Dans un flux de remplissage, cet adaptateur extrait l'intégralité du jeu de données et laisse la sonde filtrer la différence depuis la dernière exécution du travail.

- **PopulateChangesDataAdapter.** Utilisez cet adaptateur pour prendre en charge des requêtes TQL fédérées complexes et des flux de remplissage. Dans un flux de remplissage, cet adaptateur prend en charge l'extraction des seules modifications intervenues depuis la dernière exécution du travail.

Remarque : Lors du développement d'un adaptateur susceptible de renvoyer des jeux de données volumineux, il est essentiel d'autoriser la segmentation grâce à l'implémentation de l'interface ChunkGetter. Voir le document Java relatif à l'adaptateur concerné pour plus d'informations.

Interfaces de rapports de ressources

Les interfaces ci-dessous permettent à l'adaptateur de générer des rapports sur les ressources pouvant être configurées pour personnaliser son comportement. Vous pouvez ainsi modifier ces ressources directement à partir du Studio d'intégration. Ces interfaces doivent être utilisées en complément des interfaces d'adaptateur standard décrites plus haut.

- **PopulationQueriesResourcesLocator.** Définit les ressources pouvant être modifiées pour chaque requête de remplissage spécifique.

- **PushQueriesResourceLocator.** Définit les ressources pouvant être modifiées pour chaque requête d'émission de données.

- **GeneralResourcesLocator.** Définit les ressources générales pouvant être modifiées dans cet adaptateur.

Interfaces supplémentaires

- **SortResultDataAdapter.** Utilisez cette interface si vous pouvez trier les CI résultants dans le référentiel de données externe.
- **FunctionalLayoutDataAdapter.** Utilisez cette interface si vous pouvez calculer la structure fonctionnelle dans le référentiel de données externe.

Interfaces d'adaptateur pour la synchronisation

- **SourceDataAdapter.** Utilisez cette interface pour les adaptateurs source dans les flux de remplissage.
- **TargetDataAdapter.** Utilisez cette interface pour les adaptateurs cible dans les flux d'émission de données.

Ressources d'adaptateur et débogage

Cette tâche explique comment utiliser la console JMX pour créer, afficher et supprimer des ressources d'état d'adaptateur (toute ressource créée à l'aide des méthodes de manipulation des ressources dans l'interface DataAdapterEnvironment et enregistrée dans la base de données de UCMDB ou de la sonde) à des fins de débogage et de développement.

1. Lancez le navigateur Web, puis entrez l'adresse du serveur comme suit :
   - Pour le serveur UCMDB : `http://localhost:8080/jmx-console`
   - Pour la sonde : `http://localhost:1977`

   Vous devrez peut-être vous connecter à l'aide d'un nom d'utilisateur et d'un mot de passe.
2. Pour ouvrir la page JMX BEAN View, procédez de l'une des façons suivantes :
   - Sur le serveur UCMDB : cliquez sur **UCMDB:service=FCMDB Adapter State Resource Services**
   - Sur la sonde : cliquez sur **type=AdapterStateResources**
3. Saisissez des valeurs dans les opérations à utiliser, puis cliquez sur **Invoke**.

Ajout d'un adaptateur pour une nouvelle source de données externe

Cette tâche explique comment définir un adaptateur pour prendre en charge une nouvelle source de données externe.

Cette tâche comprend les étapes suivantes :

- " Conditions préalables " , ci-dessous
- " Définition de relations virtuelles valides " , ci-dessous
- " Définition d'une configuration d'adaptateur " , page suivante
- " Définition des classes prises en charge " , page 179
- " Implémentation de l'adaptateur " , page 180
- " Définition de règles de rapprochement ou implémentation du moteur de mappage " , page 181
- " Ajout des ressources jar nécessaires à l'implémentation dans le classpath " , page 181
- " Déploiement de l'adaptateur " , page 181
- " Mise à jour de l'adaptateur " , page 182

1. ## Conditions préalables

   Classes d'adaptateur prises en charge par le modèle pour les CI et relations dans le modèle de données UCMDB : En tant que développeur d'adaptateurs, vous devez présenter les aptitudes suivantes :

   - connaître la hiérarchie des types de CI UCMDB afin de comprendre le mode d'association de types de CI externes aux types de CI UCMDB ;
   - savoir modéliser les types de CI externes dans le modèle de classe UCMDB ;
   - savoir ajouter les définitions de nouveaux types de CI et de leurs relations ;
   - savoir définir des relations valides dans le modèle de classe UCMDB pour des relations valides entre classes intérieures d'adaptateur. (Les types de CI peuvent être placés à tout niveau dans l'arborescence du modèle de classe UCMDB.)

   La modélisation doit être identique quel que soit le type de fédération (à la volée ou par réplication). Pour plus d'informations sur l'ajout de nouvelles définitions de types de CI au modèle de classe UCMDB, voir Working with the CI Selector dans le *Manuel de modélisation HP Universal CMDB*.

   Pour que l'adaptateur puisse gérer des attributs fédérés sur des types de CI, ajoutez ce type de CI aux classes prises en charge, avec les attributs et la règle de rapprochement prises en charge pour ce type de CI.

2. ## Définition de relations virtuelles valides

   Remarque : Cette section s'applique uniquement à la fédération.

Pour extraire des types de CI fédérés connectés à des types de CI CMDB locaux, la base CMDB doit intégrer une définition de lien valide entre ces deux types.

a. Créez un fichier XML qui contient ces liens valides (s'ils n'existent pas encore).

b. Ajoutez le fichier XML de liens au package de l'adaptateur dans le dossier **\validlinks**. Pour plus d'informations, voir "Package Manager" dans le *Manuel d'administration HP Universal CMDB*.

Exemple de définition d'une relation valide :

Dans l'exemple suivant, la relation de type `containment` entre des instances de type `node` et de type `myclass1` est une définition de relation valide.

```
<Valid-Links>
    <Valid-Links>
        <Class-Ref class-name="containment">
        <End1 class-name="node">
        <End2 class-name="myclass1">
        <Valid-Link-Qualifiers>
    </Valid-Link>
</Valid-Links>
```

3. **Définition d'une configuration d'adaptateur**

a. Accédez à **Gestion de l'adaptateur**.

b. Cliquez sur le bouton **Créer une ressource** et sélectionnez **Nouvel adaptateur**.

c. Dans la boîte de dialogue Nouvel adaptateur, sélectionnez **Intégration** et **Adaptateur Java**.

d. Cliquez avec le bouton droit sur l'adaptateur que vous avez créé, puis sélectionnez **Modifier la source de l'adaptateur** dans le menu contextuel.

e. Modifiez les balises XML suivantes :

```
<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" id="newAdapterIdName"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
description="Adapter Description" schemaVersion="9.0"
displayName="New Adapter Display Name">

<deletable>true</deletable>

<discoveredClasses>
```

```
<discoveredClass>link</discoveredClass>

<discoveredClass>object</discoveredClass>

</discoveredClasses>

<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.cor
e.
 AdapterService">

<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.cor
e.
 AdapterServiceParams" enableAging="true"
enableDebugging="false" enableRecording=
"false" autoDeleteOnErrors="success" recordResult="false"
maxThreads="1" patternType="java_adapter"
maxThreadRuntime="25200000">

<className>com.yourCompany.adapter.MyAdapter.MyAdapterClass
</className>

</params>

<destinationInfo
className="com.hp.ucmdb.discovery.probe.tasks.BaseDestination
Data">

<!-- check -->

<destinationData name="adapterId"
description="">${ADAPTER.adapter_id}</destinationData>

<destinationData name="attributeValues"
description="">${SOURCE.attribute_values}</destinationData>

<destinationData name="credentialsId"
description="">${SOURCE.credentials_id}</destinationData>

<destinationData name="destinationId"
description="">${SOURCE.destination_id}</destinationData>

</destinationInfo>

<resultMechanism isEnabled="true">
```

```
<autoDeleteCITs isEnabled="true">

<CIT>link</CIT>

<CIT>object</CIT>

</autoDeleteCITs>

</resultMechanism>

</taskInfo>

<adapterInfo>

<adapter-capabilities>

<support-federated-query>

<!--<supported-classes/> <!—see the section about supported
classes-->

<topology>

<pattern-topology /> <!—or <one-node-topology> -->

</topology>

</support-federated-query>

<!--<support-replication-data>

<source>

<changes-source/>

</source>

<target/>

</adapter-capabilities>

<default-mapping-engine/>

<queries />

<removedAttributes />
```

```
<full-population-days-interval>-1</full-population-days-
interval>

</adapterInfo>

<inputClass>destination_config</inputClass>

<protocols />

<parameters>

<!--The description attribute may be written in simple text
or HTML.-->

<!--The host attribute is treated as a special case by UCMDB-
->

<!--and will automatically select the probe name (if
possible)-->

<!--according to this attribute’s value.-->

<parameter name="credentialsId" description="Special type of
property, handled by UCMDB for credentials menu"
type="integer" display-name="Credentials ID" mandatory="true"
order-index="12" />

<parameter name="host" description="The host name or IP
address of the remote machine" type="string" display-
name="Hostname/IP" mandatory="false" order-index="10" />

<parameter name="port" description="The remote machine's
connection port" type="integer" display-name="Port"
mandatory="false" order-index="11" />

</parameters>

<parameter name="myatt" description="is my att true?"
type="string" display-name="My Att" mandatory="false" order-
index="15" valid-values=”True;False”/>True</parameters>

<collectDiscoveredByInfo>true</collectDiscoveredByInfo>

<integration isEnabled="true">

<category >My Category</category>
```

```
</integration>

<overrideDomain>${SOURCE.probe_name}</overrideDomain>

<inputTQL>

<resource:XmlResourceWrapper
xmlns:resource="http://www.hp.com/ucmdb/1-0-
0/ResourceDefinition" xmlns:ns4="http://www.hp.com/ucmdb/1-0-
0/ViewDefinition" xmlns:tql="http://www.hp.com/ucmdb/1-0-
0/TopologyQueryLanguage">

<resource xsi:type="tql:Query" group-id="2" priority="low"
is-live="true" owner="Input TQL" name="Input TQL">

<tql:node class="adapter_config" id="-11" name="ADAPTER" />

<tql:node class="destination_config" id="-10" name="SOURCE"
/>

<tql:link to="ADAPTER" from="SOURCE" class="fcmdb_conf_
aggregation" id="-12" name="fcmdb_conf_aggregation" />

</resource>

</resource:XmlResourceWrapper>

</inputTQL>

<permissions />

</pattern>
```

Pour plus d'informations sur les balises XML, voir " Propriétés et balises de configuration XML " , page 183.

4. **Définition des classes prises en charge**

Définissez les classes prises en charge soit dans le code de l'adaptateur en implémentant la méthode *getSupportedClasses()*, soit à l'aide du fichier XML patron.

```
<supported-classes>
    <supported-class name="HistoryChange" is-derived="false" is-reconcili
ation-supported="false" federation-not-supported="false" is-id-reconcilia
tion-supported="false">
    <supported-conditions>
```

```
        <attribute-operators attribute-name="change_create_time">
            <operator>GREATER</operator>
            <operator>LESS</operator>
            <operator>GREATER_OR_EQUAL</operator>
            <operator>LESS_OR_EQUAL</operator>
            <operator>CHANGED_DURING</operator>
        </attribute-operators>
    </supported-conditions>
 </supported-class>
```

name	Nom du type de CI
is-derived	Spécifie si cette définition comprend tous les enfants héritiers.
is-reconciliation-supported	Spécifie si cette classe est utilisée pour le rapprochement.
is-id-reconciliation-supported	Spécifie si cette classe est utilisée pour le rapprochement par identifiant (ID).
federation-not-supported	Spécifie si ce type de CI doit être ou non autorisé dans le cadre d'une fédération (blocage de certains types de CI ; par exemple, un type de CI défini uniquement pour la fédération).
<supported-conditions>	Spécifie les conditions prises en charge sur chaque attribut.

5. **Implémentation de l'adaptateur**

   Sélectionnez la classe d'implémentation d'adaptateur adaptée à ses capacités définies. La classe d'implémentation de l'adaptateur met en œuvre les interfaces appropriées conformément aux capacités définies.

   Si l'adaptateur implémente **getTopologyWithReconciliationData** et que les fonctions de l'adaptateur incluent la possibilité d'utiliser un point de départ, l'adaptateur doit également prendre en charge la topologie demandante avec les données de réconciliation sans conditions (type uniquement). Dans ce cas, l'adaptateur doit renvoyer toutes les données de rapprochement des résultats recherchés.

   La prise en charge du rapprochement par adaptateur peut être définie selon **global_id**. Dans ce cas, **global_id** doit être défini comme partie intégrante des attributs de rapprochement dans les classes prises en charge par l'adaptateur. Si la prise en charge du rapprochement par adaptateur est définie selon **global_id**, **getTopologyWithReconciliationData()** doit alors renvoyer **global_id** comme partie intégrante des propriétés d'objet de rapprochement. La base UCMDB utilise **global_id** pour le rapprochement de résultats de fédération d'un type de CI à la place de la règle d'identification.

L'interface DataAdapterEnvironment constitue une partie de l'API de fédération. Cette interface représente l'environnement de l'adaptateur de données. Elle contient l'API de l'environnement nécessaire à l'adaptateur pour fonctionner. Pour plus d'informations sur l'interface DataAdapterEnvironment, voir " Interface DataAdapterEnvironment " , page 185.

6. **Définition de règles de rapprochement ou implémentation du moteur de mappage**

   Si votre adaptateur prend en charge des requêtes TQL fédérées, vous pouvez définir votre moteur de mappage de l'une des deux façons suivantes :

   - Utilisez le moteur de mappage par défaut du , qui effectue le mappage à l'aide des règles de rapprochement du CMDB. Pour l'utiliser, laissez la balise XML **<default-mapping-engine>** vide.
   - Développez votre propre moteur de mappage en implémentant l'interface de moteur de mappage et en plaçant le code JAR avec le reste du code de l'adaptateur. Pour ce faire, utilisez la balise XML suivante : **<default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>**

7. **Ajout des ressources jar nécessaires à l'implémentation dans le classpath**

   Pour implémenter vos classes, ajoutez le fichier **federation_api.jar** dans le classpath de votre éditeur de code.

8. **Déploiement de l'adaptateur**

   Déployez le package de l'adaptateur. Pour plus d'informations sur le déploiement d'un package, voir "Package Manager" dans le *Manuel d'administration HP Universal CMDB*.

   Le package doit contenir les entités suivantes :

   - Définition des nouveaux types de CI (facultatif) :
   - S'utilise uniquement si l'adaptateur prend en charge de nouveaux types de CI qui n'existent pas encore dans la base UCMDB.
   - Les définitions des nouveaux types de CI sont consignées dans le dossier `class` du package.
   - Définition des nouveaux types de données (facultatif) :
   - S'utilise uniquement si de nouveaux types de CI requièrent de nouveaux types de données.
   - Les définitions des nouveaux types de données sont consignées dans le dossier `typedef` du package.
   - Définition de nouvelles relations valides (facultatif) :

- S'utilise uniquement si l'adaptateur prend en charge le code TQL fédéré.
- Les définitions des nouvelles relations valides sont consignées dans le dossier `validlinks` du package.
- Le fichier XML de configuration de patron doit se trouver dans le dossier `discoveryPatterns` du package.
- **Descripteur.** Spécifie les définitions du package.
- Placez vos classes compilées (normalement un fichier jar) dans le package sous le dossier **adapterCode\<adapter id>**.

  > **Remarque :** Le nom de dossier `adapter id` affiche la même valeur que dans la configuration de l'adaptateur.

- Si vous créez votre propre fichier de configuration, vous devez le placer dans le package sous le dossier **adapterCode\<adapter id>**.

9. **Mise à jour de l'adaptateur**

   Les modifications de tout fichier non binaire de l'adaptateur peuvent s'effectuer dans le module Gestion de l'adaptateur. Apporter des modifications aux fichiers de configuration dans le module de gestion de l'adaptateur entraîne le rechargement de ce dernier avec les nouvelles configurations.

   Les mises à jour peuvent également s'effectuer en modifiant les fichiers dans le package (fichiers binaires et non binaires), puis en redéployant le package au moyen du Gestionnaire des packages. Pour plus d'informations, voir "How to Deploy a Package" dans le *Manuel d'administration HP Universal CMDB*.

Création d'un exemple d'adaptateur

Cet exemple illustre comment créer un adaptateur échantillon. Cette tâche comprend les étapes suivantes :

- " Sélection d'une logique d'adaptateur " , ci-dessous
- " Chargement du projet " , page suivante

1. **Sélection d'une logique d'adaptateur**

   Lorsque vous implémentez un adaptateur, vous devez choisir la manière de gérer la logique conditionnelle dans l'implémentation (conditions de propriété, d' ID, de rapprochement et de lien).

   a. Extrayez l'intégralité des données dans la mémoire de l'adaptateur et exécutez le filtrage ou la sélection des instances de CI nécessaires.

b. Convertissez toutes les conditions dans le langage de la source de données, puis filtrez et sélectionnez les données. Par exemple :

   - Convertissez la condition en une requête SQL.
   - Convertissez la condition en un objet de filtre d'API Java.

c. Filtrez certaines des données sur le service distant et laissez l'adaptateur sélectionner et filtrer le reste.

Dans l'exemple MyAdapter, la logique de l'option *a* est utilisée.

2. **Chargement du projet**

Copiez les fichiers du dossier **C:\hp\UCMDB\UCMDBServer\tools\adapter-dev-kit\SampleAdapters**et suivez les instructions qui figurent dans les fichiers Lisez-moi.

> **Remarque :** Si vous utilisez un adaptateur avec des jeux de données volumineux, vous pouvez être amené à utiliser une gestion en cache ou une indexation pour améliorer les performances de la fédération.

La documentation javadoc en ligne est disponible à l'emplacement suivant :

C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DBAdapterFramework_JavaAPI\index.html

Propriétés et balises de configuration XML

id="newAdapterIdName"		Définit le nom réel de l'adaptateur. S'utilise pour les journaux et les consultations de dossiers.
displayName="New Adapter Display Name"		Définit le nom affiché de l'adaptateur tel qu'il figure dans l'interface utilisateur.
<className>...</className>		Définit l'interface de l'adaptateur qui implémente la classe Java.
<category >My Category</category>		Définit la catégorie de l'adaptateur.
<parameters>		Définit les propriétés de la configuration disponibles dans l'interface utilisateur lors de la mise en place d'un nouveau point d'intégration.
	name	Nom de la propriété (utilisé essentiellement par le code).

	description	Indication d'affichage de la propriété.
	type	Chaîne ou entier (utilise des valeurs valides avec chaîne pour les expressions booléennes).
	display-name	Nom de la propriété dans l'interface utilisateur.
	mandatory	Indique si cette propriété de configuration est obligatoire pour l'utilisateur.
	order-index	Ordre de positionnement de la propriété (petit = haut).
	valid-values	Liste des valeurs valides possibles, séparées par des caractères ';' (par exemple, valid-values="Oracle;SQLServer;MySQL" or valid-values="True;False").
<adapterInfo>		Contient la définition des capacités et des paramètres statiques de l'adaptateur.
	<support-federated-query>	Définit cet adaptateur comme ayant une capacité de fédération.
	<start-point-adapter>	Spécifie que l'adaptateur est le point de départ du calcul de la requête TQL.
	<one-node-topology>	Capacité de fédérer des requêtes avec un seul nœud de requête fédérée.
	<pattern-topology>	Capacité de fédérer des requêtes complexes.
	<support-replicatioin-data>	Définit la capacité d'exécuter des flux de remplissage et d'émission de données.
	<source>	Cet adaptateur peut être utilisé pour les flux de remplissage.
	<push-back-ids>	Renvoie l'ID global du CI vers la colonne global_id de la table (doit être défini dans le fichier orm.xml). Le comportement peut être remplacé par l'implémentation du plug-in **FcmdbPluginPushBackIds**.
	<changes-source>	Cet adaptateur peut être utilisé pour les flux de modifications de remplissage.

	<instance-based-data>	Cette balise spécifie que l'adaptateur prend en charge un flux de remplissage basé sur une instance.
	<target>	Cet adaptateur peut être utilisé pour les flux d'émission de données.
	<default-mapping-engine>	Permet de définir un moteur de mappage pour l'adaptateur (par défaut, l'adaptateur utilise le moteur de mappage par défaut). Pour tout autre moteur de mappage, entrez le nom de la classe d'implémentation du moteur de mappage
	<removedAttributes>	Contraint la suppression d'attributs spécifiques dans le résultat.
	<full-population-days-interval>	Indique quand exécuter un travail de remplissage intégral au lieu d'un travail différentiel (tous les "x" jours). Utilise le mécanisme de vieillissement couplé au flux de modifications.
	<adapter-settings>	La liste des paramètres de l'adaptateur.
	<list.attributes.for.set>	Détermine quels sont les attributs qui remplacent la valeur précédente (s'il en existe une).

Interface DataAdapterEnvironment

OutputStream openResourceForWriting(String resourceName) throws FileNotFoundException;

Cette méthode ouvre une ressource d'un nom donné à des fins d'écriture. Elle permet d'enregistrer les données persistantes pour l'intégration. Elle doit être utilisée au lieu des méthodes Java lors d'une tentative de chargement de fichier. L'utilisateur doit s'assurer que le flux de données est fermé une fois l'écriture terminée dans le flux. La ressource sera enregistrée par close()/flush(). Cette méthode crée une ressource d'exécution (elle ne peut pas remplacer les fichiers fournis dans le package de l'adaptateur).

Paramètre

- **resourceName :** nom de la ressource à extraire. Ce nom doit être unique dans toutes les intégrations du même adaptateur.

Valeur renvoyée

Renvoie un flux de données autorisant l'accès en écriture.

Exceptions

- Cette méthode lève l'exception *FileNotFoundException* si la ressource est du type fichier et que le fichier n'existe pas, si la ressource est un répertoire au lieu d'un fichier standard ou pour toute autre raison empêchant d'ouvrir la ressource en mode lecture.
- Cette méthode lève l'exception *SecurityException* s'il existe un gestionnaire de la sécurité et que sa méthode *checkRead* refuse l'accès au fichier.

InputStream openResourceForReading(String resourceName) throws FileNotFoundException;

Cette méthode ouvre une ressource d'un nom donné à des fins de lecture. Elle permet de lire les données persistantes pour l'intégration. Elle doit être utilisée au lieu des méthodes Java lors d'une tentative de chargement de fichier. L'utilisateur doit s'assurer que le flux de données est fermé une fois la lecture terminée. Cette méthode tente d'abord de charger les fichiers livrés avec le package de l'adaptateur. S'ils sont introuvables, elle tente de charger une ressources d'exécution créée à partir de la méthode *DataAdapterEnvironment.openResourceForWriting(String)*. Il est possible d'afficher les ressources d'exécution à partir de la console JMX (de la sonde et du serveur, selon le cas).

Paramètre

- **resourceName :** nom de la ressource à extraire. Ce nom doit être unique dans toutes les intégrations du même adaptateur.

Valeur renvoyée

Renvoie un flux de données à lire.

Exceptions

- Cette méthode lève l'exception *FileNotFoundException* si la ressource est du type **fichier** et que le fichier n'existe pas, si la ressource est un répertoire au lieu d'un fichier standard ou pour toute autre raison empêchant d'ouvrir la ressource en mode lecture.
- Cette méthode lève l'exception *SecurityException* s'il existe un gestionnaire de la sécurité et que sa méthode *checkRead* refuse l'accès au fichier.

Properties openResourceAsProperties(String propertiesFile) throws IOException;

Cette méthode ouvre une ressource d'un nom donné et la charge en tant que structure de *propriétés*. Elle permet de lire les données persistantes pour l'intégration. Elle doit être utilisée au lieu des méthodes Java lors d'une tentative de chargement de fichiers **.properties**. Cette méthode tente d'abord de charger les fichiers livrés avec le package de l'adaptateur. S'ils sont introuvables, elle tente de charger une ressources d'exécution créée à partir de la méthode *DataAdapterEnvironment.openResourceForWriting(String)*. Il est possible d'afficher les ressources d'exécution à partir de la console JMX (de la sonde et du serveur, selon le cas).

Paramètre

- **propertiesFile :** nom de la ressource à extraire. Ce nom doit être unique dans toutes les intégrations du même adaptateur.

Valeur renvoyée

Renvoie le contenu du fichier représenté dans la structure de propriétés.

Exceptions

- Cette méthode lève l'exception *FileNotFoundException* si la ressource est du type **fichier** et que le fichier n'existe pas, si la ressource est un répertoire au lieu d'un fichier standard ou pour toute autre raison empêchant d'ouvrir la ressource en mode lecture.
- Cette méthode lève l'exception *SecurityException* s'il existe un gestionnaire de la sécurité et que sa méthode *checkRead* refuse l'accès au fichier.
- Cette méthode lève l'exception *IOException* si la conversion du fichier de propriétés en objet *Propriétés* a échoué.

String openResourceAsString(String resourceName) throws IOException;

Cette méthode ouvre une ressource d'un nom donné et la charge en tant que chaîne. Elle permet de lire les données persistantes pour l'intégration. Elle doit être utilisée au lieu des méthodes Java lors d'une tentative de chargement de fichier.

Cette méthode tente d'abord de charger les fichiers livrés avec le package de l'adaptateur. S'ils sont introuvables, elle tente de charger une ressources d'exécution créée à partir de la méthode *DataAdapterEnvironment.openResourceForWriting(String)*. Il est possible d'afficher les ressources d'exécution à partir de la console JMX (de la sonde et du serveur, selon le cas).

Paramètre

- **resourceName :** nom de la ressource à extraire. Ce nom doit être unique dans toutes les intégrations du même adaptateur.

Valeur renvoyée

Renvoie le contenu du fichier représenté dans le format chaîne (String).

Exceptions

- Cette méthode lève l'exception *FileNotFoundException* si la ressource est du type **fichier** et que le fichier n'existe pas, si la ressource est un répertoire au lieu d'un fichier standard ou pour toute autre raison empêchant d'ouvrir la ressource en mode lecture.
- Cette méthode lève l'exception *SecurityException* s'il existe un gestionnaire de la sécurité et que sa méthode *checkRead* refuse l'accès au fichier.
- Cette méthode lève l'exception *IOException* si une erreur d'E/S se produit.

public void saveResourceFromString(String relativeFileName, String value) throws IOException;

Cette méthode reçoit une chaîne et l'enregistre en tant que ressource. Elle permet d'enregistrer les données persistantes pour l'intégration. Elle doit être utilisée au lieu des méthodes Java lors d'une tentative d'enregistrement de fichier. Cette méthode convertit la chaîne en un flux de données et l'enregistre dans la ressource. Elle crée une ressource d'exécution, mais ne peut pas remplacer les fichiers fournis dans le package de l'adaptateur. Il est possible d'afficher les ressources d'exécution à partir de la console JMX (de la sonde et du serveur, selon le cas).

Paramètre

- **relativeFileName :** nom de la ressource à extraire. Ce nom doit être unique dans toutes les intégrations du même adaptateur.
- **value :** chaîne à enregistrer comme ressource.

Exceptions

Cette méthode lève l'exception IOException si une erreur d'E/S se produit.

boolean resourceExists(String resourceName);

Cette méthode vérifie si un nom de ressource donné existe. Elle recherche les fichiers livrés avec le package de l'adaptateur et les ressources d'exécution créées à partir de la méthode *DataAdapterEnvironment.openResourceForWriting(String)*.

Paramètre

- **resourceName :** nom de la ressource à extraire. Ce nom doit être unique dans toutes les intégrations du même adaptateur.

Valeur renvoyée

Renvoie la valeur **True** si le nom de ressource (*resourceName*) existe.

boolean deleteResource(String resourceName);

Cette méthode supprime la ressource des données persistantes. Elle supprime une ressource d'exécution, mais ne peut pas supprimer les fichiers fournis dans le package de l'adaptateur. Il est possible d'afficher les ressources d'exécution à partir de la console JMX (pour la sonde et le serveur, selon le cas).

Paramètre

- **resourceName :** nom de la ressource à supprimer. Ce nom doit être unique dans toutes les intégrations du même adaptateur.

Valeur renvoyée

Renvoie la valeur **True** si la ressource a été supprimée.

Collection<String> listResourcesInPath(String path);

Cette méthode extrait une liste de ressources dans le chemin de ressources donné. Elle recherche les fichiers livrés avec le package de l'adaptateur et les ressources d'exécution créées à partir de la méthode *DataAdapterEnvironment.openResourceForWriting(String)*. Il est possible d'afficher les ressources d'exécution à partir de la console JMX (pour la sonde et le serveur, selon le cas).

Paramètre

- **path :** chemin des ressources. Par exemple, « META-INF/myfiles/ »

Valeur renvoyée

Renvoie une liste de ressources dans le chemin.

DataAdapterLogger getLogger();

Renvoie le journal que l'adaptateur doit utiliser. Ce journal permet de consigner les événements de votre adaptateur.

Valeur renvoyée

Renvoie le journal utilisé par l'adaptateur de données.

DestinationConfig getDestinationConfig();

Cette méthode extrait la configuration de destination de l'intégration. Cette configuration inclut tous les paramètres de connexion et d'exécution de l'intégration.

Valeur renvoyée

Renvoie la configuration de destination de l'adaptateur.

int getChunkSize();

Cette méthode extrait la taille du segment de remplissage demandé pour l'intégration.

Valeur renvoyée

Renvoie la taille du segment de remplissage.

int getPushChunkSize();

Cette méthode extrait la taille du bloc d'émission de données demandé pour l'intégration.

Valeur renvoyée

Renvoie la taille du bloc d'émission de données.

ClassModel getLocalClassModel();

Cette méthode extrait un modèle de classe pour rechercher des informations relatives au modèle de classe de l'ordinateur UCMDB local. Cette méthode extrait un modèle de classe actualisé. Une fois l'objet ClassModel renvoyé, il n'est pas mis à jour pour les modifications de modèle de classe. Pour extraire un modèle de classe actualisé, vous devrez réutiliser cette méthode.

Valeur renvoyée

Renvoie le modèle de classe de UCMDB.

CustomerInformation getLocalCustomerInformation();

Cette méthode extrait les informations relatives au client qui exécute l'adaptateur.

Valeur renvoyée

Renvoie les informations relatives au client qui exécute l'adaptateur.

Object getSettingValue(String name);

Cette méthode extrait un paramètre spécifique de l'adaptateur.

Paramètre

name : nom du paramètre.

Valeur renvoyée

Renvoie la valeur du paramètre Object.

Map<String, Object> getAllSettings();

Cette méthode extrait tous les paramètres de l'adaptateur.

Valeur renvoyée

Renvoie les paramètres de l'adaptateur.

boolean isMTEnabled();

Cette méthode vérifie si l'environnement du serveur prend en charge le mode multilocataire.

Valeur renvoyée

Renvoie la valeur **true** si l'environnement du serveur prend en charge le mode multilocataire ; sinon, la valeur **false** est renvoyée.

String getUcmdbServerHostName();

Cette méthode renvoie le nom d'hôte du serveur UCMDB local.

Valeur renvoyée

Renvoie le nom d'hôte du serveur UCMDB local.

Chapitre 6 : Développement d'adaptateurs d'émission (push)

Contenu de ce chapitre :

Développement et déploiement d'adaptateurs d'émission de données 192

Création d'un package d'adaptateur 193

Création de mappages 197

Écriture de scripts Jython 201

Prise en charge de la synchronisation différentielle 205

Requêtes SQL de l'adaptateur d'émission de données XML générique 207

Adaptateur d'émission de données de service Web générique 207

Fichier de mappage de référence 227

Schéma du fichier de mappage 229

Schéma des résultats de mappage 241

Personnalisation 245

Développement et déploiement d'adaptateurs d'émission de données

Les adaptateurs d'émission de données génériques offrent une plate-forme permettant le développement rapide d'intégrations qui émettent des données UCMDB vers des référentiels de données externes (bases de données et applications tierces). Les adaptateurs d'émission de données génériques sont classés en fonction du protocole utilisé pour l'émission des données. Pour plus d'informations sur l'émission de données via XML à l'aide de l'adaptateur d'émission de données XML générique, voir " Requêtes SQL de l'adaptateur d'émission de données XML générique " , page 207. Pour plus d'informations sur l'émission de données via un service Web à l'aide de l'adaptateur d'émission de données de service Web générique, voir " Adaptateur d'émission de données de service Web générique " , page 207.

Le développement d'une intégration personnalisée sur la base d'un adaptateur d'émission générique requiert les opérations suivantes :

- La création d'un package d'adaptateur à partir des fichiers de modèles d'adaptateurs d'émission de données génériques appropriés. Pour plus d'informations, voir " Création d'un package d'adaptateur " , page suivante.

- Des mappages entre les types de lien de CI UCMDB et les éléments de données externes. Les mappages sont stockés au format XML et sont personnalisés par rapport à chaque référentiel de données externe. Pour plus d'informations, voir " Création de mappages " , page 197.

- Un script Jython pour émettre les éléments de données dans le référentiel de données externe. Pour plus d'informations, voir " Écriture de scripts Jython " , page 201.

- Des étapes supplémentaires propres à l'adaptateur. Par exemple, choix du chemin d'accès au fichier à écrire pour l'adaptateur d'émission de données XML ou création d'un récepteur de données pour l'adaptateur d'émission de données de service Web.

Création d'un package d'adaptateur

Pour créer un adaptateur d'émission propre au référentiel de données géré (MDR), copiez l'adaptateur générique puis modifiez-le pour le personnaliser en tant qu'adaptateur destiné à une cible d'émission spécifique.

Les packages d'adaptateurs génériques sont stockés dans l'un des emplacements suivants :

- Adaptateur d'émission de données XML générique : **hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip**

- Adaptateur de service Web générique : **hp\UCMDB\UCMDBServer\content\adapters\web-service-push-adapter.zip**

Pour créer un adaptateur d'émission de données à partir de l'adaptateur d'émission de données générique :

1. Extrayez le contenu du fichier zip du package sélectionné dans un dossier de travail.

2. Préparez les répertoires suivants à la phase de remplacement et de changement de nom :

   - **adapterCode :** contient le répertoire qui est déployé vers **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\répertoire CodeBase**. Les fichiers jar déployés ici ne redémarrent pas automatiquement la sonde et n'apparaissent pas automatiquement dans le CLASSPATH de la sonde.

   - **discoveryConfigFiles :** contient les définitions de mappage de l'adaptateur et pointe sur le script Jython correct (**push.properties**).

   - **discoveryPatterns :** contient la définition XML de l'adaptateur déployé sur le serveur UCMDB.

   - **discoveryScripts :** contient les scripts Jython de l'adaptateur au moyen desquels la connexion au magasin de données tiers est établie et les données émises.

   - **discoveryResources :** contient le fichier **UCMDBDataReceiver.jar** incluant les classes d'intégration Java pour le service Web.

> **Remarque :** Lorsque vous déployez ce package, la sonde est redémarrée afin que ce fichier **.jar** soit inclus dans le CLASSPATH de la sonde. Aucune action n'est nécessaire au-delà du déploiement du package.

3. Effectuez les modifications suivantes dans la structure de répertoires de l'adaptateur non décompressé :

   a. **discoveryConfigFiles\<Nom_de_votre_adaptateur_d'émission_de_données> :** renommez le répertoire PushAdapter ou XMLtoWebService en lui attributant le nom du nouvel adaptateur d'émission de données (par exemple, myPushAdapter).

   b. **discoveryConfigFiles\<Nom_de_votre_adaptateur_d'émission_de_données>\push.properties :** dans le fichier **push.properties**, procédez comme suit :

      - Mettez à jour le nom du fichier **jythonScript.name** en lui attribuant le nom du script Jython que doit utiliser le nouvel adaptateur d'émission de données (par exemple, **pushToMyService.py**).

      - Mettez à jour le nom du fichier de mappage que doit utiliser le nouvel adaptateur d'émission de données (par exemple, myPushAdapter_mappings). N'ajoutez pas l'extension **.xml**, car elle est ajoutée automatiquement.

   c. **discoveryPatterns\<push adapter name>.xml :** Renommez ce fichier en lui attribuant le nom du fichier de définition XML du nouvel adaptateur (par exemple, **my_push_adapter.xml**).

   d. **discoveryPatterns\<votre_adaptateur_d'émission_de_données>.xml :** Mettez à jour ce fichier en procédant comme suit :

      - Pour l'élément XML **<pattern> :** définissez les attributs d'ID et de description en conséquence. Exemple :

        ```
        <pattern id="PushAdapter"
        xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
        description="Discovery Pattern Description" schemaVersion="9.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ```

        est remplacé par :

        ```
        <pattern id="MyPushAdapter" displayLabel="My Push Adapter"
        xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
        description="Discovery Pattern Description" schemaVersion="9.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        ```

      - Pour l'élément XML **<parameters> :** mettez à jour les éléments enfants en fonction des besoins de votre adaptateur. Par défaut, les éléments enfants ci-après sont utilisés pour définir un adaptateur d'émission de données. Ces valeurs sont affectées lors de la

définition du point d'intégration dans le Studio d'intégration, une fois l'adaptateur configuré. Mettez à jour la liste des paramètres afin qu'elle reflète les attributs de connexion requis. Ne supprimez pas l'attribut **probeName**.

- **host :** nom du serveur hébergeant le service Web.
- **port :** port d'écoute du service de réception de données de UCMDB.
- **Adaptateur d'émission de données de service Web : uri** - Reste de l'URL formant l'adresse du point de terminaison de service du récepteur de données.
- **probeName :** définit l'instance Data Flow Probe sur laquelle le travail d'émission de données est exécuté.

- Pour l'élément XML **<integration> :** mettez à jour la valeur de l'élément enfant **<category>** en lui attribuant une valeur autre que Generic. Par défaut, les adaptateurs d'intégration appartenant à la catégorie Generic n'apparaissent pas dans le Studio d'intégration. Si vous effectuez une intégration dans un magasin de données tiers, attribuez la valeur « Third Party ». Si vous effectuez une intégration dans un produit HP BTO, attribuez la valeur « HP BTO Products ».

e. **adapterCode\PushAdapter :** renommez ce dossier avec l'ID d'adaptateur utilisé à l'étape précédente (par exemple, **adapterCode\MyPushAdapter**).

f. **discoveryScripts\<votre_script_d'émission_de_données_Jython>.py :** créez un fichier avec le même nom que celui défini dans la propriété **push.properties jythonScript.name**. Le fichier **discoveryScript** comprend un script qui insère les CI et les liens dans une base de données Oracle externe. Remplacez **discoveryScripts\pushScript.py** par le script que vous avez écrit (pour plus de détails, voir " Écriture de scripts Jython " , page 201). Si vous renommez le script, la propriété **jythonScript.name** dans **adapterCode\<ID adaptateur>\push.properties** doit être mise à jour en conséquence.

   - Adaptateur d'émission de données XML : **pushScript.py**
   - Adaptateur d'émission de données de service Web : **XMLtoWebService.py**

g. **tql\<vos_TQL_d'intégration> :** comme pour un package standard, placez la définition XML TQL de vos codes TQL d'intégration dans ce répertoire. Tous les codes TQL de ce dossier sont déployés lors du déploiement du package d'adaptateur.

h. **discoveryConfigFiles\<Nom_de_votre_adaptateur_d'émission_de_données>\mappings :** créez un fichier de mappage XML selon le code TQL que vous souhaitez utiliser dans votre intégration. Notez que l'adaptateur d'émission de données applique les transformations du fichier de mappage aux résultats des codes TQL d'intégration, puis envoie les données à l'instance Data Flow Probe dans les trois paramètres (**addResult**, **updateResult** et **deleteResult**) d'une tâche ad hoc.

i. **adapterCode\<ID adaptateur>\mappings :** remplacez le fichier **mappings.xml** par les

fichiers de mappage que vous avez préparés (pour plus d'informations, voir " Création de mappages " , page suivante).

Adaptateur d'émission de données XML : cet exemple de mappage correspond à celui des tables créées sous ORACLE dans le fichier sql_queries.

Pour utiliser un fichier de mappage pour chaque méthode TQL, affectez le nom de la méthode TQL correspondante à chaque fichier XML, suivi de l'extension **.xml**. Dans ce cas, le fichier **mappings.xml** est utilisé par défaut s'il n'existe aucun fichier de mappage spécifique pour le nom TQL actuel. Il est possible de modifier le nom du fichier de mappage par défaut en changeant la propriété **mappingFile.default** dans **adapterCode\<ID adaptateur>\push.properties**.

4. Une fois toutes les modifications ci-dessus effectuées, créez un fichier **.zip** en sélectionnant les dossiers et les fichiers définis plus haut à l'étape 3 (par exemple, **my_Push_Adapter.zip**).

5. Déployez le fichier **.zip** nouvellement créé dans le serveur UCMDB via le Gestionnaire des packages (sélectionnez **Administration > Gestionnaire des packages**).

6. Créez un point d'intégration en sélectionnant **Gestion des flux de données > Studio d'intégration**, puis définissez les codes TQL d'intégration utilisés par le point d'intégration. Planifiez l'émission de données automatique.

Résolution des problèmes

La procédure de création d'un adaptateur d'émission de données requiert une phase de remplacement et d'attribution de noms complète et correcte. Toute erreur aura une incidence sur l'adaptateur. Le package doit être décompressé puis recompressé correctement afin qu'il se comporte comme un package UCMDB. Reportez-vous aux packages prêts à l'emploi à titre d'exemple. Les erreurs courantes sont les suivantes :

- Inclusion d'un autre répertoire au début des répertoires du package dans le fichier ZIP.
  Solution : zippez la package dans le même répertoire que les répertoires de package tels que **discoveryResources**, **adapterCode**, etc. N'incluez aucun autre niveau de répertoire au-dessus de ces répertoires dans le fichier ZIP.

- Omission d'un changement de nom critique pour un répertoire, un fichier ou une chaîne dans un fichier.
  Solution : suivez attentivement les instructions de cette section.

- Changement de nom critique incorrect pour un répertoire, un fichier ou une chaîne dans un fichier.
  Solution : ne changez pas de convention de dénomination une fois que vous avez commencé la procédure de changement de nom. Si vous réalisez que vous devez modifier un nom, recommencez toute la procédure au lieu de tenter de corriger le nom rétroactivement, afin d'éviter tout risque d'erreur. De même, utilisez la fonction de recherche et de remplacement au lieu de remplacer manuellement les chaînes pour réduire le risque d'erreur.

- Déploiement d'adaptateurs portant les mêmes noms de fichier que d'autres adaptateurs, particulièrement dans les répertoires **discoveryResources** et **adapterCode**.
  Solution : vous utilisez peut-être une version UCMDB présentant un problème connu qui empêche les fichiers de mappage de porter le même nom qu'un autre adaptateur dans le même environnement UCMDB. Si vous tentez de déployer un package avec des noms en double, le déploiement risque d'échouer. Ce problème peut se produire même si les fichiers se trouvent dans des répertoires différents. Il peut également se produire que les doublons existent au sein du package ou avec des noms existant déjà dans d'autres packages déployés précédemment.

À ce stade, vous pouvez créer un travail d'adaptateur d'émission de données dans le Studio d'intégration à l'aide du nouvel adaptateur que vous venez de déployer.

Meilleures pratiques TQL recommandées pour les adaptateurs d'émission de données

1. Créez une structure de dossiers dans les arborescences de vues et TQL, et conservez tous les nouveaux codes TQL et les nouvelles vues à cet emplacement. Utilisez une convention de dénomination.

2. Copiez d'abord le code TQL le plus ressemblant, sauf s'il est de petite taille.

3. Effectuez une modification à la fois. Enregistrez, testez et prévisualisez après chaque modification. Répétez ces actions jusqu'à ce que les résultats répondent à vos besoins.

Création de mappages

Les données brutes du résultat TQL sont au format du schéma de modèle de classe UCMDB. Il est fort probable que l'abonne utilise un modèle de données différent. L'adaptateur d'émission de données fournit un mécanisme de mappage pour transformer les données dans un format plus adapté à la consommation. Les mappages effectuent des transformations à la fois directes et complexes, depuis la conversion dénomination-type directe jusqu'aux fonctions de référencement et d'agrégation parent/enfant.

La section " Fichier de mappage de référence " , page 227 décrit la définition de mappage. Utilisez cette référence pour créer un fichier de mappage.

> **Remarque :** Le fichier de propriétés d'un adaptateur se rapporte au nom du fichier de mappage. Dans les fichiers de configuration d'adaptateur, l'adaptateur implémente une structure de dossiers en utilisant le nom de l'adaptateur. Renommez ce dossier lors de l'implémentation d'un adaptateur pour conserver l'unicité requise par le Gestionnaire des packages.

Création d'un fichier de mappage

Pour créer un fichier de mappage :

1. Commencez avec un fichier de mappage par défaut.
2. Déployez l'adaptateur et exécutez-le une fois.
3. Observez les résultats.
4. Identifiez et notez les éléments à modifier.
5. Effectuez les modifications identifiées à l'étape précédente. La liste suivante peut vous aider à définir l'ordre des modifications.
   a. Commencez par la section du début, non transformable. Veillez à exécuter l'adaptateur après chaque modification.
   b. Remplacez la section des CI source par les noms UCMDB dans le résultat TQL.
   c. Mappez d'abord les clés.
   d. Ajoutez ensuite tous les mappages directs.
   e. Ajoutez les mappages complexes.
   f. Ajoutez les mappages de liens.

Répétez les étapes 2 à 5 jusqu'à ce que les données mappées soient adaptées à la consommation. Sélectionnez le package d'adapteur générique approprié pour créer le nouvel adaptateur d'émission de données.

Les fichiers de mappage fonctionnent de la même façon pour tous les types d'adaptateur d'émission de données. L'adaptateur d'émission de données XML générique écrit les résultats mappés dans un fichier. L'adaptateur d'émission de données de service Web envoie les résultats XML à un récepteur de données. Pour plus d'informations, voir " Adaptateur d'émission de données de service Web générique " , page 207.

Préparation des fichiers de mappage

> **Remarque :** Vous pouvez extraire tous les CI et relations tels qu'ils existent dans le CMDB sans mappage, en omettant la création du fichier **mappings.xml**. Ainsi, tous les CI et relations sont renvoyés avec leurs attributs.

Il existe deux façons de préparer des fichiers de mappagage :

- Vous pouvez préparer un fichier global unique.

  Tous les mappages sont alors placés dans un fichier unique appelé **mappings.xml**.

- Vous pouvez préparer un fichier distinct pour chaque requête d'émission (push).

  Chaque fichier de mappage est appelé **<nom requête>.xml**.

Pour plus d'informations, voir " Schéma du fichier de mappage " , page 229.

Cette tâche comprend les étapes suivantes :

- " Création d'un fichier mappings.xml " , ci-dessous
- " Mappage des CI " , ci-dessous
- " Mappage des liens " , page suivante

1. **Création d'un fichier mappings.xml**

   La structure des fichiers de mappage est créée comme suit (utilisez un fichier existant comme modèle) :

```
<?xml version="1.0" encoding="UTF-8"?>
<integration>
    <info>
        <source name="UCMDB"  versions="9.x" vendor="HP" >
        <!-- for example: -->
        <target name="Oracle" versions="11g" vendor="Oracle" >
    </info>
    <targetcis>
        <!--- CI Mappings --->
    </targetcis>
    <targetrelations>
        <!--- Link Mappings --->
    </ targetrelations>
</integration>
```

2. **Mappage des CI**

   Il existe deux façons de mapper les types de CI CMDB :

   - Mapper un type de CI de sorte que les CI de ce type et tous les types hérités soient mappés de la même façon :

```
<source_ci_type_tree name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey>
            <pkey>name</pkey>
        </targetprimarykey
        <target_attribute name=" name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
        <!-- more target attributes --->
```

```
    </target_ci_type>
</source_ci_type_tree>
```

- Mapper un type de CI de sorte que seuls les CI de ce type soient traités. Les CI des types hérités ne seront pas traités à moins que leur type soit également mappé (de l'une des deux façons) :

```
<source_ci_type name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey>
            <pkey>name</pkey>
        </targetprimarykey
        <target_attribute name=" name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
        <!-- more target attributes --->
    </target_ci_type>
</source_ci_type>
```

Un type de CI qui est mappé de façon indirecte (l'un de ses ancêtres est mappé à l'aide de **source_ci_type_tree**), peut également remplacer la carte de son parent en la faisant apparaître dans son propre **source_ci_type_tree** ou **source_ci_type**.

Il est recommandé d'utiliser **source_ci_type_tree** lorsque cela est possible. Sinon, les CI résultant d'un type de CI qui n'apparaît pas dans les fichiers de mappage ne seront pas transférés dans le script Jython.

3. **Mappage des liens**

Il existe deux façons de mapper des liens :

- Mapper un lien de sorte que les liens de ce type et tous les liens hérités soient mappés de la même façon :

```
<source_link_type_tree name="dependency" target_link_type="dependency"
mode="update_else_insert" source_ci_type_end1="webservice" source_ci_t
ype_end2="sap_gateway">
    <target_ci_type_end1 name="webservice" >
    <target_ci_type_end2 name="sap_gateway" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
</source_link_type_tree>
```

- Mapper un lien de sorte que seuls les liens de ce type soient traités. Les liens des types hérités ne seront pas traités à moins que leur type soit également mappé (de l'une des deux façons) :

```
<link source_link_type="dependency" target_link_type="dependency" mode
="update_else_insert" source_ci_type_end1="webservice" source_ci_type_
end2="sap_gateway">
        <target_ci_type_end1 name="webservice" >
        <target_ci_type_end2 name="sap_gateway" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
    </target_attribute>
</link>
```

Écriture de scripts Jython

Le script de mappage est un script Jython normal, qui doit respecter les règles de ce type de script. Pour plus d'informations, voir " Développement d'adaptateurs Jython " , page 39.

Le script doit contenir la fonction **DiscoveryMain**, qui peut renvoyer soit une instance **OSHVResult** vide, soit une instance **DataPushResults** en cas de réussite.

Pour signaler un échec, le script doit lever une exception, par exemple :

```
raise Exception('Failed to insert to remote UCMDB using TopologyUpdateService. S
ee log of the remote UCMDB')
```

Dans la fonction DiscoveryMain, les éléments de données à émettre vers l'application externe ou à supprimer peuvent être obtenus comme suit :

```
# get add/update/delete result objects (in XML format) from the Framework
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
```

L'objet client vers l'application externe peut être obtenu comme suit :

```
oracleClient = Framework.createClient()
```

Cet objet client utilise automatiquement l'ID, le nom d'hôte et le numéro de port des informations d'identification, qui sont transmis à l'adaptateur par le biais de l'infrastructure.

Si vous avez besoin des paramètres de connexion que vous avez définis pour l'adaptateur (voir l'étape de modification du fichier **discoveryPatterns\push_adapter.xml** à la section " Création d'un package d'adaptateur " , page 193), utilisez le code suivant :

```
propValue = str(Framework.getDestinationAttribute('<Connection Property Name'))
```

Par exemple :

```
serverName = Framework.getDestinationAttribute('ip_address')
```

Autres rubriques de cette section :

- " Utilisation des résultats du mappage " , ci-dessous
- " Gestion d'un test de connexion dans le script " , page 205

Utilisation des résultats du mappage

Les adaptateurs d'émission de données génériques créent des chaînes XML qui décrivent les données à ajouter, mettre à jour ou supprimer dans le système cible. Le script Jython doit analyser ce code XML, puis effectue l'opération d'ajout, de mise à jour ou de suppression sur la cible.

Dans le code XML de l'opération d'ajout reçue par le script Jython, l'attribut `mamId` pour les objets et liens est toujours l'identifiant UCMDB de l'objet ou du lien d'origine avant que son type, son attribut ou d'autres informations soient modifiés conformément au schéma du système distant.

Dans le code XML des opérations de mise à jour ou de suppression, l'attribut `mamId` de chaque objet ou lien contient la représentation de type chaîne du même `ExternalId` qui a été renvoyé du script Jython lors de la synchronisation précédente.

Dans le code XML, l'attribut `id` d'un CI contient l'ID `cmdbId` comme ID externe ou l'attribut `ExternalId` de ce CI si celui-ci a reçu un attribut `ExternalId` lors de son envoi au script. Les champs `end1Id` et `end2Id` du lien contiennent pour les extrémités de chaque lien l'ID `cmdbId` comme ID externe ou l'`ExternalId` de l'extrémité de ce lien si le CI situé à l'extrémité de ce lien a reçu un `ExternalId` lors de son envoi au script.

Lors du traitement des CI dans le script Jython, la valeur de retour du script constitue un mappage entre l'ID de CMDB du CI et l'ID affecté (ID attribué à chaque CI dans le script). Si un CI est émis pour la première fois, l'ID qui se trouve dans son script XML correspond à l'ID de CMDB. S'il ne s'agit pas d'une première émission, l'ID du CI est identique à celui qui lui a été attribué dans le script lorsqu'il a été émis pour la première fois.

L'ID est extrait du script XML du CI comme suit :

1. À partir de l'élément CI dans le code XML, récupérez l'ID de l'attribut id. Par exemple : `id = objectElement.getAttributeValue('id')`.

2. Après avoir récupéré l'ID du code XML, restaurez l'ID à partir de l'attribut (string). Par exemple : `objectId = CmdbObjectID.Factory.restoreObjectID(id)`.

3. Vérifiez que l'ID `objectId` reçu à l'étape précédente correspond à l'ID de CMDB. Pour ce faire, il s'agit de vérifier que le nouvel ID attribué à `objectId` est celui qui lui a été affecté par le script. Dans la négative, l'ID renvoyé ne correspond pas à l'ID de CMDB. Par exemple : `newId = objectId.getPropertyValue(<nom de l'attribut d'ID fourni par le script>)`.

   Si l'attribut `newId` a la valeur NULL, l'ID renvoyé dans l'XML est un ID de CMDB.

4. Si l'ID est un ID de CMDB (c-à-d., `newId` a la valeur NULL), effectuez les opérations suivantes (si l'ID n'est pas un ID de CMDB, passez à l'étape 5) :

   a. Créez une propriété pour le CI qui contient le nouvel ID. Par exemple : `propArray = [TypesFactory.createProperty('<nom de l'attribut d'ID fourni par le`

```
script>', '<nouvel ID>')].
```

b. Créez un attribut `externaId` pour le CI. Par exemple :
```
cmdbId = extI.getPropertyValue('internal_id')
className = extI.getType()
externalId = ExternalIdFactory.createExternalCiId(className, propArray)
```

c. Mappez l'ID de CMDB à l'attribut `externalId` nouvellement créé (et renvoyez ce mappage à l'adaptateur à l'étape suivante). Par exemple : `objectMappings.put(cmdbId, externalId)`

d. Une fois que tous les CI et liens sont mappés :
```
updateResult = DataPushResultsFactory.createDataPushResults
(objectMappings, linkMappings);
return updateResult
```

5. Si l'ID constitue le nouvel ID (la valeur de `newId` est différente de NULL), `externalId` correspond à `newId`.

Il est également possible de communiquer comme suit le statut d'émission de données pour chaque CI et lien :

1. `updateStatus = ReplicationActionDataFactory.createUpdateStatus();`

   où `updateStatus` est une instance de la classe `UpdateStatus` contenant les statuts des CI et des liens.

2. Ajoutez un statut à `updateStatus` en appelant la méthode `reportCIStatus` ou `reportRelationStatus`.

   Exemple :

   ```
   status = ReplicationActionDataFactory.createStatus(Severity.FAILURE, 'Faile
   d', ERROR_CODE_CI, errorParams,Action.ADD);
   ```

   ```
   updateStatus.reportCIStatus(externalId, status);
   ```

   où `ERROR_CODE_CI` est le numéro des messages d'erreur tels qu'ils apparaissent dans le fichier **properties.errors** (pour plus d'informations sur ce fichier, voir " Conventions d'écriture des erreurs " , page 68), et `errorParams` contient les paramètres à transmettre au message. Pour plus d'informations, voir la méthode **ReplicationActionDataFactory** de Javadoc.

3. Créez un résultat d'émission de données avec les statuts, comme suit :

   ```
   updateResult = DataPushResultsFactory.createDataPushResults(objectMappings,
   linkMappings, updateStatus);
   ```

   ```
   return updateResult
   ```

Exemple de résultat XML

```
<root>
  <data>
    <objects>
      <Object mode="update_else_insert" name="UCMDB_UNIX" operation="add" ma
mId="0c82f591bc3a584121b0b85efd90b174" id="HiddenRmiDataSource%0Aunix%0A1%0A
internal_id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A">
        <field name="NAME" key="false" datatype="char" length="255">UNIX5</field>
        <field name="DATA_NOTE" key="false" datatype="char" length="255"></field>
      </Object>
    </objects>
    <links>
        <link targetRelationshipClass="TALK" targetParent="unix" targetChild="unix"
operation="add" mode="update_else_insert"
mamId="265e985c6ec51a8543f461b30fa58f81"
id="end1id%5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%3D41372a1
cbcaba27b214b84a2ec9eb535%0A%5D%0Aend2id%
5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%3D0c82f591bc3a584121
b0b85efd90b174%0A%5D%0AHiddenRmi
DataSource%0Atalk%0A1%0Ainternal_id%3DSTRING%3D265e985c6ec51a8543f461b30fa58
f81%0A">
        <field name="DiscoveryID1">41372a1cbcaba27b214b84a2ec9eb535</field>
        <field name="DiscoveryID2">0c82f591bc3a584121b0b85efd90b174</field>
        <field name="end1Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%
3D41372a1cbcaba27b214b84a2ec9eb535%0A</field>
        <field name="end2Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%
3D0c82f591bc3a584121b0b85efd90b174%0A</field>
        <field name="NAME" key="false" datatype="char" length="255">TALK4</field>
        <field name="DATA_NOTE" key="false" datatype="char" length="255"></field>
        </link>
    </links>
  </data>
</root>
```

Remarque : Si `datatype="BYTE"`, la valeur du résultat renvoyé est une **chaîne** générée sous la forme : `new String([the byte array attribute])`. `byte[] object` peut être reconstruit par : `<chaîne reçue>.getBytes()`. S'il existe des différences dans les paramètres régionaux par défaut entre le serveur et la sonde, la reconstruction est effectuée en fonction des

paramètres régionaux par défaut du serveur.

Gestion d'un test de connexion dans le script

Un script Jython peut être appelé pour tester la connexion avec une application externe. Dans ce cas, l'attribut de destination `testConnection` aura la valeur `true`. Cet attribut peut être obtenu de l'infrastructure de la manière suivante :

```
testConnection = Framework.getTriggerCIData('testConnection')
```

En cas d'exécution en mode test de connexion, un script doit lever une exception s'il est impossible d'établir une connexion à l'application externe. Sinon, si la connexion aboutit, la fonction **DiscoveryMain** doit renvoyer un **OSHVResult** vide.

Prise en charge de la synchronisation différentielle

Pour que l'adaptateur d'émission prenne en charge la synchronisation différentielle, la fonction **DiscoveryMain** doit renvoyer un objet implémentant l'interface **DataPushResults**, qui contient les mappages entre les ID que le script Jython reçoit du langage XML et les ID qu'il crée sur la machine distante. Ces derniers ID sont de type **ExternalId**.

La commande **ExternalIdUtil.restoreExternal** qui reçoit l'ID du CI dans la base CMDB comme paramètre, restaure l'ID externe à partir de l'ID du CI dans le CMDB. Cette commande peut être utilisée, par exemple, si lors d'une synchronisation différentielle l'une des extrémités d'un lien reçu ne se trouve pas dans le bloc (la synchronisation a déjà été effectuée).

Si la méthode **DiscoveryMain** du script Jython servant de base à l'adaptateur d'émission renvoie une instance **ObjectStateHolderVector** vide, l'adaptateur ne prendra pas en charge la synchronisation différentielle. Cela signifie que même en cas d'exécution d'un travail de synchronisation différentielle, on assiste en réalité à une synchronisation complète. Par conséquent, aucune donnée ne peut être mise à jour ni supprimée sur le système distant, car toutes les données sont ajoutées à la base CMDB lors de chaque synchronisation.

Important : Si vous implémentez la synchronisation différentielle sur un adaptateur existant créé dans la version 9.00 ou 9.01, vous devez utiliser le fichier push-adapter.zip de la version 9.02 ou ultérieure pour recréer le package de votre adaptateur. Pour plus d'informations, voir " Création d'un package d'adaptateur " , page 193.

Cette tâche permet à l'adaptateur d'émission d'effectuer une synchronisation différentielle.

Le script Jython renvoie l'objet **DataPushResults** qui contient deux mappes Java : une pour les mappages d'ID d'objet (les clés et valeurs sont des objets de type ExternalCiId) et une pour les ID de lien (les clés et valeurs sont des objets de type ExternalRelationId).

- Ajoutez les instructions **from** suivantes à votre script Jython :

```
from com.hp.ucmdb.federationspi.data.query.types import ExternalIdFactory

from com.hp.ucmdb.adapters.push import DataPushResults
```

```
from com.hp.ucmdb.adapters.push import DataPushResultsFactory

from com.mercury.topaz.cmdb.server.fcmdb.spi.data.query.types import ExternalI
dUtil
```

- Utilisez la classe d'usine **DataPushResultsFactory** pour extraire l'objet **DataPushResults** de la fonction **DiscoveryMain**.

```
# Create the UpdateResult object

updateResult = DataPushResultsFactory.createDataPushResults(objectMappings, li
nkMappings);
```

- Utilisez les commandes suivantes pour créer des mappes Java pour l'objet **DataPushResults** :

```
# Prepare the maps to store the mappings if IDs

objectMappings = HashMap()

linkMappings = HashMap()
```

- Utilisez la classe **ExternalIdFactory** pour créer les ID ExternalId suivants :
  - ExternalId pour les objets ou liens ayant pour origine une base CMDB (par exemple, tous les CI d'une opération d'ajout proviennent de la base CMDB) :

```
externaCIlId = ExternalIdFactory.createExternalCmdbCiId(ciType, ciIDAsStrin
g)

externalRelationId = ExternalIdFactory.createExternalCmdbRelationId(linkTyp
e, end1ExternalCIId,
end2ExternalCIId, linkIDAsString)
```

  - ExternalId pour les objets ou liens n'ayant pas pour origine une base CMDB (généralement, chaque opération de mise à jour et de suppression contient des objets de ce type) :

```
myIDField = TypesFactory.createProperty("systemID", "1")

myExternalId = ExternalIdFactory.createExternalCiId(type, myIDField)
```

> **Remarque :** Si le script Jython a mis à jour des informations existantes et que l'ID de l'objet (ou du lien) change, vous devez renvoyer un mappage entre l'ID externe précédent et le nouveau.

- Utilisez la méthode **restoreCmdbCiIDString** ou **restoreCmdbRelationIDString** de la classe **ExternalIdFactory** pour extraire la chaîne d'ID UCMDB d'un ID externe d'un objet ou lien ayant son origine dans la base UCMDB.

- Utilisez les méthodes **restoreExternalCiId** et **restoreExternalRelationId** de la classe

ExternalIdUtil pour restaurer l'objet **ExternalId** à partir de la valeur d'attribut mamId du code XML de l'opération de mise à jour ou de suppression.

> **Remarque :** Les objets **ExternalId** constituent en fait une matrice de propriétés. Cela signifie que vous pouvez utiliser un objet **ExternalId** afin de stocker les informations nécessaires pour identifier les données sur le système distant.

Requêtes SQL de l'adaptateur d'émission de données XML générique

Dans le package de l'adaptateur, le fichier **sql_queries** inclus dans **adapterCode > PushAdapter > sqlTablesCreation,** contient les requêtes nécessaires à la création de tables dans un nouveau schéma sous Oracle, afin de tester l'adaptateur. Les tables correspondent au fichier adapterCode\<ID adaptateur>\mappings\mappings.xml.

> **Remarque :** Le fichier **sql_queries** n'est pas nécessaire pour l'adaptateur. Il s'agit juste d'un exemple.

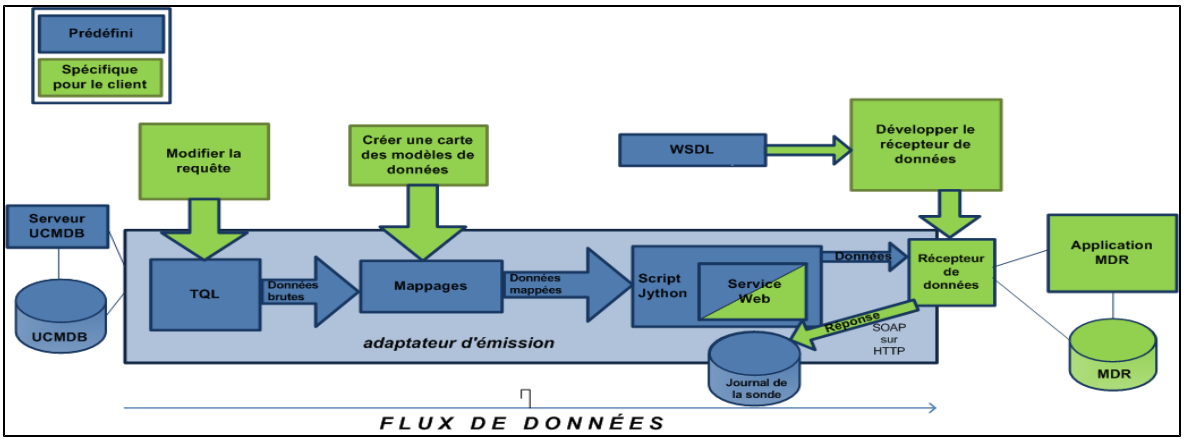
Adaptateur d'émission de données de service Web générique

L'adaptateur d'émission de données de service Web générique fournit une émission de messages SOAP initiée par UCMDB, contenant des données de requête destinées à un récepteur de données de service Web. Les résultats mappés sont envoyés au récepteur de données dans des messages SOAP standard, via le protocole HTTP POST. Le récepteur de données doit comprendre les messages SOAP produits par l'adaptateur d'émission de données. Pour faciliter le développement du récepteur de données approprié, un fichier WSDL est fourni avec cet adaptateur d'émission de données.

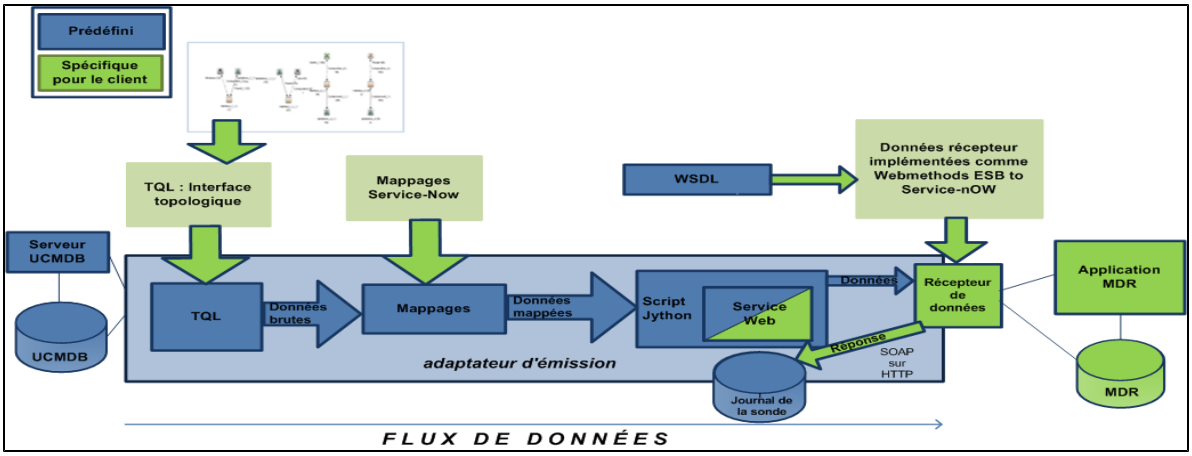
Le traitement personnalisé du fichier XML de réponse des messages SOAP est possible dans le script Jython.

Pour comprendre le format des données mappées entrantes, le développeur du récepteur de données doit communiquer avec le développeur du fichier de mappage. Aucun fichier **.xsd** n'est actuellement fourni avec cette version de l'adaptateur d'émission de données de service Web. Les données doivent donc être traitées de sorte qu'elles reflètent les données entrantes qui combinent le code TQL d'origine et les mappages appliqués.

Les fonctions de l'adaptateur d'émission de données de service Web permettant d'émettre des données vers le client sont répertoriées dans l'illustration ci-après. Les éléments en vert sont personnalisés ou fournis par le client dans le but d'implémenter l'adaptateur pour une cible d'émission de données spécifique. Les éléments en bleu sont des composants prêts à l'emploi.

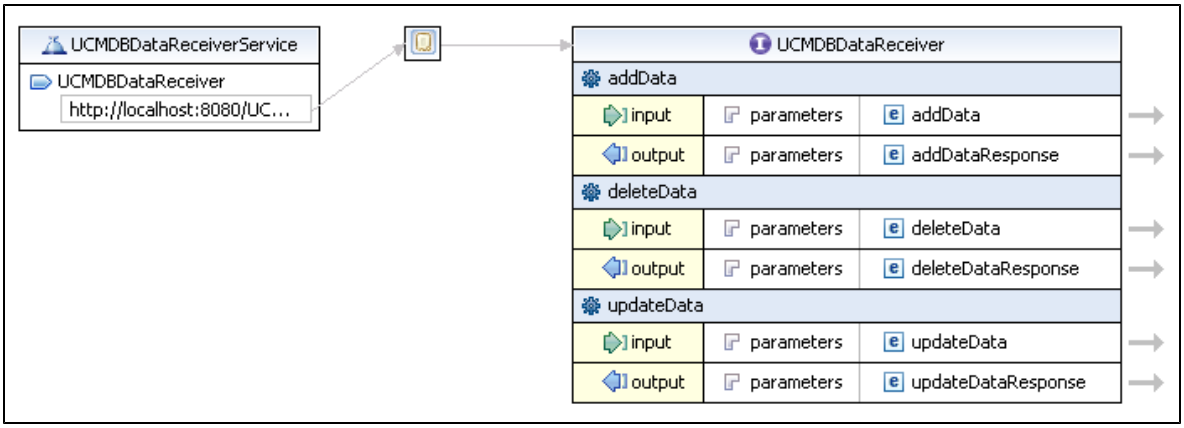


Voici un exemple d'implémentation de l'adaptateur d'émission de données de service Web générique vers un adaptateur d'émission de données propre au référentiel de données géré (MDR), via un ESB (Enterprise Service Bus) :



WSDL

Un fichier WSDL est fourni au développeur du client pour créer un récepteur de données capable de communiquer avec l'adaptateur d'émission de données UCMDB via un service Web. Le fichier **UCMDBDataReceiver.wsdl** décrit les messages SOAP permettant de communiquer des données depuis UCMDB vers le récepteur de données. Voici le schéma de conception du code WSDL :



Le récepteur de données (qui est en pratique un serveur ou un point de terminaison de service dans la terminologie SOAP) doit implémenter les trois méthodes **addData**, **deleteData** et **updateData** correspondant aux jeux de données émis par UCMDB. Les en-têtes HTTP contiennent le mot-clé **SoapAction** qui indique le type de données à envoyer. Le récepteur de données est chargé d'implémenter la logique métier et le traitement des données.

L'URL par défaut du fichier WSDL est la suivante :

- http://localhost:8080/UCMDBDataReceiver/services/UCMDBDataReceiver?wsdl

L'URL implémentée par le récepteur de données pourrait se présenter comme suit :

- http://testWSPAserver:4444/MyCo.IT.SvcMgt.ws.us:provider/UCMDBDataReceiver?wsdl

L'URL du service Web est identique à l'URL de WSDL, excepté qu'elle ne se termine pas par « ?wsdl ».

La source du fichier WSDL est la suivante :

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions targetNamespace="http://ucmdb.hp.com" xmlns:apachesoap="ht
tp://xml.apache.org/xml-soap" xmlns:impl="http://ucmdb.hp.com" xmlns:intf="h
ttp://ucmdb.hp.com" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdl
soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/20
01/XMLSchema">

<!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48
PDT)-->

    <wsdl:types>

        <schema elementFormDefault="qualified" targetNamespace="http://ucmdb
        .hp.com" xmlns="http://www.w3.org/2001/XMLSchema">

            <element name="addData">

                <complexType>

                    <sequence>

                    <element name="xmlAdded" type="xsd:string"/>

                    </sequence>

                </complexType>

            </element>

            <element name="addDataResponse">

                <complexType/>

            </element>

            <element name="deleteData">
```

```
                <complexType>
                    <sequence>
                    <element name="xmlDeleted" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="deleteDataResponse">
                <complexType/>
            </element>
            <element name="updateData">
                <complexType>
                    <sequence>
                    <element name="xmlUpdate" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="updateDataResponse">
                <complexType/>
            </element>
        </schema>
    </wsdl:types>

    <wsdl:message name="addDataRequest">
        <wsdl:part element="impl:addData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="deleteDataResponse">
        <wsdl:part element="impl:deleteDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="updateDataResponse">
```

```
        <wsdl:part element="impl:updateDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="deleteDataRequest">
        <wsdl:part element="impl:deleteData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="addDataResponse">
        <wsdl:part element="impl:addDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="updateDataRequest">
        <wsdl:part element="impl:updateData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:portType name="UCMDBDataReceiver">
        <wsdl:operation name="addData">
            <wsdlsoap:operation soapAction="addDataRequest"/>
            <wsdl:input message="impl:addDataRequest" name="addDataRequest">
            </wsdl:input>
            <wsdl:output message="impl:addDataResponse" name="addDataRespons
            e">
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="deleteData">
            <wsdlsoap:operation soapAction="deleteDataRequest"/>
            <wsdl:input message="impl:deleteDataRequest" name="deleteDataReq
            uest">
            </wsdl:input>
            <wsdl:output message="impl:deleteDataResponse" name="deleteDataR
            esponse">
            </wsdl:output>
```

```
        </wsdl:operation>
        <wsdl:operation name="updateData">
            <wsdlsoap:operation soapAction="updateDataRequest"/>
            <wsdl:input message="impl:updateDataRequest" name="updateDataReq
            uest">
            </wsdl:input>
            <wsdl:output message="impl:updateDataResponse" name="updateDataR
            esponse">
            </wsdl:output>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="UCMDBDataReceiverSoapBinding" type="impl:UCMDBDataRe
    ceiver">
        <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap
        .org/soap/http" />
        <wsdl:operation name="addData">
            <wsdl:input name="addDataRequest">
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="addDataResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="deleteData">
            <wsdl:input name="deleteDataRequest">
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="deleteDataResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="updateData">
            <wsdl:input name="updateDataRequest">
```

```
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="updateDataResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="UCMDBDataReceiverService">
        <wsdl:port binding="impl:UCMDBDataReceiverSoapBinding" name="UCMDBDa
        taReceiver">
            <wsdlsoap:address location="http://localhost:8080/UCMDBDataRecei
            ver/services/
            UCMDBDataReceiver"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Traitement des réponses

Le récepteur de données doit renvoyer une chaîne dans les structures **addDataResponse**, **deleteDataResponse** ou **updateDataResponse**. L'adaptateur transmet les données de réponse non traitées au journal **probeMgr-adaptersDebug.log** de la sonde. Le récepteur peut renvoyer n'importe quelle donnée de type chaîne (string), et les réponses sont encapsulées dans un fichier XML compatible SOAP. Dans le script Jython, vous pouvez utiliser la classe **SOAPMessage** et les classes Java associées pour analyser les messages de réponse. Voici un exemple de message de réponse provenant du récepteur de données :

```
<2012-03-16 15:47:38,080> [INFO ] [Thread-110] - XMLtoWebService.py:addData
received response:
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<intf:addDataResponse xmlns:intf="http://ucmdb.hp.com">
    <xml>&lt;result&gt;&lt;status&gt;error&lt;/status&gt;
    &lt;message&gt;Error publishing config item changes&lt;/message&gt;
    &lt;/result&gt;</xml>
</intf:addDataResponse>
</soapenv:Body>
```

Il s'agit du message d'erreur **<Error publishing config item changes>** (erreur lors de la publication des modifications de CI), mais son contenu dépend de la réponse prévue par le

récepteur de données. La réponse est un message d'erreur parce que l'auteur a simplement l'intention d'indiquer qu'il s'agit d'un message d'erreur et que l'adaptateur d'émission de données attend une réponse spécifiant un succès ou un échec. Le contenu peut comprendre les ID de rapprochement de tous les CI ajoutés ou des messages d'erreur concernant des CI spécifiques. La personnalisation du GWSPA peut inclure l'analyse du message de réponse et les actions à exécuter telles que le renvoi de certains CI ou une autre journalisation.

Test de WSDL

Le plug-in SOAPUI d'Eclipse permet de tester les couches de service Web pendant le développement. Vous pouvez utiliser SOAPUI pour personnaliser un service Web. SOAPUI offre un environnement de développement intégré (IDE) pour tester la création, l'envoi et la réception de messages SOAP. Dans la perspective de SOAPUI, le code WSDL des pages 209-213 a généré l'exemple de message suivant :

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ucm="http://ucmdb.hp.com">

    <soapenv:Header/>

    <soapenv:Body>

        <ucm:addData>

            <ucm:xmlAdded>?</ucm:xmlAdded>

        </ucm:addData>

    </soapenv:Body>

</soapenv:Envelope>
```

Le point d'interrogation (**?**) dans l'élément **xmlAdded** ci-dessus représente l'emplacement des données indiqué par l'intégration de l'adaptateur d'émission de données de service Web.

Examen des résultats

Lorsque l'adaptateur d'émission de données fonctionne normalement en mode hors débogage, les données ne sont jamais écrites dans un fichier tant que le résultat final n'est pas écrit (les résultats TQL intermédiaires et des données mappées ne sont normalement pas visibles dans un fichier journal). Cependant, les résultats peuvent être écrits dans le fichier de débogage de la sonde par les instructions **logger.debug** hors commentaires (suppression du caractère #) dans la section DiscoveryMain, comme indiqué ci-après :

```
# get referenced data - unused in this adapter implementa
                                ggerCIData('referencedAdd
                                TriggerCIData('referenced
                                TriggerCIData('referenced

                                ments to see the data
                                esult")

                                t)
  send to ESB web service
  ogger.info(SCRIPT_NAME+":sending addData Result")
  endDataToReceiver("add", URL, addResult)
  #logger.debug(addResult)

empty = isEmpty(updateResult, "updateResult")
if not empty:
```

Supprimer # pour activer la journalisation du débogage des données

Vérifiez que l'instruction du journal commence sur la même colonne que celle des autres lignes précédentes et suivantes. Comme Jython respecte les retraits, le script échouera si le retrait de toutes les lignes est incorrect.

Le fichier journal de débogage **probeMgr-adaptersDebug.log** de la sonde affiche le contenu de la sortie :

```
<2011-12-07 14:02:23,019> [INFO ] [Thread-273] - XMLtoWebService.py started

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - ESB Push parameters:

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - Wshost=harpy.trtc.com

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - WShostport=5555

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - WSuri=ws/DtITServiceManagem
ent.esla.v1.ws.provider:UMDBDataReceiver

<2011-12-07 14:02:23,019> [INFO ] [Thread-273] - URL is http://harpy.trtc.co
m:5555/ws/DtITServiceManagement.esla.v1.ws.
provider:UMDBDataReceiver

<2011-12-07 14:02:23,035> [DEBUG] [Thread-273] - Connected to http://harpy.t
rtc.com:5555/ws/DtITServiceManagement.esla.v1.ws.
provider:UMDBDataReceiver

<2011-12-07 14:02:23,035> [ERROR] [Thread-273] - sending results

<2011-12-07 14:02:23,035> [DEBUG] [Thread-273] - <?xml version="1.0" encodin
g="UTF-8"?>

<root>

    <data>

        <objects>

            <Object mode="" name="u_imp_ip_switch" operation="add" mamId="9e
            8c2f6bdfe4b7d0864c79e70833902c">
```

```
            <field name="Correlation ID" key="true" datatype="char" len
            gth="">9e8c2f6bdfe4b7d0864c79e70833902c</field>

            <field name="name" key="false" datatype="char" length="">nm
            a_09sw</field>

            <field name="location" key="false" datatype="char" length
            ="" />

            <field name="u_chassis_vendor_type" key="false" datatype="c
            har" length="">ciscoCat2960-24TT</field>

            <field name="serial_number" key="false" datatype="char" len
            gth="" />

            <field name="ram" key="false" datatype="char" length="" />

            <field name="os_version" key="false" datatype="char" length
            ="" />

        </Object>
```

Modification du script Jython XMLtoWebService.py

Le script Jython utilisé par l'adaptateur d'émission de données de service Web est très similaire à l'adaptateur d'émission de données XML. Il utilise **UCMDBDataReceiver.jar** inclus avec l'adaptateur. Il implémente la méthode **SendDataToReceiver()**. Cette méthode utilise trois paramètres :

1. l'action (ajout, mise à jour ou suppression),
2. l'URL du récepteur de données,
3. les données.

Par exemple, le bloc d'ajout est le suivant : **SendDataToReceiver("add", URL, addResult)**

Toutes les couches de service Web et SOAP sont encapsulées. L'URL est l'adresse du point de terminaison de service du récepteur de données UCMDB. Il s'agit de la même URL que celle utilisée pour obtenir le code WSDL via le suffixe « ?wsdl ».

La source du script Jython est indiquée ci-après. Les lignes de l'encapsuleur d'intégration du service Web sont mises en évidence en vert.

```
####################################
# script: XMLtoWebService.py
####################################
```

```
# This jython script accepts TQL data results (adds, updates, and deletes) f
rom the Integration adapter.

# and sends it to a web service. The web service is called UCMDBDataReceiver.

# A web service client of this name must be addressable at the URL provided
by the parameters.

# The SendDataToReceiver.jar exposes the SendDataToReceiver function, as well
as the service locator.

# examples of the service locator are in the testconnection section.

# regular expressions

import re

# logging

import logger

# web service interface

from com.hp.ucmdb import SendDataToReceiver

from com.hp.ucmdb.SendDataToReceiver import locateService

from com.hp.ucmdb.SendDataToReceiver import SendData

##############################################

########      VARIABLES             ##########

##############################################

SCRIPT_NAME = "XMLtoWebService.py"

logger.info(SCRIPT_NAME+" started")

def cleanUp(str):


    # replace mode=""

    str = re.sub("mode=\"\w+\"\s+", "", str)


    # replace mamId with id

    str = re.sub("\smamId=\"", " id=\"", str)


    # replace empty attributes

    str = re.sub("[\n|\s|\r]*<field name=\"\w+\" datatype=\"\w+\" />", "", s
    tr)
```

```
    # replace targetRelationshipClass with name
    str = re.sub("\stargetRelationshipClass=\"", " name=\"", str)

    # replace Object with object with name
    str = re.sub("<Object mode=\"", "<object mode=\"", str)
    str = re.sub("<Object operation=\"", "<object operation=\"", str)
    str = re.sub("<Object name=\"", "<object name=\"", str)
    str = re.sub("</Object>", "</object>", str)

    # replace field to attribute
    str = re.sub("<field name=\"", "<attribute name=\"", str)
    str = re.sub("</field>", "</attribute>", str)

    #logger.debug("String = %s" % str)
    #logger.debug("cleaned up")

    return str
def isEmpty(xml, type = ""):
    objectsEmpty = 0
    linksEmpty = 0

    m = re.findall("<objects />", xml)
    if m:
        #logger.warn("\t[%s] No objects found" % type)
        objectsEmpty = 1

    m = re.findall("<links />", xml)
    if m:
        #logger.warn("\t[%s] No links found" % type)
        linksEmpty = 1
```

```
    if objectsEmpty and linksEmpty:
        return 1
    return 0
##############################################
########      MAIN                 ##########
##############################################
def DiscoveryMain(Framework):
    #fix this for web service export
    errMsg = "UCMDBDataReceiver Service not found."
    testConnection = Framework.getTriggerCIData("testConnection")
    # Get Web Service Push variables
    WShostName = Framework.getTriggerCIData("Host Name")
    WShostport = Framework.getTriggerCIData("Protocol Port")
    WSuri = Framework.getTriggerCIData("URI")

    logger.info(SCRIPT_NAME+":ESB Push parameters:")
    logger.info("Host Name="+WShostName)
    logger.info("Protocol Port="+WShostport)
    logger.info("URI="+WSuri)
    URL = "http://"+WShostName+":"+WShostport+"/"+WSuri
    logger.info("URL="+URL)
    if testConnection == 'true':
        # locate the service
        test_receiver = SendDataToReceiver()
        locator = test_receiver.locateService(URL)
        #locator = locateService(URL)
        if(locator):
            logger.info(SCRIPT_NAME+":Test connection was successful")
            return
        else:
            raise Exception, errMsg
```

```
        return
# do same thing here if not just a test connection -
receiver = SendDataToReceiver()
locator = receiver.locateService(URL)
if(locator):
    logger.info(SCRIPT_NAME+":Connected to "+URL)
else:
    logger.error(SCRIPT_NAME+":no locator")
    raise Exception, errMsg
    return

# get add/update/delete result objects from the Framework
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
logger.debug(deleteResult)

# get referenced data - unused in this adapter implementation
#addRefResult = Framework.getTriggerCIData('referencedAddResult')
#updateRefResult = Framework.getTriggerCIData('referencedUpdateResult')
#deleteRefResult = Framework.getTriggerCIData('referencedDeleteResult')
# uncomment out the logger statements to see the data
empty = isEmpty(addResult, "addResult")
if not empty:
    addResult = cleanUp(addResult)
    # send to ESB web service
    logger.info(SCRIPT_NAME+":sending addData Result")
    rcvr = SendDataToReceiver()
    resp = rcvr.SendData("add", URL, addResult)
    logger.info(SCRIPT_NAME+":addData received response:"+resp)
    #logger.debug(addResult)
```

```
    empty = isEmpty(updateResult, "updateResult")
    if not empty:
        updateResult = cleanUp(updateResult)
        # send to ESB web service
        #logger.debug(updateResult)
        logger.info(SCRIPT_NAME+":sending updateData Result")
        rcvr = SendDataToReceiver()
        resp = rcvr.SendData("update", URL, updateResult)
        logger.info(SCRIPT_NAME+":received response:"+resp)

    empty = isEmpty(deleteResult, "deleteResult")
    if not empty:
        deleteResult = cleanUp(deleteResult)
        # send to ESB web service
        #logger.debug(deleteResult)
        logger.info(SCRIPT_NAME+":sending deleteData Result")
        rcvr = SendDataToReceiver()
        resp = rcvr.SendData("delete", URL, deleteResult)
        logger.info(SCRIPT_NAME+":received response:"+resp)
    logger.info(SCRIPT_NAME+" ended")
```

Personnalisation du traitement des messages de réponse

Le récepteur de données doit renvoyer une chaîne contenant la réponse ou le statut souhaité. L'adaptateur d'émission de données de service Web transmet par défaut la réponse au journal de niveau information de la sonde. Le message de réponse est au format XML SOAP contenant les chaînes de réponse renvoyées. Des données telles que des messages d'erreur ou de succès individuels ou regroupés peuvent être renvoyées par le récepteur. Si un traitement supplémentaire est souhaité, la réponse peut être traitée par le script Jython de l'adaptateur. Aucune programmation Java n'est nécessaire.

Par exemple, un message de réponse de retour envoyé à l'aide du code suivant :

```
// stub example for building your own UCMDBDataData Receiver
public class UCMDBDataReceiver {
```

```
public String addData (String xmlAdd){
System.out.println(xmlAdd); // do something with the data
// send back a response message based on what you did
String tr = new String("a test response from addData!");
return tr;
}
```

apparaît comme suit :

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <addDataResponse xmlns="http://ucmdb.hp.com">
        <addDataReturn>a test response from addData!</addDataReturn>
    </addDataResponse>
</soapenv:Body>
```

Modification du récepteur de données

Un client Java peut implémenter les classes contenues dans le fichier **UCMDBDataReceiver.jar** et appeler le service Web de la même manière que Jython. En outre, les méthodes non encapsulées peuvent également être appelées. Il existe un Javadoc pour les classes **UCMDBDataReceiver.jar**. Le code source ci-après indique comment utiliser ces méthodes essentielles pour encapsuler les données dans un message SOAP et les envoyer à un récepteur via HTTP.

Ce processus consiste à créer un objet **UCMDBDataReceiverServiceLocator**, puis à affecter **UCMDBDataReceiverEndPointAddress** à l'URL du récepteur de données.

Pour envoyer les données, la méthode **getUCMDBDataReceiver** du localisateur est appelée pour créer un objet **UCMDBDataReceiver**. L'objet **UCMDBDataReceiver** implémente les méthodes pour envoyer les données à ajouter/modifier/supprimer. Il existe trois blocs de code identiques pour traiter chaque type de demande.

Voici le code source pour la classe **SendDataToReceiver**. Les objets et méthodes mis en évidence constituent les éléments essentiels à utiliser.

```
/**
* Test SendData for the UCMDB Data Receiver for the UCMDB Web Service Push A
dapter
*/
package com.hp.ucmdb;
import com.hp.ucmdb.SendDataToReceiver;
```

```
/**
* TestSendData can be used to verify the SOAP classes are working.
* TestSendData creates a SendDataToReceiver class and invokes its SendData m
ethod.
* a response String is returned.
* The test URL is typically appended with "?wsdl" to get the WSDL of the ser
vice.
*/
public class TestSendData {
    /**
    * @param args - test SOAP message.
    *  optional arguments [0] a test string [1] a service endpoint URL of a
    Data Receiver.
    * the default URL is sent the incoming argument as a test message.
    * the default URL is "http://localhost:8080/UCMDBDataReceiver/services/U
    CMDBDataReceiver".
    * If any errors are encountered, TestClient will attempt to throw except
    ions.
    */
    public static void main(String[] args) {
        // use test message if supplied, otherwise supply a default test str
        ing
        String teststring = new String("Test SOAP message from UCMDBDataRece
        iver TestSendData.");
        if(args.length > 0) {
            teststring = args[0];
        }
        // use test URL if supplied, otherwise supply the default URL
        String URL = new String("");
        if(args.length > 1) {
            URL = args[1];
        }
        // return response
        String response = new String("");
```

```
        // perform the tests
        try{
            if(URL.equals("")) {
                UCMDBDataReceiverServiceLocator locator = new UCMDBDataRece
                iverServiceLocator();
                UCMDBDataReceiver receiver = locator.getUCMDBDataReceiver()
                ;
                URL = locator.getUCMDBDataReceiverAddress();
                System.out.println("TestClient: tested URL="+locator.getUCM
                DBDataReceiverAddress());
                System.out.println("TestClient: receiver="+receiver.toStrin
                g());
            }
            SendDataToReceiver sdtr = new SendDataToReceiver();
            // this sends a test push and gets a response message
            response = sdtr.SendData("add", URL, args[0]);
            System.out.println("Response received was:"+response);
        } catch(Exception e){
            System.out.println("TestClient: Remote Error:");
            e.printStackTrace();
        }
    }
}
```

Le code source est également inclus dans le fichier **UCMDBDataReceiver.jar** pour les autres classes :

- TestClient.java
- UCMDBDataReceiver.java
- UCMDBDataReceiverProxy.java
- UCMDBDataReceiverService.java
- UCMDBDataReceiverServiceLocator.java
- UCMDBDataReceiverSoapBindingStub.java

La source a été générée dans l'IDE d'Eclipse, puis modifiée. Exercez-vous avec prudence lors de la modification du code UCMDB, car une partie de celui-ci est généré automatiquement afin de mettre en correspondance la spécification SOAP et le récepteur de données UCMDB.

Javadoc

Un **javadoc** entièrement commenté est fourni avec l'adaptateur d'émission de données de service Web. Il est inclus dans le dossier de documentation **javadoc**. Commencez par le fichier **index.html**. La page de présentation fournit un accès à la documentation de toutes les classes et méthodes du SDK.

Toutes les classes

- **SendDataToReceiver :** API pour l'encapsuleur de service Web
- **TestClient :** test effectué sur le client pour vérifier la connectivité d'un point de terminaison de service
- **UCMDBDataReceiver :** encapsuleur de service Web

Le reste est automatiquement généré par le créateur du service Web :

- UCMDBDataReceiverProxy
- UCMDBDataReceiverService
- UCMDBDataReceiverServiceLocator
- UCMDBDataReceiverSoapBindingStub

Présentation

Le mode d'utilisation de base du SDK, notamment les exemples de code source, est expliqué dans la documentation du package. Ce **javadoc** est destiné à l'adaptateur d'émission de données de service Web UCMDB. L'API peut être appelée à partir de Jython ou Java.

Le SDK fournit deux exemples de source, **TestClient** et **SendDataToReceiver**. **TestClient** inclut un test très limité de la réponse du client local. **SendDataToReceiver** est la classe principale utilisée pour envoyer des données à un service Web.

Utilisez d'abord ce SDK (principalement le code WSDL inclus) pour implémenter un récepteur de données UCMDB qui communique avec ce service Web. Utilisez ensuite le SDK pour créer un adaptateur d'émission de données dans UCMDB, qui émet les données des résultats TQL UCMDB vers le récepteur de données. Le mode d'utilisation de base de cette API est décrit ci-après, avec des implémentations Jython et Java.

Implémentation de SendDataToReceiver()

SendDataToReceiver() encapsule toutes les fonctions à l'aide d'une seule méthode :

- Jython : `SendDataToReceiver("add",yourURL,"Hello!")`
- Java : `SendDataToReceiver("add",yourURL,"Hello!");`

Vous pouvez aussi créer un objet **SendDataToReceiver** (pour manipuler, par exemple, d'autres paramètres), puis appeler séparément la méthode **SendData** :

- Jython :

```
rcvr = SendDataToReceiver()
responseMsg = rcvr.SendData("add", yourURL, "Hello!")
```

- Java :

```
SendDataToReceiver rcvr = new SendDataToReceiver();
String responseMsg = rcvr.SendData("add", yourURL, "Hello!");
```

Si vous devez exécuter une étape à la fois, procédez comme suit :

1. Créez un objet x **UCMDBDataReceiverServiceLocator()**, puis définissez l'adresse de point de terminaison de l'objet ultérieurement :
   - **Jython :**

     ```
     x = UCMDBDataReceiverServiceLocator()
     x.setUCMDBDataReceiverEndPointAddress(URL)
     ```

   - **Java :**

     ```
     UCMDBDataReceiverServiceLocator x = new UCMDBDataReceiverServiceLocator();
     x. setUCMDBDataReceiverEndPointAddress(URL);
     ```

2. Créez ensuite un objet UCMDBDataReceiver :
   - **Jython :** `y = x.getUCMDBDataReceiver()`
   - **Java :** `UCMDBDataReceiver y = x.getUCMDBDataReceiver();`
3. Envoyez les données via le service Web SOAP :
   - **Jython :**
     - `y.addData(yourData)`
     - ou `y.updateData(yourData)`
     - ou `y.deleteData(yourData)`
   - **Java :**
     - `y.addData(yourData);`

- ou `y.updateData(yourData);`
- ou `y.deleteData(yourData);`

4. Il peut être nécessaire de tester la connectivité. Si elle est correcte, réutilisez le même objet localisateur pour renvoyer l'objet **UCDMBDataReceiver** à utiliser pour le transfert des données.

Les classes ne contiennent aucun destructeur et ne prennent pas en charge la gestion de la mémoire.

Fichier de mappage de référence

Utilisation de mappages

Un mappage doit être créé pour chaque attribut cible dans la sortie XML transformée. Les mappages spécifient l'emplacement des données et la façon de les obtenir. Si les données se trouvent dans un autre attribut correspondant dans UCDMB, un mappage direct est utilisé.

D'autres mappages complexes peuvent se révéler nécessaires pour extraire soit des données de plusieurs attributs, soit des attributs de l'enfant du CI UCMDB, soit des attributs du CI parent. Le schéma de mappage ci-dessous illustre tous les mappages possibles.

Le fichier de mappage est un fichier XML qui définit les types de CI/relation de UCMDB à mapper sur les types de CI/relation du magasin de données cible. Le format est décrit en détail ci-dessous. Le fichier de mappage contrôle les types de relation et de CI à émettre, ainsi que le nombre exact d'attributs à émettre.

Il existe une entrée de mappage pour chaque attribut à émettre dans le référentiel de données géré (MDR) cible. Chaque entrée de mappage peut comprendre un ou plusieurs attributs dans les données d'émission brutes UCMDB. Les entrées de mappage permettent d'effectuer un contrôle détaillé complet de la structure finale et de nommer les données à émettre dans le MDR cible.

Mappages directs

Les mappages transforment un modèle de données en un autre (dans ce cas, UCMDB en MDR cible d'émission). Les transformations peuvent être simples dans le cas d'une relation 1:1 entre l'attribut UCMDB et la cible qui ne diffèrent que par leur nom et, parfois, par leur type.

La plupart des mappages d'attributs sont directs. Par exemple, le nom de serveur « ServeurX » peut être représenté dans UCMDB comme un CI de type **unix** avec un nom d'attribut **nom_serveur_principal** du type **string** d'une longueur égale à 50. Le modèle de données du MDR cible peut spécifier la même entité logique avec un CI de type **linux** et un nom d'attribut **hostname** du type **char[]** d'une longueur égale à 250. Les mappages directs peuvent exécuter toutes ces tâches de conversion des types précités.

Voici un exemple de mappage direct :

```
 <target_attribute name="dns_domain" datatype="char">

<map type="direct" source_attribute="domain_name" />

</target_attribute>
```

Ce mappage direct mappe l'attribut UCMDB **dns_domain** sur l'attribut **domain_name** dans le modèle de données cible.

Utilisez le type de données **char** quel que soit le type des données réelles, sauf s'il est indispendable d'utiliser celui-ci.

Mappages complexes

Les mappages plus complexes permettent d'autres transformations :

- Mappage de valeurs d'attributs de plusieurs CI sur un seul CI cible.
- Mappage des attributs de CI enfants (ceux disposant d'une relation **container_f** ou contenu) sur le CI parent dans le magasin de données cible. Par exemple, définition d'une valeur appelée **Nombre de CPU** sur un CI d'hôte cible. La définition de la valeur **Mémoire totale** pourrait être un autre exemple sur un CI hôte cible (en additionnant les valeurs de la taille de mémoire de tous les CI de mémoire d'un CI hôte dans UCMDB).
- Mappage des attributs de CI parents (ceux disposant d'une relation **container_f** ou contenu) sur le CI du magasin de données cible. Par exemple, définition d'une valeur appelée **Serveur conteneur** sur un attribut cible appelé CI de **logiciel installlé**, en extrayant la valeur de l'hôte contenant du CI de logiciel dans UCMDB.

L'exemple ci-dessous illustre un mappage complexe utilisant deux attributs source séparés par une virgule pour créer l'attribut cible **os** :

```
<target_attribute name="os" datatype="char">
    <map type="compoundstring">
        <source_attribute name="discovered_os_name" />
        <constant value="," />
        <source_attribute name="host_osinstalltype" />
    </map>
</target_attribute>
```

Inversion de la direction des liens

Il est possible que UCMDB contienne des données dont la structure est différente d'une source à l'autre. Par exemple, la relation entre un CI d'adresse IP et un CI d'interface peut être un **parent**, comme cela pourrait être le cas lors de l'intégration de HP Network Node Manager. Ce cas peut se produire également avec un lien d'imbrication (**containment**) couramment créé par Universal Discovery. En outre, la direction de ces liens est contraire l'un par rapport à l'autre.

Il est actuellement impossible d'inverser la direction des liens dans le fichier de mappage. L'inversion des variables **_end1** et **_end2** permute l'ordre des données dans le fichier XML transformé, ou le lien est manquant dans les données source.

Une des solutions possibles à ce problème consiste à définir comme suit une règle d'enrichissement :

1. La partie TQL de l'enrichissement est un sous-ensemble d'un code TQL utilisé par l'adaptateur d'émission de données. Ce code TQL sélectionne, en particulier, tous les liens dont la direction est contraire à celle souhaitée dans le fichier xml transformé.
2. La partie enrichissement définit un nouveau lien de direction correcte et du type souhaité.
3. L'enrichissement est activé et crée les liens corrects.
4. Le travail d'intégration TQL se réfère à présent au lien enrichi au lieu du lien d'origine.
5. Les mappages de <lien> dans l'adaptateur d'émission de données se réfère ensuite au lien enrichi et produisent un jeu de liens cohérent en ce qui concerne le type et la direction.

Schéma du fichier de mappage

Nom et chemin d'accès de l'élément	Description	Attributs
integration	Définit le contenu de mappage du fichier. Il doit s'agir du bloc situé le plus à l'extérieur dans le fichier en dehors de la ligne de début et des commentaires.	
info (integration)	Définit des informations sur les référentiels de données en cours d'intégration.	

Nom et chemin d'accès de l'élément	Description	Attributs
source (integration > info)	Définit des informations sur le référentiel de données source.	1. **Nom :** type **Description :** Nom du référentiel de données cible. **Obligatoire ? :** Required **Type :** String 2. **Nom :** versions **Description :** Version(s) des référentiels de données source. **Obligatoire ? :** Required **Type :** String 3. **Nom :** vendor **Description :** Fournisseur du référentiel de données source. **Obligatoire ? :** Required **Type :** String
target (integration > info)	Définit des informations sur le référentiel de données cible.	1. **Nom :** type **Description :** Nom du référentiel de données cible. **Obligatoire ? :** Required **Type :** String 2. **Nom :** versions **Description :** Version(s) du référentiel de données cible. **Obligatoire ? :** Required **Type :** String 3. **Nom :** vendor **Description :** Fournisseur du référentiel de données source. **Obligatoire ? :** Required **Type :** String
targetcis (integration)	Élément conteneur pour tous les mappages de types de CI.	

Nom et chemin d'accès de l'élément	Description	Attributs
source_ci_type_tree (integration > targetcis)	Définit un type de CI source et tous les types de CI qui héritent de ce dernier.	1. **Nom :** name **Description :** Nom du type de CI source. **Obligatoire ? :** Required **Type :** String 2. **Nom :** mode **Description :** Type de mise à jour requis pour le type de CI actuel. **Obligatoire ? :** Required **Type :** L'une des chaînes suivantes : a. **insert :** À n'utiliser que si le CI n'existe pas déjà. b. **update :** À n'utiliser que si l'existence du CI est connue. c. **update_else_insert :** Si le CI existe, mettez-le à jour; sinon, créez-en un. d. **ignore :** Ne faites rien avec ce type de CI.
source_ci_type (integration > targetcis)	Définit un type de CI source sans les types de CI qui héritent de ce dernier.	1. **Nom :** name **Description :** Nom du type de CI source. **Obligatoire ? :** Required **Type :** String 2. **Nom :** mode **Description :** Type de mise à jour requis pour le type de CI actuel. **Obligatoire ? :** Required **Type :** L'une des chaînes suivantes : a. **insert :** À n'utiliser que si le CI n'existe pas déjà. b. **update :** À n'utiliser que si l'existence du CI est connue. c. **update_else_insert :** Si le CI existe, mettez-le à jour; sinon, créez-en un. d. **ignore :** Ne faites rien avec ce type de CI.

Nom et chemin d'accès de l'élément	Description	Attributs
target_ci_type (integration > targetcis > source_ci_type -OU- integration > targetcis > source_ci_type_ tree)	Définit un type de CI cible.	1. **Nom :** name **Description :** Nom du type de CI cible. **Obligatoire ? :** Required **Type :** String 2. **Nom :** schema **Description :** Nom du schéma qui sera utilisé pour stocker ce type de CI sur la cible. **Obligatoire ? :**Not Required **Type :** String 3. **Nom :** namespace **Description :** Indique l'espace de noms de ce type de CI sur la cible. **Obligatoire ? :**Not Required **Type :** String
targetprimarykey (integration > targetcis > source_ci_type) -OU- (integration > targetcis > source_ci_type_tree -OU- (integration > targetrelations > link) -OU- (integration > targetrelations > source_link_type_tree)	Identifie les attributs de la clé principale du type de CI cible.	

Nom et chemin d'accès de l'élément	Description	Attributs
pkey (integration > targetcis> source_ci_type > targetprimarykey -OU- integration > targetcis > source_ci_type_tree > targetprimarykey -OU- (integration > targetrelations > link > targetprimarykey) -OU- integration > targetrelations > source_link_type_tree > targetprimarykey)	Identifie un attribut de la clé principale. Obligatoire uniquement si le mode est **update** ou **insert_else_update**.	

Nom et chemin d'accès de l'élément	Description	Attributs
target_attribute (integration > targetcis > source_ci_type -OU- integration > targetcis > source_ci_type_tree -OU- integration > targetrelations > link -OU- integration > targetrelations > source_link_type_tree)	Définit l'attribut du type de CI cible.	1. **Nom :** name **Description :** Nom de l'attribut du type de CI cible. **Obligatoire ? :** Required **Type :** String 2. **Nom :** datatype **Description :** Type de données de l'attribut du type de CI cible. **Obligatoire ? :** Required **Type :** String 3. **Nom :** length **Description :** Pour les types de données chaîne/caractère, taille entière de l'attribut cible. **Obligatoire ? :**Not Required **Type.** Integer 4. **Nom.** option **Description.** Fonction de conversion à appliquer à la valeur. **Obligatoire.** False **Type.** L'une des chaînes suivantes : a. **uppercase** – Convertir en majuscules. b. **lowercase** – Convertir en minuscules. Si cet attribut est vide, aucune fonction de conversion ne sera appliquée.

Nom et chemin d'accès de l'élément	Description	Attributs
map (integration > targetcis > source_ci_type > target_attribute -OU- integration > targetcis > source_ci_type_tree > target_attribute) -OU- (integration > targetrelations > link > target_attribute -OU- integration > targetrelations > source_link_type_tree > target_attribute)	Indique comment obtenir la valeur d'attribut du type de CI source.	1. **Nom :** type **Description.** Type de mappage entre les valeurs source et cible. **Obligatoire.** Required **Type.** L'une des chaînes suivantes : a. **direct** – Indique une correspondance de un à un entre la valeur de l'attribut source et celle de l'attribut cible. b. **compoundstring –** Les sous-éléments sont joints en une chaîne unique et la valeur de l'attribut cible est définie. c. **childattr** – Les sous-éléments sont les attributs d'un ou plusieurs types de CI enfant. Les types de CI enfant sont définis comme ceux qui présentent une relation **composition** ou **containment**. d. **constant** – Chaîne statique 2. **Nom.** value **Description.** Chaîne constante pour type=**constant** **Obligatoire.** Required uniquement lorsque type=**constant** **Type.** String 3. **Nom.** attr **Description.** Nom d'attribut source pour type=**direct** **Obligatoire.** Required uniquement lorsque type=**direct** **Type.** String

Nom et chemin d'accès de l'élément	Description	Attributs
aggregation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetcis > source_ci_type_tree > target_attribute > map -OU- (integration > targetrelations > link > target_attribute > map -OU- integration > targetrelations > source_link_type_tree > target_attribute > map) Valide uniquement lorsque le type de la mappe est **childattr**	Indique comment les valeurs d'attribut des CI enfants du CI source sont combinés en une valeur unique à mapper sur l'attribut de CI cible. Facultatif.	**Nom :** type **Description.** Type de la fonction d'agrégation **Obligatoire ? :** Required **Type.** L'une des chaînes suivantes : • **csv** – Concatène toutes les valeurs incluses dans une liste séparée par des virgules (numérique ou chaîne/caractère). • **count** – Renvoie un décompte numérique de toutes les valeurs incluses. • **sum –** Renvoie la somme de toutes les valeurs numériques incluses. • **average** – Renvoie une moyenne numérique de toutes les valeurs incluses. • **min** – Renvoie la valeur incluse numérique/caractère la plus basse. • **max** – Renvoie la valeur incluse numérique/caractère la plus élevée.

Nom et chemin d'accès de l'élément	Description	Attributs
source_child_ci_type (integration > targetcis> source_ci_type > target_attribute > map -OU- integration > targetcis > source_ci_type_tree > target_attribute > map -OU- (integration > targetrelations > link > target_attribute > map -OU- integration > targetrelations > source_link_type_tree > target_attribute > map) Valide uniquement lorsque le type de la mappe est **childattr**.	Indique la provenance de l'attribut enfant parmi les CI connectés.	1. **Nom.** name **Description.** Type de CI enfant **Obligatoire.** Required **Type.** String 2. **Nom.** source_attribute **Description.**Attribut du CI enfant qui est mappé. **Obligatoire.** Obligatoire uniquement si le type d'agrégation **childAttr** (qui se trouve sur le même chemin) est différent de =count. **Type.** String

Nom et chemin d'accès de l'élément	Description	Attributs
validation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetcis > source_ci_type_tree > target_attribute > map -OU- (integration > targetrelations > link > target_attribute > map -OU- integration > targetrelations > source_link_type_tree > target_attribute > map) Valide uniquement lorsque le type de la mappe est **childattr**	Permet le filtrage d'exclusion des CI enfants du CI source sur la base des valeurs d'attribut. Utilisé avec le sous-élément aggregation pour obtenir la granularité exacte des attributs enfants mappés sur la valeur d'attribut du type de CI cible. Facultatif.	1. **Nom.** minlength **Description.** Exclut les chaînes plus courtes que la valeur indiquée. **Obligatoire ? :** Not required **Type.** Integer 2. **Nom.** maxlength **Description.** Exclut les chaînes plus longues que la valeur indiquée. **Obligatoire ? :** Not required **Type.** Integer 3. **Nom.** minvalue **Description.** Exclut les nombres plus petits que la valeur spécifiée. **Obligatoire ? :** Not required **Type.** Numeric 4. **Nom.** maxvalue **Description.** Exclut les nombres plus grands que la valeur spécifiée. **Obligatoire ? :** Not required **Type.** Numeric
targetrelations (integration)	Élément conteneur pour tous les mappages de relations. Facultatif.	

Nom et chemin d'accès de l'élément	Description	Attributs
source_link_type_tree (integration > targetrelations)	Mappe un type de relation source sur une relation cible sans les types qui héritent de ce dernier. Obligatoire uniquement si **targetrelation** est présent.	1. **Nom :** name **Description.** Nom de la relation source. **Obligatoire ? :** Required **Type.** String 2. **Nom :** target_link_type **Description.** Nom de la relation cible. **Obligatoire ? :** Required **Type.** String 3. **Nom :** nameSpace **Description :** Espace de noms du lien qui sera créé sur la cible. **Obligatoire ? :** Not required **Type :** String 4. **Nom :** mode **Description :** Type de mise à jour requis pour le lien actuel. **Obligatoire ? :** Required **Type :** L'une des chaînes suivantes : ▪ **insert** – À n'utiliser que si le CI n'existe pas déjà. ▪ **update** – À n'utiliser que si l'existence du CI est connue. ▪ **update_else_insert** – Si le CI existe, mettez-le à jour ; sinon, créez-en un. ▪ **ignore** – Ne faites rien avec ce type de CI. 5. **Nom :** source_ci_type_end1 **Description :** Type de CI End1 de la relation source. **Obligatoire ? :** Required **Type :** String 6. **Nom :** source_ci_type_end2 **Description :** Type de CI End2 de la relation source. **Obligatoire ? :** Required **Type :** String

Nom et chemin d'accès de l'élément	Description	Attributs
link (integration > targetrelations)	Mappe une relation source sur une relation cible. Obligatoire uniquement si **targetrelation** est présent.	1. **Nom :** source_link_type **Description :** Nom de la relation source. **Obligatoire ? :** Required **Type :** String 2. **Nom :** target_link_type **Description :** Nom de la relation cible. **Obligatoire ? :** Required **Type :** String 3. **Nom :** nameSpace **Description :** Espace de noms du lien qui sera créé sur la cible. **Obligatoire ? :** Not required **Type :** String 4. **Nom :** mode **Description :** Type de mise à jour requis pour le lien actuel. **Obligatoire ? :** Required **Type :** L'une des chaînes suivantes : ▪ **insert** – À n'utiliser que si le CI n'existe pas déjà. ▪ **update** – À n'utiliser que si l'existence du CI est connue. ▪ **update_else_insert** – Si le CI existe, mettez-le à jour ; sinon, créez-en un. ▪ **ignore** – Ne faites rien avec ce type de CI. 5. **Nom :** source_ci_type_end1 **Description :** Type de CI End1 de la relation source. **Obligatoire ? :** Required **Type :** String 6. **Nom :** source_ci_type_end2 **Description :** Type de CI End2 de la relation source. **Obligatoire ? :** Required **Type :** String

Nom et chemin d'accès de l'élément	Description	Attributs
target_ci_type_end1 (integration > targetrelations > link -OU- integration > targetrelations > source_link_type_tree)	Type de CI End1 de la relation cible.	1. **Nom :** name **Description :**Nom du type de CI End1 de la relation cible. **Obligatoire ? :** Required **Type :** String 2. **Nom :** superclass **Description :** Nom de la superclasse du type de CI End1. **Obligatoire ? :** Not required **Type :** String
target_ci_type_end2 (integration > targetrelations > link -OU- integration > targetrelations > source_link_type_tree)	Type de CI End2 de la relation cible.	1. **Nom :** name **Description :**Nom du type de CI End2 de la relation cible. **Obligatoire ? :** Required **Type :** String 2. **Nom :** superclass **Description :** Nom de la superclasse du type de CI End2. **Obligatoire ? :** Not required **Type :** String

Schéma des résultats de mappage

Nom et chemin d'accès de l'élément	Description	Attributs
root	Racine du document de résultat.	
data (root)	Racine des données elles-mêmes.	
objects (root > data)	Élément racine des objets à mettre à jour.	

Nom et chemin d'accès de l'élément	Description	Attributs
Object (root > data > objects)	Décrit l'opération de mise à jour d'un seul objet et de tous ses attributs.	1. **Nom :** name **Description :** Nom du type de CI **Obligatoire ? :** Required **Type :** String 2. **Nom :** mode **Description :** Type de mise à jour requis pour le type de CI actuel. **Obligatoire ? :** Required **Type :** L'une des chaînes suivantes : a. **insert** – À n'utiliser que si le CI n'existe pas déjà. b. **update** – À n'utiliser que si l'existence du CI est connue. c. **update_else_insert** – Si le CI existe, mettez-le à jour ; sinon, créez-en un. d. **ignore** – Ne faites rien avec ce type de CI. 3. **Nom :** operation **Description :**Opération à effectuer avec ce CI. **Obligatoire ? :** Required **Type :** L'une des chaînes suivantes : a. **add** – Le CI doit être ajouté. b. **update** – Le CI doit être mis à jour. c. **delete** – Le CI doit être supprimé. Si aucune valeur n'est définie, la valeur par défaut de **add** est utilisée. 4. **Nom :** mamId **Description :** ID de l'objet dans la base CMDB source. **Obligatoire ? :** Required **Type :** String

Nom et chemin d'accès de l'élément	Description	Attributs
field (root > data > objects> Object -OU- root > data > links > link)	Décrit la valeur d'un champ unique pour un objet. Le texte du champ est la nouvelle valeur du champ, et si le champ contient un lien, la valeur est l'ID d'une des extrémités. Chaque ID d'extrémité apparaît en tant qu'objet (sous <objects>).	1. **Nom :** name **Description :** Nom du champ. **Obligatoire ? :** Required **Type :** String 2. **Nom :** key **Description :** Indique si ce champ est une clé pour l'objet. **Obligatoire ? :** Required **Type :** Boolean 3. **Nom :** datatype **Description :** Type du champ. **Obligatoire ? :** Required **Type :** String 4. **Nom :** length **Description :** Pour les types de données chaîne/caractère, il s'agit de la taille entière de l'attribut cible. **Obligatoire ? :**Not Required **Type :** Integer

Nom et chemin d'accès de l'élément	Description	Attributs
links (root > data)	Élément racine des liens à mettre à jour.	1. **Nom :** targetRelationshipClass **Description :** Nom de la relation (lien) dans le système cible. **Obligatoire ? :** Required **Type :** String 2. **Nom :** targetParent **Description :** Type de la première extrémité du lien (parent). **Obligatoire ? :** Required **Type :** String 3. **Nom :** targetChild **Description :** Type de la deuxième extrémité du lien (enfant). **Obligatoire ? :** Required **Type :** String 4. **Nom :** mode **Description :** Type de mise à jour requis pour le type de CI actuel. **Obligatoire ? :** Required **Type :** L'une des chaînes suivantes : a. **insert** – À n'utiliser que si le CI n'existe pas déjà. b. **update** – À n'utiliser que si l'existence du CI est connue. c. **update_else_insert** – Si le CI existe, mettez-le à jour ; sinon, créez-en un. d. **ignore** – Ne faites rien avec ce type de CI. 5. **Nom :** operation **Description :**Opération à effectuer avec ce CI. **Obligatoire ? :** Required **Type :** L'une des chaînes suivantes : a. add – Le CI doit être ajouté. b. update – Le CI doit être mis à jour. c. delete – Le CI doit être supprimé. Si aucune valeur n'est définie, la valeur par défaut de **add** est utilisée. 6. **Nom :** mamId **Description :** ID de l'objet dans la base CMDB source.

Nom et chemin d'accès de l'élément	Description	Attributs
		Obligatoire ? : Required **Type :** String

Personnalisation

Cette section décrit certaines procédures de base des types courants de personnalisation pour les adaptateurs d'émission de données.

Ajout d'un attribut

1. Vérifiez que l'attribut est inclus dans le résultat TQL.
2. Ajoutez le mappage de l'attribut au fichier de mappage dans la section de mappage de CI appropriée.
3. Vérifiez que le récepteur de données est prêt à recevoir l'attribut supplémentaire dans les données.

Suppression d'un attribut

Pour supprimer un attribut, supprimez-le du fichier de mappage. Vous devez également le supprimer du TQL s'il n'est plus utilisé dans le résultat ou en tant que nœud conditionnel.

Ajout d'un type de CI

1. Ajoutez le type de CI au TQL.
2. Vérifiez que le type de CI et ses données d'attribut apparaissent dans le résultat TQL (utilisez les fonctions de calcul et d'aperçu).
3. Ajoutez le mappage du type de CI dans le fichier de mappage. Copiez les mappages d'un autre type de CI pour créer rapidement un nouveau type de CI.
4. Modifiez le nom du fichier XML copié et les mappages d'attributs afin qu'ils correspondent au nouveau type de CI et à ses attributs. Voir " Fichier de mappage de référence " , page 227 pour la liste des types de mappage disponibles.

Suppression d'un type de CI

1. Supprimez le type de CI du TQL.
2. Supprimez la section de mappage de ce type de CI dans le fichier de mappage.

Ajout de liens

1. Vérifiez que les deux CI d'extrémité figurent dans les données.
2. Vérifiez que le lien à ajouter est effectivement un lien valide (vérifiez-le dans le Gestionnaire des types de CI).
3. Ajoutez les éléments du lien dans la section des relations du fichier xml de mappage.

Suppression de liens

1. Supprimez la section du lien à supprimer dans le fichier de mappage.
2. Si possible, supprimez le lien du TQL (sauf si cette action a une incidence sur l'efficacité ou la fonction du TQL).

Chapitre 7 : Développement d'adaptateurs d'émission (push) génériques avancés

Contenu de ce chapitre :

Présentation 247

Fichier de mappage 247

Voyageur Groovy 250

Écriture de scripts Groovy 253

Implémentation de l'interface PushConnector 254

Création d'un package d'adaptateur 255

Schéma du fichier de mappage 256

Présentation

Un adaptateur d'émission (push) avancé est une structure de données qui représente le résultat d'une requête TQL. Chaque adaptateur construit sur l'adaptateur d'émission avancé traite la structure de données et l'émet vers la cible voulue.

La structure de données est appelée **ResultTreeNode (RTN)**. La création du RTN repose sur le fichier de mappage de l'adaptateur et les résultats de la requête TQL. Les requêtes utilisées pour l'adaptateur d'émission avancé doivent être orientées racine : la requête doit contenir un nœud de requête avec le nom d'élément **root**, ou alors un ou plusieurs éléments de relation commençant par le préfixe **root**. Ce CI ou cette relation constituent l'élément racine de la requête. Pour plus d'informations, voir Data Push dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Le développement d'adaptateurs d'émission avancés comporte deux étapes fondamentales :

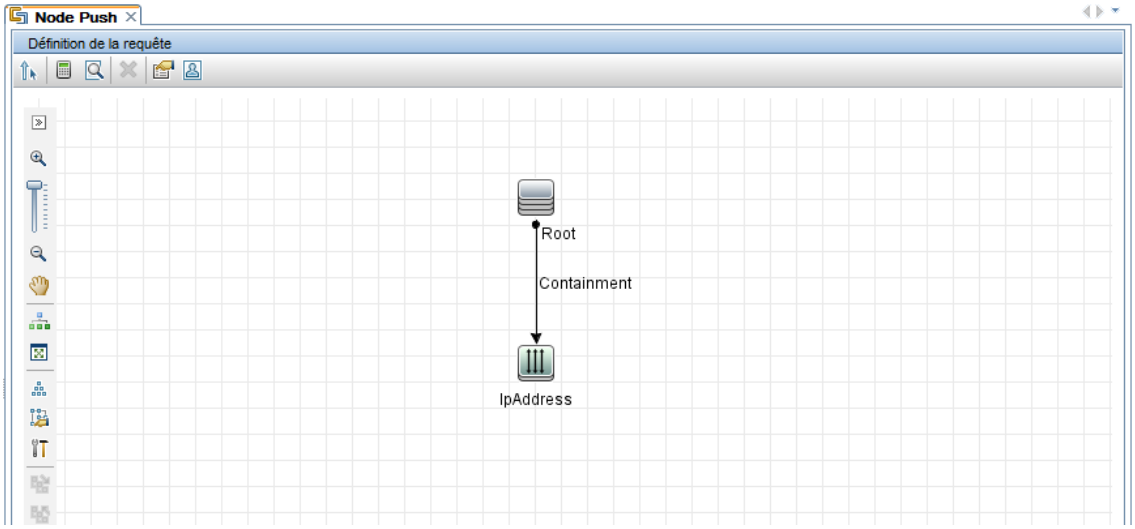
1. Implémentation de l'interface PushConnector– cette interface reçoit les données à ajouter, à mettre à jour et à supprimer comme une liste de RTN. Elle procède ensuite à l'émission dans la cible.

2. Création du fichier de mappage– le fichier de mappage détermine la création de la structure RTN en mappant les CI et les attributs du résultat du TQL.

Fichier de mappage

L'exemple suivant explique comment créer le fichier de mappage.

Nous allons ici simuler l'émission d'un nœud et d'une adresse IP. Il s'agit de créer une requête TQL appelée : **Node Push**, comme suit :

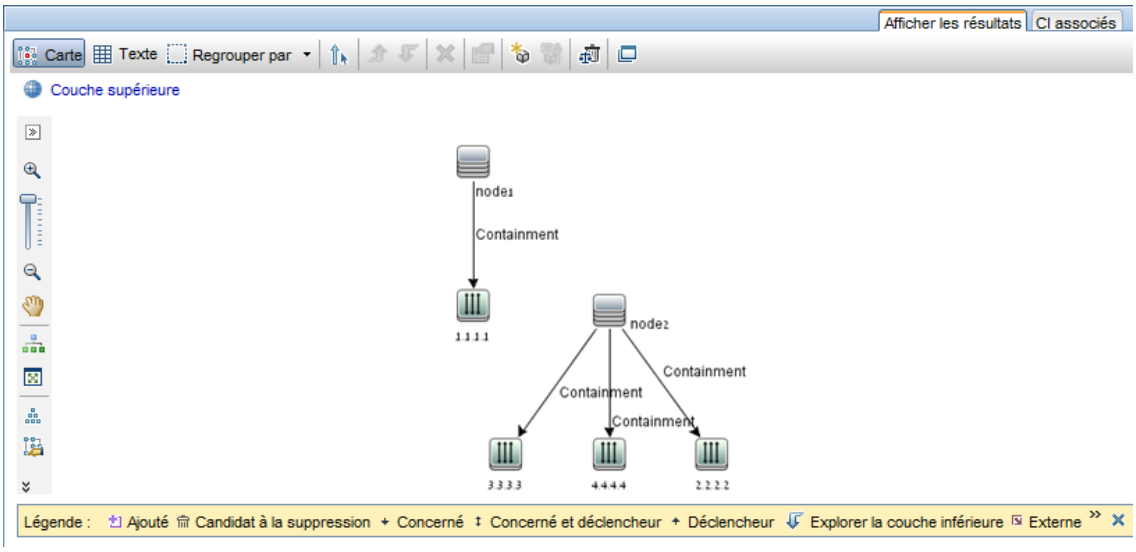


Dans le fichier de mappage, nous créons deux types de CI cible : **Computer** et **IP**. Computer dispose d'une variable et de deux attributs, tandis que IP possède un seul attribut.

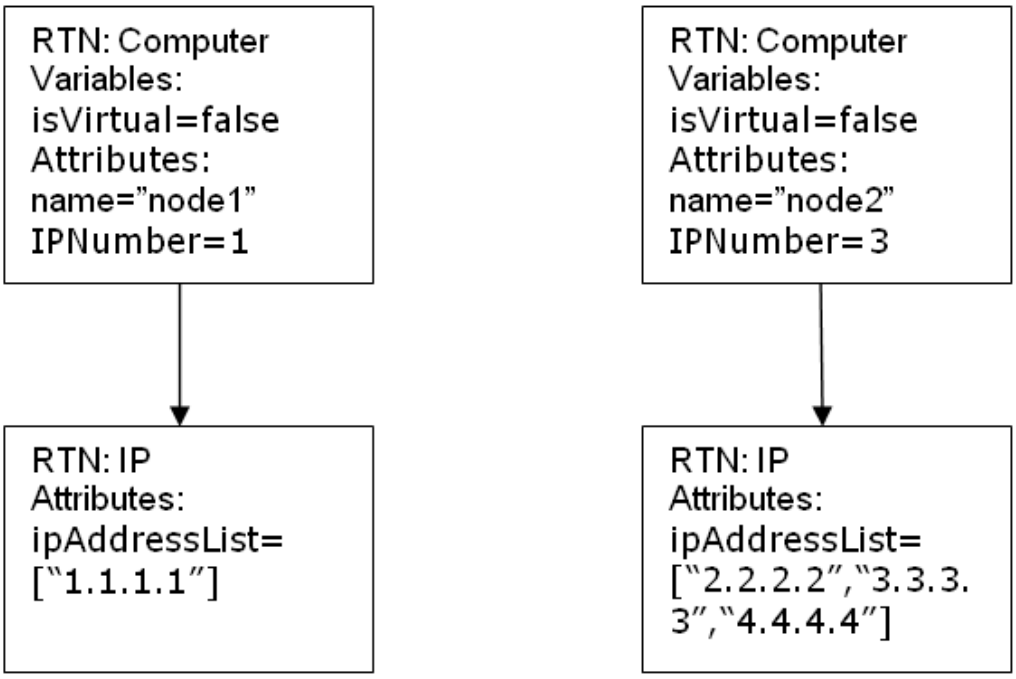
Le fichier XML de mappage est le suivant :

```
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://schemas.hp.com/ucmdb/1/pushAdap
ter">
        <info>
                <source name="UCMDB" versions="10.0" vendor="HP" />
                <target name="PushProduct" versions="9.3" vendor="HP" />
        </info>
        <import>
                <!—imports the Groovy script file. -->
                <scriptFile path="mappings.scripts.PushFunctions" />
        </import>
<targetcis>
<source_instance_type query-name="Node Push" root-element-name="Root">
 <target_ci_type name="Computer" is-
    valid="(Root['root_iscandidatefordeletion']==null) ? true :
    !Root['root_iscandidatefordeletion']">
  <variable name="isVirtual" datatype="BOOLEAN"
    value="PushFunctions.isVirtual(Root['root_class']) " />

  <target_mapping name="name" datatype="String" value="Root[`name’] " />
  <target_mapping name="ipNumber" datatype="INTEGER"
    value="Root.IpAddress.size()" />
  <target_mapping name="description" datatype="STRING" value=" PushFunctions
    .getDescription(isVirtual)" />
  <target_ci_type name="IP">
    <target_mapping name="ipAddressList" datatype="STRING_LIST"
    value="Root.IpAddress*.getAt(`name’) " />
</target_ci_type>
</target_ci_type>
</targetcis>
</integration>
```

Les résultats de la requête apparaissent comme suit :



Voici la liste RTN créée selon le fichier de mappage :



Chaque instance racine est mappée séparément à l'aide du fichier de mappage. Ainsi, dans cet exemple, PushConnector reçoit une liste de deux racines RTN.

Remarque : L'adaptateur d'émission (push) précédent offrait la possibilité de créer un mappage général pour un type de CI. Le nouveau mappage via un adaptateur d'émission s'opère par requête TQL. Lors de l'exécution d'un travail d'émission sollicitant une requête **x**, l'adaptateur recherche le fichier de mappage approprié (celui qui contient l'attribut : query-name=x).

Vous pouvez calculer les valeurs dans le fichier de mappage à l'aide du langage de script groovy. Pour plus d'informations, voir " Voyageur Groovy " , ci-dessous.

Voyageur Groovy

Les résultats d'une requête TQL sont accessibles de plusieurs façons :

- **Root[attr]** renvoie l'attribut **attr** de l'élément racine.
- **Root.Query_Element_Name** renvoie une liste d'instances de CI nommées dans le Query_ Element_Name du TQL et liées au CI racine en cours.
- **Root.Query_Element_Name[2][attr]** renvoie l'attribut **attr** du troisième Query_Element_Name lié au CI racine en cours.
- **Root.Query_Element_Name*.getAt(attr)** renvoie une liste d'attributs **attr** des instances de CI nommées Query_Element_Name dans le TQL et liées au CI racine en cours.

D'autres attributs sont également accessibles au voyageur groovy :

- **cmdb_id** – renvoie l'ID UCMDB du CI ou de la relation sous forme de chaîne.
- **external_cmdb_id** – renvoie l'ID externe du CI ou de la relation sous forme de chaîne.
- **Element_type** – renvoie le type d'élément du CI ou de la relation sous forme de chaîne.

La balise d'importation :

```
<import>
<scriptFile path="mappings.scripts.PushFunctions"/>
</import>
```

Il s'agit ici de déclarer une importation de tous les scripts groovy dans le fichier de mappage. Dans cet exemple, **PushFunctions** est un fichier script groovy qui contient certaines fonctions statiques accessibles lors du mappage (c-à-d;, value=" PushFunctions.foo()")

source_instance_type

Le mappage est effectué par TQL ; la valeur de query-name correspond au TQL lié du mappage en cours. '*' signifie que le fichier de mappage est associé à toutes les requêtes TQL commençant par le préfixe : **Node Push**.

```
<source_instance_type query-name="Node Push*" root-element-name="Root">
```

La balise source_instance_type désigne l'élément racine faisant l'objet du mappage.

La valeur de root-element-name doit être identique au nom de la racine dans le TQL.

target_ci_type

Cette balise sert à la création du RTN.

L'attribut name représente target_ci_type name : name=Computer

L'attribut **is-valid** est une valeur booléenne qui est calculée lors du mappage et détermine la validité du target_ci en cours. Les valeurs de target_ci_type non valides ne sont pas ajoutées au RTN. Dans cet exemple, il s'agit d'empêcher la création d'une instance target_ci_type pour laquelle l'attribut **root_iscandidatefordeletion** dans la base UCMDB a la valeur true.

L'attribut target_ci_type peut être accompagné de variables qui sont calculées lors du mappage :

```
<variable name="vSerialNo" datatype="STRING" value="Root['serial_number']"/>
```

La variable **vSerialNo** récupère la valeur de **serial_number** de la racine en cours.

L'attribut du RTN est créé par la balise **target_mapping**. Le résultat de l'exécution du script groovy dans le champ **value** est affecté à l'attribut RTN.

```
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
```

SerialNo affecte la valeur de la variable **vSerialNo**.

Il est possible de définir un target_ci_type comme enfant d'un autre target_ci_type comme suit :

```
<target_ci_type name="Portfolio">
<variable name="vSerialNo" datatype="STRING" value="Root['global_id']"/>
<target_mapping name="CMDBId" datatype="STRING" value="globalId"/>
<target_ci_type name=Asset">
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
</target_ci_type>
</target_ci_type>
```

Le RTN **Portfolio** aura un RTN enfant nommé **Asset**.

for-each-source-ci

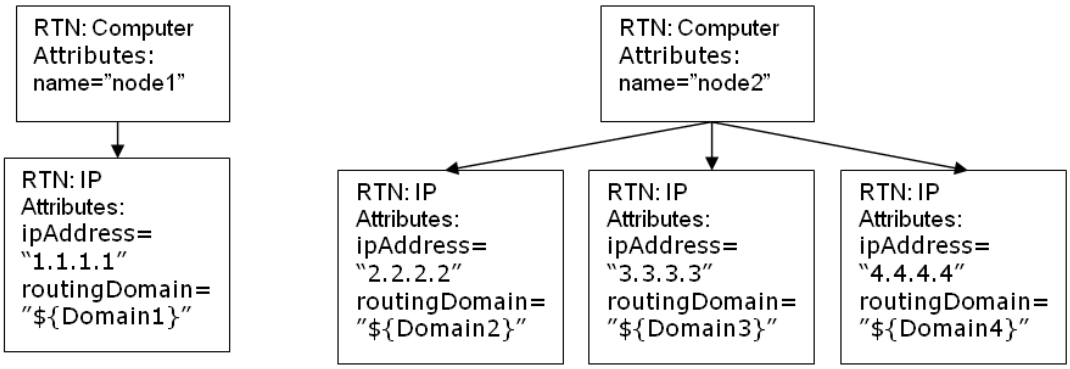
Cette balise répertorie les CI spécifiques de l'instance racine. Elle contient les champs suivants :

- source-cis=" " – liste des CI pour lesquels un CI cible est créé. C'est le voyageur groovy qui définit cette liste dans le champ **Root.IpAddress**.
- count-index=" " – variable contenant l'index du CI dans l'itération en cours de la boucle.
- var-name=" " – nom du CI dans l'itération en cours de la boucle.

Modifions notre exemple de fichier de mappage :

```
<target_ci_type name="Computer">
  <target_mapping name="name" datatype="String" value="Root[`name'] " />
   <for-each-source-ci count-index="i" var-name="currIP" source-cis="Root.IpAddress" >
     <target_ci_type name="IP">
         <target_mapping name="ipAddress" datatype="STRING"
     value="Root.IpAddress[i][`name']"/>
         <target_mapping name="routingDomain" datatype="STRING"
     value="currIP['routing_domain']"/>
  </target_ci_type>
 </for-each-source-ci>
 </target_ci_type>
```

La liste RTN issue de ce fichier de mappage présentera l'aspect suivant :



dynamic_mapping

Cette balise ajoute la capacité à créer un mappage de données à partir de la banque de données cible lors de la création de la structure RTN.

Exemple : Supposons que la cible est une base de données comportant une table nommée **Computer**, dotée des colonnes **id** et **name** en corrélation avec **Node.name** dans UCMDB. Les deux colonnes sont uniques. La base de données comporte également une table nommée **IP** qui elle-même contient une clé référencée vers l'attribut **parentID** dans la table Computer. La balise 'dynamic_mapping' peut créer une mappe stockant le nom et l'ID comme <name,id>. À partir de cette mappe, l'adaptateur peut établir une correspondance entre les ID est les ordinateurs et émettre la valeur appropriée à l'attribut parentID dans la table IP. Vous pouvez utiliser cette mappe pour affecter une valeur à l'attribut parentID lors de la création du RTN.

Le mappage est déterminé par la balise **map_property**. Le mappage dynamic_mapping est exécuté une seule fois par segment.

```
<dynamic_mapping name="IdByName " keys-unique="true">
```

L'attribut **name** représente le nom de la mappe. L'attribut **keys-unique** indique si les clés sont uniques (chaque clé est mappée sur une seule valeur ou sur un ensemble de valeurs).

Dans cet exemple, la mappe est nommée **IdByName** et dispose de clés uniques. Pour accéder au mappage dans le script, exécutez la commande suivante :

```
DynamicMapHolder.getMap('IdByName')
```

Elle renvoie une référence à la mappe.

La balise **map_property** crée la propriété sur laquelle repose le mappage.

Exemple :

```
<map_property property-name="SQLQuery" datatype="STRING"
property-value="SELECT name, id FROM Computer"/>
```

Dans cet exemple, la propriété est nommée **SQLQuery** et sa valeur est une instruction SQL qui crée la mappe. L'implémentation des méthodes **retrieveUniqueMapping** et **retrieveNonUniqueMapping** pour l'interface PushConnector détermineront le contenu réel de la mappe renvoyée.

Variables globales

Les variables globales suivantes sont accessibles au script groovy dans le fichier de mappage :

- **Topology** – Type : Topology. Instance de la topologie du segment en cours.
- **QueryDefinition** - Type : QueryDefinition. Instance de la définition de la requête du TQL en cours.
- **OutputCI** – Type : ResultTreeNode. Le RTN de l'élément racine dans le mappage d'arborescence en cours.
- **ClassModel** – Type : ClassModel. Instance du modèle de classe.
- **CustomerInformation** – Type : CustomerInformation. Informations relatives au client exécutant le travail.
- **Logger** – Type : DataAdapterLogger. L'enregistreur est disponible dans l'adaptateur pour écrire des journaux dans l'infrastructure de journalisation UCMDB.

Écriture de scripts Groovy

Cette section vise à créer le fichier **PushFunctions.groovy**. Celui-ci contiendra des fonctions statiques qui sont utilisées lors du mappage de l'instance racine.

```
package mappings.scripts

public class PushFunctions {

       public static boolean isVirtual(def nodeRole){
               return isListContainsOne(def list, "MY_VM", "MY_SIMULATOR");
```

```
        }


public static String getDescription(boolean isVirtual){
                if(isVirtual){
                        return "This is a VM";
                }
                else {
                        return "This is physical machine";
                }
        }


private static boolean isListContainsOne(def list, ...stringList){
                //returns true if the list contains one of the values.
        }
}
```

Implémentation de l'interface PushConnector

L'implémentation doit prendre en charge les étapes de base suivantes :

```
public class PushExampleAdapter implements PushAdapterConnector

{

public UpdateResult pushTreeNodes(PushConnectorInput input) throws DataAcces
sException{

// 1. build an UpdateResult instance - the UpdateResult is used to return ma
ppings between the sent ids to the actual ids that entered the data store.
// Also has an update status which allows to pass the status of data that was
actually pushed, detailed status reports on failed IDs, and actions actually
performed on successful ids.
// 2. handle the data:
// a. handle data to add. Can be retrieved by: input.getResultTreeNodes.getD
ataToAdd();
//  b. handle data to update.
//  c. handle data to delete.
// 3. Return the Update result.
        }

public void start(PushDataAdapterEnvironment env) throws DataAccessException{
        // this method is called when the integration point created,
or when the adapter is reloaded
```

```
       //(i.e after changing one of the mapping files
       // and pressing ‘save’).

}



public void testConnection(PushDataAdapterEnvironment env) throws DataAccess
Exception {
               // this method is called when pressing the 'test
connection' button in the
               //creation of the integration point.
               // For example if we push data to RDBMS this method
can create a connection
               //to the database and will run a dummy SQL statement.
               // If it fails it writes an error message to the log
and throws an exception.
       }




Map<Object, Object> retrieveUniqueMapping(MappingQuery mappingQuery){
//This method will create the map according to the given mappingQuery. It wi
ll be called in the
// mapping stage of the adapter execution, before the ‘UpdateResult pushTree
Nodes’ method.
// This method is called when the ‘keys-unique’ attribute of the ‘dynamic_ma
pping’ tag is true.
}
Map<Object, Set<Object>> retrieveNonUniqueMapping(MappingQuery mappingQuery){
 // This method is called when the ‘keys-unique’ attribute of the ‘dynamic_m
apping’ tag is false.
//  In this case a key can be mapped to several values.
}
}
```

Création d'un package d'adaptateur

Vérifiez que le package d'adaptateur contient les dossiers suivants :

- **adapterCode.** Sous ce dossier, créez un dossier **PushExampleAdapter** dans lequel sera placé le fichier .jar que nous avons créé à partir de PushExampleAdapter.java. Il contiendra également un dossier **mappings** qui pourra recevoir le fichier de mappage créé précédemment, **computerIPMapping.xml**. Il doit également contenir un autre dossier appelé **scripts** comportant le fichier **PushFunctions.groovy**.

- **discoveryConfigFiles.** Dossier destiné à recevoir les fichiers de configuration , tels que les codes d'erreur utilisés lors du signalement d'une erreur à l'aide de UpdateResult. Dans cet exemple, le dossier est vide.
- **discoveryPatterns.** Dossier destiné à recevoir le fichier **push_example_adapter.xml**.
- **tql** Dossier destiné à recevoir la requête TQL créée pour l'exemple. Ce dossier est facultatif, mais le TQL est créé automatiquement lors du déploiement du package.

Schéma du fichier de mappage

Nom et chemin d'accès de l'élément	Description	Attributs
Intégration	Définit le contenu de mappage du fichier. Il doit s'agir du bloc situé le plus à l'extérieur dans le fichier, mis à part la ligne et les commentaires.	
info (integration)	Définit les informations relatives aux référentiels en cours d'intégration.	
source (info)	Produit source	**Nom :** name **Description :** nom du produit **Obligatoire ? :** required **Type :** String
		Nom : vendor **Description :** fournisseur du produit **Obligatoire ? :** required **Type :** String
		Nom : versions **Description :** version du produit **Obligatoire ? :** required **Type :** decimal

Nom et chemin d'accès de l'élément	Description	Attributs
target (info)	Produit cible	**Nom :** name **Description :** nom du produit **Obligatoire ? :** required **Type :** String
		Nom : vendor **Description :** fournisseur du produit **Obligatoire ? :** required **Type :** String
		Nom : versions **Description :** version du produit **Obligatoire ? :** required **Type :** decimal
Import (integration)	Élément conteneur pour les fichiers de script importés.	
scriptFile (integration>import)	Définit le fichier de script groovy à importer.	**Nom :** path **Description :** chemin d'accès au fichier script **Obligatoire ? :** required **Type :** string
Targetcis (integration)	Élément conteneur pour les types de CI cible.	

Nom et chemin d'accès de l'élément	Description	Attributs
Source_instance_type (integration > targetcis)	Définit le type d'instance de CI source. Doit être l'élément racine tel qu'il est défini dans le TQL.	**Nom :** query-name. **Description :** Nom du TQL approprié **Obligatoire ? :** required **Type :** String
		Nom : name **Description :** Élément racine tel qu'il est défini dans le TQL. **Obligatoire ? :** required **Type :** String
dynamic_mapping (integration > targetcis > source_instance_type)	Définit une carte qui est créée une fois par segment.	**Nom :** name **Description :** nom de la carte **Obligatoire ? :** required **Type :** String
		Nom : keys-unique **Description :** Indique si les clés sont uniques. **Obligatoire ? :** required **Type :** boolean

Nom et chemin d'accès de l'élément	Description	Attributs
target_ci_type (integration > targetcis > source_instance_type) -OU- (integration > targetcis > source_instance_type> for-each-source-ci)	Définit le type de CI cible à ajouter au RTN.	**Nom :** name **Description :** nom du type de CI cible **Obligatoire ? :** required **Type :** String
		Nom : is-valid **Description :** vérifie la validité du CI cible actuel selon le script donné. **Obligatoire ? :** not required **Type :** String (script groovy)

Nom et chemin d'accès de l'élément	Description	Attributs
Variable (target_ci_type)	Définit une variable pour le type de CI cible.	**Nom :** name **Description :** nom de la variable **Obligatoire ? :** required **Type :** String
		Nom : datatype **Description :** type de données de la variable **Obligatoire ? :** required **Type :** type-enum (peut être l'un des suivants : INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Nom : value **Description :** Valeur à affecter à la variable **Obligatoire ? :** required **Type :** groovy script

Nom et chemin d'accès de l'élément	Description	Attributs
target_mapping (target_ci_type)	Définit un attribut pour le type de CI cible.	**Nom :** name **Description :** nom de l'attribut **Obligatoire ? :** required **Type :** String
		Nom : datatype **Description :** type de données de la variable **Obligatoire ? :** required **Type :** type-enum (peut être l'un des suivants : INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Nom : value **Description :** Valeur à affecter à la variable **Obligatoire ? :** required **Type :** groovy script
		Nom : ignore-on-null **Description :** Si la valeur de mappage cible est NULL et que l'attribut est TRUE, ignorer ce dernier. **Obligatoire ? :** not required **Type :** Boolean (script groovy)

Nom et chemin d'accès de l'élément	Description	Attributs
before-mapping (target_ci_type)	Script groovy exécuté avant le mappage du type de CI cible.	
after-mapping (target_ci_type)	Script groovy exécuté après le mappage du type de CI cible.	
for-each-source-ci (target_ci_type)	Définit une itération de CI spécifiques de l'instance racine.	**Nom :** count-index **Description :** index du CI itératif actuel **Obligatoire ? :** required **Type :** String
		Nom : var-name **Description :** variable renvoyant au CI itératif actuel. **Obligatoire ? :** not required **Type :** String
		Nom : source-cis **Description :** Nom du CI dans la requête devant faire l'objet d'une itération **Obligatoire ? :** required **Type :** String (script groovy)

Utilisation d'API

Chapitre 8 : Introduction aux API

Contenu de ce chapitre :

Présentation des API 264

Présentation des API

Les API suivantes sont incluses avec HP Universal CMDB :

- **API Java UCMDB.** Décrit l'utilisation de l'API Java par des outils tiers ou personnalisés pour extraire des données et des calculs et écrire des données dans UCMDB (base de données de gestion de la configuration universelle). Pour plus d'informations, voir " API HP Universal CMDB " , page 265.

- **API de service Web UCMDB.** Permet d'écrire les définitions d'éléments de configuration et les relations topologiques dans UCMDB, et de rechercher les informations à l'aide de requêtes TQL et ad hoc. Pour plus d'informations, voir " API de service Web HP Universal CMDB " , page 274.

- **API Java de gestion des flux de données.** Permet la gestion des sondes, des travaux, des déclencheurs et des informations d'identification dans le cadre de la gestion des flux de données. Pour plus d'informations, voir " API Java de gestion des flux de données " , page 309.

- **API de service Web de la gestion des flux de données.** Permet la gestion des sondes, des travaux, des déclencheurs et des informations d'identification dans le cadre de la gestion des flux de données. Pour plus d'informations, voir " API de service Web de la gestion des flux de données " , page 311.

Remarque : Pour tirer pleinement parti de la documentation d'une API, il est recommandé d'accéder à la documentation en ligne. En effet, la version PDF ne comporte pas de lien vers la documentation d'une API au format HTML.

Chapitre 9 : API HP Universal CMDB

Contenu de ce chapitre :

Conventions 265

Utilisation de l'API HP Universal CMDB 265

Structure générale d'une application 266

Placement du fichier Jar de l'API dans le classpath 269

Création d'un utilisateur d'intégration 269

Cas d'utilisation de l'API UCMDB 271

Exemples 273

Conventions

Ce chapitre observe les conventions suivantes :

- UCMDB désigne la base de données Universal Configuration Management proprement dite. HP Universal CMDB se rapporte à l'application.
- Les éléments et les arguments de méthode UCMDB sont cités en toutes lettres lorsqu'ils sont spécifiés dans les interfaces.

Pour une documentation complète sur les API disponibles, voir HP UCMDB API Reference.

Ces fichiers figurent dans le dossier suivant :

\\<répertoire racine de UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Utilisation de l'API HP Universal CMDB

Remarque : Utilisez ce chapitre en parallèle avec le Javadoc de l'API, disponible dans la bibliothèque de documentation en ligne.

L'API HP Universal CMDB permet d'intégrer des applications dans Universal CMDB (UCMDB). Elle fournit des méthodes pour effectuer les opérations suivantes :

- ajouter, supprimer et mettre à jour des CI et des relations dans CMDB ;
- extraire des informations sur le modèle de classe ;
- extraire des informations de l'historique UCMDB ;

- exécuter des scénarios de simulation ;
- extraire des informations sur les éléments de configuration et les relations.

Les méthodes permettant d'extraire des informations sur les éléments de configuration et les relations utilisent généralement le langage TQL (Topology Query Language). Pour plus d'informations, voir Topology Query Language dans le *Manuel de modélisation HP Universal CMDB*.

Les utilisateurs de l'API HP Universal CMDB doivent connaître :

- le langage de programmation Java ;
- HP Universal CMDB

Contenu de cette section :

- " Utilisations de l'API " , ci-dessous
- " Autorisations " , ci-dessous

Utilisations de l'API

L'API permet de satisfaire un certain nombre d'exigences métier. Ainsi, un système tiers peut interroger le modèle de classe pour obtenir des informations sur les éléments de configuration (CI) disponibles. Pour des cas d'utilisation, voir " Cas d'utilisation de l'API UCMDB " , page 271.

Autorisations

L'administrateur fournit des informations d'identification pour la connexion avec l'API. Le client de l'API a besoin du nom et du mot de passe d'un utilisateur d'intégration défini dans CMDB. Ces utilisateurs ne représentent pas des utilisateurs humains de CMDB, mais des applications qui se connectent à CMDB.

De plus, l'utilisateur doit disposer d'une autorisation pour l'action générale **Accéder à SDK** pour se connecter.

> **Attention :** Le client de l'API peut également fonctionner avec des utilisateurs réguliers dès lors qu'ils disposent d'autorisations d'authentification API. Toutefois, cette option est déconseillée.

Pour plus d'informations, voir " Création d'un utilisateur d'intégration " , page 269.

Structure générale d'une application

Il existe une seule usine statique, UcmdbServiceFactory. Cette usine est le point d'entrée d'une application. UcmdbServiceFactory présente des méthodes getServiceProvider. Ces méthodes renvoient une instance de l'interface **UcmdbServiceProvider**.

Le client crée d'autres objets utilisant des méthodes d'interface. Par exemple, pour créer une définition de requête, le client :

1. Extrait le service de requête du principal objet de service CMDB.
2. Extrait de l'objet de service un objet de requête prédéfini.
3. Extrait de l'usine une nouvelle définition de requête.

```
UcmdbServiceProvider provider =
   UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
UcmdbService ucmdbService =
   provider.connect(provider.createCredentials(USERNAME,
   PASSWORD), provider.createClientContext("Test"));
TopologyQueryService queryService = ucmdbService.getTopologyQueryService(
);
TopologyQueryFactory factory = queryService.getFactory();
QueryDefinition queryDefinition = factory.createQueryDefinition("Test Que
ry");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() + " hosts i
n uCMDB");
```

Les services accessibles depuis **UcmdbService** sont les suivants :

Méthodes de service	Utilisation
getAuthorizationModelService	Exécution d'opérations d'autorisation (création d'utilisateurs et de groupes d'utilisateurs, affectation de rôles aux utilisateurs et aux groupes, etc.).
getClassModelService	Informations sur les types de CI et les relations.
getConfigurationService	Gestion des paramètres d'infrastructure pour la configuration du serveur.
getDataStoreMgmtService	Recherche d'informations dans le magasin de données, notamment les CI et les attributs qui seront fédérés.
getDDMConfigurationService	Configuration du système de gestion des flux de données.
getDDMManagementService	Analyse et affichage de la progression, des résultats et des erreurs du système de gestion des flux de données.
getDDMZoneService	Importation et exportation des zones de gestion (avec leurs activités).

Méthodes de service	Utilisation
getHistoryService	Informations relatives à l'historique des CI contrôlés (changements, suppressions, etc.).
getImpactAnalysisService	Exécution d'un scénario d'analyse d'impact (également appelée **corrélation**).
getLicensingService	Recherche d'informations relatives aux licences installées dans le système.
getMultipleCMDBService	Conversion entre les ID globaux et les ID de UCMDB.
getMultiTenancyService	Création, lecture, mise à jour et suppression de locataires.
getPersistencyService	Persistance des données binaires dans de paires de valeurs de clé.
getQueryManagementService	Gestion de l'accès aux requêtes - enregistrement, suppression, liste des requêtes existantes. Assure également la validation des requêtes et la découverte des dépendances des requêtes.
getReconciliationService	Fonctions de fusion et d'identification.
getResourceBundleManagementService	Identification des ressources (services de regroupement). Permet la création explicite de nouveaux indicateurs et le retrait d'indicateurs de toutes les ressources identifiées.
getResourceManagementService	Déploiement de packages de ressources (requêtes TQL, vues, utilisateurs, etc.) vers le système.
getSecurityService	Vérification de la validité des informations d'identification.
getServerService	Recherche d'informations génériques relatives au système.
getSnapshotService	Services de gestion des instantanés (récupérer, enregistrer, comparer, etc.).
getSoftwareSignatureService	Définition des logiciels devant être découverts par le système de gestion des flux de données.
getStateService	Services de gestion des états (lister, ajouter, supprimer, etc).
getSystemHealthService	Services d'intégrité du système (indicateurs de performance système de base, mesures de capacité et de disponibilité).

Méthodes de service	Utilisation
getTopologyQueryService	Obtention d'informations sur l'Univers IT.
getTopologyUpdateService	Modification d'informations dans l'Univers IT.
getUcmdbVersion	Recherche d'informations relatives à la compilation et aux versions du Content Pack et de UCMDB.
getViewArchiveService	Services d'archivage des résultats de vue. Permet également d'enregistrer le résultat de la vue en cours et d'extraire les résultats précédemment enregistrés.
getViewService	Service d'exécution (définition d'exécution, exécution enregistrée) et service de gestion (enregistrement, suppression, liste des vues existantes) des vues. Assure également la validation des vues et la découverte des dépendances.

Le client communique avec le serveur sur HTTP(S).

Placement du fichier Jar de l'API dans le classpath

L'utilisation de ce jeu d'API exige le fichier **ucmdb-api.jar**. Vous pouvez télécharger le fichier en saisissant l'adresse `http://<localhost>:8080` dans un navigateur Web où `localhost` correspond à l'ordinateur sur lequel UCMDB est installé, et en cliquant sur le lien **API Client Download**.

Placez le fichier **jar** dans le classpath avant de compiler ou d'exécuter votre application.

> **Remarque :** L'utilisation de l'API Java UCMDB requiert l'installation de JRE (Java Runtime Environment) version 6 ou ultérieure.

Création d'un utilisateur d'intégration

Vous pouvez créer un utilisateur dédié pour les intégrations entre d'autres produits et UCMDB. Cet utilisateur permet à un produit qui utilise le SDK du client UCMDB d'être authentifié dans le SDK du serveur et d'exécuter les API. Les applications écrites avec ce jeu d'API doivent se connecter avec les informations d'identification de l'utilisateur d'intégration.

> **Attention :** Il est également possible de se connecter à l'aide d'un nom d'utilisateur UCMDB ordinaire (par exemple, admin). Toutefois, cette option est déconseillée. Pour se connecter avec un utilisateur UCMDB, vous devez octroyer à cet utilisateur les droits d'authentification API.

Pour créer un utilisateur d'intégration :

1. Lancez le navigateur Web, puis entrez l'adresse du serveur comme suit :

   `http://localhost:8080/jmx-console`.

   Vous devrez peut-être vous connecter à l'aide d'un nom d'utilisateur et d'un mot de passe.

2. Sous `UCMDB`, cliquez sur **service=UCMDB Authorization Services**.

3. Recherchez l'opération **createUser**. Cette méthode accepte les paramètres suivants :

   - **customerId**. ID du client.
   - **username**. Nom de l'utilisateur d'intégration.
   - **userDisplayName**. Nom d'affichage de l'utilisateur d'intégration.
   - **userLoginName**. Nom de connexion de l'utilisateur d'intégration.
   - **password**. Mot de passe de l'utilisateur d'intégration.

4. Cliquez sur **Invoke**.

5. Dans un environnement d'une seule société, accédez à la méthode **setRolesForUser** et entrez les paramètres suivants :

   - **userName**. Nom de l'utilisateur d'intégration.
   - **roles.** SuperAdmin.

   Cliquez sur **Invoke**.

6. Dans un environnement multilocataire, accédez à la méthode **grantRolesToUserForAllTenants** et entrez les paramètres suivants pour affecter le rôle lié à l'ensemble des locataires :

   - **userName**. Nom de l'utilisateur d'intégration.
   - **roles.** SuperAdmin.

   Cliquez sur **Invoke**.

   Pour affecter le rôle en lien avec des locataires particuliers, vous pouvez à appeler la méthode **grantRolesToUserForTenants** et utiliser les mêmes valeurs pour les paramètres userName et roles. Pour le paramètre **tenantNames**, entrez les locataires requis.

7. Créez des utilisateurs supplémentaires ou fermez la console JMX.

8. Connectez-vous à UCMDB en tant qu'administrateur.

9. Dans l'onglet **Administration**, exécutez le **Gestionnaire des packages**.

10. Cliquez sur l'icône **Créer un package personnalisé**.
11. Saisissez le nom du nouveau package et cliquez sur **Suivant**.
12. Dans l'onglet Sélection des ressources, sous **Paramètres**, cliquez sur **Utilisateurs**.
13. Sélectionnez un ou plusieurs utilisateurs que vous avez créés à l'aide de la console JMX.
14. Cliquez sur **Suivant** puis sur **Terminer**. Votre nouveau package apparaît dans la liste des noms de package dans le Gestionnaire des packages.
15. Déployez le package pour les utilisateurs qui exécuteront les applications API.

    Pour plus d'informations, voir « Déploiement d'un package » dans le *Manuel d'administration HP Universal CMDB*.

Remarque :

Un utilisateur d'intégration est créé pour chaque client. Pour créer un utilisateur d'intégration plus fort destiné à plusieurs clients, insérez une méthode **systemUser** en attribuant la valeur **true** à l'indicateur **isSuperIntegrationUser**. Utilisez les méthodes **systemUser** (**removeUser**, **resetPassword**, **UserAuthenticate**, etc.).

Il existe deux utilisateurs système prédéfinis. Après l'installation, il est recommandé de changer leur mot de passe à l'aide de la méthode **resetPassword**.

- **sysadmin**/**sysadmin**
- **UISysadmin**/**UISysadmin** (cet utilisateur est également le super utilisateur d'intégration **SuperIntegrationUser**).

  Si vous changez le mot de passe UISysadmin à l'aide de la méthode **resetPassword**, procédez comme suit :

  i. Dans la console JMX, recherchez le service **UCMDB-UI:name=UCMDB Integration**.

  ii. Exécutez **setCMDBSuperIntegrationUser** avec le nom d'utilisateur et le nouveau mot de passe de l'utilisateur d'intégration.

Cas d'utilisation de l'API UCMDB

Les cas d'utilisation décrits dans cette section requièrent deux systèmes :

- Serveur HP Universal CMDB
- Système tiers contenant un référentiel des éléments de configuration.

Contenu de cette section :

- " Remplissage de CMDB " , ci-dessous
- " Interrogation du CMDB " , ci-dessous
- " Interrogation du modèle de classe " , ci-dessous
- " Analyse de l'impact des modifications " , ci-dessous

Remplissage de CMDB

Cas d'utilisation :

- Un outil tiers de gestion des actifs met à jour CMDB avec des informations disponibles uniquement dans la gestion des actifs.
- Plusieurs systèmes tiers alimentent CMDB afin de créer une base CMDB centrale pour le suivi des modifications et la mise en œuvre d'analyses d'impact.
- Un système tiers crée des éléments de configuration et des relations conformes à la logique métier tierce, afin d'exploiter les fonctionnalités de requête d'UCMDB.

Interrogation du CMDB

Cas d'utilisation :

- Un système tiers obtient les éléments de configuration et les relations qui représentent le système SAP en extrayant les résultats du langage TQL SAP.
- Un système tiers obtient la liste des serveurs Oracle qui ont été ajoutés ou modifiés au cours des cinq dernières heures.
- Un système tiers obtient la liste des serveurs dont le nom d'hôte contient la sous-chaîne **lab**.
- Un système tiers trouve les éléments liés à un CI donné en extrayant ses voisins.

Interrogation du modèle de classe

Cas d'utilisation :

- Un système tiers permet aux utilisateurs de définir l'ensemble de données à extraire de CMDB. Une interface utilisateur peut être créée sur le modèle de classe pour montrer aux utilisateurs les propriétés possibles et leur demander les données requises. L'utilisateur peut alors choisir les informations à extraire.
- Un système tiers explore le modèle de classe lorsque l'utilisateur ne peut pas accéder à l'interface utilisateur UCMDB.

Analyse de l'impact des modifications

Cas d'utilisation :

- Un système tiers sort une liste des services métier sur lesquels une modification survenue sur un hôte désigné est susceptible d'avoir une incidence.

Exemples

Reportez-vous aux exemples de code suivants :

- Create a Connection
- Create and Execute an Ad Hoc Query
- Create and Execute a View
- Add and Delete Data
- Execute an Impact Analysis
- Query the Class Model
- Query a History Sample

Les fichiers se trouvent dans le répertoire

\\<répertoire racine de UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples

Chapitre 10 : API de service Web HP Universal CMDB

Contenu de ce chapitre :

Conventions 274

Présentation de l'API de service Web HP Universal CMDB 275

Appel du service Web 278

Interrogation de CMDB 278

Mise à jour d'UCMDB 282

Interrogation du modèle de classe UCMDB 283

Requête d'analyse d'impact 285

Paramètres généraux UCMDB 285

Paramètres de sortie UCMDB 288

Méthodes de requête UCMDB 289

Méthodes de mise à jour UCMDB 300

Méthodes d'analyse d'impact UCMDB 303

API de service Web d'état réel 305

Cas d'utilisation de l'API de service Web UCMDB 306

Exemples 307

Conventions

Ce chapitre observe les conventions suivantes :

- UCMDB désigne la base de données Universal Configuration Management proprement dite. HP Universal CMDB se rapporte à l'application.
- Les éléments et arguments de méthode UCMDB sont cités en toutes lettres lorsqu'ils sont spécifiés dans le schéma. Un élément ou un argument d'une méthode ne porte pas de majuscule. Par exemple, une `relation` est un élément de type `Relation` transmis à une méthode.

Pour une documentation complète sur les structures de demande et de réponse, voir HP UCMDB Web Service API Reference. Ces fichiers figurent dans le dossier suivant :

<répertoire racine UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

Présentation de l'API de service Web HP Universal CMDB

Remarque : Utilisez ce chapitre en parallèle avec la documentation sur les schémas UCMDB, disponible dans la bibliothèque de documentation en ligne.

L'API de service Web HP Universal CMDB permet d'intégrer des applications avec la base de données HP Universal CMDB (UCMDB). Elle fournit des méthodes pour effectuer les opérations suivantes :

- ajouter, supprimer et mettre à jour des CI et des relations dans CMDB ;
- extraire des informations sur le modèle de classe ;
- extraire des analyses d'impact ;
- extraire des informations sur les éléments de configuration et les relations ;
- gérer les informations d'identification : affichage, ajout, mise à jour et suppression ;
- gérer des travaux : affichage du statut, activation et désactivation ;
- gérer des plages de sonde : affichage, ajout et mise à jour ;
- gérer des déclencheurs : ajout ou suppression d'un CI déclencheur, et ajout, suppression ou désactivation d'un TQL déclencheur ;
- afficher des données générales sur les domaines et le sondes.

Les méthodes permettant d'extraire des informations sur les éléments de configuration et les relations utilisent généralement le langage TQL (Topology Query Language). Pour plus d'informations, voir Topology Query Language dans le *Manuel de modélisation HP Universal CMDB*.

Les utilisateurs de l'API de service Web HP Universal CMDB doivent connaître :

- la spécification SOAP ;
- un langage de programmation orientée objet, tel que C++, C# ou Java ;
- HP Universal CMDB
- la gestion des flux de données.

Contenu de cette section :

- " Utilisations de l'API " , page suivante
- " Autorisations " , page suivante

Utilisations de l'API

L'API de services Web UCMDB permet de satisfaire un certain nombre d'exigences métier. Par exemple :

- Un système tiers peut interroger le modèle de classe pour obtenir des informations sur les éléments de configuration (CI) disponibles.
- Un outil tiers de gestion des actifs peut mettre à jour la base CMDB avec des informations disponibles uniquement pour cet outil, unissant ainsi ses données avec les données recueillies par les applications HP.
- Plusieurs systèmes tiers peuvent alimenter la base CMDB afin de créer une base CMDB centrale pour le suivi des modifications et la mise en œuvre d'analyses d'impact.
- Un système tiers peut créer des entités et des relations conformes à sa logique métier, puis écrire les données dans la base CMDB afin d'exploiter ses fonctionnalités de requête.
- D'autres systèmes, tels que le système Release Control (CCM), peuvent utiliser les méthodes d'analyse d'impact pour l'analyse des changements.

Autorisations

Pour accéder au fichier WSDL pour le service Web, rendez-vous sur : **http://localhost:8080/axis2/services/UcmdbService?wsdl**. Vous devez fournir les informations d'identification utilisateur de l'administrateur du serveur pour afficher le fichier WSDL.

L'utilisateur doit être doté de l'autorisation d'action générale **Exécuter l'API héritée** pour se connecter.

Le tableau suivant présente les autorisations supplémentaires requises pour chaque commande API de service Web :

Commande API de service Web	Autorisations requises
addCIsAndRelations deleteCIsAndRelations updateCIsAndRelations	Action générale : **Mettre à jour les données**
executeTopologyQueryByName(AdHoc) executeTopologyQueryByNameWithParameters(AdHoc) executeTopologyQueryWithParameters(AdHoc)	Action générale : **Exécuter une requête par définition** Pour chaque requête : Autorisation **Afficher**

Commande API de service Web	Autorisations requises
getTopologyQueryExistingResultByName getTopologyQueryResultCountByName releaseChunks pullTopologyMapChunks getCINeighbours getFilteredCIsByType getCIsById getCIsByType getRelationsById	Action générale : **Afficher les CI** Pour chaque requête : Autorisation **Afficher**
getQueryNameOfView	Action générale : **Afficher les CI** Pour chaque vue : Autorisation **Afficher**
getChangedCIs	Action générale : **Afficher l'historique**, **Afficher les CI**
calculateImpact getImpactPath getImpactRulesByGroupName getImpactRulesByNamePrefix	Action générale : **Exécuter l'analyse d'impact**
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Aucune

Remarque : Lorsque le contexte racine a été modifié dans UCMDB, exécutez les étapes suivantes pour permettre l'accès à l'API de service Web :

1. Ouvrez le fichier de configuration **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\web.xml** et recherchez la section suivante :

   ```
   <servlet-class>
   org.apache.axis2.transport.http.AxisServlet
   </servlet-class>
   ```

   Ajoutez les lignes suivantes à la fin de cette section :

   ```
   <init-param>
   <param-name>axis2.find.context</param-name>
   <param-value>false</param-value>
   </init-param>
   ```

2. Ouvrez le fichier de configuration **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\conf\axis2.xml** et recherchez la ligne suivante :

```
<parameter name="enableSwA" locked="false">false</parameter>
```

Ajoutez les lignes suivantes à la fin de cette ligne :

```
<parameter name="contextRoot" locked="false">test1/setup1/axis2
</parameter>
```

où **test1/setup1** est votre contexte racine.

(Pour supprimer le contexte racine, supprimez le texte ajouté au chemin.)

3. Redémarrez le serveur UCMDB.

Appel du service Web

Utilisez les techniques de programmation SOAP standard dans le service Web HP Universal CMDB pour permettre l'appel de méthodes côté serveur. S'il est impossible d'analyser l'instruction ou si un incident se produit lors de l'appel de la méthode, les méthodes API lèvent une exception `SoapFault`. Lorsqu'une exception `SoapFault` est levée, UCMDB complète un ou plusieurs des champs de message d'erreur, de code d'erreur et de message d'exception. En l'absence d'erreur, les résultats de l'appel sont renvoyés.

Les programmateurs SOAP peuvent accéder au WSDL à l'adresse :

http://<server>[:port]/axis2/services/UcmdbService?wsdl

La spécification de port n'est nécessaire que pour les installations non standard. Demandez le numéro de port correct à votre administrateur système.

L'URL pour appeler le service est :

http://<server>[:port]/axis2/services/UcmdbService

Pour des exemples de connexion à la base CMDB, voir " Cas d'utilisation de l'API de service Web UCMDB " , page 306.

Interrogation de CMDB

La base CMDB est interrogée à l'aide des API décrites dans " Méthodes de requête UCMDB " , page 289. Les requêtes et les éléments CMDB renvoyés contiennent toujours des ID UMDB réels. Pour consulter des exemples d'utilisation de méthodes de requête, voir Query Example.

Contenu de cette section :

- " Calcul de réponse Juste-à-temps " , page suivante
- " Traitement de longues réponses " , page suivante
- " Spécification des propriétés à renvoyer " , page suivante

- " Propriétés concrètes " , page suivante
- " Propriétés dérivées " , page 281
- " Propriétés de dénomination " , page 281
- " Autres éléments de spécification de propriétés " , page 281

Calcul de réponse Juste-à-temps

Pour toutes les méthodes de requête, le serveur UCMDB calcule les valeurs demandées par la méthode à la réception de la demande, puis renvoie des résultats basés sur les données les plus récentes. Le résultat est toujours calculé au moment de la réception de la demande, même si la requête TQL est active et qu'il existe un résultat déjà calculé. Par conséquent, les résultats de l'exécution d'une requête renvoyés à l'application cliente peuvent être différents des résultats de la même requête affichés sur l'interface utilisateur.

Astuce : Si votre application utilise plusieurs fois les résultats d'une requête donnée et qu'il n'est pas prévu de modification de ces données entre les utilisations des résultats, vous pouvez améliorer les performances en demandant à l'application cliente de stocker les données plutôt que d'exécuter la requête à plusieurs reprises.

Traitement de longues réponses

La réponse à une requête comprend toujours les structures des données demandées par la méthode de requête, même si aucune donnée n'est réellement transmise. Pour de nombreuses méthodes dans lesquelles les données sont une collection ou une carte, la réponse comprend également la structure `ChunkInfo`, composée de `chunksKey` et de `numberOfChunks`. Le champ `numberOfChunks` indique le nombre de segments contenant des données à extraire.

La taille de transmission maximale des données est définie par l'administrateur système. Si les données renvoyées par la requête sont plus volumineuses que la taille maximale, les structures de données de la première réponse ne contiennent aucune information significative et la valeur du champ `numberOfChunks` est égale ou supérieure à `2`. Si les données ne sont pas plus volumineuses que la taille maximale, le champ `numberOfChunks` est égal à 0 (zéro) et les données sont transmises dans la première réponse. Par conséquent, lors du traitement d'une réponse, vérifiez d'abord la valeur `numberOfChunks`. Si elle est supérieure à `1`, supprimez les données de la transmission et demandez les segments de données. Sinon, utilisez les données de la réponse.

Pour plus d'informations sur le traitement de segments de données, voir " pullTopologyMapChunks " , page 298 et " releaseChunks " , page 300.

Spécification des propriétés à renvoyer

Les CI et relations possèdent généralement de nombreuses propriétés. Certaines méthodes qui renvoient des collections ou des graphiques de ces éléments acceptent des paramètres d'entrée qui indiquent les valeurs de propriété à renvoyer avec chaque élément correspondant à la requête. La base CMDB ne renvoie pas de propriétés vides. Par conséquent, la réponse à une requête peut comporter moins de propriétés que ce que la requête en demandait.

Cette section décrit les types d'ensembles utilisés pour spécifier les propriétés à renvoyer.

Les propriétés peuvent être référencées de deux manières :

- par leurs noms ;
- par les noms de règles de propriétés prédéfinies. Les règles de propriétés prédéfinies sont utilisées par CMDB pour créer une liste de noms de propriétés réelles.

Lorsqu'une application fait référence à des propriétés par leur nom, elle transmet un élément `PropertiesList`.

> **Astuce :** Dans la mesure du possible, utilisez `PropertiesList` pour définir les noms des propriétés qui vous intéressent, au lieu d'un ensemble reposant sur des règles. L'utilisation de règles de propriétés prédéfinies aboutit généralement au renvoi de plus de propriétés que nécessaire, au détriment des performances.

Il existe deux types de propriétés prédéfinies : les propriétés qualificatives et les propriétés simples.

- **Propriétés qualificatives.** À utiliser lorsque l'application cliente doit transmettre un élément `QualifierProperties` (liste de qualificatifs pouvant être appliqués à des propriétés). La base CMDB convertit la liste des qualificatifs transmise par l'application cliente en liste des propriétés auxquelles s'applique au moins l'un des qualificatifs. Les valeurs de ces propriétés sont renvoyées avec les éléments `CI` ou `Relation`.
- **Propriétés simples.** Pour utiliser des propriétés simples reposant sur des règles, l'application cliente transmet un élément `SimplePredefinedProperty` ou `SimpleTypedPredefinedProperty`. Ces éléments contiennent le nom de la règle selon laquelle la base CMDB génère la liste des propriétés à renvoyer. Les règles qui peuvent être spécifiées dans un élément `SimplePredefinedProperty` ou `SimpleTypedPredefinedProperty` sont `CONCRETE`, `DERIVED` et `NAMING`.

Propriétés concrètes

Les propriétés concrètes sont l'ensemble de propriétés définies pour le type de CI spécifié. Les propriétés ajoutées par des classes dérivées ne sont pas renvoyées pour des instances de ces classes.

Une collection d'instances renvoyée par une méthode peut se composer d'instances d'un type de CI spécifié dans l'appel de méthode et d'instances de type de CI qui héritent de ce dernier. Les types de CI dérivés héritent des propriétés du type de CI spécifié. En outre, les types de CI dérivés complètent le type de CI parent en ajoutant des propriétés.

> **Exemple de propriétés concrètes :**
>
> Le type de CI `T1` possède les propriétés `P1` et `P2`. Le type de CI `T11` hérite de `T1` et complète `T1` avec les propriétés `P21` et `P22`.
>
> La collection de CI de type `T1` comprend les instances de `T1` et `T11`. Les propriétés concrètes de toutes les instances de cette collection sont `P1` et `P2`.

Propriétés dérivées

Les propriétés dérivées sont l'ensemble des propriétés définies pour le type de CI spécifié et, pour chaque type de CI dérivé, les propriétés ajoutées par le type de CI dérivé.

Exemple de propriétés dérivées :

Si l'on poursuit l'exemple des propriétés concrètes, les propriétés dérivées des instances de `T1` sont `P1` et `P2`. Les propriétés dérivées des instances de `T11` sont `P1`, `P2`, `P21` et `P22`.

Propriétés de dénomination

Les propriétés de dénomination sont `display_label` et `data_name`.

Autres éléments de spécification de propriétés

- **PredefinedProperties**

  `PredefinedProperties` peut contenir un élément `QualifierProperties` et un élément `SimplePredefinedProperty` pour chacune des autres règles possibles. Un ensemble `PredefinedProperties` ne contient pas nécessairement tous les types de listes.

- **PredefinedTypedProperties**

  `PredefinedTypedProperties` est utilisé pour appliquer un jeu différent de propriétés à chaque type de CI. `PredefinedTypedProperties` peut contenir un élément `QualifierProperties` et un élément `SimpleTypedPredefinedProperty` pour chacune des autres règles applicables. Comme `PredefinedTypedProperties` est appliqué individuellement à chaque type de CI, les propriétés dérivées ne s'appliquent pas. Un ensemble `PredefinedProperties` ne contient pas nécessairement tous les types applicables de listes.

- **CustomProperties**

  `CustomProperties` peut contenir n'importe quelle combinaison de la `PropertiesList` de base et des listes de propriétés reposant sur des règles. Le filtre des propriétés réunit toutes les propriétés renvoyées par toutes les listes.

- **CustomTypedProperties**

  `CustomTypedProperties` peut contenir n'importe quelle combinaison de la `PropertiesList` de base et des listes applicables de propriétés reposant sur des règles. Le filtre des propriétés réunit toutes les propriétés renvoyées par toutes les listes.

- **TypedProperties**

  `TypedProperties` est utilisé pour transmettre un jeu différent de propriétés pour chaque type de CI. `TypedProperties` est une collection de paires composées de noms de type et d'ensembles de propriétés de tous les types. Chaque ensemble de propriétés est appliqué uniquement au type correspondant.

Mise à jour d'UCMDB

Vous mettez à jour CMDB avec les API de mise à jour. Pour plus d'informations sur les méthodes API, voir " Méthodes de mise à jour UCMDB " , page 300.

Cette tâche comprend les étapes suivantes :

- " Paramètres de mise à jour UCMDB " , ci-dessous
- " Utilisation de types d'ID avec les méthodes de mise à jour " , ci-dessous

Paramètres de mise à jour UCMDB

Cette rubrique décrit les paramètres utilisés uniquement par les méthodes de mise à jour du service.

- **CIsAndRelationsUpdates**

  Le type `CIsAndRelationsUpdates` se compose de `CIsForUpdate`, `relationsForUpdate`, `referencedRelations` et `referencedCIs`. Une instance `CIsAndRelationsUpdates` ne comporte pas nécessairement les trois éléments.

  `CIsForUpdate` est une collection de CI. `relationsForUpdate` est une collection de `Relations`. Les éléments `CI` et `relation` dans les collections comportent un élément `props`. Lors de la création d'un CI ou d'une relation, les propriétés qui possèdent soit l'attribut `required`, soit l'attribut `key` dans la définition de type de CI doivent être renseignées par des valeurs. Les éléments de ces collections sont mis à jour ou créés par la méthode.

  `referencedCIs` et `referencedRelations` sont des collections de CI qui sont déjà définies dans la base CMDB. Les éléments de la collection sont identifiés avec un ID temporaire en combinaison avec toutes les propriétés clés. Ces éléments servent à résoudre les identités des CI et relations pour mise à jour. Ils ne sont jamais créés ni mis à jour par la méthode.

  Chacun des éléments `CI` et `relation` dans ces collections comporte une collection de propriétés. De nouveaux éléments sont créés avec les valeurs de propriété dans ces collections.

Utilisation de types d'ID avec les méthodes de mise à jour

Les sections suivantes décrivent les types de CI ID, les CI et les relations. Lorsque l'ID n'est pas un ID CMDB réel, le type et les attributs clés sont obligatoires.

- **Suppression ou mise à jour d'éléments de configuration**

  Un ID temporaire ou vide peut être utilisé par le client lors de l'appel d'une méthode pour supprimer ou mettre à jour un élément. Dans ce cas, le type de CI et les " Attributs clés " qui identifient le CI doivent être définis.

- **Suppression ou mise à jour de relations**

Lors de la suppression ou de la mise à jour de relations, l'ID de relation peut être vide, temporaire ou réel.

Si l'ID d'un CI est temporaire, le CI doit être transmis dans la collection `referencedCIs` et ses attributs clés doivent être spécifiés. Pour plus d'informations, voir referencedCIs dans " CIsAndRelationsUpdates " , page précédente.

- **Insertion de nouveaux éléments de configuration dans CMDB**

  Il est possible d'utiliser un ID vide ou temporaire ou insérer un nouveau CI. Cependant, si l'ID est vide, le serveur ne peut pas renvoyer l'ID CMDB réel dans la structure `createIDsMap` car il n'y a pas de `clientID`. Pour plus d'informations, voir " addCIsAndRelations " , page 300 et " Méthodes de requête UCMDB " , page 289.

- **Insertion de nouvelles relations dans CMDB**

  L'ID de relation peut être soit temporaire, soit vide. Cependant, si la relation est nouvelle mais que les éléments de configuration à chacune de ses extrémités sont déjà définis dans CMDB, ces CI qui existent déjà doivent être identifiés par un ID CMDB réel ou spécifiés dans une collection `referencedCIs`.

Interrogation du modèle de classe UCMDB

Les méthodes de modèle de classe renvoient des informations sur les types de CI et les relations. Le modèle de classe est configuré à l'aide du Gestionnaire des types de CI. Pour plus d'informations, voir CI Type Manager dans le *Manuel de modélisation HP Universal CMDB*.

Cette section fournit des informations sur les méthodes suivantes qui renvoient des informations sur les types de CI et les relations :

- " getClassAncestors " , ci-dessous
- " getAllClassesHierarchy " , page suivante
- " getCmdbClassDefinition " , page suivante

getClassAncestors

La méthode `getClassAncestors` extrait le chemin entre le type de CI donné et sa racine, cette dernière incluse.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
className	Nom du type. Pour plus d'informations, voir " Nom de type " , page 287.

Sortie

Paramètre	Remarque
classHierarchy	Collection de paires nom de classe et nom de classe parente.
comments	À usage interne uniquement.

getAllClassesHierarchy

La méthode `getAllClassesHierarchy` extrait toute l'arborescence des modèles de classe.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page suivante.

Sortie

Paramètre	Remarque
classesHierarchy	Collection de paires nom de classe et nom de classe parente.
comments	À usage interne uniquement.

getCmdbClassDefinition

La méthode `getCmdbClassDefinition` extrait des informations sur la classe spécifiée.

Si vous utilisez `getCmdbClassDefinition` pour extraire les attributs de clé, vous devez également interroger les classes parentes jusqu'à la classe de base. `getCmdbClassDefinition` identifie comme attributs de clé uniquement les attributs pour lesquels `ID_ATTRIBUTE` est défini dans la définition de classe spécifiée par `className`. Les attributs de classe hérités ne sont pas reconnus comme attributs clés de la classe spécifiée. Par conséquent, la liste complète des attributs clés pour la classe spécifiée est l'union de toutes les clés de la classe et de tous ses parents, jusqu'à la racine.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page suivante.
className	Nom du type. Pour plus d'informations, voir " Paramètres généraux UCMDB " , page suivante.

Sortie

Paramètre	Remarque
cmdbClass	Définition de classe, composée de `name`, `classType`, `displayLabel`, `description`, `parentName`, de qualificatifs et d'attributs.
comments	À usage interne uniquement.

Requête d'analyse d'impact

Dans les méthodes d'analyse d'impact, `Identifier` désigne les données de réponse du service. Il est unique pour la réponse actuelle et est supprimé de la mémoire cache du serveur s'il n'est pas utilisé pendant 10 minutes.

Pour des exemples d'utilisation des méthodes d'analyse d'impact, voir Impact Analysis Example.

Paramètres généraux UCMDB

Cette section décrit les paramètres les plus courants des méthodes du service. schema documentation

Contenu de cette section :

- " CmdbContext " , ci-dessous
- " ID " , ci-dessous
- " Attributs clés " , page suivante
- " Types d'ID " , page suivante
- " CIProperties " , page suivante
- " Nom de type " , page 287
- " Élément de configuration (CI) " , page 287
- " Relation " , page 287

CmdbContext

Tous les appels de service API de service Web UCMDB exigent un argument `CmdbContext`. `CmdbContext` est une chaîne `callerApplication` qui identifie l'application appelant le service. `CmdbContext` est utilisé pour la journalisation et le dépannage.

ID

Chaque `CI` et Relation comporte un champ `ID`. Il se compose d'une chaîne d'ID sensible à la casse et d'un indicateur `temp` facultatif, indiquant si l'ID est temporaire.

Attributs clés

Pour identifier un `CI` ou une `Relation` dans certains contextes, il est possible d'utiliser des attributs à la place d'un ID CMDB. Les attributs clés sont ceux pour lesquels ID_ATTRIBUTE est défini dans la définition de classe.

Dans l'interface utilisateur, une icône de clé est affichée en regard des attributs clés dans la liste des attributs de type d'élément de configuration. Pour plus d'informations, voir " Boîte de dialogue Ajouter/Modifier un attribut " on page 1 dans le Manuel de modélisation HP Universal CMDB. Pour plus d'informations sur l'identification des attributs clés à partir de l'application cliente API, voir " getCmdbClassDefinition " , page 284.

Types d'ID

Un élément `ID` peut contenir un ID réel ou un ID temporaire.

Un ID réel est une chaîne affectée par la base CMDB qui identifie une entité dans la base de données. Un ID temporaire peut être n'importe quelle chaîne unique dans la demande actuelle.

Un ID temporaire peut être affecté par le client et représente souvent l'ID du CI tel qu'il est stocké par le client. Il ne représente pas nécessairement une entité déjà créée dans la base CMDB. Lorsqu'un ID temporaire est transmis par le client, si le CMDB peut identifier un élément de configuration de données existant à l'aide des propriétés clés du CI, ce dernier est utilisé selon le contexte comme s'il avait été identifié avec un ID réel.

CIProperties

Un `CIProperties` se compose de collections, contenant chacune une suite d'éléments nom-valeur qui spécifient les propriétés du type indiqué par le nom de collection. Aucune des collections n'étant obligatoire, l'élément `CIProperties` peut contenir toute combinaison de collections.

`CIProperties` est utilisé par les éléments `CI` et `Relation`. Pour plus d'informations, voir " Élément de configuration (CI) " , page suivante et " Relation " , page suivante.

Les collections de propriétés sont :

- `dateProps` - collection d'éléments `DateProp`
- `doubleProps` - collection d'éléments `DoubleProp`
- `floatProps` - collection d'éléments `FloatProp`
- `intListProps` - collection d'éléments `intListProp`
- `intProps` - collection d'éléments `IntProp`
- `strProps` - collection d'éléments `StrProp`
- `strListProps` - collection d'éléments `StrListProp`
- `longProps` - collection d'éléments `LongProp`
- `bytesProps` - collection d'éléments `BytesProp`
- `xmlProps` - collection d'éléments `XmlProp`

Nom de type

Le nom de type est le nom de classe d'un type d'élément de configuration ou de relation. Le nom de type est utilisé dans le code pour faire référence à la classe. Il ne doit pas être confondu avec le nom affiché, qui apparaît sur l'interface utilisateur où la classe est mentionnée, mais qui n'a aucune signification dans le code.

Élément de configuration (CI)

Un `CI` se compose d'un `ID`, d'un `type` et d'une collection `props`.

Lors de l'utilisation des " Méthodes de mise à jour UCMDB " pour mettre à jour un `CI`, l'élément `ID` peut contenir un ID CMDB réel ou un ID temporaire affecté par le client. Si un ID temporaire est utilisé, attribuez la valeur true à l'indicateur `temp`. En cas d'utilisation d'un élément, l'`ID` peut être vide. Les " Méthodes de requête UCMDB " prennent les `ID` réels comme paramètres d'entrée et renvoient les `ID` réels dans les résultats de requête.

Le `type` peut être n'importe quel nom de type défini dans le Gestionnaire des types de CI. Pour plus d'informations, voir " Gestionnaire des types de CI " on page 1 dans le Manuel de modélisation HP Universal CMDB.

L'élément `props` est une collection CIProperties. Pour plus d'informations, voir " Paramètres généraux UCMDB " , page 285.

Relation

Une `Relation` est une entité qui relie deux éléments de configuration. Un élément `Relation` se compose d'un `ID`, d'un `type`, des identifiants des deux éléments liés `(end1ID` et `end2ID),` et d'une collection `props`.

Lors de l'utilisation des " Méthodes de mise à jour UCMDB " pour mettre à jour une `relation`, la valeur des ID de `relation` peut être un ID CMDB réel ou un ID temporaire. En cas de suppression d'un élément, l'`ID` peut être vide. Les " Méthodes de requête UCMDB " prennent les `ID` réels comme paramètres d'entrée et renvoient les `ID` réels dans les résultats de requête.

Le type de relation est le `nom de type` de la classe UCMDB depuis laquelle la relation est instanciée. Ce type peut désigner n'importe quel type de relation défini dans la base CMDB. Pour plus d'informations sur les classes ou les types, voir " Interrogation du modèle de classe UCMDB " , page 283.

Pour plus d'informations, voir " Gestionnaire des types de CI " on page 1 dans le *Manuel de modélisation HP Universal CMDB*.

Les deux ID d'extrémité d'une relation ne doivent pas être vides car ils servent à créer l'ID de la relation actuelle. Cependant, le client peut leur affecter à tous les deux des ID temporaires.

L'élément `props` est une collection CIProperties. Pour plus d'informations, voir " CIProperties " , page précédente.

Paramètres de sortie UCMDB

Cette section décrit les paramètres de sortie les plus courants des méthodes du service. Pour plus d'informations, voir la online schema documentation.

Contenu de cette section :

- " CIs " , ci-dessous
- " ShallowRelation " , ci-dessous
- " Topology " , ci-dessous
- " CINode " , ci-dessous
- " RelationNode " , ci-dessous
- " TopologyMap " , ci-dessous
- " ChunkInfo " , page suivante

CIs

`CIs` représente une collection de CI.

ShallowRelation

`ShallowRelation` est une entité qui relie deux éléments de configuration, composée d'un `ID`, d'un `type` et des identifiants de deux éléments liés `(end1ID` et `end2ID)`. Le type de relation est le `nom de type` de la classe CMDB depuis laquelle la relation est instanciée. Ce type peut désigner n'importe quel type de relation défini dans la base CMDB.

Topology

`Topology` est un graphique des éléments et relations de `CI`. `Topology` se compose d'une collection `CIs` et d'une collection `Relations` contenant un ou plusieurs éléments `Relation`.

CINode

`CINode` se compose d'une collection `CIs` avec une `étiquette`. `label` contenu dans `CINode` est l'étiquette définie dans le nœud du code TQL utilisé dans la requête.

RelationNode

`RelationNode` est un ensemble de collections `Relations` avec une `étiquette`. Le `label` contenu dans `RelationNode` est l'étiquette définie dans le nœud du code TQL utilisé dans la requête.

TopologyMap

`TopologyMap` est la sortie d'un calcul de requête qui correspond à une requête TQL. Les `labels` contenus dans `TopologyMap` sont les étiquettes de nœud définies dans le code TQL utilisé dans la requête.

Les données de `TopologyMap` sont renvoyées sous la forme suivante :

- `CINodes`. Il s'agit d'un ou de plusieurs `CINode` (voir " CINode " , page précédente).
- `relationNodes`. Il s'agit d'un ou de plusieurs `RelationNode` (voir " RelationNode " , page précédente).

Les `labels` contenus dans ces deux structures ordonnent les listes d'éléments de configuration et de relations.

ChunkInfo

Lorsqu'une requête renvoie une grande quantité de données, le serveur stocke ces données divisées en segments (chunks). Les informations que le client utilise pour extraire les segments de données figurent dans la structure `ChunkInfo` renvoyée par la requête. `ChunkInfo` se compose de `numberOfChunks` qui doit être extrait et de `chunksKey`. `chunksKey` est un identifiant unique des données sur le serveur pour cet appel de requête particulier.

Pour plus d'informations, voir " Traitement de longues réponses " , page 279.

Méthodes de requête UCMDB

Cette section fournit des informations sur les méthodes suivantes :

- " executeTopologyQueryByNameWithParameters " , ci-dessous
- " executeTopologyQueryWithParameters " , page suivante
- " getChangedCIs " , page 291
- " getCINeighbours " , page 292
- " getCIsByID " , page 292
- " getCIsByType " , page 293
- " getFilteredCIsByType " , page 293
- " getQueryNameOfView " , page 297
- " getTopologyQueryExistingResultByName " , page 297
- " getTopologyQueryResultCountByName " , page 297
- " pullTopologyMapChunks " , page 298
- " releaseChunks " , page 300

executeTopologyQueryByNameWithParameters

La méthode `executeTopologyQueryByNameWithParameters` extrait un élément `topologyMap` qui correspond à la requête paramétrée spécifiée.

Les valeurs des paramètres de requête sont transmises dans l'argument `parameterizedNodes`. Le code TQL spécifié doit comporter des étiquettes uniques définies pour chaque `CINode` et chaque `relationNode`, sinon l'appel de méthode échoue.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
queryName	Nom du code TQL paramétré dans la base CMDB pour lequel extraire la carte.
parameterizedNodeList	Conditions que chaque nœud doit satisfaire pour être inclus dans les résultats de la requête.
queryTypedProperties	Collection d'ensembles de propriétés à extraire pour des éléments d'un type d'élément de configuration particulier.

Sortie

Paramètre	Remarque
topologyMap	Pour plus d'informations, voir " TopologyMap " , page 288.
chunkInfo	Pour plus d'informations, voir " ChunkInfo " , page précédente et " Traitement de longues réponses " , page 279.

executeTopologyQueryWithParameters

La méthode `executeTopologyQueryWithParameters` extrait un élément `topologyMap` qui correspond à la requête paramétrée.

La requête est transmise dans l'argument `queryXML`. Les valeurs des paramètres de requête sont transmises dans l'argument `parameterizedNodeList`. Le code TQL doit comporter des étiquettes uniques définies pour chaque `CINode` et chaque `relationNode`.

La méthode `executeTopologyQueryWithParameters` permet de transmettre des requêtes ad hoc, au lieu d'accéder à une requête définie dans le CMDB. Vous pouvez utiliser cette méthode lorsque vous n'avez pas accès à l'interface utilisateur UCMDB pour définir une requête, ou lorsque vous ne souhaitez pas enregistrer la requête dans la base de données.

Pour utiliser un code TQL exporté comme entrée de la méthode, procédez comme suit :

1. Lancez le navigateur Web et entrez l'adresse **http://localhost:8080/jmx-console**.

   Vous devrez peut-être vous connecter à l'aide d'un nom d'utilisateur et d'un mot de passe.

2. Cliquez sur **UCMDB:service=TQL Services**.

3. Recherchez l'opération **exportTql**.

- Dans le champ du paramètre **customerId**, entrez **1** (valeur par défaut).
- Dans le champ du paramètre **patternName**, entrez un nom TQL valide.

4. Cliquez sur **Invoke**.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
queryXML	Chaîne XML représentant un code TQL sans balises de ressource.
parameterizedNodeList	Conditions que chaque nœud doit satisfaire pour être inclus dans les résultats de la requête.

Sortie

Paramètre	Remarque
topologyMap	Pour plus d'informations, voir " TopologyMap " , page 288.
chunkInfo	Pour plus d'informations, voir " ChunkInfo " , page 289 et " Traitement de longues réponses " , page 279.

getChangedCIs

La méthode `getChangedCIs` renvoie les données de changement pour tous les CI liés aux CI spécifiés.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
ids	Liste des ID des CI racine dont les CI liés sont vérifiés pour la présence de modifications. Seuls les ID CMDB réels sont valides dans cette collection..
fromDate	Début de la période à laquelle vérifier si des CI ont été modifiés.
toDate	Fin de la période à laquelle vérifier si des CI ont été modifiés.

Sortie

Paramètre	Remarque
getChangedCIsResponseList	Zéro collection ou plus d'éléments `ChangedDataInfo`.

getCINeighbours

La méthode `getCINeighbours` renvoie les voisins immédiats du CI spécifié.

Par exemple, si la requête porte sur les voisins du CI `A`, et que le CI `A` contient le CI `B` qui utilise le CI `C`, le CI `B` est renvoyé, mais pas le CI `C`. Autrement dit, seuls les voisins du type spécifié sont renvoyés.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
ID	ID du CI pour lequel les voisins sont extraits. Il doit s'agir d'un ID CMDB réel.
neighbourType	Nom de type de CI des voisins à extraire. Les voisins du type spécifié et des types dérivés de ce type sont renvoyés. Pour plus d'informations, voir " Nom de type " , page 287.
CIProperties	Données à renvoyer sur chaque élément de configuration (appelées Mise en page dans l'interface utilisateur). Pour plus d'informations, voir " TypedProperties " , page 281.
relationProperties	Données à renvoyer sur chaque relation (appelées Mise en page dans l'interface utilisateur). Pour plus d'informations, voir " TypedProperties " , page 281.

Sortie

Paramètre	Remarque
topology	Pour plus d'informations, voir " Topology " , page 288.
comments	À usage interne uniquement.

getCIsByID

La méthode `getCIsByID` extrait des éléments de configuration par leurs ID CMDB.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
CIsTypedProperties	Collection de propriétés typées. Pour plus d'informations, voir " Autres éléments de spécification de propriétés " , page 281.
IDs	Seuls les ID CMDB réels sont valides dans cette collection.

Sortie

Paramètre	Remarque
CIs	Collection d'éléments de CI.
chunkInfo	Pour plus d'informations, voir " ChunkInfo " , page 289 et " Traitement de longues réponses " , page 279.

getCIsByType

La méthode `getCIsByType` renvoie la collection d'éléments de configuration du type spécifié et de tous les types qui héritent de ce dernier.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
type	Nom de classe. Pour plus d'informations, voir " Nom de type " , page 287.
properties	Données à renvoyer sur chaque élément de configuration. Pour plus d'informations, voir " CustomProperties " , page 281.

Sortie

Paramètre	Remarque
CIs	Collection d'éléments de CI.
chunkInfo	Pour plus d'informations, voir : " ChunkInfo " , page 289, " Traitement de longues réponses " , page 279.

getFilteredCIsByType

La méthode `getFilteredCIsByType` extrait les CI du type spécifié qui répondent aux conditions utilisées par la méthode. Une condition se compose des éléments suivants :

- un champ de nom contenant le nom d'une propriété ;
- un champ d'opérateur contenant un opérateur de comparaison ;
- un champ de valeur facultatif contenant une valeur ou une liste de valeurs.

Ensemble, ces éléments forment une expression booléenne :
`<item>.property.value [operator] <condition>.value`

Par exemple, si le nom de la condition est `root_actualdeletionperiod`, sa valeur est `40` et l'opérateur est `Equal`, l'instruction booléenne est la suivante :

```
<item>.root_actualdeletionperiod.value = = 40
```

La requête renvoie tous les éléments pour lesquels `root_actualdeletionperiod` a la valeur `40`, en supposant qu'il n'existe aucune autre condition.

Si l'argument `conditionsLogicalOperator` est `AND`, la requête renvoie les éléments qui répondent à toutes les conditions dans la collection `conditions`. Si `conditionsLogicalOperator` est `OR`, la requête renvoie les éléments qui répondent à l'une au moins des conditions dans la collection `conditions`.

Le tableau suivant dresse la liste des opérateurs de comparaison :

Opérateur	Type de condition/Remarques
ChangedDuring	Date Il s'agit d'un contrôle de plage. La valeur de la condition est exprimée en heures. Si la valeur de la propriété de date se situe dans la plage de temps dans laquelle la méthode est appelée plus ou moins la valeur de la condition, cette dernière est vraie. Par exemple, si la valeur de la condition est `24`, la condition est vraie si la valeur de la propriété de date est comprise entre hier à cette heure et demain à cette heure. **Remarque :** Le nom `ChangedDuring` est conservé pour préserver la compatibilité rétroactive. Dans les versions précédentes, l'opérateur était uniquement utilisé avec les propriétés de création et de modification d'heure.
Equal	Chaîne et numérique
EqualIgnoreCase	Chaîne
Greater	Numérique
GreaterEqual	Numérique
In	Chaîne, numérique et liste La valeur de la condition est une liste. La condition est vraie si la valeur de la propriété est l'une des valeurs de la liste.
InList	Liste Les valeurs de la condition et de la propriété sont des listes. La condition est vraie si toutes les valeurs dans la liste de la condition apparaissent aussi dans la liste des propriétés de l'élément. La condition reste vraie même s'il y a plus de valeurs de propriété que le nombre spécifié dans la condition.

Opérateur	Type de condition/Remarques
IsNull	Chaîne, numérique et liste La propriété de l'élément n'a pas de valeur. Lorsque l'opérateur `IsNull` est utilisé, la valeur de la condition est ignorée et dans certains cas peut être nulle.
Less	Numérique
LessEqual	Numérique
Like	Chaîne La valeur de la condition est une sous-chaîne de la valeur de la propriété. La valeur de la condition doit être entourée de signes de pourcentage (%). Par exemple, `%Bi%` correspond à `Bismark` et à `Golfe de Biscaye`, mais pas à `biscuit`.
LikeIgnoreCase	Chaîne Utilisez l'opérateur `LikeIgnoreCase` de la même manière que l'opérateur `Like`. Toutefois, la correspondance n'est pas sensible à la casse. Par conséquent, `%Bi%` correspond à `biscuit`.
NotEqual	Chaîne et numérique
UnchangedDuring	Date Il s'agit d'un contrôle de plage. La valeur de la condition est exprimée en heures. Si la valeur de la propriété de date se situe dans la plage de temps dans laquelle la méthode est appelée plus ou moins la valeur de la condition, cette dernière est fausse. Si elle se situe en dehors de cette plage, la condition est vraie. Par exemple, si la valeur de la condition est `24`, la condition est vraie si la valeur de la propriété de date est antérieure à hier à cette heure ou postérieure à demain à cette heure. **Remarque :** Le nom `UnchangedDuring` est conservé pour préserver la compatibilité rétroactive. Dans les versions précédentes, l'opérateur était uniquement utilisé avec les propriétés de création et de modification d'heure.

Exemple de définition d'une condition :

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11f);
```

```
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.FloatOperator.EQUAL);
```

Exemple de recherche de propriétés héritées :

Le CI cible est `sample` et comporte deux attributs, `name` et `size`. `sampleII` complète le CI avec deux attributs, `level` et `grade`. Cet exemple définit une requête pour les propriétés de `sampleII` , héritées de `sample`, en les désignant par leur nom.

```
GetFilteredCIsByType request = new GetFilteredCIsByType()
request.setCmdbContext(cmdbContext)
request.setType("sampleII");
CustomProperties customProperties = new CustomProperties();
PropertiesList propertiesList = new PropertiesList();
propertiesList.setPropertyNames(Arrays.asList("name","size"));
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties);
```

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
type	Nom de classe. Pour plus d'informations, voir " Nom de type " , page 287. Le type peut être n'importe quel type défini à l'aide du Gestionnaire des types de CI. Pour plus d'informations, voir CI Type Manager dans le *Manuel de modélisation HP Universal CMDB*.
properties	Données à renvoyer sur chaque CI (appelées Mise en page dans l'interface utilisateur). Pour plus d'informations, voir " CustomProperties " , page 281.
conditions	Collection de paires nom-valeur avec les opérateurs qui les relient entre elles. Par exemple, `host_hostname like QA`.
conditionsLogicalOperator	• **AND.** Toutes les conditions doivent être satisfaites. • **OR**. Au moins l'une des conditions doit être satisfaite.

Sortie

Paramètre	Remarque
CIs	Collection d'éléments de configuration.
chunkInfo	Pour plus d'informations, voir " ChunkInfo " , page 289 et " Traitement de longues réponses " , page 279.

getQueryNameOfView

La méthode `getQueryNameOfView` extrait le nom du code TQL sur lequel la vue spécifiée est basée.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
viewName	Nom d'une vue, c'est-à-dire d'un sous-ensemble du modèle de classe dans le CMDB.

Sortie

Paramètre	Remarque
queryName	Nom du code TQL dans la base CMDB sur lequel la vue es basée.

getTopologyQueryExistingResultByName

La méthode `getTopologyQueryExistingResultByName` extrait le résultat le plus récent de l'exécution du code TQL spécifié. L'appel n'exécute pas le code TQL. S'il n'y a pas de résultats issus d'une exécution antérieure, aucun résultat n'est renvoyé.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
queryName	Nom d'un code TQL.
queryTypedProperties	Collection d'ensembles de propriétés à extraire pour des éléments d'un type d'élément de configuration particulier.

Sortie

Paramètre	Remarque
topologyMap	Pour plus d'informations, voir " TopologyMap " , page 288.
chunkInfo	Pour plus d'informations, voir " ChunkInfo " , page 289 et " Traitement de longues réponses " , page 279.

getTopologyQueryResultCountByName

La méthode `getTopologyQueryResultCountByName` extrait le nombre d'instances de chaque nœud qui correspond à la requête spécifiée.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
queryName	Nom d'un code TQL.
countInvisible	Si ce paramètre a la valeur true, la sortie inclut des CI définis comme invisibles dans la requête.

Sortie

Paramètre	Remarque
getTopologyQueryResultCountByNameResponse	Nombre d'instances correspondant à la requête.

pullTopologyMapChunks

La méthode `pullTopologyMapChunks` extrait l'un des segments qui contiennent la réponse à une méthode.

Chaque segment contient un élément `topologyMap` qui constitue une partie de la réponse. Le premier segment porte le numéro `1`, de sorte que le compteur de boucle d'extraction se répète de 1 à <*objet de réponse*>`.getChunkInfo().getNumberOfChunks()`.

Pour plus d'informations, voir " ChunkInfo " , page 289 et " Interrogation de CMDB " , page 278.

L'application cliente doit être capable de traiter les cartes partielles.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
ChunkRequest	Numéro du segment à extraire et `ChunkInfo` qui est renvoyé par la méthode de requête.
queryTypedProperties	Collection d'ensembles de propriétés à extraire pour les éléments d'un type de CI spécifique.

Sortie

Paramètre	Remarque
topologyMap	Pour plus d'informations, voir " TopologyMap " , page 288.
comments	À usage interne uniquement.

Exemple de traitement de segments :

```
GetCIsByType request =
     new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
     ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1; j<=response.getChunkInfo().getNumberOfChunks(); j++){
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req =new PullTopologyMapChunks(cmdbContext,chunkRe
quest);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().sizeCINodeList() ;
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINode(m).getCIs();
            for(int i=0 ; i < cis.sizeCIList() ; i++) {
                // your code to process the CIs
            }
        }
}
```

```
GetCIsByType request =
     new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
     ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1; j<=response.getChunkInfo().getNumberOfChunks(); j++){
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req = new PullTopologyMapChunks(cmdbContext, chunk
Request);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().getCINodes().size();
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINodes().get(m).getCIs();
            for(int i=0 ; i < cis.getCIs().size(); i++) {
                // your code to process the CIs
            }
        }
}
```

releaseChunks

La méthode `releaseChunks` libère la mémoire des segments qui contiennent les données de la requête.

> **Astuce :** Le serveur supprime les données au bout de dix minutes. L'appel de cette méthode pour supprimer les données dès qu'elles ont été lues permet de préserver les ressources du serveur.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
chunksKey	Identifiant des données sur le serveur qui ont été divisées en segments. La clé est un élément de `ChunkInfo`.

Méthodes de mise à jour UCMDB

Cette section fournit des informations sur les méthodes suivantes :

- " addCIsAndRelations " , ci-dessous
- " addCustomer " , page suivante
- " deleteCIsAndRelations " , page suivante
- " removeCustomer " , page 302
- " updateCIsAndRelations " , page 302

addCIsAndRelations

La méthode `addCIsAndRelations` ajoute ou met à jour des CI et des relations.

Si les CI ou relations n'existent pas dans la base CMDB, ils y sont ajoutés et leurs propriétés sont définies en fonction du contenu de l'argument `CIsAndRelationsUpdates`.

Si les CI ou relations existent dans la base CMDB, ils sont mis à jour avec les nouvelles données, si `updateExisting` a la valeur **true**.

Si `updateExisting` a la valeur **false**, `CIsAndRelationsUpdates` ne peut pas référencer des relations ou éléments de configuration existants. Toute tentative de référence à des éléments existants lorsque `updateExisting` a la valeur false entraîne une exception.

Si `updateExisting` a la valeur **true**, l'opération d'ajout ou de mise à jour est effectuée sans validation des CI, quelle que soit la valeur de `ignoreValidation`.

Si `updateExisting` a la valeur **false** et que `ignoreValidation` a la valeur **true**, l'opération d'ajout est effectuée sans validation des CI.

Si `updateExisting` a la valeur **false** et `ignoreValidation` a aussi la valeur **false**, les CI sont validés avant l'opération d'ajout.

Les relations ne sont jamais validées.

`CreatedIDsMap` est une carte ou un dictionnaire de type `ClientIDToCmdbID` qui connecte les ID temporaires du client avec les ID CMDB réels correspondants.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
updateExisting	Définir la valeur **true** pour mettre à jour les éléments qui existent déjà dans la base CMDB. Définir la valeur **false** pour lever une exception si un élément existe déjà.
CIsAndRelationsUpdates	Éléments à mettre à jour ou à créer. Pour plus d'informations, voir " CIsAndRelationsUpdates " , page 282.
ignoreValidation	Si la valeur est true, aucun contrôle n'est effectué avant la mise à jour de la base CMDB.
dataStore	Informations du modificateur.

Sortie

Paramètre	Remarque
createdIDsMapList	Liste des ID de client mappés sur des ID CMDB. Pour plus d'informations, voir la description plus haut.
comments	À usage interne uniquement.

addCustomer

La méthode `addCustomer` ajoute un client.

Entrée

Paramètre	Remarque
CustomerID	ID numérique du client.

deleteCIsAndRelations

La méthode `deleteCIsAndRelations` supprime les relations et éléments de configuration spécifiés de la base CMDB.

Lorsqu'un CI est supprimé alors qu'il constitue une extrémité d'une ou de plusieurs éléments `Relation`, ces éléments `Relation` sont également supprimés.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
CIsAndRelationsUpdates	Éléments à supprimer. Pour plus d'informations, voir " CIsAndRelationsUpdates " , page 282.
dataStore	Informations du modificateur.

removeCustomer

La méthode `removeCustomer` supprime un enregistrement de client.

Entrée

Paramètre	Remarque
CustomerID	ID numérique du client.

updateCIsAndRelations

La méthode `updateCIsAndRelations` met à jour les CI et relations spécifiés.

La mise à jour utilise les valeurs de propriété de l'argument `CIsAndRelationsUpdates`. Si certains des CI ou des relations n'existent pas dans la base CMDB, une exception est levée.

`CreatedIDsMap` est une carte ou un dictionnaire de type `ClientIDToCmdbID` qui connecte les ID temporaires du client avec les ID CMDB réels correspondants.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
CIsAndRelationsUpdates	Éléments à mettre à jour. Pour plus d'informations, voir " CIsAndRelationsUpdates " , page 282.
ignoreValidation	Si la valeur est true, aucun contrôle n'est effectué avant la mise à jour de la base CMDB.
dataStore	Informations du modificateur.

Sortie

Paramètre	Remarque
createdIDsMapList	Liste des ID de client mappés sur des ID CMDB. Pour plus d'informations, voir " addCIsAndRelations " , page 300.

Méthodes d'analyse d'impact UCMDB

Cette section fournit des informations sur les méthodes suivantes :

- " calculateImpact " , ci-dessous
- " getImpactPath " , ci-dessous
- " getImpactRulesByNamePrefix " , page suivante

calculateImpact

La méthode `calculateImpact` calcule quels CI sont affectés par un CI donné selon les règles définies dans la base CMDB.

Cela montre l'effet d'un déclenchement d'événement de la règle. La sortie `identifier` de `calculateImpact` est utilisée comme entrée de " getImpactPath " , ci-dessous.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
impactCategory	Type d'événement qui déclencherait la règle simulée.
IDs	Collection d'éléments ID.
impactRulesNames	Collection d'éléments `ImpactRuleName`.
severity	Gravité de l'événement déclencheur.

Sortie

Paramètre	Remarque
impactTopology	Pour plus d'informations, voir " Topology " , page 288.
identifier	Clé de la réponse du serveur.

getImpactPath

La méthode `getImpactPath` extrait le graphique topologique du chemin entre le CI affecté et le CI qui l'affecte.

La sortie `identifier` de " calculateImpact " , page précédente est utilisée comme argument d'entrée `identifier` de `getImpactPath`.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
identifier	Clé de la réponse du serveur qui a été renvoyée par calculateImpact.
relation	Relation basée sur l'une des entités " ShallowRelation " renvoyées par calculateImpact dans l'élément `impactTopology`.

Sortie

Paramètre	Remarque
impactPathTopology	Collection `CIs` et collection `ImpactRelations`.
comments	À usage interne uniquement.

Un élément `ImpactRelations` se compose d'un `ID`, d'un `type`, d'un ID d'extrémité 1 (`end1ID`), d'un ID d'extrémité 2 (`end2ID`), d'une règle (`rule`) et d'une `action`.

getImpactRulesByNamePrefix

La méthode `getImpactRulesByNamePrefix` extrait des règles à l'aide d'un filtre de préfixe.

Cette méthode s'applique aux règles d'impact dont le nom contient un préfixe indiquant le contexte auquel elles s'appliquent, par exemple, `SAP_myrule`, `ORA_myrule`, etc. Cette méthode filtre tous les noms de règle d'impact pour trouver ceux qui commencent par le préfixe spécifié par l'argument `ruleNamePrefixFilter`.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
ruleNamePrefixFilter	Chaîne contenant les premières lettres des noms de règle à rechercher.

Sortie

Paramètre	Remarque
impactRules	`impactRules` se compose de zéro `impactRule` ou plus. Une `impactRule`, qui indique l'effet d'une modification, se compose de `ruleName`, `description`, `queryName` et `isActive`.

API de service Web d'état réel

L'API de service Web d'état réel permet principalement à Service Manager de récupérer les informations d'état réel liées à un ID de CMDB ou à un ID global spécifique et à un ID d'abonné spécifique. L'API recherche une requête correspondante dans le dossier **Integration/SM Query** et exécute le code TQL associé à l'ID de CMDB ou à l'ID global sous la forme d'une condition ; elle renvoie ensuite le résultat de la requête.

URL du service Web : http://[nom_machine]:8080/axis2/services/ucmdbSMService

Schéma du service Web : http://[nom_machine]:8080/axis2/services/ucmdbSMService?xsd=xsd0

Flux

Lorsque la méthode API est appelée, elle tente de rechercher une requête appropriée dans le dossier **Integration/SM Query**. Elle essaie d'établir une correspondance entre le type de l'identifiant CMDBID/GlobalID demandé et une des requêtes contenues dans le dernier dossier. Elle commence par rechercher un élément de requête **QueryElement** nommé **Root**. Si la recherche est positive, elle utilise ensuite un nœud de requête **QueryNode** du même type que le CMDBID/GlobalID demandé. Une fois que la requête et le nœud de requête appropriés ont été trouvés, la méthode place CMDBID/GlobalID comme une condition sur le nœud de requête, puis exécute cette dernière. Le résultat est ensuite renvoyé à l'appelant de l'API.

Manipulation du résultat à l'aide de transformations

Il peut arriver de vouloir appliquer des transformations supplémentaires au code XML résultant (par exemple, pour additionner les tailles de l'ensemble des disques et ajouter le total trouvé comme attribut supplémentaire au CI). Pour ajouter des transformations supplémentaires sur le résultat d'une requête TQL, il convient de placer une ressource **[tql_name].xslt** dans la configuration de l'adaptateur comme suit : **Gestion de l'adaptateur > ServiceDeskAdapter7-1 >Fichiers de configuration > [tql_name].xslt.**

Journaux associés à l'API du service Web d'état réel

La configuration des journaux d'UCMDB est définie dans les différents fichiers ***.properties** du dossier **UCMDBServer/Conf/log**.

Pour consulter les journaux du flux d'état réel de SM :

1. Ouvrez le fichier **cmdb_soaapi.properties** et modifiez le niveau de journalisation sur DEBUG comme suit : **loglevel=DEBUG**.

2. Ouvrez le fichier **fcmdb.properties** et modifiez le niveau de journalisation sur DEBUG comme suit : **loglevel=DEBUG**.

3. Patientez une minute, le temps pour le serveur de récupérer les modifications.

4. Exécutez l'état réel à partir de SM.

5. Affichez les fichiers journaux suivants situés dans le dossier **UCMDBServer/Runtime/log** :

- cmdb.soaapi.log
- fcmdb.log

Activation de l'état réel des CI répliqués après la modification du contexte racine

Si vous avez modifié le contexte racine qui est utilisé pour accéder à UCMDB, vous devez apporter les modifications de configuration suivantes afin d'activer l'état réel des CI répliqués :

1. Sous **UCMDBServer\deploy\axis2\WEB-INF**, ouvrez le fichier **web.xml**.
2. Ajoutez le paramètre **servlet init** suivant à AxisServlet (collez ces quatre lignes après la ligne 28) :

   ```
   <init-param>
   <param-name>axis2.find.context</param-name>
   <param-value>false</param-value>
   </init-param>
   ```

   Ce paramètre empêche Axis2 de calculer la racine du contexte et lui demande de la rechercher explicitement dans **axis2.xml**.
3. Sous **UCMDBServer\deploy\axis2\WEB-INF\conf**, ouvrez le fichier **axis2.xml**.
4. À la ligne 58, supprimez les commentaires du paramètre **contextRoot** et modifiez ce dernier comme suit :

   ```
   <parameter name="contextRoot" locked="false">test/axis2</parameter>
   ```

   (où **test** est le nouveau contexte racine dans le fichier **cmdb.xml**).

   > **Remarque :** Il n'y a pas de barre oblique au début de **test/axis2**.

Cas d'utilisation de l'API de service Web UCMDB

Les cas d'utilisation suivants nécessitent deux systèmes :

- Serveur HP Universal CMDB
- Système tiers contenant un référentiel des éléments de configuration.

Contenu de cette section :

- " Remplissage du CMDB " , page suivante
- " Interrogation du CMDB " , page suivante

- " Interrogation du modèle de classe " , ci-dessous
- " Analyse de l'impact des modifications " , ci-dessous

Remplissage du CMDB

Cas d'utilisation :

- Un outil tiers de gestion des actifs met à jour CMDB avec des informations disponibles uniquement dans la gestion des actifs.
- Plusieurs systèmes tiers alimentent la base CMDB afin de créer une base CMDB centrale pour le suivi des modifications et la mise en œuvre d'analyses d'impact.
- Un système tiers crée des éléments de configuration et des relations conformes à la logique métier tierce, afin d'exploiter les fonctionnalités de requête de CMDB.

Interrogation du CMDB

Cas d'utilisation :

- Un système tiers obtient les éléments de configuration et les relations qui représentent le système SAP en extrayant les résultats du langage TQL SAP.
- Un système tiers obtient la liste des serveurs Oracle qui ont été ajoutés ou modifiés au cours des cinq dernières heures.
- Un système tiers obtient la liste des serveurs dont le nom d'hôte contient la sous-chaîne *lab*.
- Un système tiers trouve les éléments liés à un CI donné en extrayant ses voisins.

Interrogation du modèle de classe

Cas d'utilisation :

- Un système tiers permet aux utilisateurs de définir l'ensemble de données à extraire de CMDB. Une interface utilisateur peut être créée sur le modèle de classe pour montrer aux utilisateurs les propriétés possibles et leur demander les données requises. L'utilisateur peut alors choisir les informations à extraire.
- Un système tiers explore le modèle de classe lorsque l'utilisateur ne peut pas accéder à l'interface utilisateur UCMDB.

Analyse de l'impact des modifications

Cas d'utilisation :

Un système tiers sort une liste des services métier sur lesquels une modification survenue sur un hôte désigné est susceptible d'avoir une incidence.

Exemples

Reportez-vous aux exemples de code suivants :

- The Example Base Class
- Query Example
- Update Example
- Class Model Example
- Impact Analysis Example

Les fichiers se trouvent dans le répertoire

\\<UCMDB root directory>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\WebServiceAPI_Samples

Chapitre 11 : API Java de gestion des flux de données

Contenu de ce chapitre :

Utilisation de l'API Java de gestion des flux de données 309

Utilisation de l'API Java de gestion des flux de données

Remarque : Utilisez ce chapitre en parallèle avec le Javadoc de l'API de gestion des flux de données, disponible dans la bibliothèque de la documentation en ligne.

Ce chapitre explique comment les outils tiers ou personnalisés exploitent l'API Java de gestion des flux de données HP pour administrer la gestion des flux de données. L'API fournit les méthodes nécessaires pour effectuer les opérations suivantes :

- **gérer les informations d'identification** : afficher, ajouter, mettre à jour et supprimer ;
- **gérer les travaux** : afficher le statut, activer et désactiver ;
- **gérer des plages de sonde** : afficher, ajouter et mettre à jour ;
- **gérer les déclencheurs** : ajouter ou supprimer un CI déclencheur, et ajouter, supprimer ou désactiver un TQL déclencheur ;
- **afficher les données générales** : données sur les domaines et les sondes.

Les services suivants sont disponibles dans le package des services de découverte :

- **DDMConfigurationService.** Services permettant de configurer les instances de Data Flow Probe, les clusters, les plages IP et les informations d'identification. Le serveur Universal Discovery peut être configuré avec un fichier XML ou via Data Flow Probe.
- **DDMManagementService.** Services permettant d'analyser et d'afficher l'avancement, les résultats et les erreurs de l'exécution de Universal Discovery.
- **DDMSoftwareSignatureService.** Services permettant de définir les éléments logiciels que doivent découvrir les composants Data Flow Probe. Les définitions s'appliquent à l'ensemble du système. Si plusieurs composants Data Flow Probe sont définis, les définitions s'appliquent à tous.
- **DDMZoneService.** Services permettant de gérer la découverte basée sur une zone de gestion.

En plus de ces services, les API clientes de la gestion des flux de données permettent de créer des adaptateurs Jython. Pour plus d'informations, voir " Développement d'adaptateurs Jython " , page 39.

Autorisations

L'administrateur fournit des informations d'identification pour la connexion avec l'API. Le client de l'API a besoin du nom et du mot de passe d'un utilisateur d'intégration défini dans CMDB. Ces utilisateurs ne représentent pas des utilisateurs humains de CMDB, mais des applications qui se connectent à CMDB.

De plus, l'utilisateur doit disposer d'une autorisation pour l'action générale **Accéder à SDK** pour se connecter.

> **Attention :** Le client de l'API peut également fonctionner avec des utilisateurs réguliers dès lors qu'ils disposent d'autorisations d'authentification API. Toutefois, cette option est déconseillée.

Pour plus d'informations, voir " Création d'un utilisateur d'intégration " , page 269.

Chapitre 12 : API de service Web de la gestion des flux de données

Contenu de ce chapitre :

API de service Web de la gestion des flux de données - Présentation 311
Conventions 312
Appel du service Web 312
Structures de données et méthodes de gestion des flux de données 312
Exemple de code 323
Exemple d'ajout d'informations d'identification 326

API de service Web de la gestion des flux de données - Présentation

Ce chapitre explique comment des outils tiers ou personnalisés peuvent exploiter l'API de service Web de la gestion des flux de données HP pour gérer les flux de données.

L'API de service Web de la gestion des flux de données HP permet d'intégrer des applications dans HP Universal CMDB. Elle fournit des méthodes pour effectuer les opérations suivantes :

- **gérer les informations d'identification** : afficher, ajouter, mettre à jour et supprimer ;
- **gérer les travaux** : afficher le statut, activer et désactiver ;
- **gérer des plages de sonde** : afficher, ajouter et mettre à jour ;
- **gérer les déclencheurs** : ajouter ou supprimer un CI déclencheur, et ajouter, supprimer ou désactiver un TQL déclencheur ;
- **afficher les données générales** : données sur les domaines et les sondes.

Les utilisateurs du service Web de la gestion des flux de données HP doivent maîtriser les éléments suivants :

- la spécification SOAP ;
- un langage de programmation orientée objet, tel que C++, C# ou Java ;
- HP Universal CMDB ;
- la gestion des flux de données.

L'utilisateur doit disposer d'une autorisation pour l'action générale **Exécuter l'API héritée** pour se connecter.

Pour consulter la documentation complète sur les opérations disponibles, voir *HP Universal Discovery Schema Reference*. Ces fichiers figurent dans le dossier suivant :

<répertoire racine de UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

Conventions

Ce chapitre observe les conventions suivantes :

- Le style `Elément` signale qu'un élément est une entité de la base de données ou un élément défini dans le schéma, y compris les structures transmises ou renvoyées par des méthodes. Le texte ordinaire indique que l'élément abordé est traité dans un contexte général.
- Les éléments et arguments de méthode de la gestion des flux de données sont cités en toutes lettres lorsqu'ils sont spécifiés dans le schéma. Cela implique généralement qu'un nom de classe ou une référence générique à une instance de la classe apparaît en majuscules. Un élément ou un argument d'une méthode ne porte pas de majuscule. Par exemple, `credential` est un élément de type `Credential` transmis à une méthode.

Appel du service Web

L'API de service Web de la gestion des flux de données HP permet d'appeler des méthodes côté serveur à l'aide des techniques de programmation SOAP standard. S'il est impossible d'analyser l'instruction ou si un incident se produit lors de l'appel de la méthode, les méthodes API lèvent une exception `SoapFault`. Lorsqu'une exception `SoapFault` est levée, le service complète un ou plusieurs des champs de message d'erreur, de code d'erreur et de message d'exception. En l'absence d'erreur, les résultats de l'appel sont renvoyés.

Pour appeler le service, utilisez les éléments suivants :

- `Protocole : http ou https (selon la configuration du serveur)`
- `URL : <Serveur UCMDB>:8080/axis2/services/DiscoveryService`
- `Mot de passe par défaut : "admin"`
- `Nom d'utilisateur par défaut : "admin"`

Les programmateurs SOAP peuvent accéder au WSDL à l'adresse :

- `axis2/services/DiscoveryService?wsdl`

Structures de données et méthodes de gestion des flux de données

Cette section répertorie les structures de données et les méthodes de l'API de service Web de la gestion des flux de données, avec un bref aperçu de leur utilisation. Pour consulter la documentation complète sur la demande et la réponse de chaque opération, voir *HP Universal Discovery Schema Reference*.

Contenu de cette section :

- " Structures de données " , ci-dessous
- " Méthodes de gestion des travaux de découverte " , ci-dessous
- " Méthodes de gestion de déclencheurs " , page 315
- " Méthodes de données de domaine et de sonde " , page 317
- " Méthodes de données d'informations d'identification " , page 319
- " Méthodes d'actualisation de données " , page 321

Structures de données

L'API du service Web de la gestion des flux de données utilise certaines structures de données.

CIProperties

`CIProperties` est un ensemble de collections. Chacune d'elles contient des propriétés d'un type de données différent. Par exemple, les collections peuvent être de type `dateProps`, `strListProps` ou encore `xmlProps`, etc.

Chaque collection contient les propriétés individuelles du type donné. Les noms de ces éléments de propriétés reprennent ceux des conteneurs, mais au singulier. C'est ainsi que le conteneur `dateProps` contient des éléments `dateProp`. Chaque propriété constitue une paire nom-valeur.

Voir CIProperties dans *HP Universal Discovery Schema Reference*.

IPList

Liste d'éléments `IP`, chacun d'eux contenant une adresse IPv4 ou IPv6.

Voir IPList dans *HP Universal Discovery Schema Reference*.

IPRange

`IPRange` comprend deux éléments, `Start` (début) et `End` (fin). Chacun d'eux contient un élément `Address` qui correspond à une adresse IPv4 ou IPv6.

Voir IPRange dans *HP Universal Discovery Schema Reference*.

Étendue

Deux structures `IPRange`. `Exclude` est une collection de structures `IPRange` à exclure du travail. `Include` est une collection de structures `IPRange` à inclure dans le travail.

Voir Scope dans *HP Universal Discovery Schema Reference*.

Méthodes de gestion des travaux de découverte

activateJob

Active le travail spécifié.

Voir " Exemple de code " , page 323.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail.

deactivateJob

Désactive le travail spécifié.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail.

dispatchAdHocJob

Envoie un travail sur la sonde ad hoc. Le travail doit être actif et contenir le CI déclencheur spécifié.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail.
CIID	ID du CI déclencheur.
ProbeName	Nom de la sonde.
Dépassement de délai	En millisecondes

getDiscoveryJobsNames

Renvoie la liste des noms de travaux.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.

Sortie

Paramètre	Remarque
strList	Liste des noms de travaux.

isJobActive

Vérifie si le travail est actif.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail à vérifier.

Sortie

Paramètre	Remarque
JobState	La valeur est "true" si le travail est actif.

Méthodes de gestion de déclencheurs

addTriggerCI

Ajoute un nouveau CI déclencheur au travail spécifié.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail.
CIID	ID du CI déclencheur.

addTriggerTQL

Ajoute un nouveau code TQL déclencheur au travail spécifié.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail.
TqlName	Nom du TQL à ajouter.

disableTriggerTQL

Empêche le code TQL de déclencher le travail, mais ne le supprime pas de façon permanente de la liste des requêtes qui déclenchent le travail.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail.

removeTriggerCI

Supprime le CI spécifié de la liste des CI qui déclenchent le travail.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail.
CIID	ID du CI déclencheur.

removeTriggerTQL

Supprime le code TQL spécifié de la liste des requêtes qui déclenchent le travail.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Ensemble de noms de travail à vérifier.
CIID	ID du TQL à supprimer.

setTriggerTQLProbesLimit

Restreint à la liste spécifiée les sondes dans lesquelles le code TQL est actif dans le travail.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
JobName	Nom du travail.
tqlName	Nom du TQL.
probesLimit	Liste des sondes pour lesquelles le TQL est actif.

Méthodes de données de domaine et de sonde

getDomainType

Renvoie le type de domaine.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Nom du domaine.

Sortie

Paramètre	Remarque
domainType	Type de domaine.

getDomainsNames

Renvoie les noms des domaines actuels.

Voir " Exemple de code " , page 323.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.

Sortie

Paramètre	Remarque
domainNames	Liste des noms de domaine.

getProbeIPs

Renvoie les adresses IP de la sonde spécifiée.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Le domaine à vérifier.
probeName	Le nom de la sonde utilisée dans ce domaine.

Sortie

Paramètre	Remarque
probeIPs	Liste " IPList " des adresses dans la sonde.

getProbesNames

Renvoie les noms des sondes dans le domaine spécifié.

Voir " Exemple de code " , page 323.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Le domaine à vérifier.

Sortie

Paramètre	Remarque
probesName	La liste des sondes dans le domaine.

getProbeScope

Renvoie la définition d'étendue de la sonde spécifiée.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Le domaine à vérifier.
probeName	Nom de la sonde.

Sortie

Paramètre	Remarque
probeScope	" Étendue " de la sonde.

isProbeConnected

Vérifie si la sonde spécifiée est connectée.

Voir " Exemple de code " , page 323.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Le domaine à vérifier.
probeName	La sonde à vérifier.

Sortie

Paramètre	Remarque
isConnected	A la valeur "true" si la sonde est connectée.

updateProbeScope

Définit l'étendue de la sonde spécifiée, en remplacement de la sonde existante.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Domaine.
probeName	Sonde à mettre à jour.
newScope	" Étendue " à définir pour la sonde.

Méthodes de données d'informations d'identification

addCredentialsEntry

Ajoute une entrée d'informations d'identification au protocole spécifié pour le domaine spécifié.

Voir " Exemple de code " , page 323.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Domaine à mettre à jour.
protocolName	Nom du protocole.
credentialsEntryParameters	Collection " CIProperties " des nouvelles informations d'identification.

Sortie

Paramètre	Remarque
credentialsEntryID	ID de CI de la nouvelle entrée réservée aux informations d'identification.

getCredentialsEntriesIDs

Renvoie les ID des informations d'identification définies pour le protocole spécifié.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Domaine pour lequel les informations d'identification doivent être récupérées.
protocolName	Nom d'un protocole utilisé dans ce domaine.

Sortie

Paramètre	Remarque
credentialsEntryIDs	Liste d'ID d'informations d'identification associées au protocole du domaine.

getCredentialsEntry

Renvoie les informations d'identification définies pour le protocole spécifié. Les attributs chiffrés sont renvoyés vides.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Domaine pour lequel les informations d'identification doivent être récupérées.
protocolName	Nom d'un protocole utilisé dans ce domaine.
credentialsEntryID	ID d'informations d'identification à obtenir.

Sortie

Paramètre	Remarque
credentialsEntryParameters	Collection " CIProperties " des informations d'identification.

removeCredentialsEntry

Supprime les informations d'identification spécifiées du protocole.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Domaine.
protocolName	Nom d'un protocole utilisé dans le domaine.
credentialsEntryID	ID des informations d'identification à supprimer.

updateCredentialsEntry

Définit de nouvelles valeurs pour les propriétés de l'entrée des informations d'identification spécifiées.

Les propriétés existantes sont supprimées tandis que ces propriétés sont définies. Toute propriété dont la valeur n'est pas définie dans cet appel demeure indéfinie.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
domainName	Domaine dans lequel les informations d'identification doivent être mises à jour.
protocolName	Nom d'un protocole utilisé dans le domaine.
credentialsEntryID	ID des informations d'identification à mettre à jour.
credentialsEntryParameters	Collection " CIProperties " à définir comme propriétés pour les informations d'identification.

Méthodes d'actualisation de données

rediscoverCIs

Recherche les déclencheurs ayant découvert les objets CI spécifiés et réexécute ces déclencheurs. **rediscoverCIs** s'exécute en mode asynchrone. Appelez **checkDiscoveryProgress** pour déterminer quand la redécouverte est terminée.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
CmdbIDs	Collection d'ID des objets à redécouvrir.

Sortie

Paramètre	Remarque
isSucceed	La valeur est "true" si la redécouverte des CI a abouti.

checkDiscoveryProgress

Renvoie la progression du dernier appel **rediscoverCIs** sur les ID spécifiés. La réponse est une valeur comprise entre 0 et -1. Lorsque la réponse est 1, l'appel **rediscoverCIs** est terminé.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
CmdbIDs	Collection d'ID des objets contenus dans l'appel de redécouverte à suivre.

Sortie

Paramètre	Remarque
progress	Lorsqu'un travail est terminé, sa progression s'élève à 1. Lorsqu'il n'est pas terminé, la progression indiquée est une fraction inférieure à 1.

rediscoverViewCIs

Recherche les déclencheurs ayant créé les données pour alimenter la vue spécifiée et les réexécute. **rediscoverViewCIs** s'exécute en mode asynchrone. Appelez **checkViewDiscoveryProgress** pour déterminer quand la redécouverte est terminée.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
viewName	Vues à vérifier.

Sortie

Paramètre	Remarque
isSucceed	La valeur est "true" si la redécouverte des CI a abouti.

checkViewDiscoveryProgress

Renvoie la progression du dernier appel **rediscoverViewCIs** sur la vue spécifiée. La réponse est une valeur comprise entre 0 et 1. Lorsque la réponse est 1, l'appel **rediscoverCIs** est terminé.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir " CmdbContext " , page 285.
viewName	Collection de vues à vérifier.

Sortie

Paramètre	Remarque
progress	Lorsqu'un travail est terminé, sa progression s'élève à 1. Lorsqu'il n'est pas terminé, la progression indiquée est une fraction inférieure à 1.

Exemple de code

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.*;
import com.hp.ucmdb.generated.types.*;
public class test {
    static final String HOST_NAME = "<my_hostname>";
    static final int PORT = 8080;
    private static final String PROTOCOL = "http";
    private static final String FILE = "/axis2/services/DiscoveryService";

    private static final String PASSWORD = "<my_password>";
    private static final String USERNAME = "<my_username>";

    private static CmdbContext cmdbContext = new CmdbContext("ws tests");


    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest(
            "Range IPs by ICMP", cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protcol
        addNTCMDCredentialsEntry();
    }
```

```
    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(
                new GetDomainsNamesRequest(cmdbContext)).
                getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credential
s");
            return;
        }
        String domainName = domains.getStrValue(0);
        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user", newCredsPropertie
s);
        setStringProperties("ntadminprotocol_ntdomain",
            "test doamin", newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(
            new AddCredentialsEntryRequest(domainName,
            "ntadminprotocol",  newCredsProperties, cmdbContext));
        System.out.println("new credentials craeted for domain: " + domainName +
" in ntcmd protocol");
    }


    private static void setPasswordProperty(CIProperties newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }


    private static void setStringProperties(String propertyName, String value, C
IProperties newCredsProperties) {
        StrProp strProp = new StrProp();
```

```
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }



    private static void getProbesInfo(DiscoveryService discoveryService) throws
Exception {
        GetDomainsNamesResponse result = discoveryService.getDomainsNames(new Ge
tDomainsNamesRequest(cmdbContext ));
        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName =
            result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(
                    new GetProbesNamesRequest(domainName, cmdbContext));
            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames().sizeStrValueList(); i+
+) {
                String probeName = probesResult.getProbesNames().getStrValue(i);
                // Check if connected
                IsProbeConnectedResponce connectedRequest =
                    discoveryService.isProbeConnected(
                        new IsProbeConnectedRequest(
                        domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected();
                // Do something ...
                System.out.println("probe " + probeName + " isconnect=" + isConn
ected);
            }
        }
    }


    private static DiscoveryService getDiscoveryService() throws Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub =
                new DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth =
                new HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
```

```
            serviceStub._getServiceClient().getOptions().setProperty(
                HTTPConstants.AUTHENTICATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }
        return discoveryService;
    }
}
```

Exemple d'ajout d'informations d'identification

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.DiscoveryService;
import com.hp.ucmdb.generated.services.DiscoveryServiceStub;
import com.hp.ucmdb.generated.types.BytesProp;
import com.hp.ucmdb.generated.types.BytesProps;
import com.hp.ucmdb.generated.types.CIProperties;
import com.hp.ucmdb.generated.types.CmdbContext;
import com.hp.ucmdb.generated.types.StrList;
import com.hp.ucmdb.generated.types.StrProp;
import com.hp.ucmdb.generated.types.StrProps;

public class test {
    static final String HOST_NAME = "hostname";
    static final int PORT = 8080;
    private static final String PROTOCOL = "http";
    private static final String FILE = "/axis2/services/DiscoveryService";

    private static final String PASSWORD = "admin";
    private static final String USERNAME = "admin";

    private static CmdbContext cmdbContext = new CmdbContext("ws tests");

    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest("Range IPs by ICMP",
cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protcol
```

```
        addNTCMDCredentialsEntry();
    }

    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(new GetDomainsNamesRequest(cmdbCont
ext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credential
s");
            return;
        }
        String domainName = domains.getStrValue(0);
        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user", newCredsPropertie
s);
        setStringProperties("ntadminprotocol_ntdomain", "test doamin", newCredsP
roperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(new AddCredentialsEntryRequest(doma
inName,  "ntadminprotocol",  newCredsProperties, cmdbContext));
        System.out.println("new credentials craeted for domain: " + domainName +
" in ntcmd protocol");
    }

    private static void setPasswordProperty(CIProperties newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }

    private static void setStringProperties(String propertyName, String value, C
IProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
```

```
        newCredsProperties.getStrProps().addStrProp(strProp);
    }

    private static void getProbesInfo(DiscoveryService discoveryService) throws
Exception {
        GetDomainsNamesResponse result = discoveryService.getDomainsNames(new Ge
tDomainsNamesRequest(cmdbContext ));
        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName = result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(new GetProbesNamesRequest(domain
Name, cmdbContext));
            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames().sizeStrValueList(); i+
+) {
                String probeName = probesResult.getProbesNames().getStrValue(i);
                // Check if connected
                IsProbeConnectedResponce connectedRequest =
                    discoveryService.isProbeConnected(new IsProbeConnectedReques
t(domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected();
                // Do something ...
                System.out.println("probe " + probeName + " isconnect=" + isConn
ected);
            }
        }
    }

    private static DiscoveryService getDiscoveryService() throws Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub = new DiscoveryServiceStub(url.toSt
ring());

            // Authenticate info
            HttpTransportProperties.Authenticator auth = new HttpTransportProper
ties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty(HTTPConstan
ts.AUTHENTICATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }
        return discoveryService;
```

```
        }
    } // End class
```

Vos commentaires sont toujours les bienvenus.

Pour soumettre vos commentaires relatifs à ce document, vous pouvez contacter l'équipe de documentation par e-mail. Si un client de messagerie est configuré sur ce système, cliquez sur le lien ci-dessus pour accéder à une fenêtre contenant le libellé suivant sur la ligne Objet :

Commentaires sur Manuel de référence du développeur (Universal CMDB 10.10)

Il vous suffit ensuite d'ajouter vos commentaires et de cliquer sur Envoyer.

Si aucun client de messagerie n'est disponible, copiez le libellé ci-dessus dans une fenêtre d'un client de messagerie Web et envoyez votre message de commentaires à SW-Doc@hp.com.