HP Universal CMDB

pour systèmes d'exploitation Windows et Linux

Version du logiciel : 9.02

Manuel de référence du développeur

Date de publication de la documentation : Octobre 2010
Date de lancement du logiciel : Octobre 2010

Mentions légales

Garantie

Les seules garanties relatives aux produits et services HP sont exposées dans les déclarations de garantie expresses accompagnant lesdits produits et services. Le présent avis ne constitue en aucun cas une garantie supplémentaire. HP ne saurait être tenu pour responsable des erreurs ou omissions techniques ou rédactionnelles que pourrait comporter ce document.

Les informations contenues dans ce manuel pourront faire l'objet de modifications sans préavis.

Droits limités

Logiciel confidentiel. L'acquisition, l'utilisation et la copie en sont interdites sans une licence valide délivrée par HP. Conformément aux directives FAR 12.211 et 12.212, les droits s'appliquant aux logiciels commerciaux, à la documentation informatique et aux informations techniques des articles commerciaux concédés aux Gouvernement des États-Unis sont ceux concédés habituellement par une licence commerciale standard.

Copyright

Marques

Adobe® et Acrobat® sont des marques d'Adobe Systems Incorporated.

AMD et le symbole AMD Arrow sont des marques d'Advanced Micro Devices, Inc.

Google™ et Google Maps™ sont des marques de Google Inc.

Intel®, Itanium®, Pentium® et Intel® Xeon® sont des marques d'Intel Corporation aux États-Unis et dans les autres pays.

Java™ est une marque de Sun Microsystems, Inc. aux États-Unis.

Microsoft®, Windows®, Windows NT®, Windows® XP et Windows Vista® sont des marques déposées aux États-Unis de Microsoft Corporation.

Oracle est une marque déposée d'Oracle Corporation et/ou de ses filiales.

UNIX® est une marque déposée de The Open Group.

Crédits

- Ce produit inclut un logiciel développé par Apache Software Foundation (http://www.apache.org/licenses).
- Ce produit contient du code OpenLDAP d'OpenLDAP Foundation (http://www.openldap.org/foundation/).
- Ce produit contient du code GNU de Free Software Foundation, Inc. (http://www.fsf.org/).
- Ce produit contient du code JiBX de Dennis M. Sosnoski.
- Ce produit inclut dans la distribution l'analyseur XPP3 XMLPull qui est utilisé via JiBX d'Extreme! Lab, Indiana University.
- Ce produit inclut la licence Office Look and Feels de Robert Futrell (http://sourceforge.net/projects/officelnfs).
- Ce produit contient du code JEP (Java Expression Parser) de Netaphor Software, Inc. (http://www.netaphor.com/home.asp).

Mises à jour de la documentation

La page de titre de ce document contient les informations d'identification suivantes :

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

Pour rechercher les dernières mises à jour ou vérifier que vous utilisez l'édition la plus récente d'un document, visitez le site Web :

http://h20230.www2.hp.com/selfsolve/manuals

L'accès à ce site requiert la création d'un compte HP Passport. Pour obtenir un identifiant HP Passport, accédez à la page :

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

Vous pouvez également cliquer sur le lien d'**inscription des nouveaux utilisateurs** disponible dans la page de connexion de HP Passport.

Vous recevrez également les nouvelles éditions ou les mises à jour si vous vous abonnez au service d'assistance du produit approprié. Pour plus d'informations, contactez votre représentant commercial HP.

Assistance

Visitez le site Web d'assistance HP Software à l'adresse :

http://www.hp.com/go/hpsoftwaresupport

Ce site Web fournit des informations sur les contacts, les produits, les services et l'assistance HP Software.

L'assistance en ligne HP Software fournit aux clients des fonctions de résolution des problèmes. Elle offre un moyen rapide et efficace d'accéder aux outils interactifs d'assistance technique nécessaires à la gestion de votre activité. En tant que client bénéficiant de l'assistance HP, vous pouvez effectuer les opérations suivantes :

- effectuer des recherches dans les documents qui vous intéressent ;
- soumettre des incidents et suivre leur résolution ou demander des améliorations ;
- télécharger des correctifs logiciels ;
- gérer vos contrats d'assistance ;
- rechercher des contrats d'assistance HP ;
- consulter des informations sur les services disponibles ;
- participer à des discussions avec d'autres utilisateurs du logiciel ;
- rechercher des formations et vous y inscrire.

La plupart des domaines d'assistance nécessitent une inscription en tant qu'utilisateur HP Passport et, le cas échéant, un contrat d'assistance. Pour obtenir un identifiant HP Passport, accédez à la page :

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

Pour plus d'informations sur les niveaux d'accès, accédez à la page :

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

Table des matières

Bienvenue dans ce manuel

Ce manuel explique comment créer et gérer des adaptateurs permettant d'envoyer et de recevoir des données à partir de référentiels de données externes et d'autres bases CMDB.

Contenu de ce chapitre :

Structure du manuel

Le présent document comprend les parties suivantes :

Partie I Création d'adaptateurs de découverte et d'intégration

Explique comment créer des adaptateurs.

Partie II Utilisation d'API

Explique comment utiliser les API pour extraire des données de configuration de HP Universal CMDB.

À qui s'adresse ce manuel

Ce manuel s'adresse aux utilisateurs suivants de HP Universal CMDB :

- Administrateurs de HP Universal CMDB
- Administrateurs de plates-formes HP Universal CMDB
- Administrateurs d'applications HP Universal CMDB
- Administrateurs de gestion de données HP Universal CMDB

Ces utilisateurs doivent déjà posséder des connaissances sur l'administration des systèmes d'entreprise, les concepts ITIL et le module HP Universal CMDB.

Documentation en ligne de HP Universal CMDB

HP Universal CMDB inclut la documentation en ligne suivante :

Lisez-moi. Répertorie les limites de la version et les mises à jour de dernière minute. Dans le répertoire racine de HP Universal CMDB, double-cliquez sur **readme.html**. Vous pouvez également accéder au fichier Lisez-moi actualisé à partir du site Web Assistance HP Software.

Nouveautés. Répertorie les nouvelles fonctions et les points forts de la version. Dans HP Universal CMDB, sélectionnez **Aide** > **Nouveautés**.

Documentation imprimable. Sélectionnez **Aide** > **Aide UCMDB**. Les manuels suivants sont publiés au format PDF uniquement :

- *Manuel de déploiement HP Universal CMDB* PDF. Décrit la configuration matérielle et logicielle requise pour installer HP Universal CMDB et explique comment installer ou mettre à niveau HP Universal CMDB, renforcer le système et se connecter à l'application.
- *Manuel de base de données HP Universal CMDB* PDF. Décrit la configuration de la base de données (MS SQL Server ou Oracle) nécessaire à HP Universal CMDB.

- *HP Universal CMDB Discovery and Integration Content Guide* PDF. Explique comment exécuter le module de découverte pour détecter les applications, systèmes d'exploitation et composants réseau qui sont exécutés sur votre système. Explique également comment détecter des données dans d'autres référentiels de données via une intégration.

L'**aide en ligne de HP Universal CMDB** comprend les sections suivantes :

- **Modélisation**. Permet de gérer le contenu du modèle Univers IT.
- **gestion des flux de données**. Explique comment intégrer HP Universal CMDB dans d'autres référentiels de données et configurer HP Universal CMDB pour détecter les composants réseau.
- **Administration UCMDB**. Explique comment utiliser HP Universal CMDB.
- **Références pour développeurs**. S'adresse aux utilisateurs qui connaissent bien HP Universal CMDB. Explique comment définir et utiliser des adaptateurs et accéder aux données à l'aide d'API.

L'aide en ligne est également disponible à partir de certaines fenêtres de HP Universal CMDB en cliquant sur la fenêtre puis sur le bouton **Aide**.

Les manuels en ligne peuvent être affichés et imprimés à l'aide d'Adobe Reader, téléchargeable à partir du site Web d'Adobe (www.adobe.com).

Types de rubrique

Dans ce manuel, chaque domaine est organisé en rubriques. Une rubrique contient un module d'informations par sujet. Les rubriques sont généralement classées selon le type d'information qu'elles contiennent.

Cette structure a été conçue pour faciliter l'accès à des informations spécifiques, en divisant la documentation selon les différents types d'information dont vous pourriez avoir besoin à différents moments.

Trois types de rubrique principaux sont utilisés : **Concepts**, **Tâches** et **Références**. Des icônes permettent de distinguer visuellement ces types de rubrique.

Type de rubrique	Description	Utilisation
Concepts 	Informations générales, descriptives ou conceptuelles.	Obtenir des informations générales sur une fonctionnalité.
Tâches 	**Tâches pas à pas.** Ces tâches vous aident à utiliser l'application et à réaliser vos objectifs. Certaines tâches comprennent des exemples de données. Les étapes des tâches peuvent être numérotées ou non : ➤ **Étapes numérotées.** Les tâches sont exécutées en suivant chaque étape par ordre consécutif. ➤ **Étapes non numérotées.** Liste d'opérations autonomes que vous pouvez effectuer dans n'importe quel ordre.	➤ Obtenir des informations sur le déroulement global d'une tâche. ➤ Suivre les étapes numérotées d'une tâche pour exécuter celle-ci. ➤ Exécuter les opérations indépendantes correspondant aux étapes non numérotées d'une tâche.
	Tâches de scénario de cas d'utilisation. Exemples d'exécution d'une tâche propre à une situation.	Obtenir des informations sur l'exécution d'une tâche dans le cadre d'un scénario réaliste.

Type de rubrique	Description	Utilisation
Références 	**Références générales.** Listes détaillées et explications de la documentation de référence.	Rechercher un élément d'information de référence propre à un contexte donné.
	Références d'interface utilisateur. Rubriques de référence spéciales décrivant en détail une interface utilisateur donnée. La sélection de l'option **Aide sur cette page** dans le menu Aide du produit permet en général d'ouvrir les rubriques relatives à l'interface utilisateur.	Rechercher des informations spécifiques sur les entrées à effectuer ou sur l'utilisation d'un ou de plusieurs éléments d'une interface utilisateur donnée, tels qu'une fenêtre, une boîte de dialogue ou un assistant.
Résolution des problèmes et limites 	**Résolution des problèmes et limites**. Rubriques de référence spéciales qui décrivent les problèmes fréquents et leurs solutions, et indiquent les limites d'une fonction ou d'une zone du produit.	Permet de prendre connaissance des problèmes importants avant d'utiliser une fonction ou de consulter des informations sur un problème lié à l'utilisation du logiciel.

Autres ressources en ligne

Résolution des problèmes et Base de connaissances - Permet d'accéder à la page Résolution des problèmes du site Web d'assistance HP Software et d'effectuer des recherches dans la base de connaissances. Sélectionnez **Aide** > **Résolution des problèmes et Base de connaissances**. URL du site Web : http://h20230.www2.hp.com/troubleshooting.jsp.

Assistance HP Software - Permet d'accéder au site Web d'assistance HP Software. À partir de ce site, vous pouvez consulter la base de connaissances pour résoudre vous-même vos problèmes. Vous pouvez également publier des messages et rechercher des informations sur les forums de discussion des utilisateurs, soumettre des demandes d'assistance, télécharger des correctifs et des documents mis à jour, etc. Sélectionnez **Aide** > **Assistance HP Software**. URL du site Web : www.hp.com/go/hpsoftwaresupport.

La plupart des domaines d'assistance nécessitent une inscription en tant qu'utilisateur HP Passport et, le cas échéant, un contrat d'assistance.

Pour plus d'informations sur les niveaux d'accès, accédez à la page :

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

Pour obtenir un identifiant HP Passport, accédez à la page :

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

Page d'accueil HP Software - Permet d'accéder au site Web de HP Software. Sur ce site, vous trouverez les dernières informations relatives aux produits HP Software, notamment les nouvelles mises à jour des logiciels, les séminaires, les salons, l'assistance clientèle, etc. Sélectionnez **Aide** > **Page d'accueil HP Software**. URL du site Web : www.hp.com/go/software.

Mise à jour de la documentation

HP Software assure en permanence la mise à jour de la documentation de ses produits.

Pour rechercher les dernières mises à jour ou vérifier que vous utilisez l'édition la plus récente d'un document, visitez le site Web HP Software relatif à la documentation des produits (http://h20230.www2.hp.com/selfsolve/manuals).

Partie I

Création d'adaptateurs de découverte et d'intégration

1

Écriture et développement d'adaptateurs

Contenu de ce chapitre :

Concepts

Tâches

Concepts

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 :

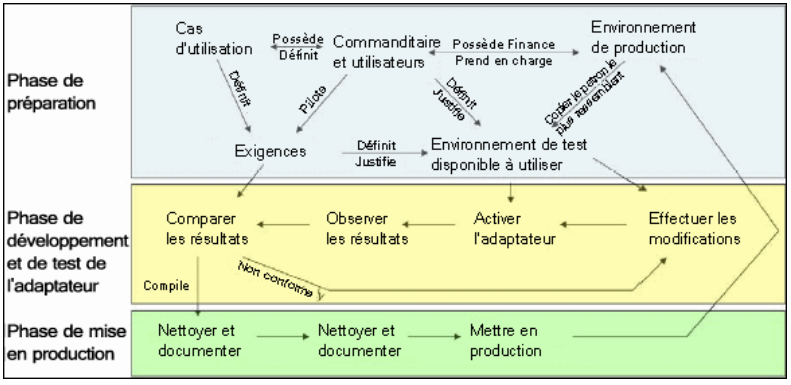
- Présuppose que vous disposiez d'une connaissance pratique de HP Universal CMDB et que les éléments du système vous soient quelque peu familiers. Vise à vous aider au cours du processus d'apprentissage et ne saurait constituer un guide exhaustif.
- Couvre les étapes de la planification, de la recherche 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 plus d'informations sur les API disponibles, voir le manuel *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

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.



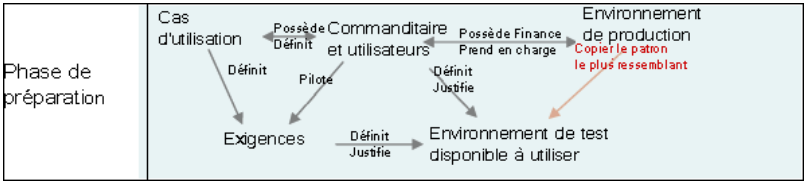
Chaque phase du développement d'un adaptateur repose sur celle qui la précède.

Une fois satisfait de l'aspect et du fonctionnement de l'adaptateur, vous voici prêt à le compresser. Utilisez le Gestionnaire des composants applicatifs UCMDB ou effectuez une exportation manuelle des composants pour créer un fichier *.zip compressé. Pour une pratique optimale, déployez et testez ce composant applicatif 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 la compression, voir "Gestionnaire des composants applicatifs" 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
- Développement et test de l'adaptateur
- Compression et mise en production de l'adaptateur

Phase de préparation et de recherche



La phase 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 manière dont l'adaptateur doit collecter des données :

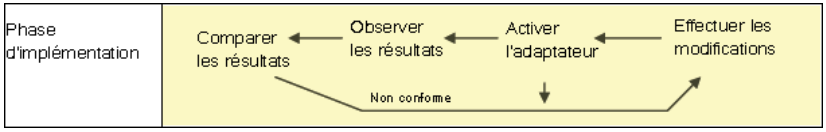
- ➤ Utilisez des outils/protocoles externes pour obtenir les données.
- ➤ Développez la manière dont l'adaptateur devra créer des éléments de configuration (CI) en fonction des données.
- ➤ Vous savez à présent à quoi doit ressembler un adaptateur similaire.

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 composant applicatif.

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



Développement et test de l'adaptateur

La phase 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 composant applicatif

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

- Utilisez le Gestionnaire des composants applicatifs pour compresser votre composant au format .zip. Pour plus d'informations, voir "Gestionnaire des composants applicatifs" 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 composants applicatifs 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 composants applicatifs 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. Quel que soit l'usage final du contenu, votre plan doit répondre aux questions suivantes relatives à cette approche :

- 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 20.

Cette section comprend les rubriques suivantes :

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 composant applicatif avec ces types de CI (qui parallèlement renfermeront plus tard l'ensemble de la découverte, des vues et des autres éléments associés au composant) 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 investit.

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 consulter les protocoles de connexion HP Universal CMDB actuellement pris en charge, voir : **Panneau de configuration de la découverte > Configuration des sondes des flux de données > volet Domaines et sondes**. Pour plus d'informations, voir "Volet Domaines et sondes" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

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). Celles-ci sont entrées dans la base CMDB et sont conservées de manière sécurisée sur l'ensemble du 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 ?

  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 ?

  Une **fédération** est nécessaire.

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

  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'intégrations, voir "Studio d'intégration" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

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

- Adaptateur Jython
  - Patron de découverte classique
  - Écrit en Jython
  - Utilisé pour le remplissage

  Pour plus d'informations, voir "Développement d'adaptateurs Jython", page 65.

- 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 tâches qui se connectent individuellement à une seule cible ou source de données.

  Pour plus d'informations, voir "Développement d'adaptateurs Java", page 225.
- Adaptateur de base de données générique
  - Adaptateur abstrait fondé sur l'adaptateur Java et 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 à la fois 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 tâches qui se connectent individuellement à une seule source de données.

  Pour plus d'informations, voir "Développement d'adaptateurs de base de données génériques", page 129.
- 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 tâches qui se connectent individuellement à une seule cible de données.
- Utilisé pour l'émission de données.

Pour plus d'informations, voir "Développement d'adaptateurs d'émission (push)", page 265.

Le tableau suivant présente les capacités de chaque adaptateur :

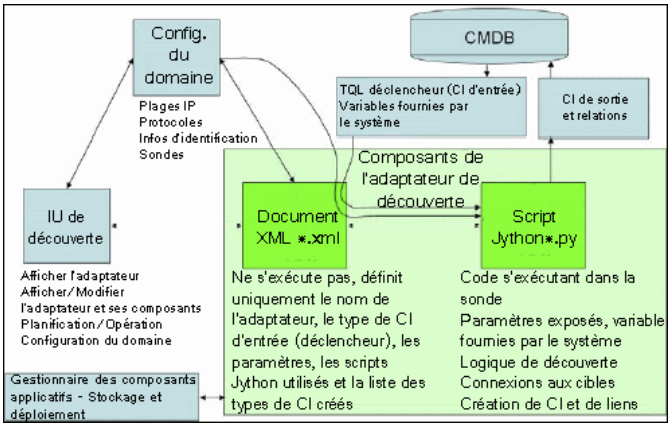
Flux/ Adaptateur	Adaptateur Jython	Adaptateur Java	Adaptateur de BD générique	Adaptateur d'émission
Remplissage	X	X	X	
Fédération		X	X	
Émission de données		X		X

Développement d'un contenu de découverte

Contenu de cette section :

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 des flux de données. Ils sont enregistrés sous forme opérationnelle dans la base CMDB proprement dite ; si le composant applicatif externe reste présent, il n'est pas référencé dans le cadre d'une exploitation. Le Gestionnaire des composants applicatifs 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 Panneau de configuration de la découverte, Gestion de l'adaptateur et Configuration des sondes des flux de données.

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

Séparation d'adaptateurs

Techniquement, une découverte entière peut être définie par 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.

Tâches

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 Adapter GFD.

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 la section suivante.

2 Travail de découverte.

Vous configurez un travail au moyen d'informations de planification et d'un code TQL déclencheur. Pour plus d'informations, voir "Étape 2 : Affectation d'un travail à l'adaptateur", page 52.

3 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 53.

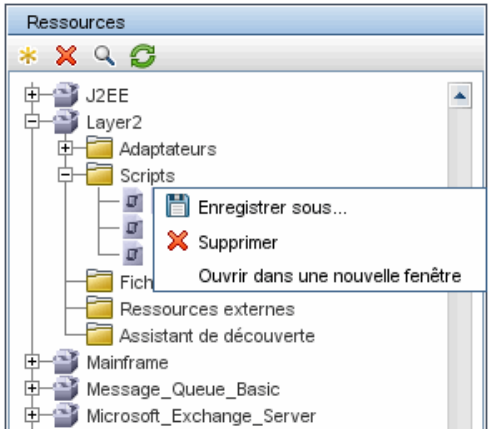
Pour écrire de nouveaux adaptateurs, vous créez chacun des composants ci-dessus, chacun d'eux étant automatiquement lié au composant de l'étape précédente. Par exemple, une fois que vous avez créé un travail et sélectionné l'adaptateur pertinent, le fichier d'adaptateur est lié à ce travail.

Code d'adaptateur

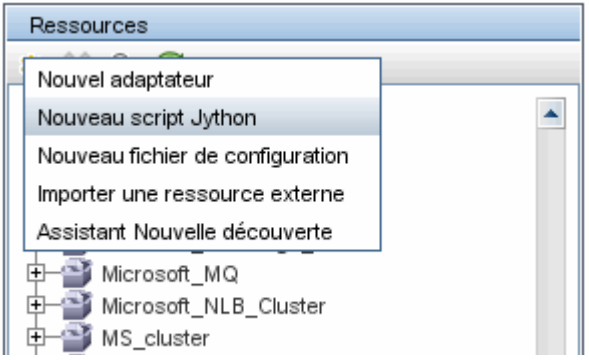
L'implémentation proprement dite de la connexion au système distant, de l'interrogation de ses données et de leur mappage en tant que données CMDB s'effectue au moyen du code Jython. Par exemple, le code contient la logique nécessaire à la connexion à une base de données et à l'extraction des données 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.

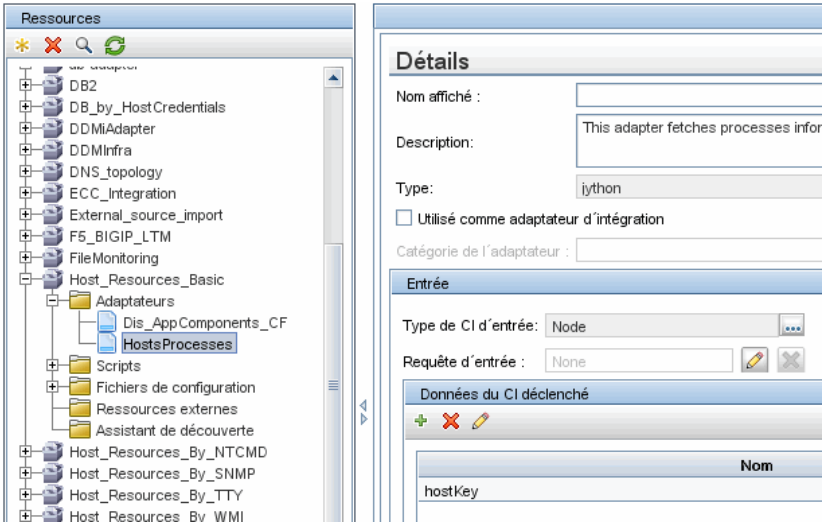
Un adaptateur peut contenir une liste de scripts utilisés lors de l'exécution de la découverte. Lors de la création d'un adaptateur, vous créez généralement un script que vous affectez à l'adaptateur. Un nouveau script comprend des modèles de base, mais vous pouvez utiliser un des autres scripts en tant 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 53. 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 normaux par l'absence de 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 "CI et requêtes déclencheurs" 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)

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.

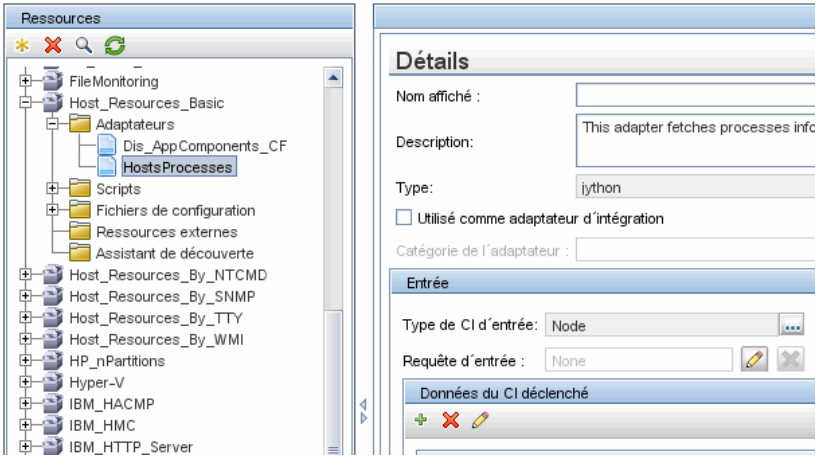
  Si l'adaptateur requiert des informations supplémentaires de la part des CI associés au CI déclencheur, vous pouvez ajouter des nœuds supplémentaires au code TQL d'entrée. Pour plus d'informations, voir "Exemple de définition de requête d'entrée", page 46 et "Ajouter des nœuds de requête et des relations 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.

Exemple de définition d'un type de CI déclencheur :

Dans cet exemple, un type de CI déclencheur spécifie que des CI IP sont autorisés dans l'adaptateur.

1. Accédez à la page **Gestion des flux de données** > **Gestion de l'adaptateur**. Sélectionnez l'adaptateur HostProcesses (**Composants applicatifs** > **Host_Resources_Basic** > **Adaptateurs** > **HostProcesses**).
2. Repérez le champ Type de CI d'entrée. Pour plus d'informations, voir "Données du CI déclenché" dans le *Manuel de gestion des flux de données HP Universal CMDB*.
3. Cliquez sur le bouton pour ouvrir la boîte de dialogue Sélectionner la classe détectée. Pour plus d'informations, voir "Boîte de dialogue Sélectionner la classe détectée" dans le *Manuel de gestion des flux de données HP Universal CMDB*.
4. Sélectionnez le type de CI.

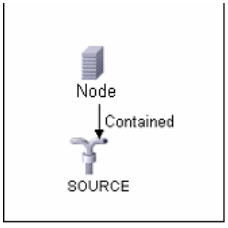
   Dans cet exemple, le CI IP (Host) est autorisé dans l'adaptateur :



Exemple de définition de requête d'entrée

Dans cet exemple, la requête TQL d'entrée spécifie que le CI IP (configuré dans l'exemple précédent en tant que type de CI déclencheur) doit être connecté à un CI Host.

1. Accédez à la page **Gestion des flux de données** > **Gestion de l'adaptateur**. Repérez le champ TQL d'entrée. Cliquez sur le bouton **Modifier** pour ouvrir la fenêtre Éditeur de requête d'entrée. Pour plus d'informations, voir "Fenêtre Éditeur de requête d'entrée" dans le manuel *Manuel de gestion des flux de données HP Universal CMDB*.
2. Dans l'Éditeur de requête d'entrée, nommez le nœud de CI déclencheur **SOURCE** : cliquez avec le bouton droit de la souris sur le nœud, puis sélectionnez **Propriétés du nœud**. Dans le champ **Nom de l'élément**, attribuez le nom **SOURCE**.
3. Ajoutez un CI Host et une relation Contains au CI IP. Pour plus d'informations sur l'utilisation de l'Éditeur de requête d'entrée, voir "Fenêtre Éditeur de requête d'entrée" dans le manuel *Manuel de gestion des flux de données HP Universal CMDB*.



Le CI **IP** est connecté à un CI **HOST**. Le code TQL d'entrée est constitué de deux nœuds, **HOST** et **IP**, et d'un lien qui les relie. Le CI **IP** se nomme **SOURCE**.

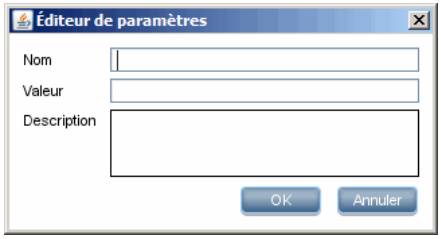
Exemple d'ajout de variables à la requête TQL d'entrée :

Dans cet exemple, vous ajoutez les variables DIRECTORY et CONFIGURATION_FILE à la requête TQL d'entrée créée dans l'exemple précédent. Ces variables contribuent à définir ce qui doit être découvert ; dans le cas présent, il s'agit des fichiers de configuration qui se trouvent sur les hôtes liés aux adresses IP que vous devez découvrir.

1. Affichez le code TQL d'entrée créé dans l'exemple précédent.

   Accédez à **Gestion des flux de données** > **Gestion de l'adaptateur**. Repérez le volet Données du CI déclenché. Pour plus d'informations, voir "Données du CI déclenché" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

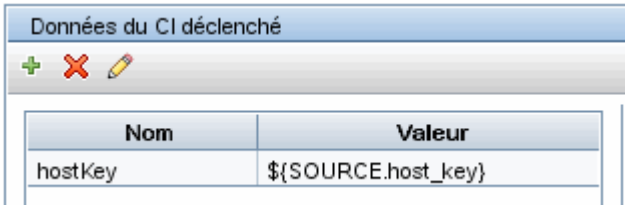
2. Ajoutez des variables au code TQL d'entrée. Pour plus d'informations, consultez **Gestion des flux de données** > **Gestion de l'adaptateur**. Repérez le volet Données du CI déclenché. Pour plus d'informations, voir le champ Variables dans "Données du CI déclenché" dans le *Manuel de gestion des flux de données HP Universal CMDB*.



Exemple de remplacement de variables par des données réelles :

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

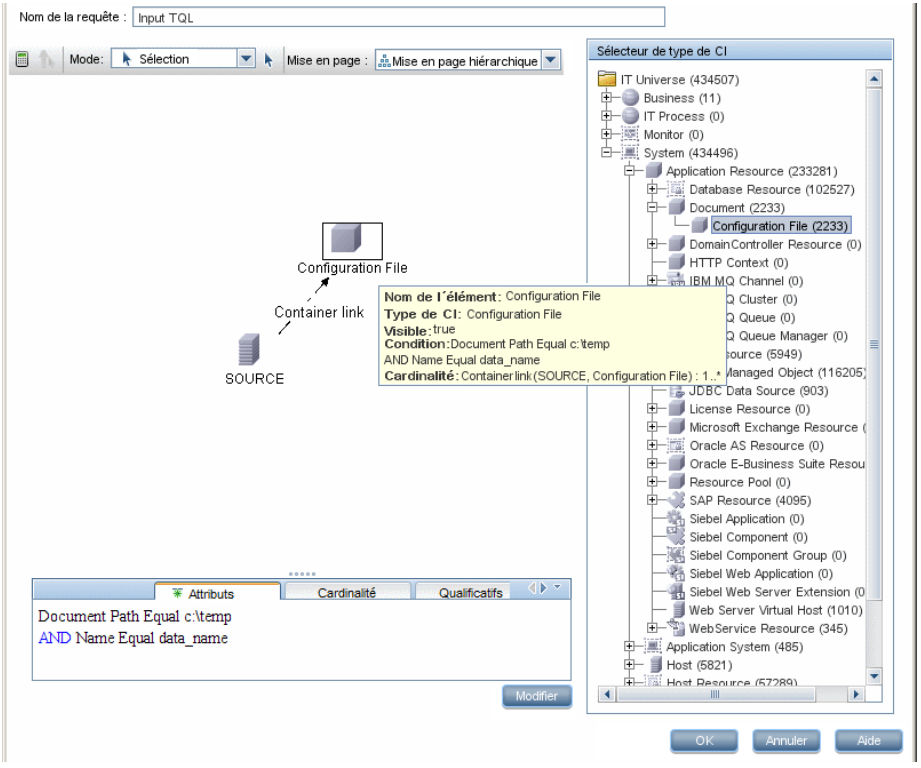
Les données du CI déclenché pour le CI **IP** comprennent une variable fileName. Cette variable permet le remplacement du nœud CONFIGURATION_FILE dans le code TQL d'entrée par les valeurs réelles du fichier de configuration installé sur un hôte :



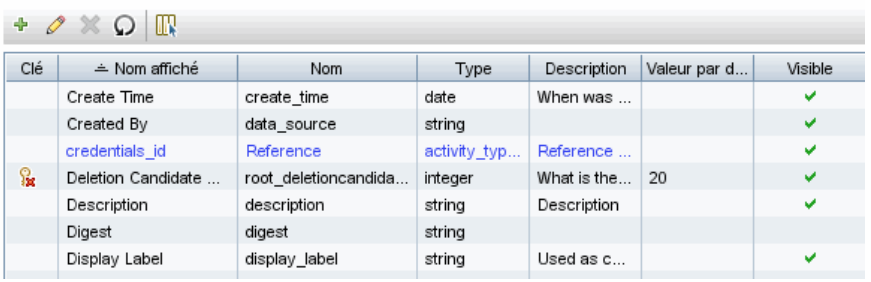
Les données du CI déclencheur sont téléchargées vers la sonde avec toutes les variables remplacées par de vraies valeurs. Le script de l'adaptateur comprend une commande qui permet d'utiliser l'infrastructure GFD pour extraire les valeurs réelles des variables définies :

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

Les variables fileName et path utilisent les attributs data_name et document_path du nœud du fichier de configuration (défini dans le code TQL d'entrée ; voir exemple précédent).



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

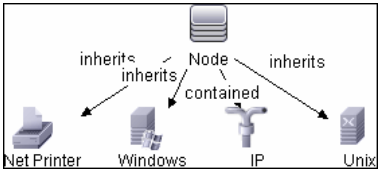


Clé	Nom affiché	Nom	Type	Description	Valeur par d...	Visible
	Create Time	create_time	date	When was ...		✔
	Created By	data_source	string			✔
	credentials_id	Reference	activity_typ...	Reference ...		✔
	Deletion Candidate ...	root_deletioncandida...	integer	What is the...	20	✔
	Description	description	string	Description		✔
	Digest	digest	string			
	Display Label	display_label	string	Used as c...		✔

Définition d'une sortie d'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 74.

Exemple de sortie d'adaptateur :

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

1. Accédez à **Gestion des flux de données** > **Gestion de l'adaptateur**.
2. Dans le volet Ressources, sélectionnez **Network** > **Adaptateurs** > **NSLOOKUP_on_Probe**.
3. Dans l'onglet Définition de l'adaptateur, accédez au volet Types de CI détectés.
4. 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 "Volet Types de CI détectés" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

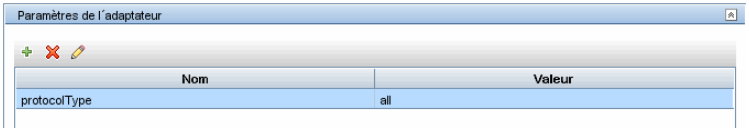
Remplacement de paramètres d'adaptateur

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.

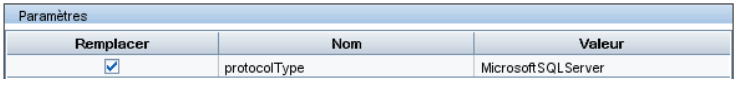
Exemple de remplacement d'un paramètre d'adaptateur :

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.

1. Accédez à **Gestion des flux de données** > **Gestion de l'adaptateur**.
2. Dans le volet Ressources, sélectionnez **Database Basic** > **Adaptateurs** > **SQL_NET_Dis_Connection**.
3. Dans l'onglet Définition de l'adaptateur, recherchez le volet **Paramètres de l'adaptateur**. Le paramètre protocolType a pour valeur **all** :



4. 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**.
5. Affichez l'onglet Propriétés. Accédez au volet Paramètres :



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.

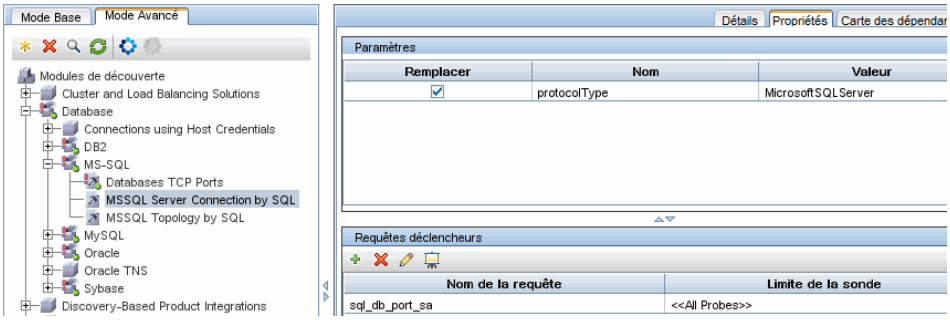
Pour plus d'informations sur l'ajout, la suppression ou la modification de paramètres, voir "Volet Paramètres de l'adaptateur" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

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

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. C'est l'entité que l'utilisateur active.



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.

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 "Boîte de dialogue Planificateur de découverte" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Paramètres

Lorsque vous configurez un travail, vous pouvez remplacer les paramètres d'adaptateur. Pour plus d'informations, voir "Remplacement de paramètres d'adaptateur", page 51.

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 de code Jython", page 67.

2

Consignes de migration du contenu de découverte

Contenu de ce chapitre :

Concepts

Tâches

Références

Concepts

Consignes de migration du contenu de découverte - Présentation

Dans HP Universal CMDB version 9.0x, le modèle de données a beaucoup évolué, imposant des modifications correspondantes dans le code antérieur du contenu de Discovery and Dependency Mapping (DDM). En conséquence, certains mécanismes principaux du contenu DDM ont changé. Ainsi, le contenu développé pour UCMDB avant la version 9.0x doit être mis à niveau pour correspondre au modèle de données 9.0x (BDM : modèle de données BTO). Cette section décrit la procédure d'adoption du contenu DDM et son alignement avec BDM.

Pour plus d'informations sur la mise à niveau de HP Universal CMDB, voir "Mise à niveau de HP Universal CMDB version 8.0x vers la version 9.0x" dans le *Manuel de déploiement HP Universal CMDB* PDF.

Nouvelles fonctions de l'infrastructure de la version 9.0x

Remarque : Pour plus d'informations sur l'accès à la documentation BDM en ligne, voir "Accès à la documentation du modèle de données BTO en ligne", page 62.

Contenu de cette section :

Modèle de données BTO (BDM)

- Pour plus de détails sur le modèle de données BTO, voir le document Conceptual Data Model. Ce document cartographie les concepts modélisés, ainsi que l'étendue du modèle. Ce modèle de données conceptuel offre un point de départ pour comprendre la sémantique du domaine modélisé.
- Pour plus d'informations sur les classes BDM, voir le document HP Software BTO Data Model Reference. Ce document couvre toutes les classes BDM, incluant des informations sur la description d'une classe, ses attributs, ses qualificatifs et sa hiérarchie.

Différences entre le modèle de classe UCMDB 8.0x et le modèle de données UCMDB 9.0x

Les modifications apportées entre le modèle de classe UCMDB version 8.0x et BDM sont téléchargées sur la sonde dans le fichier de configuration de découverte suivant :
C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles\flat-class-model-changes.xml.

bdm_changes.xml. Ce fichier XML contient des informations sur les modifications apportées aux noms de classe et d'attribut, aux classes supprimées, aux attributs, aux qualificatifs, etc.

- Pour plus de détails sur le mappage entre le modèle de classe UCMDB version 8.0x et BDM, voir le document Mapping of UCMDB 9.0x (BTO Data Model) to UCMDB 8.0x Class Model.
- Pour plus d'informations sur les modifications apportées au modèle de classe entre les versions 8.0x et 9.0x, voir le document UCMDB Class Model Changes Report.

Nouveau mécanisme d'identification des types de CI

Dans les versions d'UCMDB antérieures à la 9.0x, des attributs clés servent à identifier les CI. Dans UCMDB version 9.0x, ce concept est généralisé et l'identification s'effectue désormais dans un composant serveur appelé moteur de rapprochement. Le moteur de rapprochement est capable d'identifier les CI par des règles logiques appelées règles DDA (algorithme de définition de données).

Ce nouveau mécanisme est principalement utile pour les types de CI dont l'identification dépend de la topologie associée (par exemple, le type de CI Node (Host dans les versions antérieures) est identifié par son nom et la topologie associée, par exemple les types de CI IP Address et Interface). Certains types de CI sont toujours identifiés par des attributs clés ; pour ceux-là, aucune règle DDA n'est définie.

Pour plus d'informations sur le moteur de rapprochement, voir "Présentation du rapprochement" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

Mécanisme des logiciels en exécution

Le CI **Software Element** de la version 8.0x est appelé **Running Software** dans BDM version 9.0x. Ce type de CI est identifié dans la version 9.0x par une règle DDA, au lieu d'attributs clés.

Supposons que vous ayez ajouté un type de CI personnalisé dérivé du type de CI **Running Software**. Dans les versions antérieures, ce type de CI personnalisé était identifié par ses attributs clés. Dans la version 9.0x, cependant, il est identifié par une règle DDA héritée, et les attributs clés définis sont ignorés.

Donc, si vous ajoutez un type de CI dérivé, vous devez tenir compte des points suivants :

- Pour identifier le nouveau type de CI par la même règle DDA que tous les type de CI Running Software, vous devez conserver la configuration actuelle.
- Pour identifier le nouveau type de CI par des attributs clés, vous devez créer une nouvelle règle DDA, définissant l'identification par des attributs clés. Voici un exemple de règle DDA de ce type, définie pour le type de CI **object** :

```
<identification-config type="object">
    <identification-criteria>
        <identification-criterion targetType="root">
            <key-attributes-condition/>
        </identification-criterion>
</identification-criteria>
</identification-config>
```

Identification côté sonde

DDM_ID_ATTRIBUTE. La sonde des flux de données de la version 9.0x identifie les CI uniquement par leurs attributs clés (c'est-à-dire **ID_ATTRIBUTE**). Si un type de CI comprend une règle DDA (c'est-à-dire une règle de rapprochement), il peut ne pas inclure d'attribut clé. Dans ce cas, les attributs principaux du type de CI sont marqués par un qualificatif **DDM_ID_ATTRIBUTE**. Par conséquent, pour identifier un CI, la sonde prend en compte tous les qualificatifs **DDM_ID_ATTRIBUTE** et **ID_ATTRIBUTE**.

DDM_REQUIRED_TOPOLOGY. Une règle DDA pour un type de CI particulier peut dépendre de différents CI signalés dans le même lot que le CI examiné. Par exemple, l'identification du type de CI **J2EE Domain** s'effectue non seulement par l'attribut du nom de domaine, mais aussi par le type de CI **J2EE Application Server** auquel il est connecté avec un lien member.

Pour vous assurer que tous les CI requis seront signalés avec le CI examiné, vous devez marquer chaque CI examiné avec le qualificatif **DDM_REQUIRED_TOPOLOGY** qui contient un élément de données spécifiant le type de lien requis. Ainsi, dans l'exemple ci-dessus, le type de CI **J2EE Domain** est marqué avec le qualificatif **DDM_REQUIRED_TOPOLOGY** et un élément de données de lien **member** afin que, lorsque la découverte signale un domaine J2EE, les serveurs soient également signalés.

Pour plus d'informations sur les qualificatifs, voir "Page Qualificatifs" dans le *Manuel de modélisation HP Universal CMDB*.

Couche de transformation

Pour garantir la compatibilité rétroactive, un nouveau mécanisme de transformation est introduit dans la version 9.0x sur la sonde. Ce nouveau mécanisme est capable de convertir des topologies de la version 8.0x en topologies 9.0x lors de l'exécution. Il permet à la sonde de continuer à exécuter des tâches, comme des scripts Jython, qui rapportent des topologies compatibles avec la version 8.0x.

Le nouveau mécanisme de transformation utilise les données conservées dans le fichier **bdm_changes.xml** et effectue les modifications requises (changement de nom de classe et d'attribut, suppression d'attribut, modification de la hiérarchie, etc.) pour rendre les topologies 8.0x compatibles avec BDM. En même temps (et indépendamment des topologies signalées par les tâches exécutées par la sonde), le serveur UCMDB reçoit des topologies compatibles avec BDM.

Utilitaire de migration des composants applicatifs

L'installation de UCMDB 9.0x comprend un utilitaire externe de migration de composants applicatifs qui permet aux développeurs de convertir un composant applicatif de contenu du modèle de classe 8.0x au modèle de données 9.0x. Cet utilitaire convertit les ressources des composants applicatifs, sous-système par sous-système, afin de les rendre compatibles avec le nouveau modèle de classe. Les définitions de type de CI, les requêtes, les travaux, les adaptateurs et les modules sont transformés en fonction des données contenues dans le fichier **bdm_changes.xml**. En conséquence, ils peuvent être déployés et utilisés par un serveur UCMDB 9.0x.

Pour plus d'informations, voir "Mise à niveau des composants applicatifs de la version 8.04 vers 9.02" dans le *Manuel de déploiement HP Universal CMDB* PDF.

Limites de l'utilitaire de migration de composants applicatifs

- L'utilitaire de migration de composants applicatifs ne met pas à niveau les scripts Jython. Pour la prise en charge des scripts conçus pour correspondre au modèle de classe UCMDB version 8.0x, un nouveau module **Couche de transformation** est introduit dans UCMDB 9.0x. Pour plus d'informations, voir "Couche de transformation", page 59.
- L'utilitaire de migration de composants applicatifs ne met pas à niveau les adaptateurs de découverte de type Intégration, qui doivent donc être mis à niveau manuellement.
- Le travail de découverte Topologie de couche 2 (et ses ressources correspondantes, comme l'adaptateur de découverte, TQL, etc.) a beaucoup changé et l'utilitaire de migration de composants applicatifs le supprime au lieu de le mettre à niveau.

Consignes de développement de scripts de modèles de données croisées

Les consignes suivantes s'appliquent aux deux versions, 8.0x et 9.0x.

Bibliothèque d'API de scripts de découverte

La bibliothèque d'API de découverte est entièrement compatible rétroactivement et, par conséquent, toutes les bibliothèques et API de la version 8.0x sont prises en charge. Pour plus d'informations, voir "Bibliothèques et utilitaires Jython", page 116.

L'API 9.0x comprend plus d'éléments et de méthodes. Par exemple, un script Jython indique désormais un code d'erreur (entier) au lieu d'un message d'erreur de type chaîne, ce qui permet de localiser les messages d'erreur de découverte. Pour plus d'informations, voir "Conventions d'écriture d'erreurs", page 123.

Conseils d'implémentation

- Utilisez le module **Modélisation** pour créer un type de CI **Running Software** ou tout descendant qui comporte la méthode concernée.
- Utilisez **HostBuilder** pour créer un CI de type **Node**.
- Utilisez **modeling.createOshByCmdbIdString** pour restaurer OSH par son ID.
- Utilisez l'instance **ShellUtils** du module **shellutils** pour toutes les connexions basées sur le shell.
- Utilisez le mécanisme intégré pour extraire la version UCMDB : logger.Version().getVersion(framework). Par exemple, si un attribut supplémentaire application_ip est ajouté uniquement pour UCMDB version 9.0x ou ultérieure :

```
versionAsDouble = logger.Version().getVersion(Framework)
if versionAsDouble >= 9:
    appServerOSH.setAttribute('application_ip', ip)
```

- Utilisez **wmiutils** pour créer une découverte basée sur WMI.
- Utilisez **snmputils** pour créer une découverte basée sur SNMP.

Tâches

Accès à la documentation du modèle de données BTO en ligne

Pour accéder à la documentation BDM :

1. Connectez-vous à HP Universal CMDB.
2. Cliquez sur **Aide** > **Aide UCMDB**.
3. Dans la page d'accueil, cliquez sur le lien **Modélisation** sous **Applications** pour accéder au portail **Modélisation**.
4. Cliquez sur l'onglet **Data Model.**

Références

Résolution des problèmes et limites

- La valeur **ip_address** n'est pas transmise par défaut au patron. Elle devrait y être ajoutée explicitement sous la forme de données de CI déclencheur.
- Si un script Jython non prêt à l'emploi exige un fichier jar ou une ressource externe dans le classpath, cet élément doit figurer dans le composant applicatif concerné, dans un sous-dossier appelé **discoveryResources**.
- En travaillant avec des attributs de type **List** comme **StringVector** et **IntegerVector** (hérités de **BaseVector**), vous ne pouvez pas utiliser les opérations **add element** et **remove element** sur le même objet de liste.

3

Développement d'adaptateurs Jython

Contenu de ce chapitre :

Concepts

Tâches

Références

Concepts

Référence des API de gestion des flux de données HP

Pour consulter la documentation complète sur les API disponibles, voir *HP Universal CMDB Data Flow Management API Reference*. Ces fichiers figurent dans le dossier suivant :

C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\doc_lib\DevRef_guide\DDM_JavaDoc\index.html

Tâches

Création de 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

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 la gestion des flux de données doivent être compatibles avec Jython version 2.1.
- Pour consulter la documentation complète sur les API disponible, voir *HP Universal CMDB Data Flow Management API Reference*.

Contenu de cette section :

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 la forme soit d'archives utilitaires Java, soit d'archives de connexion telles que des fichiers JAR du pilote JDBC Driver, soit de fichiers exécutables (par exemple, **nmap.exe** est utilisé pour la découverte sans informations d'identification).

Ces ressources doivent être regroupées dans le composant applicatif 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_utils)
```

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 httpplugin

#

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 find. this join
can't be done with IP address:port

                # because multiple apps on a web server share the same IP:port for
multiple websphere applications

                logger.debug('httpcontext_webapplicationserver attribute has been
changed from [' + servername + '] to [' + transportHN[:5] + '] to facilitate websphere
enrichment')

                servername = transportHN[:5]

        return servername
```

Enregistrez le script externe dans le dossier des ressources externes. Pour plus d'informations, voir "Volet Ressources" dans le *Manuel de gestion des flux de données HP Universal CMDB*. Si vous ajoutez ce script à un composant applicatif, vous pourrez également l'utiliser pour d'autres travaux. Pour plus d'informations sur l'utilisation du Gestionnaire des composants applicatifs, voir "Gestionnaire des composants applicatifs" 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 :

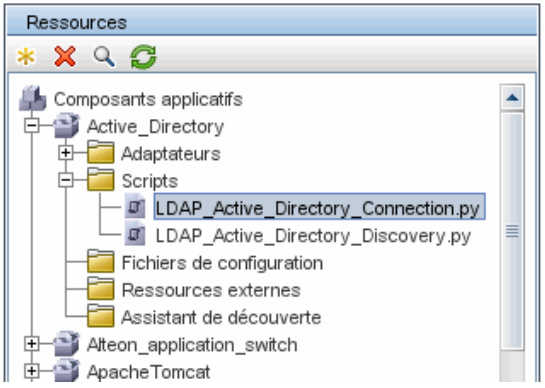
```
def DiscoveryMain(Framework):
```

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 composant applicatif :



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,string'
    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.getClientFactory(ClientsConsts.SNMP_PROTOCOL_NAME).createClie
nt()
    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" 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" 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 de détails sur le signalement de ce résultat composite à l'infrastructure pour permettre son envoi 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 les types de CI communs, voir **modeling.py** dans "Bibliothèques et utilitaires Jython", page 116.

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 66.

Cette section décrit les utilisations les plus importantes de Framework :

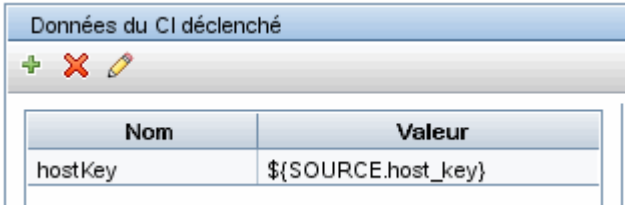
- "Framework.getTriggerCIData(String attributeName)", page 76
- "Framework.createClient(credentialsId, props)", page 77
- "Framework.getParameter (String parameterName)", page 78
- "Framework.reportError(String message) et Framework.reportWarning(String message)", page 79

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 "Informations d'identification de domaine" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

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.

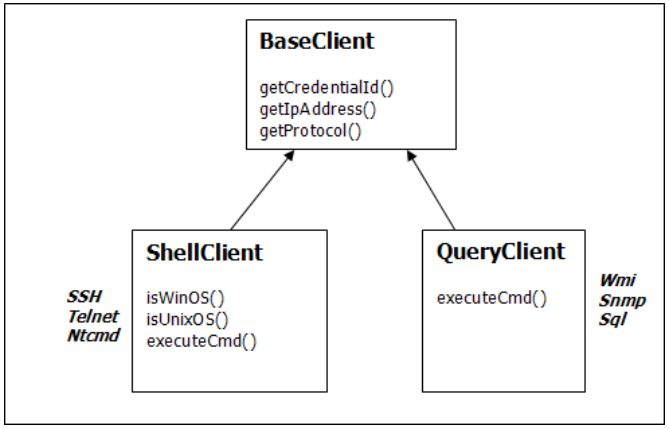
Exemple de création d'un client WMI et d'exécution d'une requête WMI :

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
                                    FROMWin32_LogicalMemoryConfiguration")
```

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 les API prises en charge, voir BaseClient, ShellClient et QueryClient dans le manuel *HP Universal CMDB Data Flow Management API Reference*.

Framework.getParameter (String parameterName)

En plus d'extraire des informations sur le CI déclencheur, vous devez souvent extraire une valeur de paramètre d'adaptateur. 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.getClientFactory(ClientsConsts.SNMP_PROTOCOL_NAME)
        createClient()
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 121.

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'un des paramètres nécessaires lors de la création d'un client (via ClientFactory) est l'ID d'informations d'identification. 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 clientFactory.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).

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 38.

```
import logger
from appilog.collectors.clients import ClientsConsts
from appilog.common.system.types.vectors import ObjectStateHolderVector

        def mainFunction(Framework):
resultVector = ObjectStateHolderVector()

        # Get the Trigger CI data values
        ip_address = Framework.getDestinationAttribute('ip_address')
        ip_domain  = Framework.getDestinationAttribute('ip_domain')

        # Create the client factory for SNMP
        clientFactory = framework.getClientFactory(ClientsConsts.SNMP_PROTOCOL_NAME)
        protocols = clientFactory.getAvailableProtocols(ip_address, ip_domain)
```

```
connected = 0
# Go over all the protocol credentials
for credentials_id in protocols:
    client = None
    try:
        # try to connect to the snmp agent
        client = clientFactory.createClient(credentials_id)

        // Query the agent
        ….

        # connection succeed
        connected = 1
    except:
        if client != None:
            client.close()
if (not connected):
    logger.debug('Failed to connect using all credentials')
else:
    // return the results as OSHV
    return resultVector
```

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, par exemple :

```
try:
        client = Framework.getClientFactory().createClient()
except Exception, msg:
        Framework.reportError('Connection failed')
        logger.debugException('Exception while connecting: %s' % (msg))
        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.

Localisation de la prise en charge 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.

Précédemment, avant le Content Pack 3.00, la gestion des flux de données utilisait un codage défini statiquement pour traiter les sorties de toutes les cibles réseau. Cependant, cette approche ne convient pas à un réseau informatique multilingue : pour découvrir des hôtes avec des SE en langues différentes, les administrateurs de la sonde devaient réexécuter manuellement les travaux GFD à plusieurs reprises avec des paramètres différents à chaque fois. Cette procédure générait une importante surcharge du réseau, mais évitait surtout plusieurs fonctions clés de GFD, telles que l'appel de travail immédiat sur un CI déclencheur ou l'actualisation automatique des données dans UCMDB par le Gestionnaire de planification.

Les langues suivantes des paramètres régionaux sont prises en charge par défaut : japonais, russe et allemand. Les paramètres régionaux par défaut sont ceux de l'anglais.

Contenu de cette section :

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 :

➤ "Ajout d'un groupe de ressources (fichiers *.properties)", page 83

➤ "Déclaration et enregistrement de l'objet Language", page 84

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 88.

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 composant applicatif AutoDiscoveryContent. Pour visualiser le script, accédez à la fenêtre Gestion de l'adaptateur. Pour plus d'informations, voir "Fenêtre Gestion de l'adaptateur" 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'),
(1049,), 866)
```

Pour plus d'informations sur la langue des classes, voir "Référence des API", page 89. 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_FRENCH, LANG_SPANISH,
LANG_RUSSIAN, LANG_JAPANESE)
```

Changement de 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 83.

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 réginaux, 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, vous utilisez l'API pour la classe **ShellUtils**. Pour plus d'informations, voir "API de service Web HP Universal CMDB - Présentation", page 296.

Ce code prend généralement la forme suivante :

```
client = Framework.createClient(protocol, properties)
shellUtils = shellutils.ShellUtils(client)
languageBundle = shellutils.getLanguageBundle (’langNetwork’, shellUtils.osLanguage,
Framework)
strWindowsIPAddress = languageBundle.getString(’windows_ipconfig_str_ip_address’)
ipconfigOutput = shellUtils.executeCommandAndDecode(’ipconfig /all’,
strWindowsIPAddress)
#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.osLanguage, Framework)
```

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

```
strWindowsIPAddress =
languageBundle.getString('windows_ipconfig_str_ip_address')
```

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

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

L'objet ShellUtils est également nécessaire pour renvoyer un utilisateur à la 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 4 page 87 dans "Définition d'un nouveau travail fonctionnant avec des données localisées", page 86.

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**.

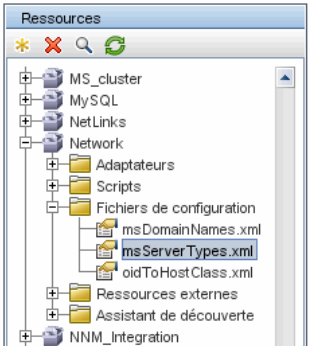
3 Faites en sorte que **shellUtils** utilise ce jeu de caractères par défaut, en appelant la méthode **useCharset** sur l'objet **ShellUtils**.

4 Appelez la méthode **execCmd** de **ShellUtils** une ou plusieurs fois. La sortie est renvoyée avec le jeu de caractères spécifié à l'étape 3. 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 "Volet Fichier de configuration" 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 les données de langues différentes, dans la gestion des flux de données un groupe contient toujours 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

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", page 91.

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.

Utilisation de Discovery Analyzer

L'outil Discovery Analyzer est conçu à des fins de débogage lors du développement de composants applicatifs, de scripts ou de tout autre contenu. L'outil exécute un travail sur une destination distante et renvoie des journaux contenant des informations, des avertissements et des erreurs, ainsi que les résultats de la découverte de CI.

Notez que les résultats ne sont pas toujours indiqués dans l'interface utilisateur. En effet, il existe deux façons d'indiquer les résultats dont une seule est prise en charge. De plus, le journal de communication n'est pas pris en charge par Eclipse.

Lors de l'exécution de l'outil à partir d'Eclipse, le fichier **DiscoveryProbe.properties** (**C:\hp\UCMDB\DataFlowProbe\conf\DiscoveryProbe.properties**) doit contenir le paramètre suivant, avec la valeur **true** :

```
appilog.agent.local.discoveryAnalyzerFromEclipse = true
```

Pour plus d'informations, voir "Exécution de Discovery Analyzer à partir d'Eclipse", page 103.

Dans tous les autres cas (lorsque l'outil est exécuté à partir du fichier **cmd** ou pendant que la sonde est en cours d'exécution), cet indicateur doit avoir la valeur **false**:

```
appilog.agent.local.discoveryAnalyzerFromEclipse = false
```

Tâches et enregistrements

Un fichier de tâche contient des données sur une tâche à exécuter. La tâche se compose d'informations telles que le nom du travail et les paramètres obligatoires qui définissent le CI déclencheur, par exemple l'adresse de la destination distante.

Un fichier d'enregistrement contient des informations de tâche, ainsi que les résultats d'une exécution particulière, c'est-à-dire la communication détaillée (réponse comprise) entre la sonde ou Discovery Analyzer (selon le module qui a exécuté la tâche) et la destination distante.

Une tâche définie par un fichier de tâche peut être exécutée sur une destination distante, tandis qu'une tâche définie par un fichier d'enregistrement (qui contient des données supplémentaires sur une exécution particulière) peut être exécutée ainsi que lue (c'est-à-dire qu'elle peut reproduire la même exécution consignée dans le fichier d'enregistrement).

Journaux

Les journaux fournissent des informations sur la dernière exécution, comme suit :

- **Journal General**. Ce journal renferme l'ensemble des informations, erreurs et avertissements survenus au cours de l'exécution.
- **Journal Communication**. Ce journal contient la communication détaillée entre Discovery Analyzer et la destination distante (réponse comprise). Après l'exécution, le journal peut être enregistré sous forme de fichier d'enregistrement.
- **Journal Results**. Affiche une liste des CI découverts. La durée d'apparition de chaque CI dépend de la conception des adaptateurs et scripts.

Vous pouvez enregistrer tous les journaux ensemble ou séparément. Lorsque vous enregistrez tous les journaux, ils sont enregistrés ensemble sous un seul nom.

Si vous relisez un fichier d'enregistrement, les mêmes données sont affichées dans le journal de communication, la seule différence portant sur l'heure d'exécution.

Limite : Les journaux Communication et Results ne sont pas disponibles lors de l'exécution de Discovery Analyzer par le biais d'Eclipse.

Contenu de cette section :

1 Conditions préalables

- La sonde doit être installée. (Discovery Analyzer est installé dans le cadre du processus d'installation de la sonde et partage des ressources avec elle.)
- Il n'est pas nécessaire que la sonde soit en cours d'exécution lorsque vous utilisez Discovery Analyzer.

  Cependant, si la sonde a déjà été exécutée sur un serveur UCMDB, toutes les ressources requises sont déjà téléchargées dans le système de fichiers. Si la sonde n'a pas été exécutée, vous pouvez télécharger les ressources dont Discovery Analyzer a besoin par le biais du menu Settings. Pour plus d'informations, voir "Importation de paramètres", page 101.
- Il n'est pas nécessaire d'installer le serveur CMDB.

2 Accès à Discovery Analyzer

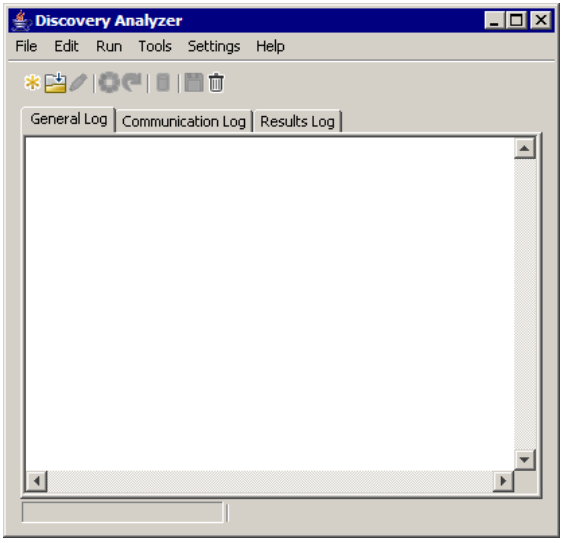
Vous accédez à Discovery Analyzer de différentes manières :

➤ En utilisant Eclipse.

L'installation de la sonde comporte un espace de travail Eclipse par défaut situé dans **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**. Cet espace comprend un script Jython pour démarrer Discovery Analyzer (**startDiscoveryAnalyzerScript.py**), ainsi qu'un lien à tous les scripts GFD. Si vous démarrez l'outil de cette façon, vous pouvez rechercher des points d'arrêt dans les scripts Jython à des fins de débogage.

➤ Directement, en double-cliquant sur le fichier dans le dossier suivant : **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzer.cmd**. Pour plus d'informations, voir la section suivante.

La fenêtre Discovery Analyzer s'ouvre :



3 Définition d'une tâche

Vous définissez une tâche par l'une des méthodes suivantes :

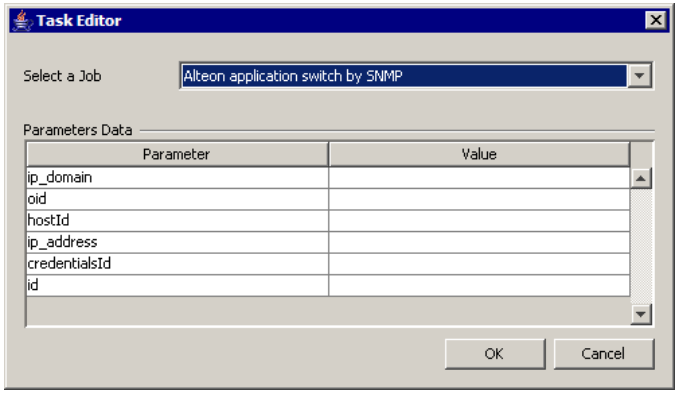
➤ En définissant une nouvelle tâche. Pour plus d'informations, voir "Définition d'une nouvelle tâche", page 97.

- En important une tâche à partir d'un fichier d'enregistrement. Pour plus d'informations, voir "Extraction d'un enregistrement", page 98.
- En important une tâche enregistrée à partir d'un fichier de tâche. Pour plus d'informations, voir "Ouverture d'un fichier de tâche", page 98.
- En extrayant un travail de la base de données interne de la sonde. Pour plus d'informations, voir "Importation d'une tâche depuis la base de données", page 99.

4 Définition d'une nouvelle tâche

a Affichez Task Editor : cliquez sur le bouton **New Task**..

Task Editor affiche une liste des travaux qui existent actuellement dans le système de fichiers. Cette liste est mise à jour chaque fois que la sonde reçoit des tâches du serveur ou que des composants applicatifs sont déployés manuellement à partir du menu Settings.



b Sélectionnez un travail.

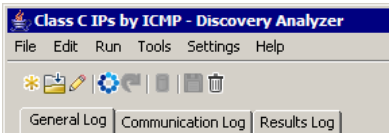
c Entrez des valeurs pour tous les paramètres.

Les paramètres affichés ici sont des paramètres d'adaptateur GFD. Ils peuvent être affichés dans le volet Discovery Pattern Parameters de l'onglet Pattern Signature. Pour plus d'informations, voir "Volet Paramètres de l'adaptateur" dans le manuel *Manuel de gestion des flux de données HP Universal CMDB*.

Tous les champs sont obligatoires (sauf si le script d'un travail exige qu'un champ soit vide).

Pour les paramètres exigeant une valeur d'entrée d'ID ou d'ID d'informations d'identification, vous pouvez utiliser des ID créés de façon aléatoire : cliquez avec le bouton droit sur la zone de valeur et sélectionnez **Generate random CMDB ID** ou **Credential Chooser**.

La tâche est désormais active et son nom est affiché dans la barre de titre :



d Poursuivez la procédure de définition d'une tâche. Pour plus d'informations, voir "Enregistrement de la tâche et des journaux", page 99.

5 Extraction d'un enregistrement

Vous pouvez définir une tâche en ouvrant un fichier d'enregistrement contenant des données relatives à une exécution particulière. Si une tâche est définie de cette manière, vous pouvez reproduire l'exécution particulière en sélectionnant l'option de lecture. (Si une tâche est relue, les réponses proviennent des données stockées dans le fichier d'enregistrement et non de la destination distante).

Sélectionnez **File** > **Open Record**. Naviguez jusqu'au dossier où vous avez enregistré l'enregistrement. L'enregistrement est désormais actif et le nom de la tâche est affiché dans la barre de titre.

Pour plus d'informations sur l'acquisition d'un fichier d'enregistrement, voir "Enregistrement de code GFD", page 114.

6 Ouverture d'un fichier de tâche

Vous pouvez définir une tâche à partir d'un fichier de tâche : Sélectionnez **File** > **Open Task**.

7 Importation d'une tâche depuis la base de données

Vous pouvez extraire une tâche à partir de la base de données de la sonde, à condition que cette dernière ait déjà été exécutée et contienne des tâches actives dans sa base interne. Vous pouvez utiliser les valeurs de paramètre pour définir la tâche.

a Sélectionnez **File** > **Import Task from Probe Database**.

b Dans la boîte de dialogue qui s'ouvre, sélectionnez la tâche à exécuter et cliquez sur **OK**.

c Poursuivez la procédure de définition d'une tâche. Pour plus d'informations, voir "Enregistrement de la tâche et des journaux", page 99.

8 Modification d'une tâche

Une fois qu'une tâche est définie, son nom (ou celui du fichier) est affiché dans la barre de titre. Il est alors possible de modifier le fichier.

a Sélectionnez **Edit** > **Edit Task**.

b Apportez des modifications à la tâche et cliquez sur **OK**.

9 Enregistrement de la tâche et des journaux

Vous pouvez enregistrer les paramètres de la tâche : Sélectionnez **File** > **Save Task**.

Les option suivantes ne sont disponibles qu'après l'exécution d'une tâche.

- Sauvegarde d'un enregistrement de la tâche. Vous pouvez enregistrer les paramètres de la tâche et les résultats de son exécution : Sélectionnez **File** > **Save Record**.
- Enregistrement d'un journal de la tâche : Sélectionnez **File** > **Save General Log**.
- Enregistrement des résultats : Sélectionnez **File** > **Save Results**.

10 Exécution de la tâche

L'étape suivante de la procédure consiste à exécuter la tâche que vous venez de créer.

a Importez le fichier de configuration des informations d'identification/plages. Pour plus d'informations, voir "Importation de paramètres", page 101.

b Pour exécuter la tâche uniquement sur une destination distante, cliquez sur le bouton **Run Task**.

Discovery Analyzer exécute le travail et affiche des informations dans les trois fichiers journaux : **General**, **Communication** et **Results**.

c Vous pouvez enregistrer les fichiers journaux ensemble ou séparément : Sélectionnez **File** > **Save General Log**, **Save Record**, **Save Results** ou **Save All Logs**. Pour plus d'informations sur les fichiers journaux, voir "Journaux", page 94.

d Si une tâche est extraite d'un fichier d'enregistrement, il est possible de reproduire l'exécution décrite dans ce fichier en cliquant sur le bouton **Playback**. Le même journal de communication est affiché, mais l'heure d'exécution est mise à jour.

11 Envoi des résultats de la tâche au serveur

Si l'exécution d'une tâche se termine avec des résultats (c'est-à-dire que l'onglet Results Log affiche une liste des CI découverts), vous pouvez envoyer les résultats au serveur UCMDB. Cela est utile si, par exemple, vous étiez en train de tester un script lors de l'interruption du serveur.

Remarque : Vous pouvez envoyer les résultats uniquement à un serveur UCMDB qui reçoit des tâches de la sonde installée sur la même machine que Discovery Analyzer.

---

12 Importation de paramètres

Pour exécuter des tâches ou le fichier d'enregistrement de lecture, vous devez importer le fichier **domainScopeDocument.bin**. Au cours de l'importation, un mot de passe vous est demandé.

a Lancez un navigateur Web et entrez l'URL suivante : **http://localhost:8080/jmx-console**. Vous pouvez être amené à vous connecter à l'aide d'un nom d'utilisateur et d'un mot de passe.

b Cliquez sur **UCMDB:service=DiscoveryManager** pour ouvrir la page JMX MBEAN View.

c Recherchez l'opération **exportCredentialsAndRangesInformation**. Procédez comme suit :

- Entrez l'ID du client (la valeur par défaut est **1**).
- Entrez le nom du fichier exporté.
- Entrez le mot de passe.
- Attribuez la valeur **False** à **isEncrypted**.

d Cliquez sur **Invoke** pour exporter le fichier **domainScopeDocument.bin**.

Dès que le processus d'exportation a abouti, le fichier est enregistré à l'emplacement suivant : **C:\hp\UCMDB\UCMDBServer\conf\discovery\<rép_client>**.

e Copiez le fichier **domainScopeDocument.bin** dans le système de fichiers de la sonde des flux de données et importez-le en sélectionnant : **Settings** > **Import domainScopeDocument**.

Remarque : Au cours de l'importation du fichier **domainScopeDocument**, vous êtes invité à fournir un mot de passe. Cette demande est également affichée à chaque redémarrage de Discovery Analyzer et avant l'exécution de la première tâche ou du premier enregistrement.

---

13 Points d'arrêt

Si vous exécutez Discovery Analyzer à partir du script Python, vous pouvez ajouter des points d'arrêt dans le script.

14 Configuration d'Eclipse

Pour plus d'informations sur l'exécution de vos scripts Jython en mode de débogage, voir "Exécution de Discovery Analyzer à partir d'Eclipse", page 103.

Exécution de Discovery Analyzer à partir d'Eclipse

Cette tâche explique comment configurer Eclipse afin de pouvoir exécuter vos scripts Jython en mode de débogage, ce qui donne une meilleure visibilité des threads de travaux, des CI déclencheurs et des résultats.

Contenu de cette section :

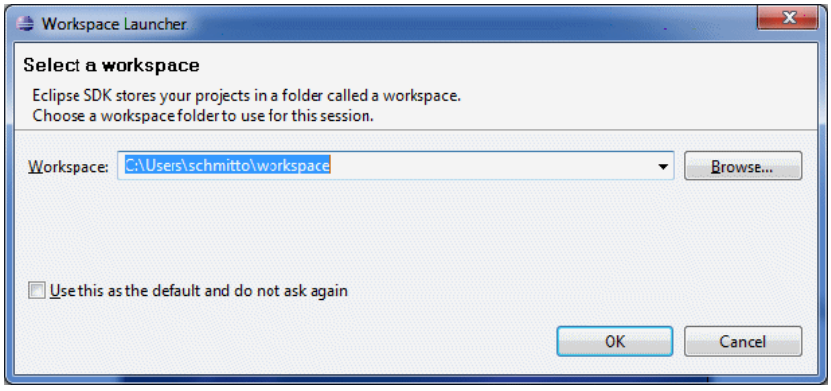
1 Conditions préalables

- Installez la dernière version d'Eclipse sur votre ordinateur. L'application est disponible à l'adresse www.eclipse.org.
- Vérifiez que la sonde des flux de données est installée sur le même ordinateur.
- Vérifiez que le paramètre **appilog.agent.local.discoveryAnalyzerFromEclipse** du fichier **DiscoveryProbe.properties** a la valeur **true**.

2 Décompression et démarrage d'Eclipse

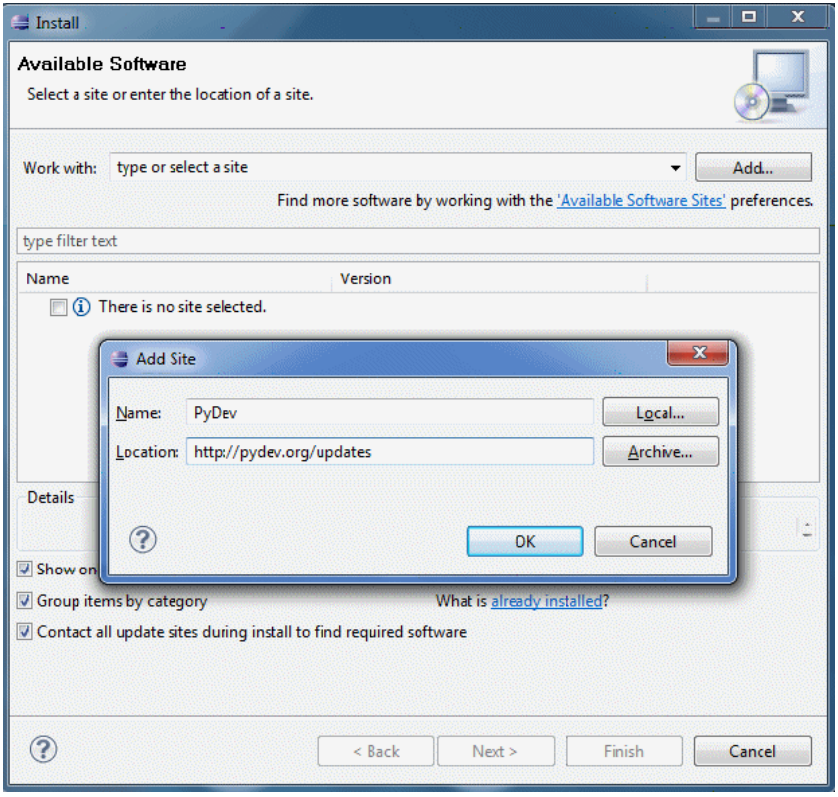
3 Configuration de l'espace de travail par défaut

Configurez l'espace de travail par défaut dans lequel Eclipse enregistre et stocke tous les projets et les données associées.



4 Configuration des extensions PyDev

a Accédez à **Help** > **Install New Software**, cliquez sur **Add**, tapez un nom pour le plug-in PyDev et, dans le champ Location, ajoutez l'URL du site depuis lequel pydev peut être téléchargé : **http://pydev.org/updates**. Cliquez sur **OK**.

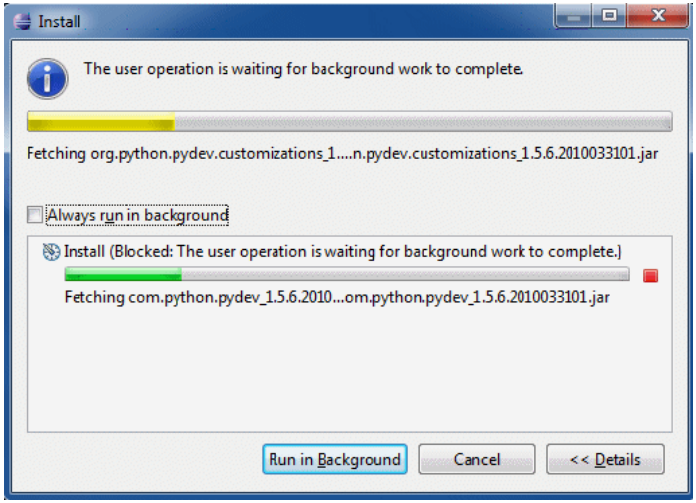


Remarque : PyDev et les extensions PyDev sont maintenant fusionnés en un seul plug-in car les extensions PyDev sont désormais Open Source. Pour plus d'informations, rendez-vous sur http://pydev.org.

b Dans la fenêtre qui s'ouvre, sélectionnez **Pydev**. Le deuxième plug-in est réservé aux interfaces utilisateur axées sur les tâches. Cliquez sur **Next**, vérifiez les détails d'installation puis cliquez de nouveau sur **Next**.

c Acceptez le contrat de licence et cliquez sur **Next**.

d Pydev est installé. Si vous êtes invité à installer un contenu non signé, confirmez en cliquant sur **OK**.

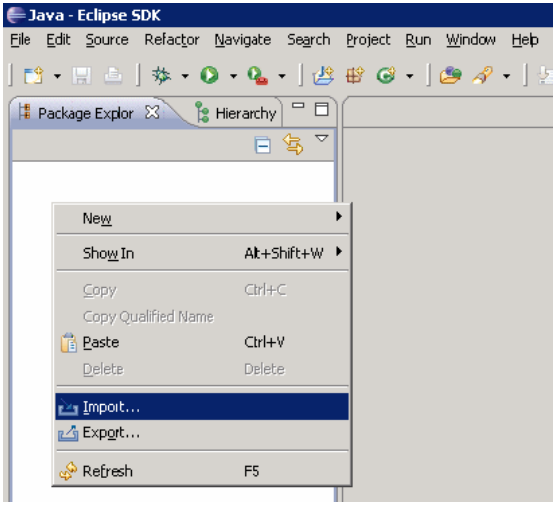


e Redémarrez Eclipse.

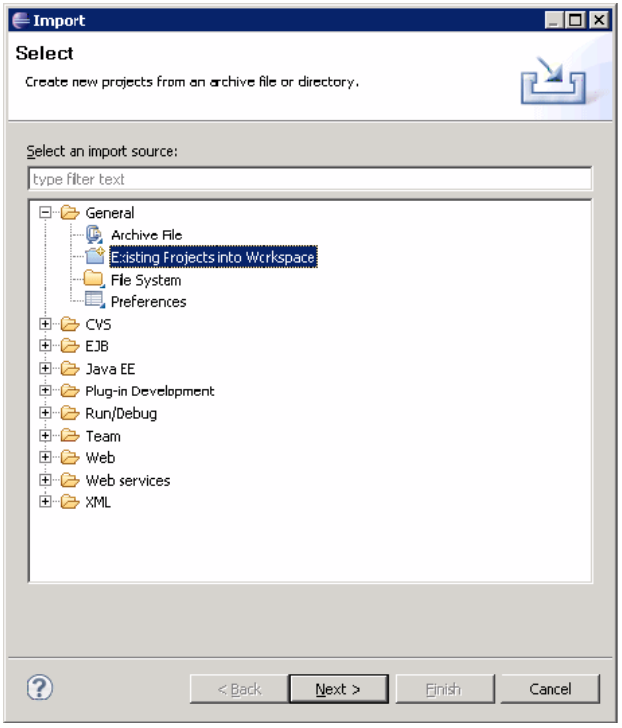
PyDev est maintenant installé dans votre IDE Eclipse. Vous disposez de nouvelles perspectives dans Eclipse et l'IDE est en mesure d'interpréter les scripts Python (texte en surbrillance, options de configuration supplémentaires, etc.).

5 Configuration de l'espace de travail Discovery Analyzer

a Importez les fichiers nécessaires : Cliquez avec le bouton droit de la souris dans la zone blanche de Package Explorer et cliquez sur **Import** pour importer l'espace de travail préconfiguré **discoveryAnalyzerWorkspace**, inclus avec l'installation de la sonde.



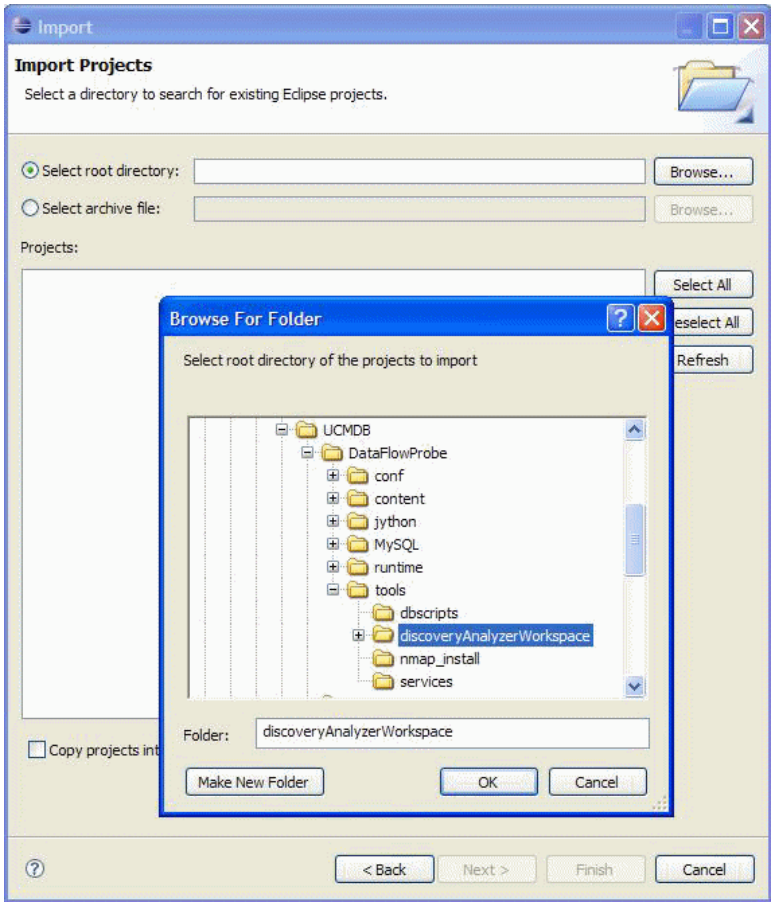
b Sous **General**, sélectionnez **Existing projects into Workspace** pour importer le projet dans l'espace de travail Eclipse.

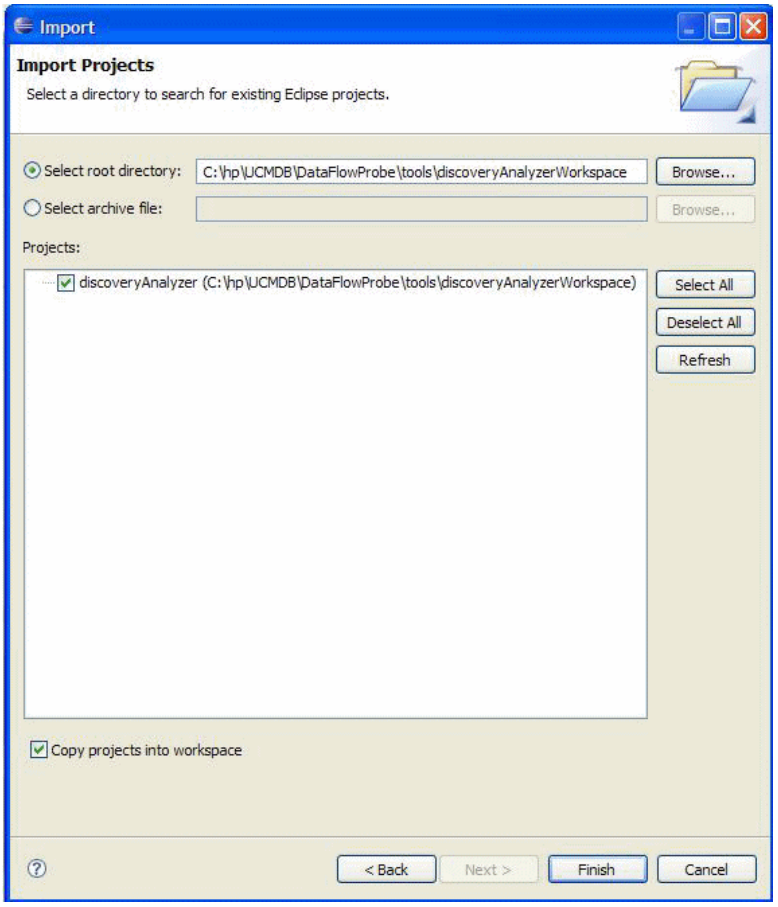


c Sous **Select root directory**, sélectionnez l'espace de travail Analyzer, généralement situé sous **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**.

d Sélectionnez **Copy projects into workspace** pour créer une copie réelle de l'espace de travail existant. Il s'agit d'une étape importante : en cas d'échec, vous pouvez réimporter l'espace **discoveryAnalyserWorkspace** d'origine.

e Cliquez sur **Finish** pour lancer l'importation.





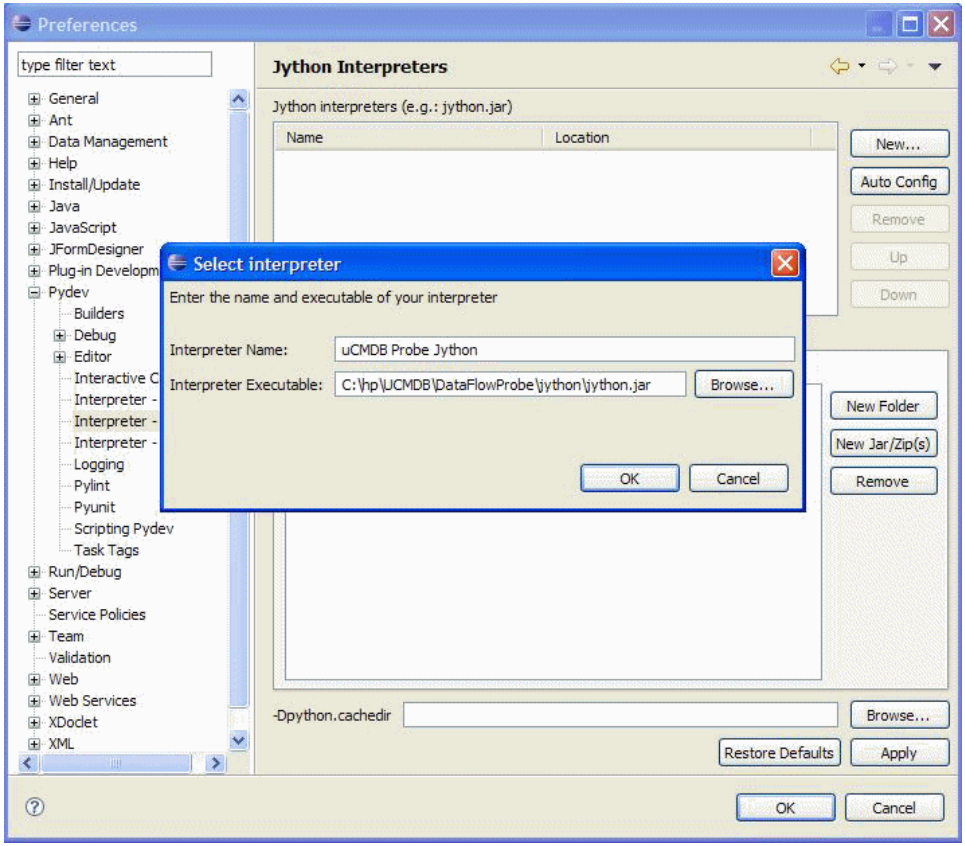
6 Configuration du classpath et de l'interpréteur

a Cliquez avec le bouton droit sur **discoveryAnalyzerWorkspace** et sélectionnez **Properties** pour afficher les paramètres propres au projet.

b Allez dans **Pydev > Interpreter/Grammar** et cliquez sur **Please configure an interpreter in the related preferences before proceeding**.

Cette étape configure le même interpréteur Jython que celui de la sonde, afin de garantir que les scripts ne soient pas interprétés par une version différente de Jython.

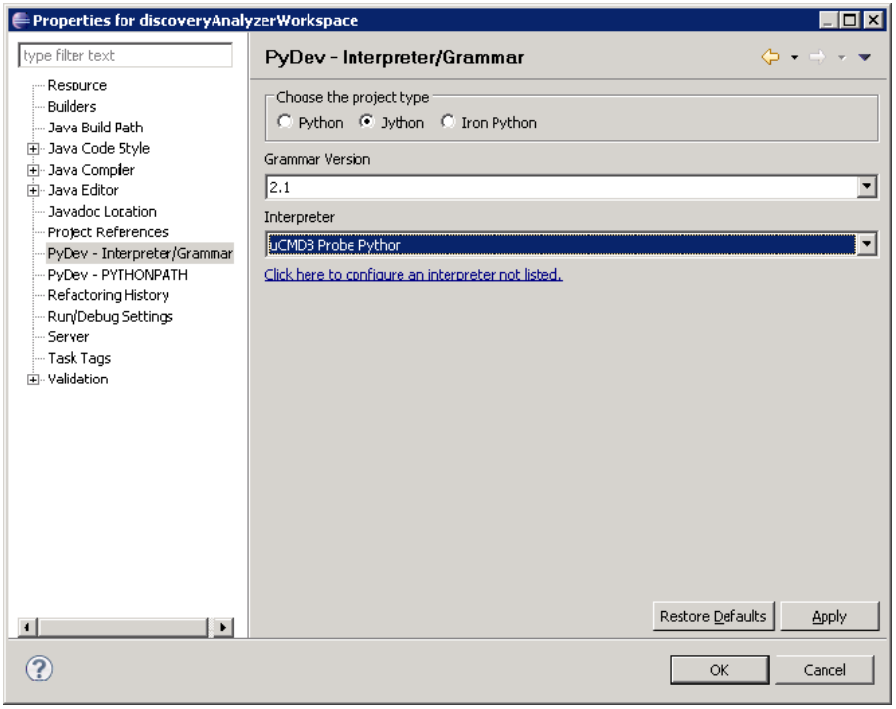
c Cliquez sur **New**, entrez un nom pour l'interpréteur et sélectionnez le fichier dans le dossier suivant :
C:\hp\UCMDB\DataFlowProbe\jython\jython.jar.



d Cliquez sur **OK**. Si une fenêtre apparaît, vous invitant à sélectionner les dossiers qui doivent être importés dans votre chemin système Python, ne changez rien (il doit s'agir des dossiers **C:\hp\UCMDB\DataFlowProbe\jython** et **C:\hp\UCMDB\DataFlowProbe\jython\lib**) et cliquez sur **OK**.

e Cliquez sur **Apply** puis sur **OK**.

f Cliquez sur **Interpreter** et sélectionnez l'interpréteur que vous venez de créer.



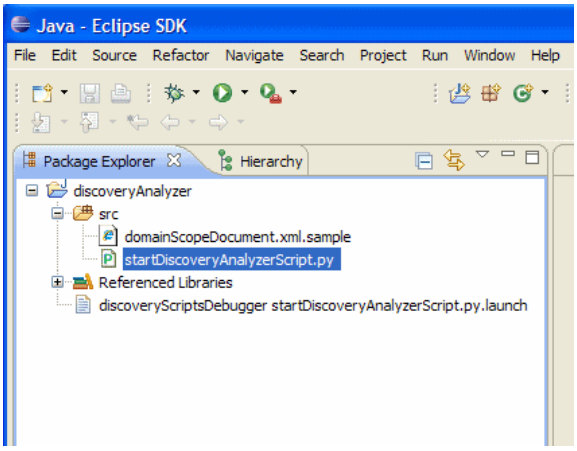
g Cliquez sur **Apply** puis sur **OK**.

L'interpréteur Jython est maintenant le même que celui qu'utilise la sonde.

7 Exécution de Discovery Analyzer

a Ajoutez un point d'arrêt dans le script Jython à déboguer.

b Pour démarrer Discovery Analyzer, sélectionnez **startDiscoveryAnalyzerScript.py** dans le projet **discoveryAnalyzerWorkspace\src**. Cliquez avec le bouton droit sur le fichier et choisissez **Debug as** > **Jython run**.

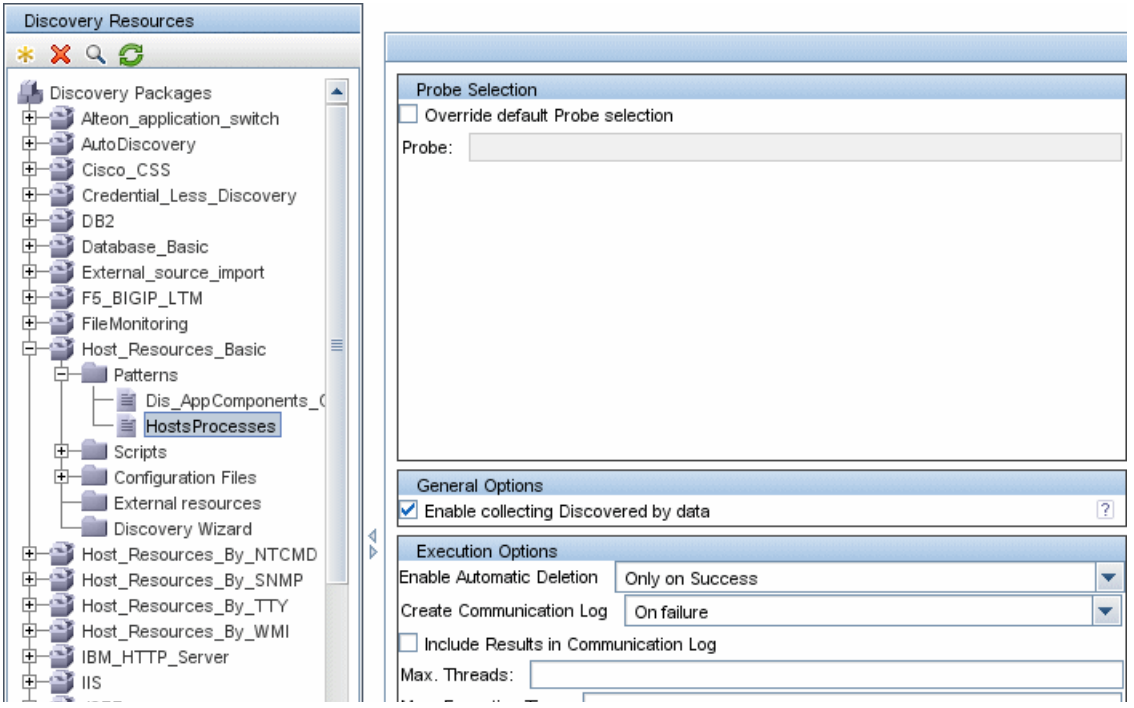


Enregistrement de code GFD

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.

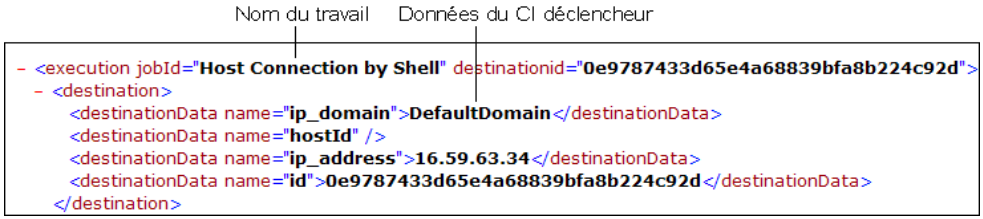
Pour enregistrer du code GFD :

1. Accédez à la page **Gestion des flux de données** > **Panneau de configuration de la découverte**. Cliquez avec le bouton droit de la souris sur le travail dont l'exécution doit être consignée et sélectionnez **Modifier l'adaptateur** pour ouvrir l'application Gestion de l'adaptateur.
2. Recherchez le volet **Options d'exécution** dans l'onglet Pattern Management:

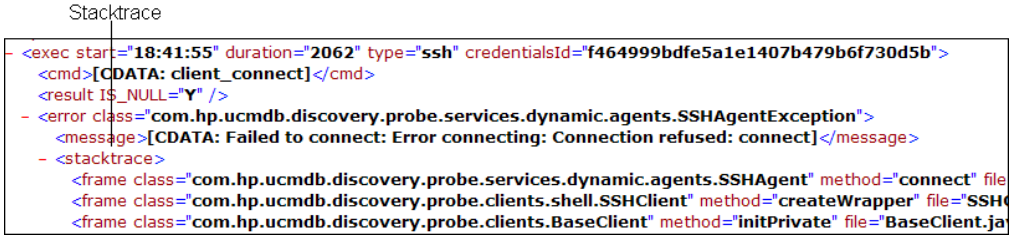


3 Dans la zone **Créer un journal de communication**, sélectionnez **Toujours**. Pour plus d'informations sur la définition des options de consignation, voir "Volet Options d'exécution" dans le manuel *Manuel de gestion des flux de données HP Universal CMDB*.

L'exemple suivant représente le fichier journal XML qui est créé lorsque le travail Host Connection by Shell est exécuté et que la zone **Créer un journal de communication** a la valeur **Toujours** ou **Sur échec**:



L'exemple suivant contient les paramètres message et stacktrace :



Références

Bibliothèques et utilitaires Jython

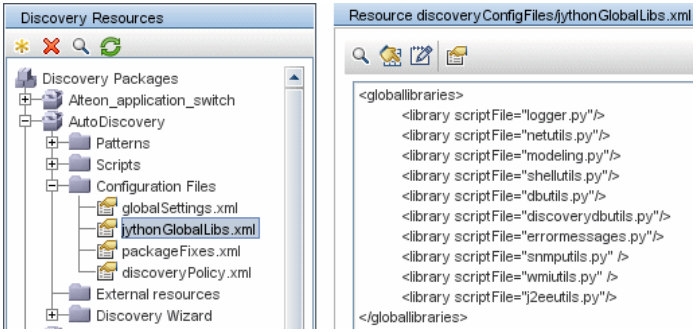
Plusieur scripts utilitaires sont couramment employés dans les adaptateurs. Ces scripts font partie du composant applicatif AutoDiscovery et résident sous :
C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts avec les autres scripts qui sont téléchargés sur la sonde.

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

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 d'erreur", page 127.

```
#############################################################################
#################       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/pr
obeMgr-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. 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. 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. Exemple :

```
ttyClient = clientFactory.createClient(Props)
clientShUtils = shellutils.ShellUtils(ttyClient)
if (clientShUtils.isWinOs()):
        logger.debug ('discovering Windows..')
```

4

Messages d'erreur

Contenu de ce chapitre :

Concepts

Messages d'erreur - Présentation

Au cours de la découverte, de nombreuses erreurs peuvent survenir, par exemple, des échecs de connexion, des problèmes matériels, des exceptions, des dépassements de délai, etc. Le processus GFD affiche ces erreurs dans le Panneau de configuration de la découverte, dans les modes Base et Avancé, dès que le flux de découverte normal n'aboutit pas. Vous pouvez explorer à partir du CI déclencheur ayant provoqué l'incident pour afficher le message d'erreur lui-même.

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 composant applicatif, vous pouvez ajouter des messages appropriés comme ressources de ce composant. Au déploiement du composant applicatif, les messages sont également déployés à l'emplacement correct. Les messages doivent être conformes aux conventions décrites à la section "Conventions d'écriture d'erreurs", page 123.

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 "Volet Statut de Découverte" 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*.

Références

Conventions d'écriture d'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

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 composant applicatif **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.
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 d'erreur

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 **Réussi**.

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.

Avertissement

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 correctement calculé en raison de données incorrectes.

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

5

Développement d'adaptateurs de base de données génériques

Contenu de ce chapitre :

Concepts

Tâches

Références

Concepts

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.

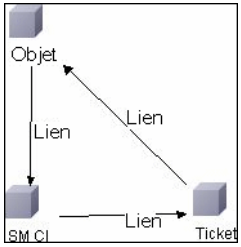
Cette version de l'implémentation d'un adaptateur de base de données repose sur une norme JPA (API de persistance Java) avec la bibliothèque Hibernate ORM comme fournisseur de persistance.

Requêtes TQL non prises en charge

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

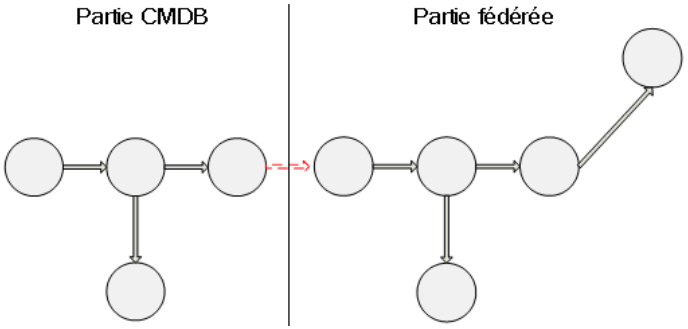
- les sous-graphiques ne sont pas pris en charge ;
- les relations composées ne sont pas prises en charge ;
- les cycles ou parties de cycle ne sont pas pris en charge ;

  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 137.
- Le rapprochement avec l'ID CMDB 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 224.

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 135.

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.

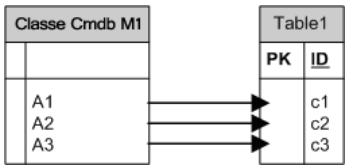
Cette section comprend aussi les rubriques suivantes :

Exemples de mappage relationnel-objet

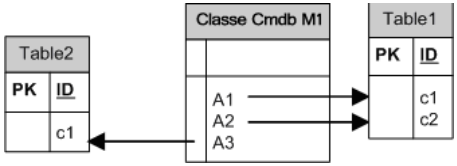
Les exemples suivants décrivent un mappage relationnel-objet :

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.

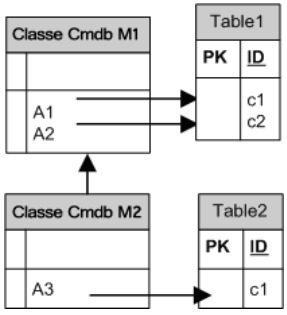


Exemple d'une classe CMDB mappée sur deux tables de base de données :



Exemple d'héritage :

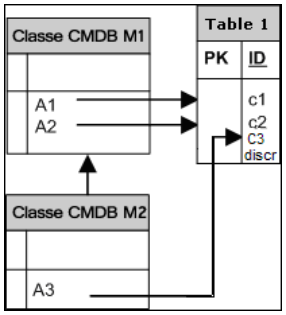
Ce cas est utilisé dans la base CMDB, où chaque classe possède sa propre table de base de données.



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.

Lorsque vous utilisez les fonctions de discriminateur, vous ne pouvez pas sauter une classe de la hiérarchie ; autrement dit, comme C3 hérite de C2, qui hérite elle-même de C1, vous ne pouvez pas vous contenter de définir C1 et C3, vous devez définir les trois classes.



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 les définitions.

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.

Tâches

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 137

➤ "Création d'un type de CI", page 139

➤ "Création d'une relation", page 140

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 132.

ID	NOM	ADRESSE IP
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 133.

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
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 fédéré qui doit être mappé sur des données dans le SGBDR (la source de données externe).

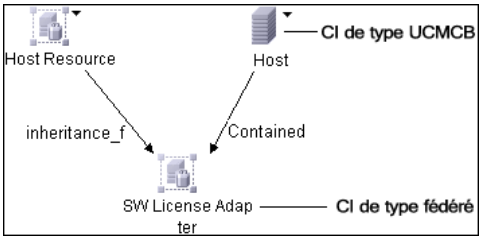
a Dans UCMDB, accédez au Gestionnaire des types de CI et créez un nouveau type de CI. Pour plus d'informations, voir "Créer un type de CI" dans le *Manuel de modélisation HP Universal CMDB*.

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.

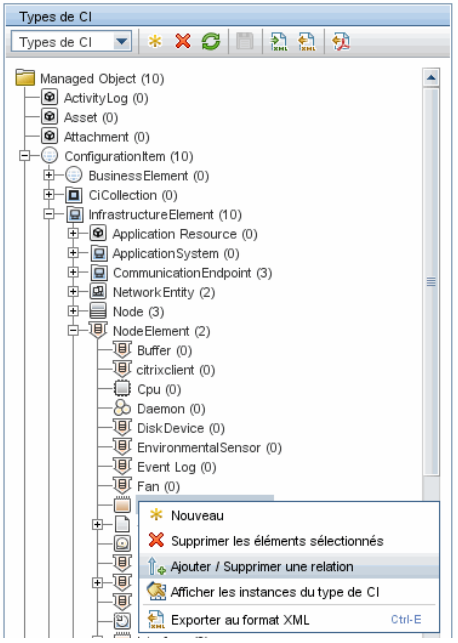
Ajoutez des relations appropriées valides au nouveau type de CI. Pour plus d'informations, voir "Boîte de dialogue Ajouter/Supprimer une relation" dans le *Manuel de modélisation HP Universal CMDB*.



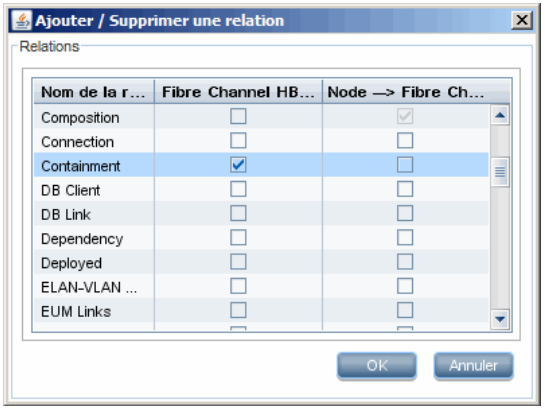
Remarque : À ce stade, vous ne pouvez pas encore visualiser les données fédérées, puisque vous n'avez pas encore défini la méthode d'importation des données.

Exemple de création d'une relation contenant-contenu :

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



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



Préparation du composant applicatif de l'adaptateur

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

1. Recherchez le composant applicatif **db-adapter.zip** dans le dossier **C:\hp\UCMDB\UCMDBServer\content\adapters**.
2. Extrayez le composant applicatif 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:noNamespaceSchemaLocation="../../Patterns.xsd" description="Discovery
Pattern Description"
        schemaVersion="9.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" displayName="UCMDB API Population">
```

Si l'adaptateur prend en charge les données de réplication, 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 Points d'intégration dans HP Universal CMDB.

Pour plus d'informations sur l'insertion de données dans 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.AdapterMa
ppingEngine</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.

---

4 Dans le répertoire temporaire, ouvrez le dossier **adapterCode** et attribuez à **GenericDBAdapter** la valeur d'**adapter id** utilisée à l'étape 3.

Ce dossier contient les fichiers jar qui exécutent la logique de fédération, par exemple, le nom de l'adaptateur, les requêtes et les classes dans la base CMDB et 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", page 145.

6 Créez un fichier *.zip portant le même nom que celui donné à l'attribut **adapter id** à l'étape 3 page 142.

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

---

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

Mise à niveau de l'adaptateur de base de données générique de la version 9.00 ou 9.01 aux versions 9.02 et ultérieures

1 Copiez le composant applicatif de votre adaptateur dans un répertoire temporaire local.

2 Extrayez les fichiers.

3 Supprimez les fichiers suivants du dossier **adapterCode\<Nom de votre adaptateur>** :

- **asm.jar**
- **asm-attrs.jar**
- **cglib.jar**
- **db-adapter.jar**
- **jboss-archive-browsing.jar**
- **saxon-b.jar**

4 Recréez le composant applicatif de votre adaptateur.

Remarque : Pour les adaptateurs de base de données générique déployés dont vous disposez, le programme d'installation UCMDB supprime les fichiers nécessaires de la base UCMDB et du système de fichiers de la sonde. Cependant, vous devez quand même corriger vous-même le composant applicatif, afin de le redéployer en cas de besoin.

Configuration de l'adaptateur

Vous pouvez utiliser l'une des méthodes suivantes pour configurer l'adaptateur :

- "Configuration de l'adaptateur – Méthode minimale", page 145
- "Configuration de l'adaptateur – Méthode avancée", page 149

Ces fichiers de configuration figurent dans le composant applicatif **db-adapter.zip**, dans le dossier **C:\hp\UCMDB\UCMDBServer\content\adapters** que vous avez extrait à l'étape 2 de la section "Préparation du composant applicatif de l'adaptateur", page 142.

Configuration de l'adaptateur – Méthode minimale

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.

La procédure suivante décrit une méthode de mappage du modèle de classe dans la base CMDB sur un SGBDR. Cette méthode minimale est recommandée pour :

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

Cette méthode :

- ne prend en charge que la fédération sur un nœud ;
- ne prend en charge que les relations virtuelles de plusieurs à un.

Cette tâche comprend les étapes suivantes :

- "Configuration du fichier adapter.conf", page 146
- "Configuration du fichier simplifiedConfiguration.xml", page 146

Configuration du fichier adapter.conf

Au cours de cette étape, vous modifiez les paramètres du fichier adapter.conf afin que les données soient fédérées automatiquement.

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**.

Configuration du fichier simplifiedConfiguration.xml

Au cours de cette étape, vous configurez le fichier **simplifiedConfiguration.xml** en mappant le type de CI dans la base CMDB sur les champs de la table du SGBDR.

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

   Ce fichier comprend un modèle que vous pouvez utiliser pour chaque entité à mapper.

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

---

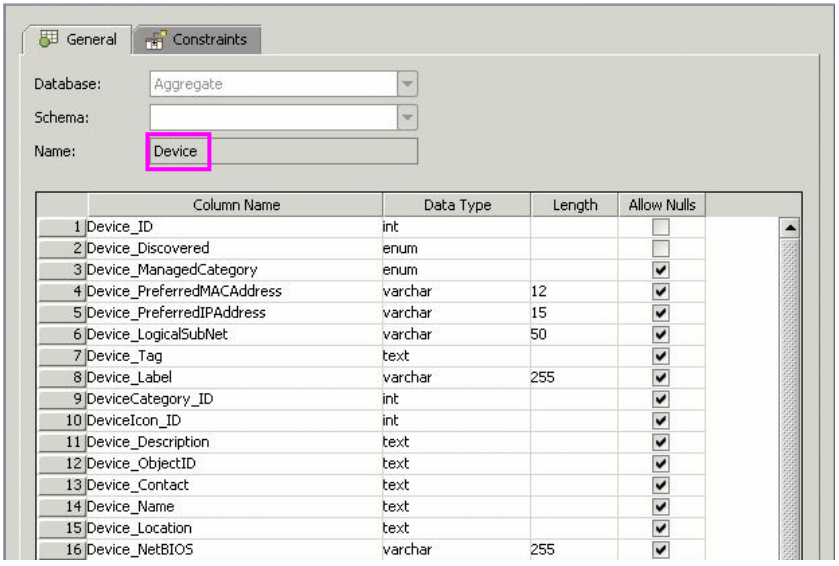
2 Apportez des modifications aux attributs suivants :

➤ le nom du type de CI dans UCMDB (cmdb-class-name) et le nom de table correspondant dans le SGBDR (default-table-name) :

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

l'attribut cmdb-class-name provient du type de CI node :

l'attribut default-table-name provient de la table Device :



- l'identifiant unique dans le SGBDR :

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

- la règle de rapprochement (reconciliation-by-two-nodes) :

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

- l'attribut de rapprochement dans UCMDB (cmdb-attribute-name) et dans le SGBDR (column-name) :

```
<connected-node-attribute cmdb-attribute-name="name"
column-name="[nom_colonne]"/>
```

- le nom du type de CI (cmdb-class-name) et le nom de la table correspondante dans le SGBDR (default-table-name) ; également la relation CMDB (connected-cmdb-class-name) et la relation du type de CI (link-class-name) :

```
<class cmdb-class-name="sw_sub_component"
default-table-name="SWSubComponent" connected-cmdb-class-name="node"
link-class-name="composition">
```

- la clé principale et la clé étrangère :

```
<foreign-primary-key column-name="Device_ID"
cmdb-class-primary-key-column="Device_ID"/>
```

- l'identifiant unique dans le SGBDR :

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

- le mappage entre l'attribut CMDB (cmdb-attribute-name) et le nom de colonne dans le SGBDR (column-name) :

```
<attribute cmdb-attribute-name="last_access_time"
column-name="SWSubComponent_LastAccess TimeStamp"/>
```

3 Enregistrez le fichier.

Configuration de l'adaptateur – Méthode avancée

Cette tâche comprend les étapes suivantes :

- "Configuration du fichier orm.xml", page 149
- "Configuration du fichier reconciliation_types.txt", page 154
- "Configuration du fichier reconciliation_rules.txt", page 154

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 pour la fédération.

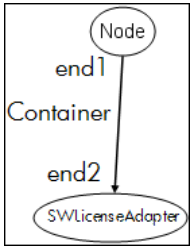
Remarque : N'éditez le fichier **orm.xml** dans aucune version du Bloc-notes de Microsoft Corporation. Utilisez Notepad++, UltraEdit ou un autre éditeur de texte tiers.

---

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** :

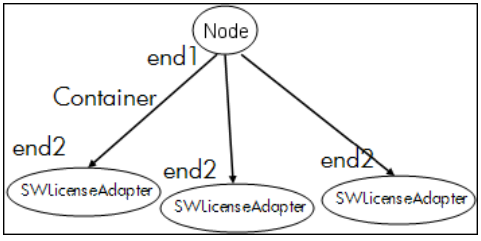
- De un à un :



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>
```

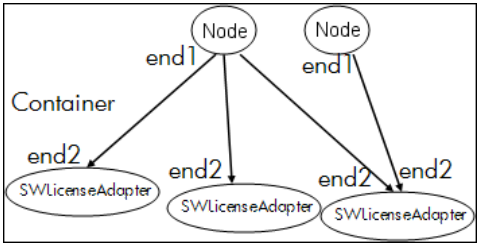
- De plusieurs à un :



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 189.

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 139:

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

- Nom de la table correspondante dans le SGBDR :

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

- Identifiant 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>
```

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

```
<basic name="license_required">
        <column updatable="false" insertable="false"
name="MyAdapter_LicenseRequired"/>
```

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="[nom-colonne]" name="Device_ID"/>
```

Configuration du fichier reconciliation_types.txt

Ouvrez le fichier **reconciliation_types.txt** dans un éditeur de texte.

Pour plus d'informations, voir "Fichier reconciliation_types.txt", page 197.

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 198.

Implémentation d'un plug-in

Cette tâche décrit comment implémenter et déployer un adaptateur de BD générique avec des plug-in.

Remarque : Avant d'écrire un plug-in pour un adaptateur, assurez-vous d'avoir effectué toutes les étapes nécessaires dans "Préparation du composant applicatif de l'adaptateur", page 142.

---

1 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**.
- Copiez les fichiers **federation-api.jar** et **federation-api-javadoc.jar** depuis le dossier **\tools\adapter-dev-kit\SampleAdapters\production-lib**.

Remarque : Vous trouverez plus d'informations sur le développement d'un plug-in dans les fichiers **db-interfaces-javadoc.jar** et **federation-api-javadoc.jar** et dans la documentation en ligne suivante :

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

---

2 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	FcmdbPluginForSyncGetFull Topology	getFullTopology
Synchronisation des modifications	FcmdbPluginForSyncGet ChangesTopology	getChangesTopology
Synchronisation de la mise en page	FcmdbPluginForSyncGet Layout	getLayout
Extraction des requêtes prises en charge	FcmdbPluginForSyncGet SupportedQueries	getSupportedQueries
Modification de la définition de requête TQL et des résultats	FcmdbPluginGetTopology CmdbFormat	getTopologyCmdbFormat
Modification de la demande de mise en page de CI	FcmdbPluginGetCIsLayout	getCisLayout
Modification de la demande de mise en page de liens	FcmdbPluginGetRelations Layout	getRelationsLayout

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.

3 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**.

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

Les plug-in sont configurés à l'aide du fichier **plugins.txt**, situé 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]
```

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.

5 Compressez votre adaptateur avec le nouveau fichier jar et le fichier **plugins.xml** mis à jour. Les autres fichiers du composant applicatif 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 composants applicatifs. Pour plus d'informations, voir "Page Gestionnaire des composants applicatifs" dans le *Manuel d'administration HP Universal CMDB*.

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

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

Modification de l'adaptateur

Après avoir créé et déployé l'adaptateur, vous pouvez le modifier dans UCMDB. Pour plus d'informations, voir "Gestion des adaptateurs", page 115.

Création d'un point d'intégration

Au cours de cette étape, vous contrôlez que la fédération fonctionne, c'est-à-dire que la connexion et le fichier XML sont valides. 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 "Boîte de dialogue Nouveau point d'intégration/Modifier le point d'intégration" 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 "Onglet Fédération" dans le *Manuel de gestion des flux de données HP Universal CMDB*.

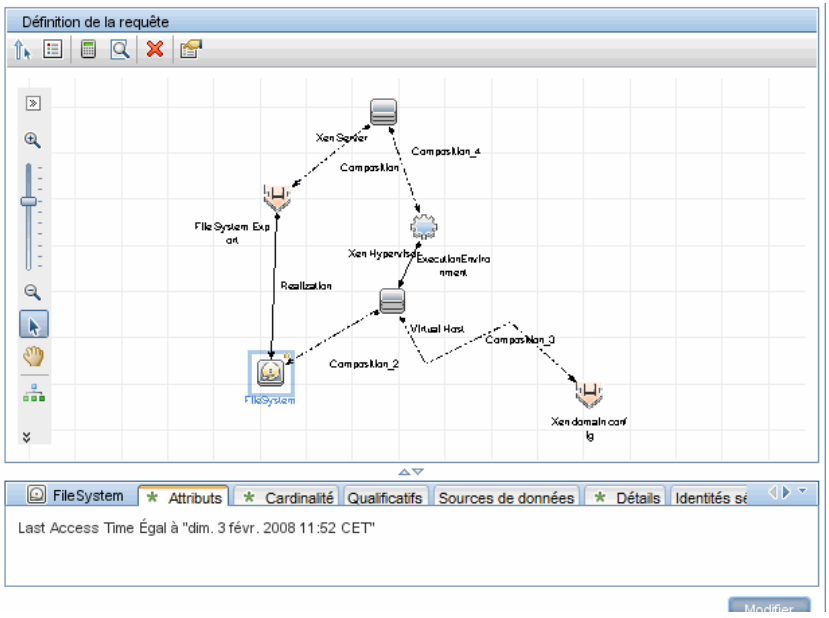
Création d'une vue

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 "Créer une vue basée sur maquette" dans le *Manuel de modélisation HP Universal CMDB*.

3 Vous pouvez ajouter des conditions au code TQL, par exemple, un dernier accès antérieur à six mois :



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 de rapports

Cette étape consiste à afficher les rapports topologiques. Pour plus d'informations, voir "Présentation des rapports topologiques" 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 220.

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 décrit 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.

Cette tâche comprend les étapes suivantes :

1 Conditions préalables

Installez **Java Runtime Environment (JRE) 6 Update 7** sur la machine sur laquelle vous allez exécuter Eclipse, à partir du site suivant : **http://java.sun.com/javase/downloads/index.jsp**.

La procédure fonctionne avec l'environnement d'exécution Java 5 (ou ultérieur).

2 Installation

a Téléchargez et extrayez **Eclipse IDE for Java EE Developers** à partir du site <**http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR1/eclipse-jee-ganymede-SR1-win32.zip**> dans 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** (exige au moins un environnement d'exécution Java 5). 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

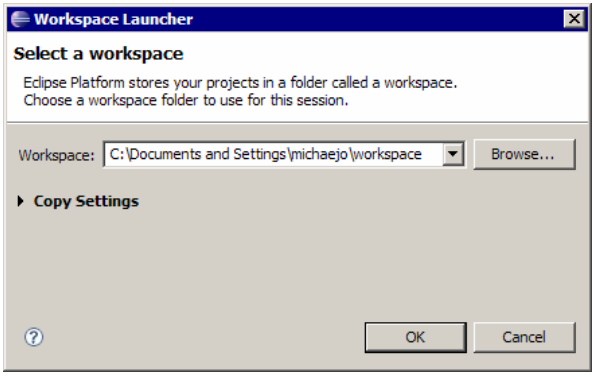
Cette étape consiste à configurer l'espace de travail, la base de données, les connexions et les propriétés des pilotes.

a Extrayez le fichier **workspaces_gdb.rar** à partir de **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** dans **C:\Documents and Settings\All Users\workspaces**.

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** :

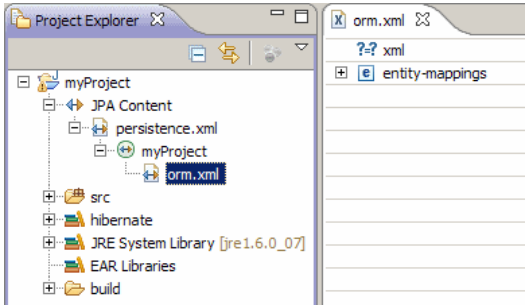


Si vous travaillez avec :

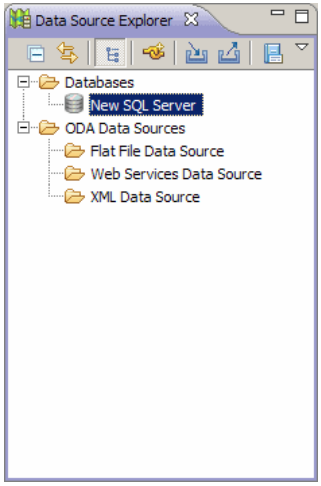
- SQL Server, sélectionnez le dossier suivant : **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.
- MySQL, sélectionnez le dossier suivant : **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
- Oracle, sélectionnez le dossier suivant : **C:\Documents and Settings\All Users\workspace_gdb_oracle**.

c Cliquez sur **OK**.

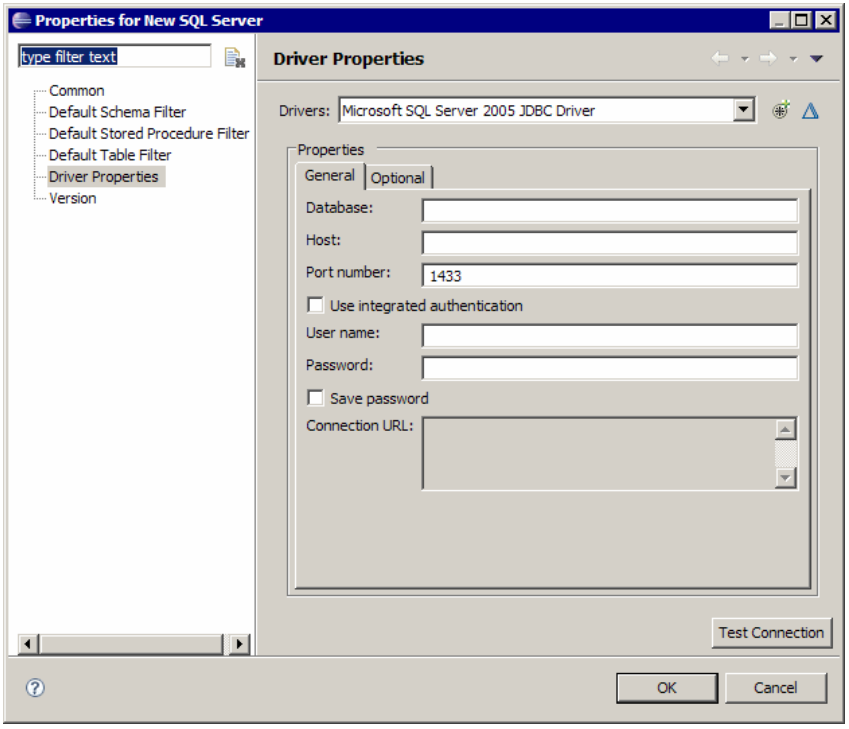
d Dans Eclipse, affichez la vue Project Explorer et sélectionnez **<Active project> > JPA Content > persistence.xml > <nom du projet actif> > orm.xml**.



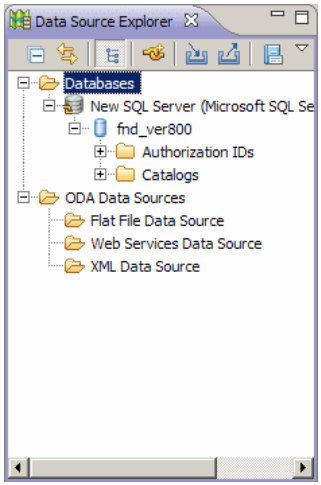
e Dans la vue Data Source Explorer (volet inférieur gauche), cliquez avec le bouton droit sur la connexion de base de données et sélectionnez le menu **Propriétés**.



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.

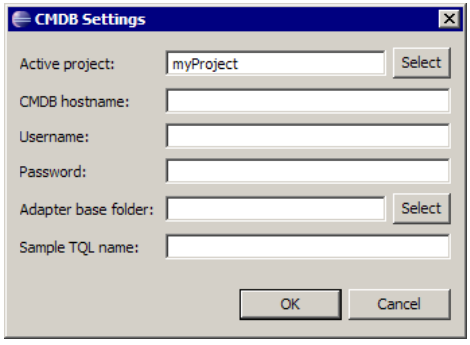


4 Création d'un adaptateur

Créez un adaptateur à l'aide des instructions de la section "Étape 1 : Création d'un adaptateur", page 43.

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**.

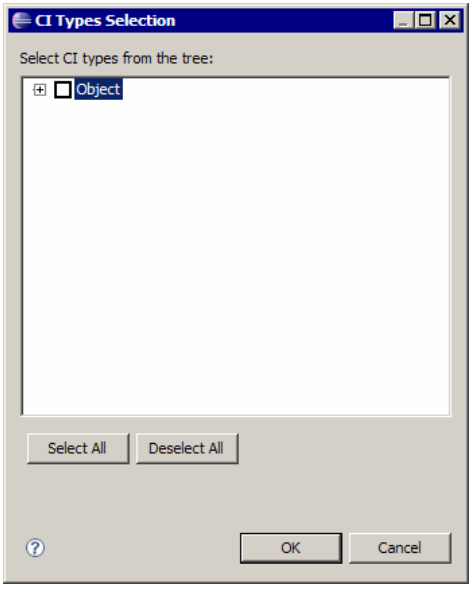
e Assurez-vous que le dossier **C:\hp** sur le serveur CMDB est mappé comme lecteur réseau.

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 d'accès 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.

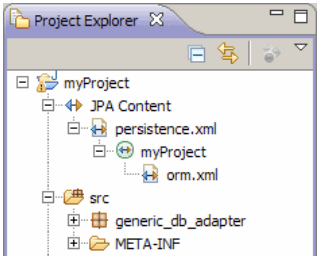
6 Importation du modèle de classe UCMDB

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 Type Selection** :



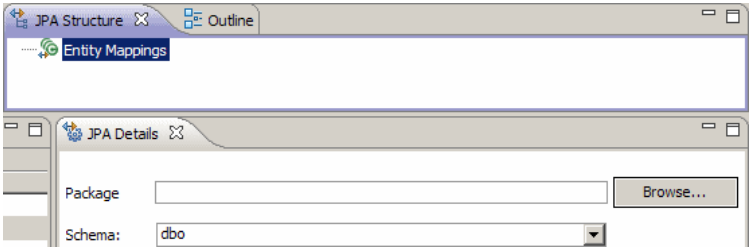
b Sélectionnez les types de CI que vous comptez mapper comme entités JPA. Cliquez sur **OK**. Les types de CI sont importés en tant que classes Java. Vérifiez qu'ils apparaissent sous le dossier **src** du projet actif :



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

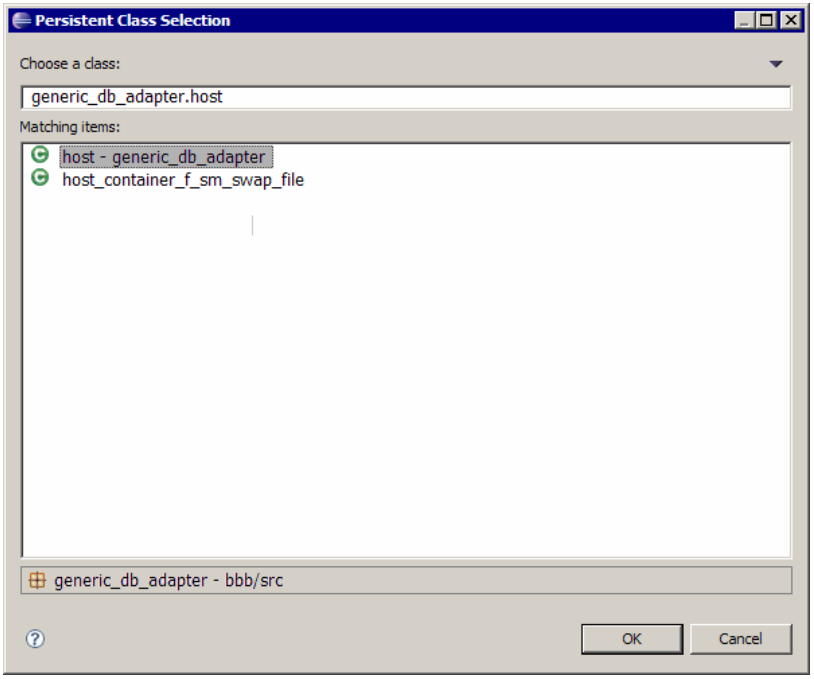
Au cours de cette étape, vous mappez les classes Java (que vous avez importées à l'étape précédente) sur les tables de base de données.

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 composant applicatif **generic_db_adaptor**).

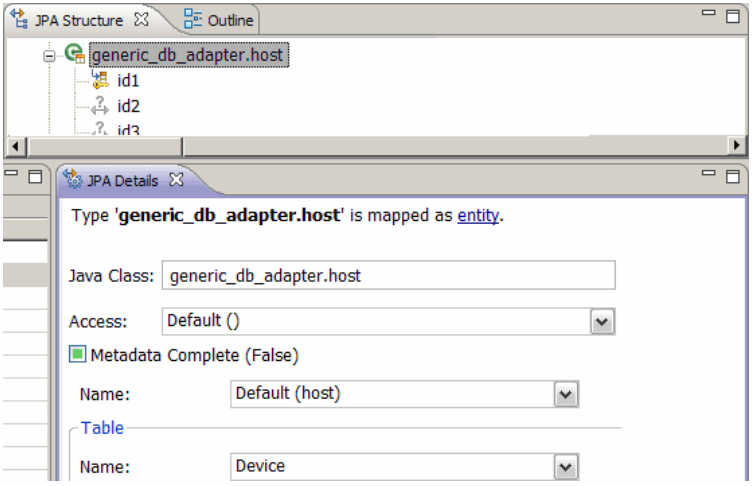


d Cliquez sur **OK** dans les deux boîtes de dialogue. La classe sélectionnée est affichée sous la branche **Entity Mappings** dans la vue JPA Structure.

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**.

---

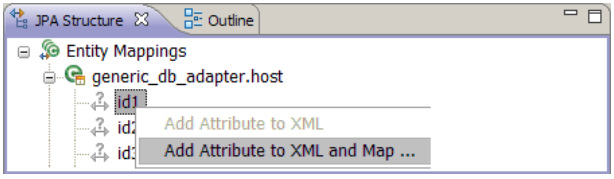
e Dans la vue JPA Details, sélectionnez la table de base de données principale à laquelle la classe UCMDB doit être mappée. Ne modifiez pas les autres champs.



8 Mappage d'ID

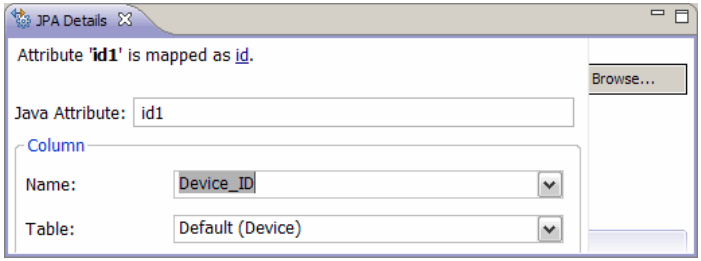
Selon les normes JPA, chaque classe persistante doit comporter au moins un attribut ID. Pour les classes UCMDB, vous pouvez mapper jusqu'à trois attributs comme ID. Les attributs ID potentiels sont appelés **id1**, **id2** et **id3**. Pour mapper un attribut ID :

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**), puis 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**….

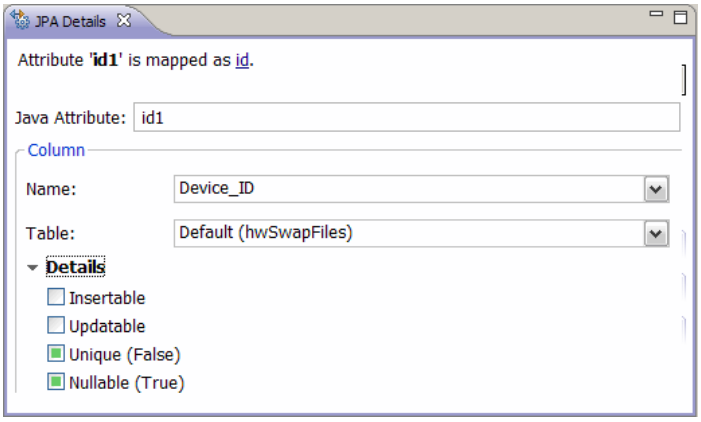
b La boîte de dialogue **Add Persistent Attribute** s'ouvre. Sélectionnez **Basic** dans le champ **Map as** et cliquez sur **OK**.

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 à l'étape b page 169 pour mapper une classe UCMDB dénotant 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 198.

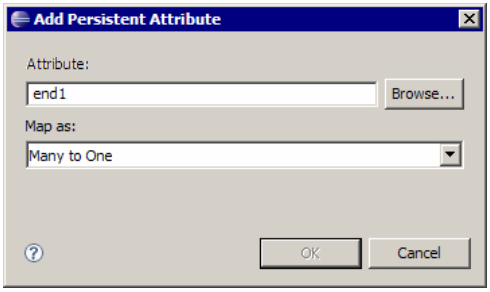
a Mappez les attributs d'ID de la classe de lien comme décrit à l'étape "Mappage d'ID", page 171. Pour chaque attribut d'ID, développez le groupe de cases à cocher **Details** dans la vue JPA Details et désactivez les cases **Insertable** et **Updateable**.



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...**.

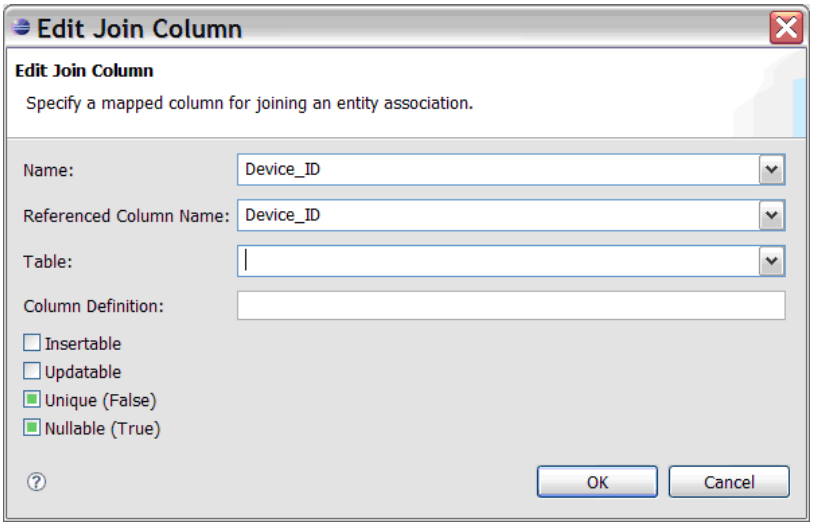
➤ Dans la boîte de dialogue **Add Persistent Attribute**, sélectionnez **Many to One** ou **One to One** dans le champ **Map as**.



➤ 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**.



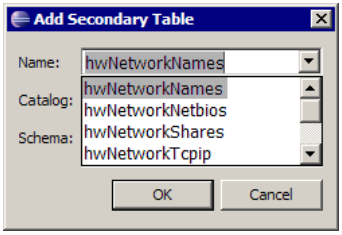
➤ Si l'entité cible **end1/end2** a plusieurs ID, cliquez sur le bouton **Add** pour ajouter des colonnes de jointure supplémentaires et les mapper de la même manière qu'à l'étape précédente.

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 à l'étape "Mappage d'attributs", page 172.

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.

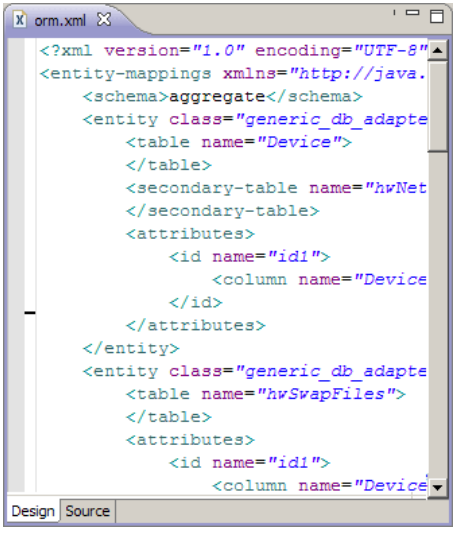
14 Utilisation d'un fichier ORM existant comme base

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 à l'étape b page 169. 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

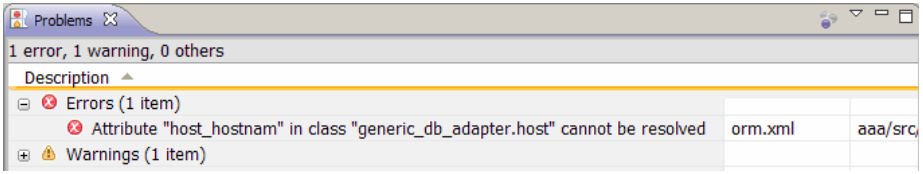
S'il existe déjà un adaptateur, il est possible d'utiliser le plug-in Eclipse pour modifier graphiquement son fichier ORM. Importez le fichier ORM dans Eclipse, éditez-le à l'aide du plug-in puis redéployez-le sur la machine UCMDB. Pour importer le fichier ORM, clique sur le bouton approprié dans la 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 179.

16 Vérification de l'exactitude du fichier ORM – Contrôle d'exactitude intégré

Le plug-in JPA Eclipse vérifie la présence d'erreurs 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 nouveau 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 "Studio d'intégration" 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 Multinœud, s'ils sont configurés dans le fichier reconciliation_rules.txt. Pour plus d'informations, voir "Fichier reconciliation_rules.txt (pour compatibilité rétroactive)", page 198.

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 à l'aide de du Gestionnaire des requêtes et non du Gestionnaire des vues.

b Créez un point d'intégration à l'aide de l'adaptateur **GenericDBAdapter**. Pour plus d'informations, voir "Boîte de dialogue Nouveau point d'intégration/Modifier le point d'intégration" 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 "Configuration du plug-in CMDB", page 166.

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**.

Références

Fichiers de configuration de l'adaptateur

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

Cette section comprend les rubriques suivantes :

Configuration générale

- **adapter.conf.** Fichier de configuration de l'adaptateur. Pour plus d'informations, voir "Fichier adapter.conf", page 181.

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 185.
- **reconciliation_types.txt**. Contient les règles utilisées pour configurer les types de rapprochement. Pour plus d'informations, voir "Fichier reconciliation_types.txt", page 197.
- **reconciliation_rules.txt**. Contient les règles de rapprochement. Pour plus d'informations, voir "Fichier reconciliation_rules.txt (pour compatibilité rétroactive)", page 198.
- **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 200.

- **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 202.
- **Replication_config.txt.** Ce fichier contient une liste séparée par des virgules de types de CI et de relations 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 204.
- **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 204.

Configuration Hibernate

- **persistence.xml.** Utilisé pour remplacer les configurations Hibernate prêtes à l'emploi. Pour plus d'informations, voir "Fichier persistence.xml", page 201.

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 182.

 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 sont remplacés par un moins grand nombre 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 persistence.xml.

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

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 permettant de modifier le fichier, accédez à **Gestion de l'adaptateur** > **db-adapter** > **Fichiers de configuration**.

Cette section comprend les rubriques suivantes :

- "Modèle de fichier simplifiedConfiguration.xml", page 182
- "Limites", page 185

Modèle de fichier simplifiedConfiguration.xml

La propriété **CMDB-class-name** est le type Multinœud (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-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="[nom_table]">
        <primary-key column-name="[nom_colonne]"/>
```

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="[nom_colonne]"
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="[nom_colonne]"/>
```

S'il y a peu de colonnes de clé principale, dupliquez cette colonne.

```
<primary-key column-name="[nom_colonne]"/>
```

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. Par exemple, le nom de nœud dans CMDB porte le suffixe mer.com.

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="[nom_colonne]"
from-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.i
mpl.GenericEnumTransformer(generic-enum-transformer-example.xml)"
to-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)"/>
        <attribute CMDB-attribute-name="[CMDB_attribute_name]"
column-name="[nom_colonne]"/>
        <attribute CMDB-attribute-name="[CMDB_attribute_name]"
column-name="[nom_colonne]"/>
    </class>
</generic-DB-adapter-config>
```

Limites

- Peut servir à mapper les requêtes TQL sur un nœud seulement (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 nouveau fichier figure dans le répertoire **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INFMETA-INF**.

Pour éditer le fichier XML pour un adaptateur déployé, accédez à **Gestion de l'adaptateur** > **db-adapter** > **Fichiers de configuration**.

Cette section comprend les rubriques suivantes :

- "Modèle de fichier orm.xml", page 186
- "Plusieurs fichiers ORM", page 189
- "Conventions de dénomination", page 189
- "Utilisation d'instructions SQL en ligne au lieu de noms de table", page 190
- "Schéma orm.xml", page 191

Modèle de fichier orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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 http://java.sun.com/xml/ns/persistence/
orm_1_0.xsd">
    <description>ORM de l'adaptateur de BD générique</description>
```

Ne modifiez pas le nom du composant applicatif.

```
    <package>generic_db_adapter</package>
```

entity. Nom du type de CI CMDB. Il s'agit de l'entité Multinœud.

Assurez-vous que **class** comprend un préfixe **generic_db_adapter.**.

```
    <entity class="generic_db_adapter.node">
        <table name="[nom_table]"/>
```

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="[nom_colonne]"/>
```

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="[nom_colonne]"/>
            <generated-value strategy="TABLE"/>
        </id>
```

basic. Sert à déclarer les attributs CMDB. Veillez à ne modifier que les propriétés **name** and **column_name**.

```
        <basic name="name">
            <column updatable="false" insertable="false" name="[nom_colonne]"/>
        </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_class_name]">
        <table name="[default_table_name]"/>
        <secondary-table name=""/>
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[nom_colonne]"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="[nom_colonne]"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="[nom_colonne]"/>
                <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="[nom_colonne]"/>
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[nom_colonne]"/>
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[nom_colonne]"/>
            </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_adapter.node_composition_[CMDB_class_name]">
        <table name="[default_table_name]"/>
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[nom_colonne]"/>
                <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="[nom_colonne]" name="[nom_colonne]"/>
            </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="[nom_colonne]"/>
            </one-to-one>
        </attributes>
    </entity>
</entity-mappings>
```

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. Exemple :

```
    <entity class="generic_db_adapter.node">
        <table name="(select d.id as id1, d.name as name , d.os as host_os from
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**. Vous trouverez le schéma complet à l'adresse 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.

Élément		Attributs
Nom et chemin d'accès	**Description**	
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.	

Élément		Attributs
Nom et chemin d'accès	**Description**	
package (entity-mappings)	Nom du composant applicatif Java qui va contenir les classes de mappage. Doit toujours contenir le texte generic_db_adapter.	**Nom**. name **Description.** Nom du type de CI UCMDB sur lequel cette entité est mappée. Si cette entité est mappée sur un lien dans la base CMDB, le nom de l'entité doit avoir le format <end_1>_<nom_lien>_<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 composant applicatif, ce champ peut être omis. **Obligatoire.** Optional **Type.** String
		Nom. class **Description.** Nom complet de la classe Java qui sera créée pour cette entité de base de données. Le nom du composant applicatif 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

Élément		Attributs
Nom et chemin d'accès	**Description**	
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 quelconque. 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 quelconque. S'il s'agit d'une instruction SELECT, elle doit être encapsulée entre parenthèses. **Obligatoire.** Required **Type.** String

Élément		Attributs
Nom et chemin d'accès	**Description**	
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
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

Élément		Attributs
Nom et chemin d'accès	**Description**	
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.	
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 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

Élément		Attributs
Nom et chemin d'accès	**Description**	
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.	**Nom**. name **Description.** Nom du champ. **Obligatoire.** Required **Type.** String
		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
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.	**Nom**. name **Description.** Indique laquelle des deux extrémités est représentée par ce champ. **Obligatoire.** Required **Type.** end1 ou end2
		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

Élément		Attributs
Nom et chemin d'accès	**Description**	
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.	**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
		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

Fichier reconciliation_types.txt

Ce fichier permet de configurer les types de rapprochement.

Chaque ligne du fichier représente un type de CI CMDB qui est connecté à un type de CI de base de données fédérée dans la requête TQL.

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. Exemple :

```
multinode[node] expression[^node.name OR ip_address.name] end1_type[node]
end2_type[ip_address] link_type[containment]
```

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 le code 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 ensuite 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 le code TQL de rapprochement contient deux nœuds au lieu d'un simple multinœud. Dans ce cas, le code 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 node AND/OR name 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.

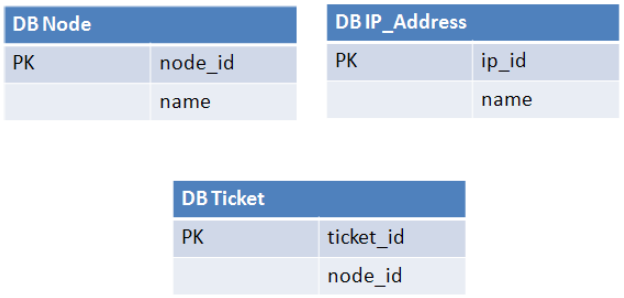
Remarque : En conséquence, il existe une limite selon laquelle tous les attributs de rapprochement de la source de données doivent résider dans des tables qui partagent la même clé principale.

Une autre limite indique que le code TQL de rapprochement ne doit pas comporter plus de deux nœuds. Par exemple, le code 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) :



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.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)]
```

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.FcmdbDalTransformerToExternalDB**. 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 204.

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 MSSQL 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**.

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.mercury.jdbc.MercOracleDriver"/>
            <!--The connection url"/-->
            <property name="hibernate.connection.url" value="jdbc:mercury:oracle://
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>
```

 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.

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 :

Le fichier **discriminator.properties** contient le code suivant :

```
node=10001, 10005,10010,10011,10012
nt=10002,10003
unix=10004,10006,10008
```

Le fichier orm.xml comprend le code suivant :

```
<entity class="generic_db_adapter.node" >
    <table name="[nom_table]"/>
    …
    <inheritance strategy="SINGLE_TABLE"/>
    <discriminator-value>node</discriminator-value>
    <discriminator-column name="[discriminator_column]"/>
    …
</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 calculé comme suit :

- La colonne de discriminateur de la table correspondante contient 10002 pour une certaine entrée. L'entrée est mappée sur le type de CI **nt**.
- La colonne de discriminateur de la table correspondante contient 10006 pour une certaine entrée. L'entrée est mappée sur le type de CI **unix**.
- La colonne de discriminateur de la table correspondante contient 10010 pour une certaine entrée. L'entrée est mappée sur le type de CI **node**.

Notez que le type de CI **node** est également le parent de **nt** et **unix**.

Fichier replication_config.txt

Ce fichier contient une liste séparée par des virgules de types de CI et de relations dont les conditions de propriété sont prises en charge par le plug-in de réplication. Pour plus d'informations, voir "Plug-ins", page 208.

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>]
```

Exemple :

```
entity[ip_address] attribute[ip_domain] value[DefaultDomain]
```

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.

Cette section comprend les rubriques suivantes :

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.

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**.

Exemple de fichier XSD d'entrée :

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
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: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="required"/>
        </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-action="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>
```

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.AdapterToCmdbAddSuffixTransformer**. 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.AdapterToCmdbRemoveSuffixTransformer**. 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.AdapterToCmdbAddPrefixTransformer**. 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.AdapterToCmdbRemovePrefixTransformer**. 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.

Plug-ins

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.

Pour plus d'informations sur l'implémentation et le déploiement de plug-in, voir "Implémentation d'un plug-in", page 155.

Exemples de configuration

Cette section comprend des exemples de configuration.

Elle comprend les rubriques suivantes :

Cas d'utilisation

Use case. Un code 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-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <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_Name"/>
            </or>
        </reconciliation-by-single-node>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-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="hwCardVendor"/>
        <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 185.

Exemple de fichier orm.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xsi="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>ORM de l'adaptateur de BD générique</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_vendor">
                <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_types.txt

Pour plus d'informations, voir "Fichier reconciliation_types.txt", page 197.

```
node
```

Fichier reconciliation_rules.txt

Pour plus d'informations, voir "Fichier reconciliation_rules.txt (pour compatibilité rétroactive)", page 198.

```
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.

Le code TQL de rapprochement est **node** > **(containment)** > **ip_address**.

Définition simplifiée

Le rapprochement s'effectue sur name de node OR de ip_address :

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <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"/>
                <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-CMDB-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="hwCardVendor"/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName"/>
    </class>
</generic-DB-adapter-config>
```

Le rapprochement s'effectue sur name de node AND de ip_address :

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <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">
            <and>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"/>
                <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-CMDB-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="hwCardVendor"/>
        <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-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <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>
                <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-CMDB-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="hwCardVendor"/>
        <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_types.txt

Pour plus d'informations, voir "Fichier reconciliation_types.txt", page 197.

```
node
```

Fichier reconciliation_rules.txt

Pour plus d'informations, voir "Fichier reconciliation_rules.txt (pour compatibilité rétroactive)", page 198.

```
multinode[node] expression[ip_address.name OR node.name] end1_type[node]
end2_type[ip_address] link_type[containment]
```

```
multinode[node] expression[ip_address.name AND node.name] end1_type[node]
end2_type[ip_address] link_type[containment]
```

```
multinode[node] expression[ip_address.name] end1_type[node] end2_type[ip_address]
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-CMDB-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 185.

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="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <id name="id2">
            <column name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
            <generated-value strategy="TABLE"/>
        </id>
        <id name="id3">
            <column name="hwCards_Seq" insertable="false" updatable="false"/>
            <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="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <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="Device_ID" referenced-column-name="Device_ID" insertable="false"
updatable="false"/>
            <join-column name="hwBusesSupported_Seq"
referenced-column-name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
            <join-column name="hwCards_Seq" referenced-column-name="hwCards_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:noNamespaceSchemaLocation="../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.i
mpl.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.Generic
EnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Gene
ricEnumTransformer(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.

Cette section comprend les rubriques suivantes :

- "Niveaux de journalisation", page 220
- "Emplacements des journaux", page 221

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 sont placés 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 le code TQL de rapprochement utilisé 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 133.

Références externes

Pour plus d'informations sur la spécification JavaBeans 3.0, voir http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

Résolution des problèmes et limites

Cette section décrit la résolution des problèmes et les limites correspondant à l'adaptateur de base de données générique.

Limites générales

- L'authentification NTLM SQL Server n'est pas prise en charge.
- Lorsque vous mettez à jour un composant applicatif 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 composant applicatif 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, les noms 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**.
- L'adaptateur de base de données générique ne prend pas en charge l'authentification Windows pour SQL Server.

6

Développement d'adaptateurs Java

Contenu de ce chapitre :

Concepts

Tâches

Références

Concepts

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.
- Pour obtenir un glossaire des définitions et des termes, voir "Glossaire" dans le *Manuel d'administration HP Universal CMDB*.

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 vers une source externe.

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", page 227
- "Émission de données", page 228
- "Remplissage", page 229

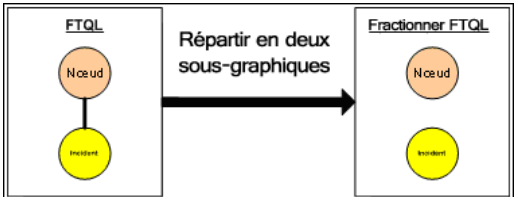
Fédération à la volée

Des requêtes TQL fédérées permettent l'extraction d'informations de n'importe quel référentiel sans réplication des données.

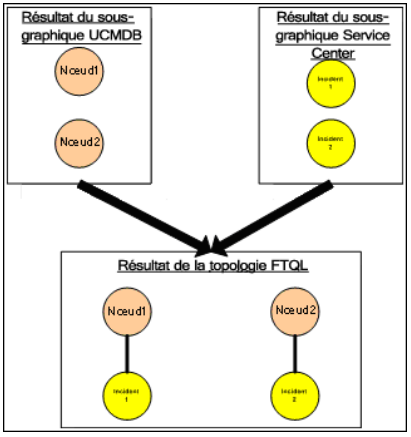
Une requête TQL fédérée exploite des adaptateurs qui représentent des référentiels de données externes. Elle permet d'établir des relations externes appropriées entre CI issus de référentiels de données externes distincts et CI UCMDB.

Exemple de flux de fédération à la volée :

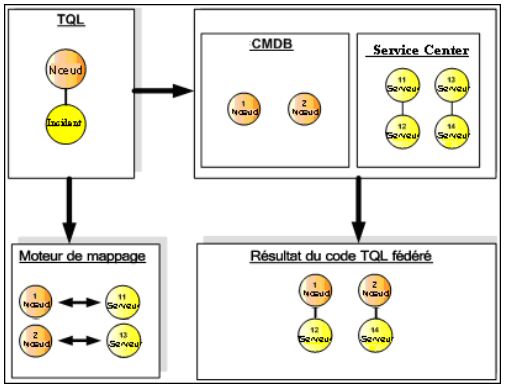
1. L'infrastructure de fédération divise une requête TQL fédérée en plusieurs sous-graphiques, dans lesquels tous les nœuds d'un sous-graphique référencent le même référentiel de données. Chaque sous-graphique est connecté aux autres sous-graphiques par une Relation virtuelle (sans en contenir aucune 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 la sonde de flux de données 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 la sonde de flux de données, 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.

 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", page 232
- "Méthodes assist de l'adaptateur", page 232

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 la sonde de flux de données, 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.

Flux de l'infrastructure de fédération pour requêtes TQL fédérées

Cette section comprend les rubriques suivantes :

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.

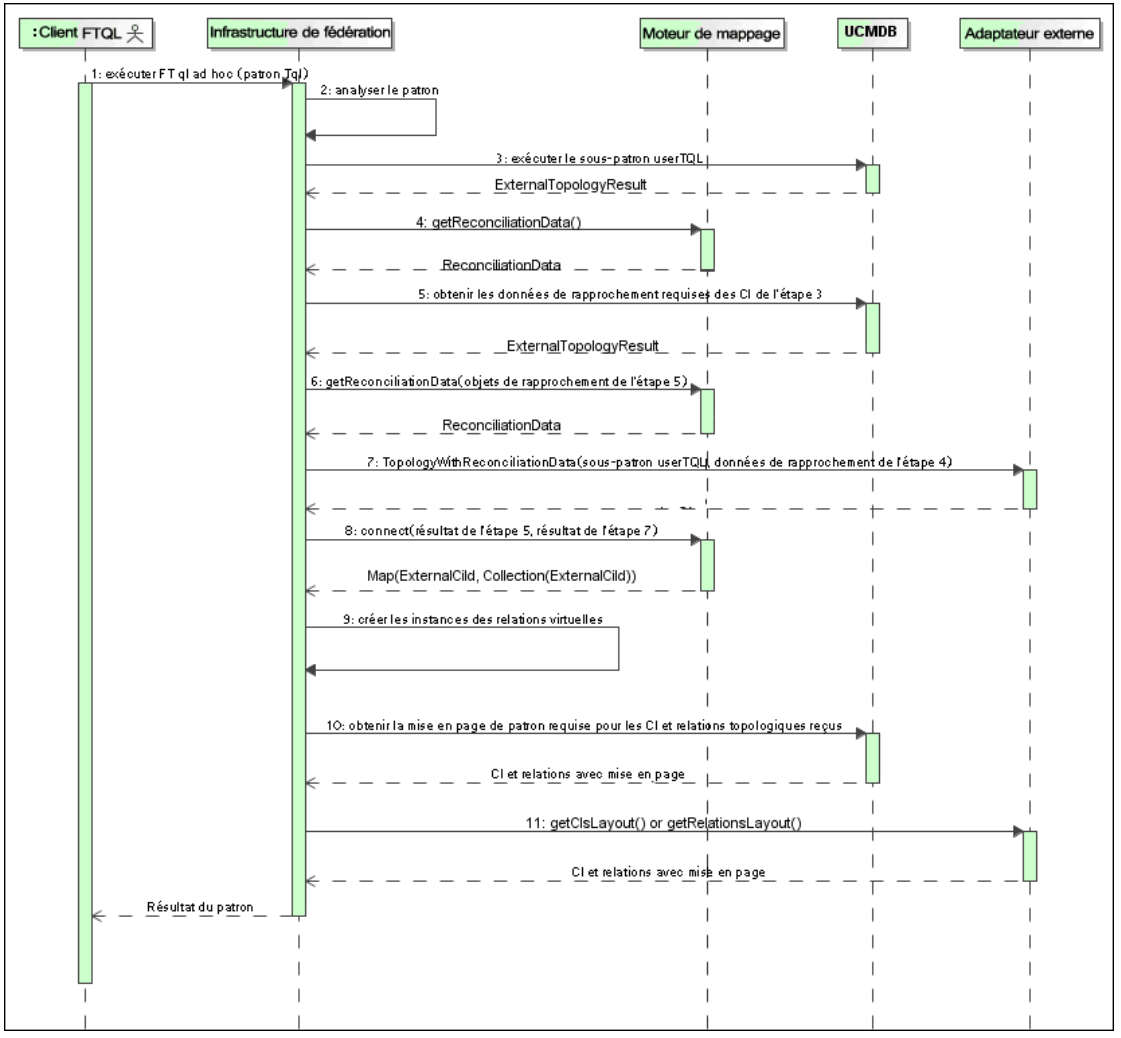
Diagrammes de flux

Les diagrammes suivants illustrent l'interaction entre l'infrastructure de fédération, la base 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.

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 débute au niveau de l'extrémité constituée par la base HP Universal CMDB.

Le diagramme de séquence suivant illustre l'interaction entre l'infrastructure de fédération, la base 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.

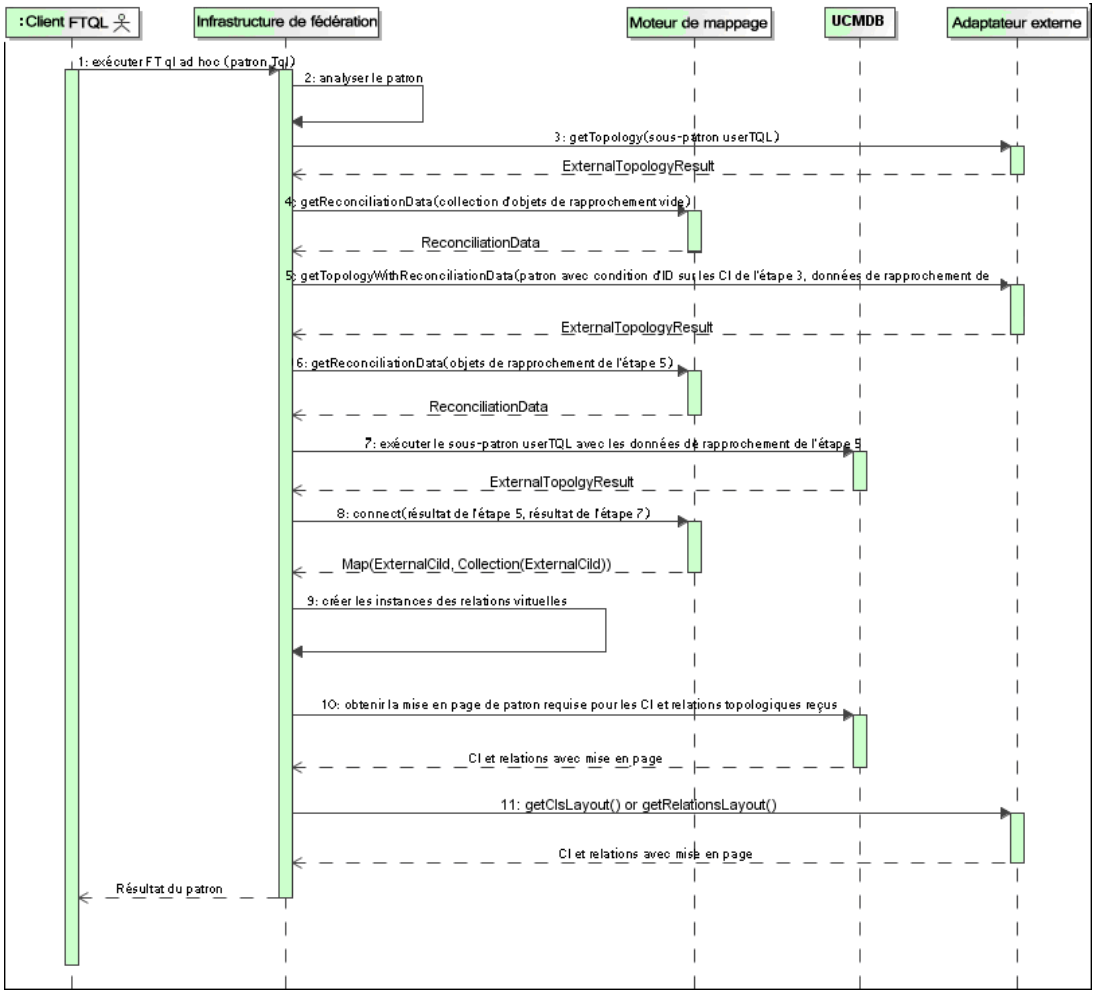


Les numéros qui figurent sur cette image sont expliqués ci-dessous :

Numéro	Explication
1	L'infrastructure de fédération reçoit un appel pour un calcul de requête TQL fédérée.
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 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), 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.

Numéro	Explication
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 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 la base UCMDB à l'étape 5, tandis que le paramètre secondResult correspond au résultat topologique reçu de l'adaptateur externe à l'étape 7. Le moteur de mappage renvoie une carte sur laquelle l'identifiant du CI externe issu du premier référentiel de données (UCMDB dans le cas présent) est mappé sur les identifiants des CI externes issus du second référentiel de données (externe).
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.

Le calcul débute au niveau de l'extrémité constituée par l'adaptateur externe.



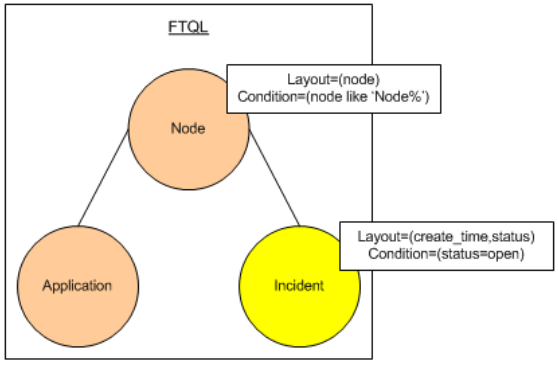
Les numéros qui figurent sur cette image sont expliqués ci-dessous :

Numéro	Explication
1	L'infrastructure de fédération reçoit un appel pour un calcul de TQL fédérée.
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.

Numéro	Explication
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.
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

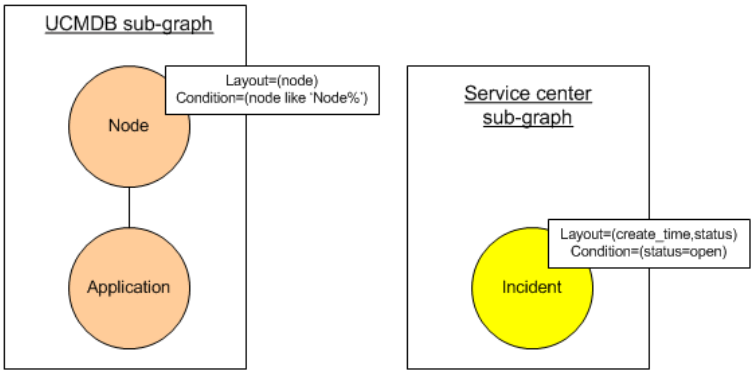
Cet exemple explique comment afficher tous les incidents en cours sur des nœuds spécifiques. Le référentiel de données ServiceCenter est un référentiel de données externe. Les instances de nœud sont enregistrées dans la base UCMDB et les instances d'incident dans ServiceCenter. Pour connecter les instances 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.



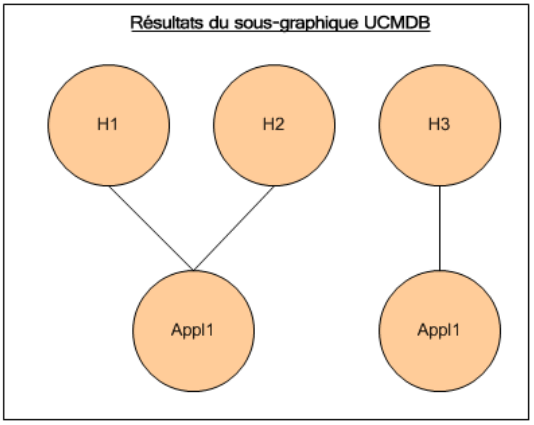
Remarque : Pour la fédération d'attributs, la méthode **getTopology** de l'adaptateur est appelée. Les données de rapprochement sont adaptées dans le code TQL utilisateur (dans le cas présent, l'élément CI).

---

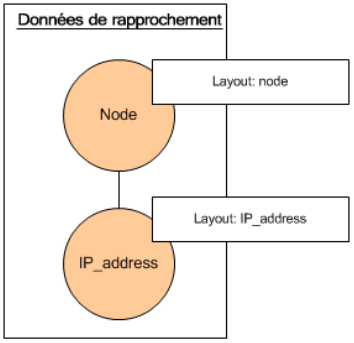
1 Une fois l'adaptateur analysé, l'infrastructure de fédération identifie la relation virtuelle entre les propriétés Node et Incident, et scinde la requête TQL fédérée en deux sous-graphiques :



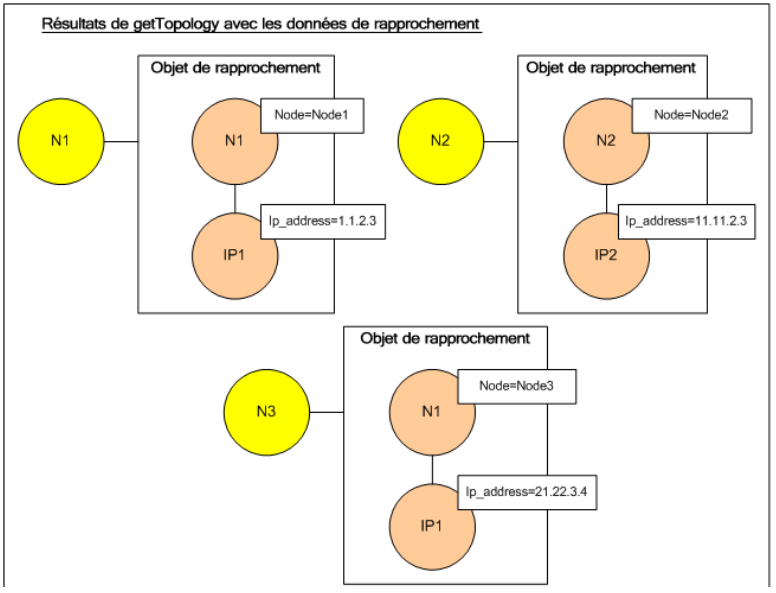
2 L'infrastructure de fédération exécute le sous-graphique UCMDB pour solliciter la topologie et reçoit les résultats suivants :



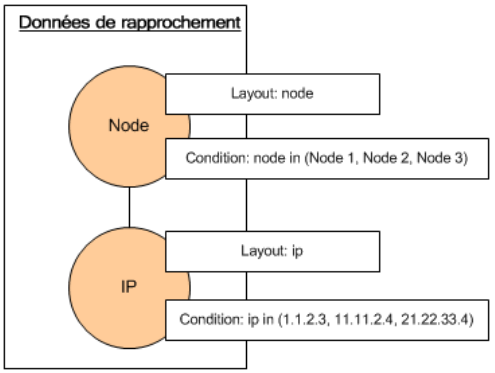
3 L'infrastructure de fédération sollicite auprès du moteur de mappage approprié les données de rapprochement pour le premier référentiel de données (la base UCMDB) qui contient les informations nécessaires pour établir une connexion entre les données reçues de deux référentiels. Dans ce cas, les données de rapprochement sont les suivantes :



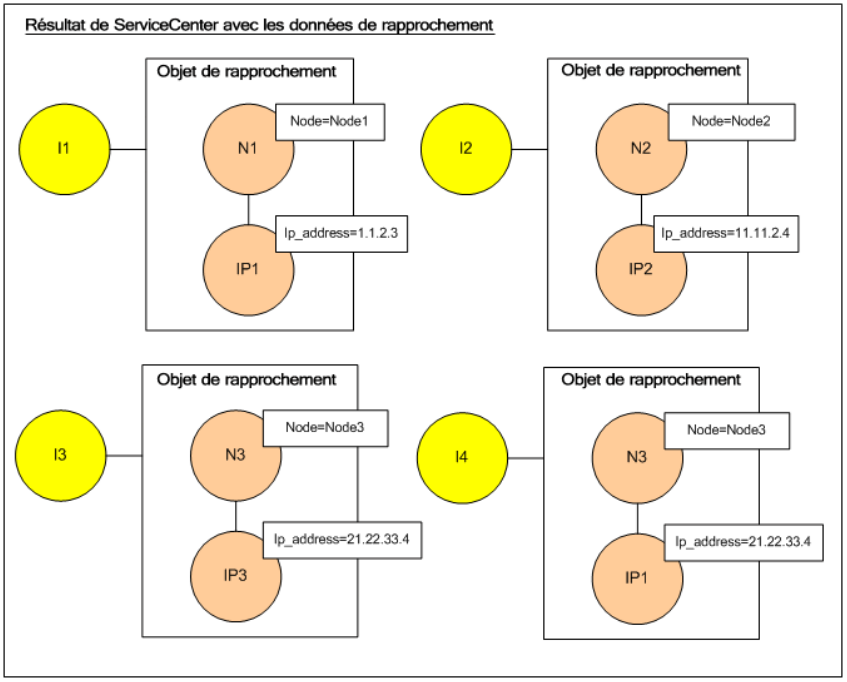
4 L'infrastructure de fédération crée une requête de topologie à un nœud intégrant les conditions Node et ID issues du résultat précédent (node en H1, H2, H3), puis applique cette requête avec les données de rapprochement requises à la base UCMDB. Le résultat comprend les CI de nœud pertinents vis-à-vis de la condition d'identifiant (ID), ainsi que l'objet de rapprochement approprié pour chaque CI :



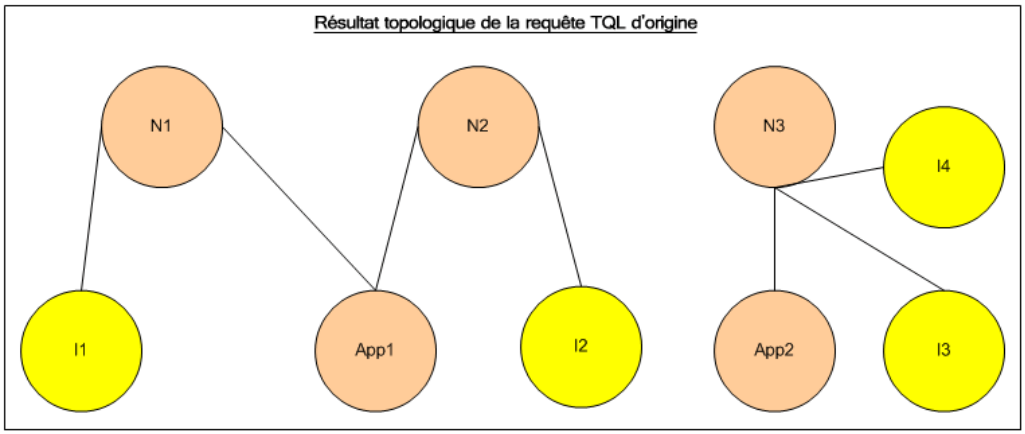
5 Les données de rapprochement destinées à ServiceCenter devraient contenir une condition pour les propriétés node et ip dérivées des objets de rapprochement reçus de la base UCMDB :



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

Cette section comprend les rubriques suivantes :

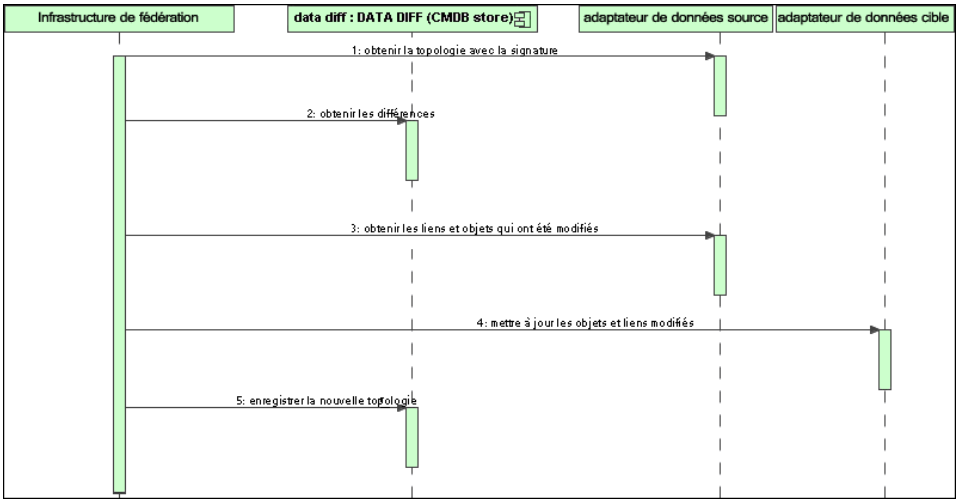
- "Définitions et termes", page 246
- "Diagramme de flux", page 247

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

Cette section comprend les rubriques suivantes :

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 adaptée à chaque adaptateur, comme suit.

Une **interface topologique à un nœud** s'utilise lorsque l'adaptateur ne prend aucune relation externe en charge ; en d'autres termes, lorsque l'adaptateur n'est jamais censé recevoir de requête impliquant plus d'un CI externe. Toutes les interfaces à un nœud (OneNode) sont créées pour simplifier le flux de tâches. Dans les cas où vous devez utiliser une requête plus étendue, utilisez l'interface DataAdapter.

Obsolète à compter de la version 9.00 d'UCMDB : interface topologique de patron

Une interface **DataAdapter** s'utilise pour définir des adaptateurs qui prennent en charge des requêtes fédérées complexes. La requête de rapprochement intégrée à ces adaptateurs fait partie du paramètre unique QueryDefinition. Ces adaptateurs peuvent également s'utiliser pour le remplissage.

Interfaces OneNode

Les interfaces suivantes présentent différents types de données de rapprochement :

- **OneNodeTopologyIdReconciliationDataAdapter.** 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).
- **OneNodeTopologyPropertyReconciliationDataAdapter.** 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.
- **OneNodeTopologyDataAdapter.** 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.

Interfaces d'adaptateur de données

- **DataAdapter.** Utilisez cet adaptateur pour prendre en charge des requêtes TQL fédérées complexes. Autorise la plus grande diversité.
- **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.

Interfaces topologiques de patron (obsolètes à compter de la version 9.00 d'UCMDB)

Les interfaces suivantes présentent différents types de données de rapprochement :

- **PatternTopologyIdReconciliationDataAdapter.** S'utilise si l'adaptateur prend en charge un **code TQL complexe** et si le rapprochement entre des référentiels de données s'effectue en fonction de l'identifiant (ID).
- **PatternTopologyPropertyReconciliationDataAdapter.** S'utilise si l'adaptateur prend en charge un **code TQL complexe** et si le rapprochement entre des référentiels de données s'effectue en fonction de propriétés de nœud unique.
- **PatternTopologyDataAdapter.** S'utilise si l'adaptateur prend en charge un **code TQL complexe** et si le rapprochement entre des référentiels de données s'effectue en fonction de la topologie.

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.

Tâches

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.

Elle comprend les étapes suivantes :

- "Conditions préalables", page 251
- "Définition de relations virtuelles valides", page 252
- "Définition d'une configuration d'adaptateur", page 253
- "Définition des classes prises en charge", page 256
- "Implémentation de l'adaptateur", page 257
- "Définition de règles de rapprochement ou implémentation du moteur de mappage", page 257
- "Ajout des ressources jar nécessaires à l'implémentation dans le classpath", page 257
- "Déploiement de l'adaptateur", page 258
- "Mise à jour de l'adaptateur", page 259

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 "Utilisation du Sélecteur de CI" 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 est pertinente uniquement dans le cadre de 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 composant applicatif de l'adaptateur dans le dossier **\validlinks**. Pour plus d'informations, voir "Gestionnaire des composants applicatifs" 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-Link>
        <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 nouvelle ressource**.

c Dans la boîte de dialogue Nouvel adaptateur, sélectionnez **Intégration** et **Adaptateur Java**.

d Cliquez avec le bouton droit de la souris 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="Nom affiché du nouvel adaptateur">
    <deletable>true</deletable>
    <discoveredClasses>
        <discoveredClass>link</discoveredClass>
        <discoveredClass>object</discoveredClass>
    </discoveredClasses>
    <taskInfo className="com.hp.ucmdb.discovery.probe.services.dynamic.core.AdapterService">
        <params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.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.BaseDestinationData">
            <!-- 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 263.

4 Définition des classes prises en charge

Définissez des classes prises en charge soit dans le code de l'adaptateur, en implémentant la méthode *getSupportedClasses()*, soit en utilisant le fichier XML patron.

```
<supported-classes>
    <supported-class name="HistoryChange" is-derived="false"
is-reconciliation-supported=”false” federation-not-supported=”false”
is-id-reconciliation-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.

6 Définition de règles de rapprochement ou implémentation du moteur de mappage

Si votre adaptateur prend en charge des requêtes TQL fédérées, vous avez trois possibilités pour définir votre moteur de mappage :

- Utiliser le moteur de mappage par défaut CMDB 9.0x. Celui-ci utilise les règles de rapprochement internes CMDB. Pour l'utiliser, laissez la balise XML **<default-mapping-engine/>** vide.

  Pour plus d'informations, voir "Fichier reconciliation_types.txt", page 197.

- Utiliser le moteur de mappage CMDB 8.0x. Pour ce faire, utilisez la balise XML suivante :
  <default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>

  Pour plus d'informations, voir "Fichier reconciliation_rules.txt (pour compatibilité rétroactive)", page 198.

- Développer 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

a Déployez le composant applicatif de l'adaptateur. Pour plus d'informations sur le déploiement d'un composant applicatif, voir "Gestionnaire des composants applicatifs" dans le *Manuel d'administration HP Universal CMDB*.

Le composant applicatif 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 composant applicatif.

- 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 composant applicatif.

- 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 composant applicatif.

- Le fichier XML de configuration de patron doit se trouver dans le dossier discoveryPatterns du composant applicatif.

- **Descripteur.** Spécifie les définitions du composant applicatif.

- Placez vos classes compilées (normalement un fichier jar) dans le composant applicatif 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 composant applicatif 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 composant applicatif (fichiers binaires et non binaires), puis en redéployant le composant applicatif au moyen du module Gestion de l'adaptateur. Pour plus d'informations, voir "Déployer un composant applicatif" dans le *Manuel d'administration HP Universal CMDB*.

Implémentation du moteur de mappage

La configuration du moteur de mappage dépend du moteur de mappage utilisé.

Cette tâche comprend les étapes suivantes :

- "Configuration du fichier reconciliation_types.txt (pour le moteur de mappage par défaut UCMDB 9.0x)", page 259
- "Configuration du fichier reconciliation_rules.txt (pour le moteur de mappage UCMDB 8.0x)", page 260

1 Configuration du fichier reconciliation_types.txt (pour le moteur de mappage par défaut UCMDB 9.0x)

Le fichier permet de définir les types de CI utilisés pour le rapprochement dans l'adaptateur.

Écrivez chaque type de CI utilisé pour le rapprochement sur une ligne indépendante, comme suit :

```
node
business_application
```

Placez le fichier dans le composant applicatif de l'adaptateur, dans le dossier **adapterCode**\<**AdapterID**>\
META-INF\.

2 Configuration du fichier reconciliation_rules.txt (pour le moteur de mappage UCMDB 8.0x)

Ce fichier permet de configurer les règles de rapprochement. Chaque ligne du fichier représente une règle. Exemple :

```
reconcilition_type[node] expression[^node.name OR ip_address.name]
end1_type[node] end2_type[ip_address] link_type[containment]
```

Le paramètre **reconcilition_type** est rempli avec le type de CI sur lequel le rapprochement s'effectue (le nom de classe UCMDB connecté à la classe fédérée dans le code TQL).

Le paramètre **expression** constitue la logique qui décide si deux objets de rapprochement sont égaux (un objet de rapprochement côté UCMDB et l'autre côté adaptateur fédéré).

L'expression est composée d'opérateurs OR et AND.

La convention relative aux noms d'attributs dans la partie expression est la suivante : **[className].[attributeName]**. Par exemple, l'attribut **ip_address** de la classe **ip** s'écrit **ip.ip_address**.

Vous pouvez définir des correspondances ordonnées. La correspondance ordonnée contrôle la première sous-expression OR. Si deux objets de rapprochement affichent la valeur figurant au niveau des attributs de la sous-expression et si une valeur false est renvoyée (les objets de rapprochement ne sont pas égaux), alors la deuxième expression OR n'est pas comparée.

Dans le cas d'une correspondance ordonnée, utilisez **ordered expression** au lieu d'**expression**.

Le symbole circonflexe (^) sert à ignorer la casse lors des comparaisons.

Les autres paramètres (**end1_type**, **end2_type** et **link_type**) ne s'utilisent que si les données de rapprochement contiennent deux nœuds et pas seulement le nœud du type de rapprochement (les données de rapprochement topologiques). Dans ce cas, les données de rapprochement sont **end1_type -(link_type)> end2_type**.

Il n'est pas nécessaire d'ajouter la structure pertinente car celle-ci est extraite de l'expression.

Pour effectuer un rapprochement au moyen de l'identifiant (ID) UCMDB, utilisez **cmdb_id** en tant que nom d'attribut dans l'expression.

Placez le fichier dans le composant applicatif de l'adaptateur, dans le dossier **adapterCode\<AdapterID>\META-INF**.

Exemples :

- Vous pouvez ajouter une règle de rapprochement uniquement pour un type de CI de nœud. En effet, seuls ces types de CI affichent des relations valides avec des types de CI externes. Par exemple, un CI de nœud dans la base CMDB est mis en correspondance avec un CI de nœud dans ServiceCenter par le biais de l'attribut node.name ou de l'attribut ip_address.name.
- Dans ce cas, la règle de rapprochement correspond à une règle de topologie et l'expression est ordonnée. La règle procède aux contrôles suivants sur les CI soumis à la comparaison :
  - Si l'attribut node.name est égal, la règle associe les nœuds.
  - Si l'attribut node.name n'est pas égal, la règle n'associe pas les nœuds.
  - Si l'attribut node.name est de type NULL dans l'un des CI comparés, la règle contrôle l'attribut ip_address.name. Si l'attribut ip_address.name est égal, la règle associe les nœuds.

Création d'un exemple d'adaptateur

Cet exemple illustre comment créer un exemple d'adaptateur.

Cette tâche comprend les étapes suivantes :

- "Sélection d'une logique d'adaptateur", page 262
- "Chargement du projet", page 262

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. Exemple :

- ➤ Convertissez la condition en une requête SQL.
- ➤ Convertissez la condition en un objet de filtre d'API Java.

c La solution intermédiaire consiste à filtrer certaines données au niveau du service distant, et à faire en sorte que l'adaptateur sélectionne et filtre le reste.

Dans l'exemple MyAdapter, la logique de l'étape a est utilisée.

2 Chargement du projet

Copiez les fichiers depuis le dossier **C:\hp\UCMDB\UCMDBServer\tools\adapter-dev-kit\SampleAdapters** et suivez les instructions contenues 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\doc_lib\DevRef_guide\DBAdapterFramework_JavaAPI\index.html

Références

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="Nom affiché du nouvel adaptateur"		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" ou 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.
	<one-node-topology>	Capacité de fédérer des requêtes avec un seul nœud de requête fédérée.

	<pattern-topology>	Capacité de fédérer des requêtes complexes.
	<support-replication-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.
	<changes-source/>	Cet adaptateur peut être utilisé pour les flux de modifications de remplissage.
	<target>	Cet adaptateur peut être utilisé pour les flux d'émission de données.
	<default-mapping-engine>	Permet la définition d'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 (pour le moteur de mappage UCMDB 8.0x, utilisez : com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine).
	<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.

7

Développement d'adaptateurs d'émission (push)

Contenu de ce chapitre :

Concepts

Tâches

Références

Concepts

Présentation du développement d'adaptateurs d'émission (push)

L'adaptateur d'émission générique offre une plate-forme permettant le développement rapide d'intégrations qui émettent (push) les données UCMDB 9.0x en direction de référentiels de données externes (bases de données et applications tierces). Le développement d'une intégration personnalisée basée sur l'adaptateur d'émission générique exige les éléments suivants :

- un fichier de mappage XML entre les types de lien de CI UCMDB et les éléments de données externes ;
- un script Jython pour émettre les éléments de données dans le référentiel de données externe.

Synchronisation différentielle

Si la méthode **DiscoveryMain** du script Jython servant de base à l'adaptateur d'émission renvoie une instance **OSHVResult** 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.

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.

Tâches

Préparation des fichiers de mappage

Il existe deux façons de préparer des fichiers de mappage :

- 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).

  Dans ce cas, chaque fichier de mappage est appelé **<nom requête>.xml**.

Pour plus d'informations, voir "Schéma du fichier de mappage", page 275.

Cette tâche comprend les étapes suivantes :

- "Création du fichier de mappage", page 268
- "Mappage des CI", page 268
- "Mappage des liens", page 269

1 Création du fichier de mappage

Le fichier de mappage possède la structure suivante :

```
<?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

Chaque type de CI CMDB est mappé comme dans l'exemple suivant :

```
<source_ci_type name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey><pkey>host_key</pkey></targetprimarykey>
        <target_attribute name="host_os" datatype="STRING">
            <map type="direct" source_attribute="discovered_os_name" />
        </target_attribute>
        <!-- more target attributes --->
    </target_ci_type>
</source_ci_type>
```

Remarque : Les valeurs possibles de mode dépendent de l'implémentation du script.

---

3 Mappage des liens

Chaque lien valide est mappé comme dans l'exemple suivant :

```
<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" />
</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 65.

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. See
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 devez vous servir des paramètres de connexion que vous avez définis pour l'adaptateur (pour plus de détails, voir l'étape 2 de la section "Création d'un composant applicatif d'adaptateur", page 274), utilisez le code suivant :

```
propValue = str(Framework.getDestinationAttribute('<Connection Property Name'))
```

Exemple :

```
serverName = Framework.getDestinationAttribute('ip_address')
```

Autre contenu de cette section :

- "Utilisation des résultats du mappage", page 270
- "Gestion d'un test de connexion dans le script", page 271

Utilisation des résultats du mappage

L'adaptateur d'émission générique crée 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 soit modifié 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.

Exemple de résultat XML

```
<root>
    <data>
        <objects>
            <Object mode="update_else_insert" name="ip" operation="add"
mamId="2ebdc7a93dc7f5bcb33a444763c2a16c">
                <field name="root_lastaccesstime" key="false" datatype="DATE"
length="">1275469266</field>
                <field name="display_label" key="false" datatype="STRING"
length="">16.59.61.67</field>
                <field name="ip_probename" key="false" datatype="STRING"
length="">VMUCMDB05</field>
            </Object>
        </objects>
        <links>
            <link targetRelationshipClass="contained" targetParent="nt"
targetChild="ip" operation="add" mode="update_else_insert"
mamId="8c0a38d53c74c3cc972d6254fb50adba">
                <field name="DiscoveryID1">d5aac653aff428b4a3780111f6389d53</
field>
                <field
name="DiscoveryID2">2ebdc7a93dc7f5bcb33a444763c2a16c</field>
            </link>
        </links>
    </data>
</root>
```

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

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 composant applicatif de votre adaptateur. Pour plus d'informations, voir "Création d'un composant applicatif d'adaptateur", page 274.

Cette tâche permet à l'adaptateur d'émission d'effectuer une synchronisation différentielle. Pour plus d'informations, voir "Synchronisation différentielle", page 266.

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
ExternalIdUtil
```

➤ Utilisez la classe d'usine **DataPushResultsFactory** pour extraire l'objet **DataPushResults** de la fonction **DiscoveryMain**.

```
# Create the UpdateResult object
updateResult = DataPushResultsFactory.createDataPushResults(objectMappings,
linkMappings);
```

➤ 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) :

```
externaCIIId = ExternalIdFactory.createExternalCmdbCiId(ciType, ciIDAsString)
externalRelationId = ExternalIdFactory.createExternalCmdbRelationId(linkType,
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.

Création d'un composant applicatif d'adaptateur

1. Extrayez le contenu de **C:\hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip** dans un dossier temporaire.
2. Éditez le fichier **discoveryPatterns\push_adapter.xml**.
   a. Modifiez la balise <pattern> avec un nouvel ID et un nouveau nom affiché. Remplacez :

   ```
   <pattern id="PushAdapter" xsi:noNamespaceSchemaLocation="../../
   Patterns.xsd" description="Discovery Pattern Description" schemaVersion="9.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   ```

   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">
   ```

   b. Mettez à jour la liste des paramètres, afin qu'elle reflète les attributs de connexion requis. Ne supprimez pas l'attribut **probeName**.
3. Renommez le dossier **adapterCode\PushAdapter** avec l'ID d'adaptateur utilisé à l'étape 2 (par exemple, **adapterCode\MyPushAdapter**).
4. Remplacez **discoveryScripts\pushScript.py** par le script que vous avez écrit (pour plus de détails, voir "Écriture de scripts Jython", page 269). Si vous renommez le script, la propriété jythonScript.name dans **adapterCode\<adapter ID>\push.properties** doit être mise à jour en conséquence.
5. Remplacez le fichier **adapterCode\<adapter ID>\mappings\mappings.xml** par les fichiers de mappage que vous avez préparés (pour plus de détails, voir "Préparation des fichiers de mappage", page 267.

   Si vous souhaitez 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** sera utilisé par défaut, si aucun fichier de mappage particulier est trouvé 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\<adapter ID>\push.properties**.

Références

Schéma du fichier de mappage

Élément		Attributs
Nom et chemin d'accès	**Description**	
integration	Définit le contenu de mappage du fichier. Il doit s'agir du bloc situé le plus 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.	
source (integration > info)	Définit des informations sur le référentiel de données source.	**Nom**. type **Description.** Nom du référentiel de données source. **Obligatoire.** Required **Type.** String
		Nom. versions **Description.** Version(s) des référentiels de données source. **Obligatoire.** Required **Type.** String
		Nom. vendor **Description.** Fournisseur du référentiel de données source. **Obligatoire.** Required **Type.** String

Élément		Attributs
Nom et chemin d'accès	**Description**	
target (integration > info)	Définit des informations sur le référentiel de données cible.	**Nom.** type **Description.** Nom du référentiel de données cible. **Obligatoire.** Required **Type.** String
		Nom. versions **Description.** Version(s) du référentiel de données cible. **Obligatoire.** Required **Type.** String
		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	

Élément		Attributs
Nom et chemin d'accès	**Description**	
source_ci_type (integration > targetcis)	Définit un type de CI source.	**Nom.** name **Description.** Nom du type de CI source. **Obligatoire.** Required **Type.** String
		Nom. mode **Description.** Type de mise à jour requis pour le type de CI 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.

Élément		Attributs
Nom et chemin d'accès	**Description**	
target_ci_type (integration > targetcis > source_ci_type)	Définit un type de CI cible.	**Nom.** name **Description.** Nom du type de CI cible. **Obligatoire.** Required **Type.** String
		Nom. schema **Description.** Nom du schéma qui sera utilisé pour stocker ce type de CI sur la cible. **Obligatoire.** Not Required **Type.** String
		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 > targetrelations > link)	Identifie les attributs de la clé principale du type de CI cible.	
pkey (integration > targetcis > source_ci_type > targetprimarykey -OU- integration > targetrelations > link > targetprimarykey)	Identifie un attribut de la clé principale. Obligatoire uniquement si le mode is **update** ou **insert_else_update**	

Élément		Attributs
Nom et chemin d'accès	**Description**	
target_attribute (integration > targetcis > source_ci_type -OU- integration > targetrelations > link)	Définit l'attribut du type de CI cible.	**Nom.** name **Description.** Nom de l'attribut du type de CI cible. **Obligatoire.** Required **Type.** String
		Nom. datatype **Description.** Type de données de l'attribut du type de CI cible. **Obligatoire.** Required **Type.** String
		Nom. length **Description.** Pour les types de données chaîne/caractère, taille entière de l'attribut cible. **Obligatoire.** Not Required **Type.** Integer
		Nom. option **Description.** Fonction de conversion à appliquer à la valeur. **Obligatoire.** False **Type.** L'une des chaînes suivantes : ➤ **uppercase** – Convertir en majuscules. ➤ **lowercase** – Convertir en minuscules. ➤ Si cet attribut est vide, aucune fonction de conversion ne sera appliquée.

Élément		Attributs
Nom et chemin d'accès	**Description**	
map (integration > targetcis > source_ci_type > target_attribute -OU- integration > targetrelations > link > target_attribute)	Indique comment obtenir la valeur d'attribut du type de CI source.	**Nom.** type **Description.** Type de mappage entre les valeurs source et cible. **Obligatoire.** Required **Type.** L'une des chaînes suivantes : ➤ **direct** – Indique une correspondance de un à un entre la valeur de l'attribut source et celle de l'attribut cible. ➤ **compoundstring** – Les sous-éléments sont joints en une chaîne unique et la valeur de l'attribut cible est définie. ➤ **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 **container_f** ou **contained**. ➤ **constant** – Chaîne statique
		Nom. value **Description.** Chaîne constante pour type=**constant** **Obligatoire.** Required uniquement lorsque type=**constant** **Type.** String
		Nom. attr **Description.** Nom d'attribut source pour type=**direct** **Obligatoire.** Required uniquement lorsque type=**direct** **Type.** String

Élément		Attributs
Nom et chemin d'accès	**Description**	
aggregation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetrelations > link > 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 un décompte numérique de toutes les valeurs 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.

Élément		Attributs
Nom et chemin d'accès	**Description**	
validation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetrelations > link > target_attribute > map Valide uniquement lorsque le type de la mappe est **childatt**)	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.	**Nom.** minlength **Description.** Exclut les chaînes plus courtes que la valeur indiquée. **Obligatoire.** Not required **Type.** Integer
		Nom. maxlength **Description.** Exclut les chaînes plus longues que la valeur indiquée. **Obligatoire.** Not required **Type.** Integer
		Nom. minvalue **Description.** Exclut les nombres plus petits que la valeur spécifiée. **Obligatoire.** Not required **Type.** Numeric
		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.	

Élément		Attributs
Nom et chemin d'accès	**Description**	
link (integration > targetrelations)	Mappe une relation source sur une relation cible. Obligatoire uniquement si **targetrelation** est présent.	**Nom.** source_link_type **Description.** Nom de la relation source. **Obligatoire.** Required **Type.** String
		Nom. target_link_type **Description.** Nom de la relation cible. **Obligatoire.** Required **Type.** String
		Nom. nameSpace **Description.** Espace de noms du lien qui sera créé sur la cible. **Obligatoire.** Not required **Type.** String
		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.

Élément		Attributs
Nom et chemin d'accès	**Description**	
link (suite)		**Nom.** source_ci_type_end1 **Description.** Type de CI End1 de la relation source **Obligatoire.** Required **Type.** String
		Nom. source_ci_type_end2 **Description.** Type de CI End2 de la relation source **Obligatoire.** Required **Type.** String
target_ci_type_end1 (integration > targetrelations > link)	Type de CI End1 de la relation cible	**Nom.** name **Description.** Nom du type de CI End1 de la relation cible. **Obligatoire.** Required **Type.** String
		Nom. superclass **Description.** Nom de la superclasse du type de CI End1. **Obligatoire.** Not required **Type.** String
target_ci_type_end2 (integration > targetrelations > link)	Type de CI End2 de la relation cible	**Nom.** name **Description.** Nom du type de CI End2 de la relation cible. **Obligatoire.** Required **Type.** String
		Nom. superclass **Description.** Nom de la superclasse du type de CI End2. **Obligatoire.** Not required **Type.** String

Schéma des résultats de mappage

Élément		Attributs
Nom et chemin d'accès	**Description**	
root	Racine du document de résultat	
data (root)	Racine des données elles-mêmes	
objects (root > data)	Élément racine des objets à mettre à jour	
Object (root > data > objects)	Décrit l'opération de mise à jour pour un objet unique et tous ses attributs.	**Nom.** name **Description.** Nom du type de CI **Obligatoire.** Required **Type.** String
		Nom. mode **Description.** Type de mise à jour requis pour le type de CI 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.

Élément		Attributs
Nom et chemin d'accès	**Description**	
Object (suite)		**Nom.** operation **Description.** Opération à effectuer avec ce CI. **Obligatoire.** Required **Type.** L'une des chaînes suivantes : ➤ **add** – Le CI doit être ajouté. ➤ **update** – Le CI doit être mis à jour. ➤ **delete** – Le CI doit être supprimé. Si aucune valeur n'est définie, la valeur par défaut de **add** est utilisée.
		Nom. mamId **Description.** ID de l'objet dans la base CMDB source. **Obligatoire.** Required **Type.** String

Élément		Attributs
Nom et chemin d'accès	**Description**	
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 contenue dans le champ et, si ce dernier contient un lien, la valeur est l'ID de l'une des extrémités. Chaque ID de fin apparaît comme un objet (sous <objects>).	**Nom.** name **Description.** Nom du champ. **Obligatoire.** Required **Type.** String
		Nom. key **Description.** Indique si ce champ est une clé pour l'objet. **Obligatoire.** Required **Type.** Boolean
		Nom. datatype **Description.** Type du champ. **Obligatoire.** Required **Type.** String
		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

Élément		Attributs
Nom et chemin d'accès	**Description**	
links (root > data)	Élément racine des liens à mettre à jour	**Nom.** targetRelationshipClass **Description.** Nom de la relation (lien) dans le système cible. **Obligatoire.** Required **Type.** String
		Nom. targetParent **Description.** Type de la première extrémité du lien (parent). **Obligatoire.** Required **Type.** String
		Nom. targetChild **Description.** Type de la deuxième extrémité du lien (enfant). **Obligatoire.** Required **Type.** String

Élément		Attributs
Nom et chemin d'accès	**Description**	
links (suite)		**Nom.** mode **Description.** Type de mise à jour requis pour le type de CI 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.
		Nom. operation **Description.** Opération à effectuer avec ce CI. **Obligatoire.** Required **Type.** L'une des chaînes suivantes : ➤ **add** – Le CI doit être ajouté. ➤ **update** – Le CI doit être mis à jour. ➤ **delete** – Le CI doit être supprimé. Si aucune valeur n'est définie, la valeur par défaut de **add** est utilisée.
		Nom. mamId **Description.** ID de l'objet dans la base CMDB source. **Obligatoire.** Required **Type.** String

Partie II

Utilisation d'API

8

Introduction aux API

Contenu de ce chapitre :

Concepts

Concepts

Présentation des API

Les API suivantes sont incluses avec HP Universal CMDB :

- **API de service Web UCMDB.** Permet d'écrire les définitions de éléments de configuration et les relations topologiques avec UCMDB (base de données de gestion de la configuration universelle), ainsi que d'interroger les informations avec des requêtes TQL et ad hoc. Pour plus d'informations, voir "API de service Web HP Universal CMDB", page 295.
- **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 383.

9

API de service Web HP Universal CMDB

Contenu de ce chapitre :

Concepts

Tâches

Références

Concepts

Conventions

Ce chapitre observe les conventions suivantes :

- **UCMDB** se rapporte à la base de données de gestion de la configuration universelle elle-même. **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 initiale. Par exemple, une relation est un élément de type Relation transmis à une méthode.

API de service Web HP Universal CMDB - Présentation

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 la base 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 : afficher le statut, activer et désactiver ;
- 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 "TQL (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.

Cette section comprend les rubriques suivantes :

- "Utilisations de l'API", page 297
- "Autorisations", page 298

Utilisations de l'API

L'API permet de satisfaire un certain nombre d'exigences métier. 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

L'administrateur fournit des informations d'identification pour la connexion au service Web. Les informations d'identification requises dépendent si HP Universal CMDB est utilisé comme application autonome ou à partir de Business Service Management :

- **HP Universal CMDB autonome.** Connectez-vous à l'aide des informations d'identification d'un utilisateur UCMDB auquel des autorisations ont été accordées sur les ressources de découverte et d'intégration.

  Pour plus d'informations, voir "Page Gestionnaire des sécurités" dans le *Manuel d'administration HP Universal CMDB*.

- **HP Universal CMDB intégré dans Business Service Management.** Connectez-vous à l'aide des informations d'identification d'un utilisateur Business Service Management. Les autorisations pertinentes doivent avoir été accordées à l'utilisateur sur la ressource HP Universal CMDB dans Business Service Management.

Lorsque des autorisations sont accordées par le biais de HP Universal CMDB, les niveaux d'autorisation sont Affichage, Mise à jour et Exécution. Lorsqu'elles sont affectées par le biais de Business Service Management, les niveaux sont Affichage et Mise à jour, ce dernier incluant également le niveau Exécution. Pour afficher les autorisations requises pour chaque opération et des informations sur la demande de chaque opération, voir *gestion des flux de données Schema Reference*.

HP Universal CMDB Web Service API Reference

Pour une documentation complète sur les structures de demande et de réponse, voir Référence des API de service Web HP UCMDB. Ces fichiers figurent dans le dossier suivant :

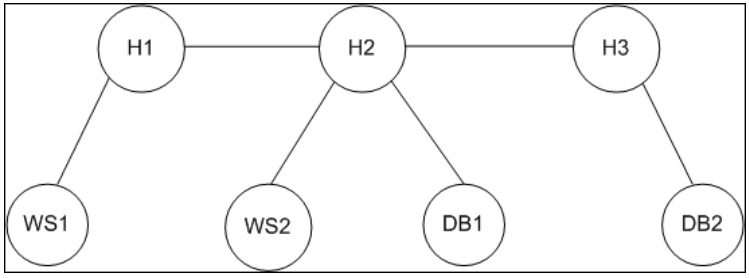
C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\doc_lib\DevRef_guide\CMDB_Schema\webframe.html

Renvoi d'éléments de carte topologique non ambigus

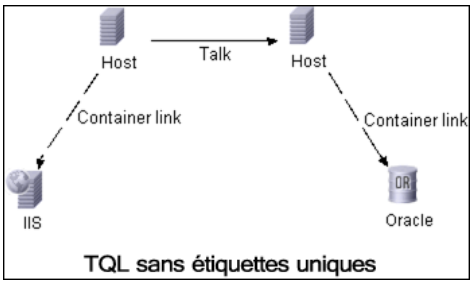
Les méthodes de requête qui renvoient les données dans des éléments topology ou topologyMap recherchent sur le système une correspondance avec une requête TQL. Les diagrammes suivants illustrent l'incidence sur les structures topology et topologyMap résultantes de l'emploi d'étiquettes uniques dans la requête.

Les étiquettes sont des noms spécifiés par l'utilisateur dans la requête pour désigner des relations et des éléments de configuration dans des configurations particulières. Les étiquettes spécifiées dans la requête sont utilisées comme étiquettes de nœud dans la carte renvoyée. Si aucune étiquette n'est spécifiée, le Nom de type CI ou Relation est utilisé comme étiquette dans la carte résultante. L'exemple suivant illustre la définition des étiquettes IISHost et DBHost à la place de l'étiquette Host par défaut et des étiquettes ContainerIIS et ContainsDB à la place de l'étiquette Container Link par défaut.

L'exemple suivant représente un petit modèle d'Univers IT. Il existe trois hôtes : H1, H2, H3, qui hébergent des serveurs Web (WS) et des gestionnaires de base de données (DB). WS1 réside sur H1. DB1 et WS2 résident sur H2. DB2 réside sur H3.



Cette requête est définie à l'aide des étiquettes par défaut :



Le résultat de l'exécution de cette requête TQL sur l'Univers IT peut être un élément Topology ou TopologyMap.

Réponse Topology

```
CIs: H1, H2, H3, WS1, WS2, DB1, DB2
Relations: H1-WS1, H1-H2, H2-H3, WS2-H2, DB1-H2, DB2-H3
```

Réponse TopologyMap

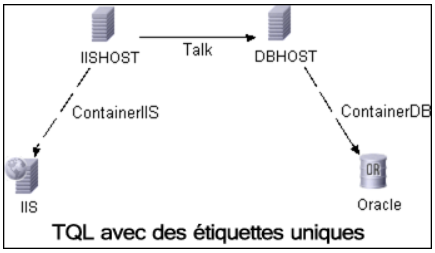
```
CINode:
    label: Host
    CIs: H1, H2
CINode:
    label: Host
    CIs: H2, H3
CINode:
    label: DB
    CIs: DB1, DB2
CINode:
    label: Webserver
CIs: IIS
relationNode:
    label: talk
    relations: H1-H2, H2-H3

relationNode:
    label: Container Link
    relations: WS1-H1, WS2-H2
relationNode:
    label: Container Link
    relations: DB2-H3, DB1-H2
```

Dans la réponse TopologyMap ci-dessus, les deux premiers CINode contiennent des étiquettes Host identiques, correspondant aux deux CI Host de la requête. Ces deux CINode contiennent l'hôte H2, sans qu'il soit indiqué pourquoi H2 est dupliqué.

Les deux derniers relationNode contiennent des étiquettes Contained identiques, correspondant aux deux relations Container link de la requête.

Les duplications ont lieu parce qu'aucune étiquette unique n'est spécifiée dans la requête, ce qui entraîne l'utilisation d'étiquettes par défaut (les noms de type Host et Container) dans la carte. Pour extraire une carte plus exploitable, définissez des requêtes avec des étiquettes uniques pour chaque configuration à faire correspondre, comme indiqué dans la requête suivante :



Le résultat topology est identique à celui du code TQL sans étiquettes uniques. Le résultat de topologyMap, cependant, est différent : Chaque étiquette est maintenant unique.

```
CINode:
    label: IISHOST
    CIs: H1, H2
CINode:
    label: DBHOST
    CIs: H2, H3

...

relationNode:
    label: ContainerIIS
    relations: WS1-H1, WS2-H2
relationNode:
    label: ContainerDB
    relations: DB2-H3, DB1-H2
```

Cette carte montre clairement pourquoi H2 est renvoyé deux fois. Les étiquettes uniques indiquent qu'il est renvoyé une fois en tant qu'hôte de serveur Web et une fois en tant qu'hôte de base de données.

Conseil : Lorsque cela est possible dans la base CMDB, appliquez à des configurations particulières des étiquettes uniques définies par l'utilisateur.

Tâches

Appel du service Web

Vous 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", page 341.

Interrogation de la base CMDB

La base CMDB est interrogée à l'aide des API décrites dans "Méthodes de requête UCMDB", page 314.

Les requêtes et les éléments CMDB renvoyés contiennent toujours des ID UMDB réels.

Pour des exemples d'utilisation des méthodes de requête, voir "Exemple de requête", page 347.

Cette section comprend les rubriques suivantes :

Calcul de réponse Juste-à-temps

Pour toutes les méthodes de requête, le serveur UMDB calcule les valeurs demandées par la méthode à la réception de la demande et 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.

Conseil : 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 328 et "releaseChunks", page 330.

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 référence des propriétés par leur nom, elle transmet un élément PropertiesList.

Conseil : 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 presque toujours 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 (une 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 type de CI. 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 331.

Pour des exemples d'utilisation des méthodes de mise à jour, voir "Exemple de mise à jour", page 363.

Cette tâche comprend les étapes suivantes :

- "Paramètres de mise à jour UCMDB", page 309
- "Utilisation de types d'ID avec les méthodes de mise à jour", page 310
- "Méthodes de mise à jour UCMDB", page 331

Paramètres de mise à jour UCMDB

Cette rubrique décrit les paramètres utilisés uniquement par les méthodes de mise à jour du service. Pour plus d'informations, voir la documentation sur les schémas.

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 309.

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 331 et "Méthodes de requête UCMDB", page 314.

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 "Gestionnaire des types de CI" dans le *Manuel de modélisation HP Universal CMDB*.

Pour des exemples d'utilisation des méthodes de modèle de classe, voir "Exemple de modèle de classe", page 367.

Cette section fournit des informations sur les méthodes suivantes qui renvoient des informations sur les types de CI et les relations :

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 377.
className	Nom du type. Pour plus d'informations, voir "Nom de type", page 379.

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 377.

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 clés, vous devez également interroger les classes parentes jusqu'à la classe de base. getCmdbClassDefinition identifie comme attributs clés 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 377.
className	Nom du type. Pour plus d'informations, voir "Nom de type", page 379.

Sortie

Paramètre	Remarque
cmdbClass	Définition de classe, composée de name, classType, displayLabel, description, parentName, de qualificatifs et d'attributs.
comments	À usage interne uniquement.

Interrogation de l'analyse d'impact

Identifier dans les méthodes d'analyse d'impact 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 "Exemple d'analyse de l'impact", page 369.

Références

Méthodes de requête UCMDB

Cette section fournit des informations sur les méthodes suivantes :

executeTopologyQueryByName

La méthode executeTopologyQueryByName extrait la carte topologique qui correspond à la requête spécifiée.

Conseil : La carte contient plus d'informations et est plus facile à comprendre si l'étiquette de chaque CINode et de chaque relationNode dans le code TQL est unique. Pour plus d'informations, voir "Renvoi d'éléments de carte topologique non ambigus", page 299.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir "CmdbContext", page 377.
queryName	Nom du code TQL dans CMDB avec lequel extraire la carte.
queryTypedProperties	Collection d'ensembles de propriétés à extraire vers des éléments d'un type d'élément de configuration particulier.

Sortie

Paramètre	Remarque
topologyMap	Pour plus d'informations, voir "TopologyMap", page 382.

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 377.
queryName	Nom du code TQL paramétré dans la base CMDB pour lequel extraire la carte.
parameterizedNodes	Conditions que chaque nœud doit satisfaire pour être inclus dans les résultats de la requête.
queryTypedProperties	Collection d'ensembles de propriétés à extraire vers des éléments d'un type d'élément de configuration particulier.

Sortie

Paramètre	Remarque
topologyMap	Pour plus d'informations, voir "TopologyMap", page 382.
chunkInfo	Pour plus d'informations, voir : "ChunkInfo", page 382, "Traitement de longues réponses", page 305.

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 parameterizedNodes. 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 la base 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.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir "CmdbContext", page 377.
queryXML	Chaîne XML représentant un code TQL sans balises de ressource.
parameterizedNodes	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 382.
chunkInfo	Pour plus d'informations, voir "ChunkInfo", page 382 et "Traitement de longues réponses", page 305.

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 377.
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
changeDataInfo	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 377.
ID	ID du CI avec lequel extraire les voisins. 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 379.
CIProperties	Données à renvoyer sur chaque élément de configuration (appelées Mise en page dans l'interface utilisateur). Pour plus d'informations, voir "TypedProperties", page 308.
relationProperties	Données à renvoyer sur chaque relation (appelées Mise en page dans l'interface utilisateur). Pour plus d'informations, voir "TypedProperties", page 308

Sortie

Paramètre	Remarque
topology	Pour plus d'informations, voir "Topology", page 381.
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 377.
CIsTypedProperties	Collection de propriétés typées. Pour plus d'informations, voir "Autres éléments de spécification de propriétés", page 308.
IDs	Seuls les ID CMDB réels sont valides dans cette collection.

Sortie

Paramètre	Remarque
CIs	Collection d'éléments de configuration.
chunkInfo	Pour plus d'informations, voir : "ChunkInfo", page 382, "Traitement de longues réponses", page 305.

getCIsByType

La méthode getCIsByType renvoie la collection d'éléments de configuration du type spécifié et de tous les types hérités du type spécifié.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir "CmdbContext", page 377.
type	Nom de classe. Pour plus d'informations, voir "Nom de type", page 379.
properties	Données à renvoyer sur chaque élément de configuration. Pour plus d'informations, voir "CustomProperties", page 308.

Sortie

Paramètre	Remarque
CIs	Collection d'éléments de configuration.
chunkInfo	Pour plus d'informations, voir : "ChunkInfo", page 382, "Traitement de longues réponses", page 305.

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'y a 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 dans un souci de 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 dans un souci de 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(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.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.addPropertyName("name");
propertiesList.addPropertyName("size");
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties)
```

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir "CmdbContext", page 377.
type	Nom de classe. Pour plus d'informations, voir "Nom de type", page 379. Le type peut être n'importe quel type défini à l'aide du Gestionnaire des types de CI. Pour plus d'informations, voir "Gestionnaire des types de CI" 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 308.
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 382 et "Traitement de longues réponses", page 305.

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 377.
viewName	Nom d'une vue, c'est-à-dire d'un sous-ensemble du modèle de classe dans la base CMDB.

Sortie

Paramètre	Remarque
queryName	Nom du code TQL dans la base CMDB sur lequel la vue est 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 377.
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
queryName	Nom du code TQL dans la base CMDB sur lequel la vue es basée.

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 377.
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
queryName	Nom du code TQL dans la base CMDB sur lequel la vue es basée.

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 382 et "Interrogation de la base CMDB", page 304.

L'application cliente doit être capable de traiter les cartes partielles. Reportez-vous à l'exemple suivant de traitement d'une collection de CI et à l'exemple de fusion de segments dans une carte à la section "Exemple de requête", page 347.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir "CmdbContext", page 377.
ChunkRequest	Numéro du segment à extraire et ChunkInfo qui est renvoyé par la méthode de requête.

Sortie

Paramètre	Remarque
topologyMap	Pour plus d'informations, voir "TopologyMap", page 382.
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,
chunkRequest);
     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
               }
          }
}
```

releaseChunks

La méthode releaseChunks libère la mémoire des segments qui contiennent les données de la requête.

Conseil : 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 377.
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

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 377.
updateExisting	À définir comme *true* pour mettre à jour les éléments qui existent déjà dans la base CMDB. À définir comme *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 309.
ignoreValidation	Si la valeur est true, aucun contrôle n'est effectué avant la mise à jour de la base CMDB.

Sortie

Paramètre	Remarque
CreatedIDsMap	Mappe des ID de client sur des ID CMDB. Pour plus d'informations, voir "addCIsAndRelations", page 331.
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 377.
CIsAndRelationsUpdates	Éléments à supprimer. Pour plus d'informations, voir "CIsAndRelationsUpdates", page 309.

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 377.
CIsAndRelationsUpdates	Éléments à mettre à jour. Pour plus d'informations, voir "CIsAndRelationsUpdates", page 309.
ignoreValidation	Si la valeur est true, aucun contrôle n'est effectué avant la mise à jour de la base CMDB.

Sortie

Paramètre	Remarque
CreatedIDsMap	Mappe des ID de client sur des ID CMDB. Pour plus d'informations, voir "addCIsAndRelations", page 331.

Méthodes d'analyse d'impact UCMDB

Cette section fournit des informations sur les méthodes suivantes :

- "calculateImpact", page 335
- "getImpactPath", page 336
- "getImpactRulesByNamePrefix", page 337

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.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir "CmdbContext", page 377.
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 381.
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 est utilisée comme argument d'entrée identifier de getImpactPath.

Entrée

Paramètre	Remarque
cmdbContext	Pour plus d'informations, voir "CmdbContext", page 377.
identifier	Clé de la réponse du serveur qui a été renvoyée par calculateImpact.
relation	Relation basé sur l'un des ShallowRelation renvoyés 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 end1ID, d'un end2ID, d'un rule et d'un 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 377.
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.

Méthodes de gestion des flux de données

Cette section contient une liste des opérations de service Web et un court résumé de leur utilisation. Pour une documentation complète sur la demande et la réponse de chaque opération, voir *gestion des flux de données Schema Reference*.

Contenu de cette section :

Méthodes de gestion des travaux de gestion des flux de données

- **activateJob**

  Active le travail spécifié.

- **deactivateJob**

  Désactive le travail spécifié.

- **dispatchAdHocJob**

  Envoie un travail sur la sonde ad hoc. Le travail doit être actif et contenir le CI déclencheur spécifié.

- **getDiscoveryJobsNames**

  Renvoie la liste des noms de travaux.

- **isJobActive**

  Vérifie si le travail est actif.

Méthodes de gestion de déclencheurs

- **addTriggerCI**

  Ajoute un nouveau CI déclencheur au travail spécifié.

➤ **addTriggerTQL**

Ajoute un nouveau code TQL déclencheur au travail spécifié.

➤ **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.

➤ **removeTriggerCI**

Supprime le CI spécifié de la liste des CI qui déclenchent le travail.

➤ **removeTriggerTQL**

Supprime le code TQL spécifié de la liste des requêtes qui déclenchent le travail.

➤ **setTriggerTQLProbesLimit**

Restreint à la liste spécifiée les sondes dans lesquelles le code TQL est actif dans le travail.

Méthodes de données de domaine et de sonde

➤ **getDomainType**

Renvoie le type de domaine.

➤ **getDomainsNames**

Renvoie les noms des domaines actuels.

➤ **getProbeIPs**

Renvoie les adresses IP de la sonde spécifiée.

➤ **getProbesNames**

Renvoie les noms des sondes dans le domaine spécifié.

➤ **getProbeScope**

Renvoie la définition d'étendue de la sonde spécifiée.

➤ **isProbeConnected**

Vérifie si la sonde spécifiée est connectée.

➤ **updateProbeScope**

Définit l'étendue de la sonde spécifiée, en remplacement de la sonde existante.

Méthodes de données d'informations d'identification

➤ **addCredentialsEntry**

Ajoute une entrée d'informations d'identification au protocole spécifié pour le domaine spécifié.

➤ **getCredentialsEntriesIDs**

Renvoie les ID des informations d'identification définies pour le protocole spécifié.

➤ **getCredentialsEntry**

Renvoie les informations d'identification définies pour le protocole spécifié. Les attributs chiffrés sont renvoyés vides.

➤ **removeCredentialsEntry**

Supprime les informations d'identification spécifiées du protocole.

➤ **updateCredentialsEntry**

Définit de nouvelles valeurs pour les propriétés de l'entrée des informations d'identification spécifiées.

Méthodes d'actualisation de données

➤ **rediscoverCIs**

Recherche les déclencheurs ayant découvert les objets CI spécifiés et réexécute ces déclencheurs. (Notez que la commande de réexécution a une priorité supérieure aux autres éléments planifiés.)

rediscoverCIs s'exécute de façon asynchrone. Appelez **checkDiscoveryProgress** pour déterminer quand la redécouverte est terminée.

➤ **checkDiscoveryProgress**

Renvoie la progression du dernier appel **rediscoverCIs** sur les ID spécifiés. La réponse est une valeur de 0 à 1. Lorsque la réponse est 1, l'appel **rediscoverCIs** est terminé.

➤ **rediscoverViewCIs**

Recherche les déclencheurs ayant créé les données pour alimenter la vue spécifiée et les réexécute. (Notez que la commande de réexécution a une priorité supérieure aux autres éléments planifiés.)

rediscoverViewCIs s'exécute de façon asynchrone. Appelez **checkViewDiscoveryProgress** pour déterminer quand la redécouverte est terminée.

- **checkViewDiscoveryProgress**

  Renvoie la progression du dernier appel **rediscoverViewCIs** sur la vue spécifiée. La réponse est une valeur de 0 à 1. Lorsque la réponse est 1, l'appel **rediscoverCIs** est terminé.

Cas d'utilisation

Les cas d'utilisation suivants nécessitent deux systèmes :

- serveur HP Universal CMDB ;
- système tiers contenant un référentiel des éléments de configuration.

Cette section comprend les rubriques suivantes :

- "Remplissage de CMDB", page 341
- "Interrogation de CMDB", page 342
- "Interrogation du modèle de classe", page 342
- "Analyse de l'impact des modifications", page 342

Remplissage de CMDB

Cas d'utilisation :

- Un outil tiers de gestion des actifs met à jour la base 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 de CMDB.

Interrogation de 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 la base 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

Cette section comprend les rubriques suivantes :

Exemple de classe de base

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.services.UcmdbServiceStub;
import com.hp.ucmdb.generated.types.CmdbContext;
import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;
```

```
import org.apache.axis2.transport.http.HttpTransportProperties;

import java.net.MalformedURLException;
import java.net.URL;
```

```
/**
* User: hbarkai
* Date: Jul 12, 2007
*/
abstract class Demo {
```

```
UcmdbService stub;
CmdbContext context;
```

```
public void initDemo() {
    try {
        setStub(createUcmdbService("admin", "admin"));
        setContext();
    } catch (Exception e) {
        //handle exception
    }
}
```

```
public UcmdbService getStub() {
    return stub;
}
```

```
public void setStub(UcmdbService stub) {
    this.stub = stub;
}
```

```
public CmdbContext getContext() {
    return context;
}
```

```
public void setContext() {
    CmdbContext context = new CmdbContext();
    context.setCallerApplication("demo");
    this.context = context;
}
```

```
//connection to service  - for axis2/jibx client

private static final String PROTOCOL = "http";
private static final String HOST_NAME = "host_name";
private static final int PORT = 8080;
private static final String FILE = "/axis2/services/UcmdbService";

protected UcmdbService createUcmdbService
    (String username, String password) throws Exception{
        URL url;
        UcmdbServiceStub serviceStub;
```

```
        try {
            url = new URL
                (Demo.PROTOCOL, Demo.HOST_NAME,
                Demo.PORT, Demo.FILE);
            serviceStub = new UcmdbServiceStub(url.toString());
            HttpTransportProperties.Authenticator auth =
                new HttpTransportProperties.Authenticator();
            auth.setUsername(username);
            auth.setPassword(password);
serviceStub._getServiceClient().getOptions().setProperty
    (HTTPConstants.AUTHENTICATE,auth);
```

```
    } catch (AxisFault axisFault) {
            throw new Exception
            ("Failed to create SOAP adapter for "
                + Demo.HOST_NAME , axisFault);
```

```
        } catch (MalformedURLException e) {

            throw new Exception
                ("Failed to create SOAP adapter for "
                    + Demo.HOST_NAME, e);
        }
        return serviceStub;
 }
}
```

Exemple de requête

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.query.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.props.*;

import java.rmi.RemoteException;
```

```
public class QueryDemo extends Demo{

    UcmdbService stub;
    CmdbContext context;
```

```
    public void getCIsByTypeDemo() {
        GetCIsByType request = new GetCIsByType();
        //set cmdbcontext
        CmdbContext cmdbContext = getContext();
        request.setCmdbContext(cmdbContext);
        //set CIs type
        request.setType("anyType");
        //set CIs propeties to be retrieved
        CustomProperties customProperties = new CustomProperties();
        PredefinedProperties predefinedProperties =
            new PredefinedProperties();
        SimplePredefinedProperty simplePredefinedProperty =
            new SimplePredefinedProperty();
        simplePredefinedProperty.setName
            (SimplePredefinedProperty.nameEnum.DERIVED);
        SimplePredefinedPropertyCollection
            simplePredefinedPropertyCollection =
                new SimplePredefinedPropertyCollection();
```

```
        simplePredefinedPropertyCollection.addSimplePredefinedProperty
            (simplePredefinedProperty);
        predefinedProperties.setSimplePredefinedProperties
            (simplePredefinedPropertyCollection);
        customProperties.setPredefinedProperties(predefinedProperties);
        request.setProperties(customProperties);
        try {
            GetCIsByTypeResponse response =
                getStub().getCIsByType(request);
            TopologyMap map =
                getTopologyMapResultFromCIs
                    (response.getCIs(), response.getChunkInfo());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }
```

```
    public void getCIsByIdDemo() {
        GetCIsById request = new GetCIsById();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set ids
        ID id1 = new ID();
        id1.setBase("cmdbobjectidCIT1");
        ID id2 = new ID();
        id2.setBase("cmdbobjectidCIT2");
        IDs ids = new IDs();
        ids.addID(id1);
        ids.addID(id2);
        request.setIDs(ids);
        //set CIs properties to be retrieved
        TypedPropertiesCollection properties =
            new TypedPropertiesCollection();
```

```
    TypedProperties typedProperties1 =
            new TypedProperties();
        typedProperties1.setType("CIT1");
```

```
        CustomTypedProperties customProperties1 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties1 =
            new PredefinedTypedProperties();
        SimpleTypedPredefinedProperty simplePredefinedProperty1 =
            new SimpleTypedPredefinedProperty();
        simplePredefinedProperty1.setName
            (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
        SimpleTypedPredefinedPropertyCollection
            simplePredefinedPropertyCollection1 =
                new SimpleTypedPredefinedPropertyCollection();
        simplePredefinedPropertyCollection1
            .addSimpleTypedPredefinedProperty
                (simplePredefinedProperty1);
```

```
predefinedProperties1.
        setSimpleTypedPredefinedProperties
            (simplePredefinedPropertyCollection1);
        customProperties1.
            setPredefinedTypedProperties
                (predefinedProperties1);
        typedProperties1.setProperties(customProperties1);
        properties.addTypedProperties(typedProperties1);
```

```
        TypedProperties typedProperties2 =
            new TypedProperties();
        typedProperties2.setType("CIT2");
        CustomTypedProperties customProperties2 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties2 =
            new PredefinedTypedProperties();
        SimpleTypedPredefinedProperty simplePredefinedProperty2 =
            new SimpleTypedPredefinedProperty();
        simplePredefinedProperty2.setName
            (SimpleTypedPredefinedProperty.nameEnum.NAMING);
        SimpleTypedPredefinedPropertyCollection
            simplePredefinedPropertyCollection2 =
                new SimpleTypedPredefinedPropertyCollection();
```

```
        simplePredefinedPropertyCollection2.
            addSimpleTypedPredefinedProperty
                (simplePredefinedProperty2);
```

```
        predefinedProperties2.setSimpleTypedPredefinedProperties
            (simplePredefinedPropertyCollection2);
        customProperties2.setPredefinedTypedProperties
            (predefinedProperties2);
        typedProperties2.setProperties(customProperties2);
        properties.addTypedProperties(typedProperties2);
```

```
    request.setCIsTypedProperties(properties);
        try {
            GetCIsByIdResponse response =
                getStub().getCIsById(request);
            CIs cis = response.getCIs();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getFilteredCIsByTypeDemo() {
        GetFilteredCIsByType request = new GetFilteredCIsByType();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set CIs type
        request.setType("anyType");
        //sets Filter conditions
        Conditions conditions = new Conditions();
        IntConditions intConditions = new IntConditions();
        IntCondition intCondition = new IntCondition();
        IntProp intProp = new IntProp();
        intProp.setName("int_attr1");
```

```
intProp.setValue(100);
intCondition.setCondition(intProp);
intCondition.setIntOperator
    (IntCondition.intOperatorEnum.Greater);
intConditions.addIntCondition(intCondition);
```

```
conditions.setIntConditions(intConditions);
request.setConditions(conditions);
//set logical operator for conditions
request.setConditionsLogicalOperator
    (GetFilteredCIsByType.conditionsLogicalOperatorEnum.AND);
//set CIs properties to be retrieved
CustomProperties customProperties =
    new CustomProperties();
PredefinedProperties predefinedProperties =
    new PredefinedProperties();
SimplePredefinedProperty simplePredefinedProperty =
    new SimplePredefinedProperty();
simplePredefinedProperty.setName
    (SimplePredefinedProperty.nameEnum.NAMING);
```

```
SimplePredefinedPropertyCollection
    simplePredefinedPropertyCollection =
        new SimplePredefinedPropertyCollection();
simplePredefinedPropertyCollection.
    addSimplePredefinedProperty
        (simplePredefinedProperty);
predefinedProperties.setSimplePredefinedProperties
    (simplePredefinedPropertyCollection);
customProperties.setPredefinedProperties
    (predefinedProperties);
```

```
request.setProperties(customProperties);
try {
    GetFilteredCIsByTypeResponse response =
        getStub().getFilteredCIsByType(request);
    TopologyMap map =
        getTopologyMapResultFromCIs
            (response.getCIs(), response.getChunkInfo());
```

```
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void executeTopologyQueryByNameDemo() {
        ExecuteTopologyQueryByName request = new
ExecuteTopologyQueryByName();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set query name
        request.setQueryName("queryName");
```

```
        try {
            ExecuteTopologyQueryByNameResponse response =
                getStub().executeTopologyQueryByName(request);
            TopologyMap map =
                getTopologyMapResult
                    (response.getTopologyMap(), response.getChunkInfo());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
// assume the follow query was defined at UCMDB
// Query Name: exampleQuery
// Query sketch:
//                                  Host
//                                  / \
//                                ip  Disk
// Query Parameters:
//     Host-
//          host_os (like)
//     Disk-
//          disk_failures (equal)
```

```
public void executeTopologyQueryByNameWithParametersDemo() {
    ExecuteTopologyQueryByNameWithParameters request =
        new ExecuteTopologyQueryByNameWithParameters();
    CmdbContext cmdbContext = getContext();
    //set cmdbcontext
    request.setCmdbContext(cmdbContext);
    //set query name
    request.setQueryName("queryName");
    //set parameters
    ParameterizedNode hostParametrizedNode =
        new ParameterizedNode();
    hostParametrizedNode.setNodeLabel("Host");
    CIProperties parameters = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp = new StrProp();
    strProp.setName("host_os");
    strProp.setValue("%2000%");
    strProps.addStrProp(strProp);
    parameters.setStrProps(strProps);
    hostParametrizedNode.setParameters(parameters);
    request.addParameterizedNodes(hostParametrizedNode);
    ParameterizedNode diskParametrizedNode =
        new ParameterizedNode();
```

```
    diskParametrizedNode.setNodeLabel("Disk");
    CIProperties parameters1 = new CIProperties();
    IntProps intProps = new IntProps();
```

```
    IntProp intProp = new IntProp();
        intProp.setName("disk_failures");
        intProp.setValue(30);
        intProps.addIntProp(intProp);
        parameters1.setIntProps(intProps);
        diskParametrizedNode.setParameters(parameters1);
```

```
        request.addParameterizedNodes(diskParametrizedNode);
        try {
            ExecuteTopologyQueryByNameWithParametersResponse
                response =
                getStub().executeTopologyQueryByNameWithParameters
                    (request);
            TopologyMap map =
                getTopologyMapResult
                    (response.getTopologyMap(), response.getChunkInfo());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
/   // assume the follow query was defined at UCMDB
    // Query Name: exampleQuery
    // Query sketch:
    //                              Host
    //                              / \
    //                             ip  Disk
    // Query Parameters:
    //     Host-
    //         host_os (like)
    //     Disk-
    //         disk_failures (equal)
```

```
public void executeTopologyQueryWithParametersDemo() {
    ExecuteTopologyQueryWithParameters request =
        new ExecuteTopologyQueryWithParameters();
    CmdbContext cmdbContext = getContext();
    //set cmdbcontext
    request.setCmdbContext(cmdbContext);
    //set query definition
    String queryXml = "<xml that represents the query above>";
    request.setQueryXml(queryXml);
    //set parameters
    ParameterizedNode hostParametrizedNode =
        new ParameterizedNode();
```

```
hostParametrizedNode.setNodeLabel("Host");
    CIProperties parameters = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp strProp = new StrProp();
    strProp.setName("host_os");
    strProp.setValue("%2000%");
    strProps.addStrProp(strProp);
    parameters.setStrProps(strProps);
    hostParametrizedNode.setParameters(parameters);
    request.addParameterizedNodes(hostParametrizedNode);
    ParameterizedNode diskParametrizedNode =
        new ParameterizedNode();
    diskParametrizedNode.setNodeLabel("Disk");
    CIProperties parameters1 = new CIProperties();
    IntProps intProps = new IntProps();
    IntProp intProp = new IntProp();
    intProp.setName("disk_failures");
    intProp.setValue(30);
    intProps.addIntProp(intProp);
    parameters1.setIntProps(intProps);
    diskParametrizedNode.setParameters(parameters1);
    request.addParameterizedNodes(diskParametrizedNode);
```

```
        try {
            ExecuteTopologyQueryWithParametersResponse
            response = getStub().executeTopologyQueryWithParameters
                (request);
            TopologyMap map =
                getTopologyMapResult
                    (response.getTopologyMap(), response.getChunkInfo());
```

```
         } catch (RemoteException e) {
              //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getCINeighboursDemo() {
        GetCINeighbours request = new GetCINeighbours();
        //set cmdbcontext
        CmdbContext cmdbContext = getContext();
        request.setCmdbContext(cmdbContext);
        // set CI id
        ID id = new ID();
        id.setBase("cmdbobjectidCIT1");
        request.setID(id);
        //set neighbour type
        request.setNeighbourType("neighbourType");
        //set Neighbours CIs propeties to be retrieved
        TypedPropertiesCollection properties =
            new TypedPropertiesCollection();
        TypedProperties typedProperties1 = new TypedProperties();
        typedProperties1.setType("neighbourType");
        CustomTypedProperties customProperties1 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties1 =
            new PredefinedTypedProperties();
```

```
QualifierProperties qualifierProperties =
    new QualifierProperties();
qualifierProperties.addQualifierName("ID_ATTRIBUTE");
predefinedProperties1.setQualifierProperties(qualifierProperties);
customProperties1.setPredefinedTypedProperties
    (predefinedProperties1);
typedProperties1.setProperties(customProperties1);
properties.addTypedProperties(typedProperties1);
request.setCIProperties(properties);
```

```
TypedPropertiesCollection relationsProperties =
    new TypedPropertiesCollection();
TypedProperties typedProperties2 = new TypedProperties();
typedProperties2.setType("relationType");
CustomTypedProperties customProperties2 =
    new CustomTypedProperties();
```

```
PredefinedTypedProperties predefinedProperties2 =
    new PredefinedTypedProperties();
SimpleTypedPredefinedProperty simplePredefinedProperty2 =
    new SimpleTypedPredefinedProperty();
simplePredefinedProperty2.setName
```

```
    (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
SimpleTypedPredefinedPropertyCollection
    simplePredefinedPropertyCollection2 =
        new SimpleTypedPredefinedPropertyCollection();
simplePredefinedPropertyCollection2.
    addSimpleTypedPredefinedProperty
        (simplePredefinedProperty2);
predefinedProperties2.
    setSimpleTypedPredefinedProperties
        (simplePredefinedPropertyCollection2);
customProperties2.setPredefinedTypedProperties
    (predefinedProperties2);
typedProperties2.setProperties(customProperties2);
relationsProperties.addTypedProperties(typedProperties2);
request.setRelationProperties(relationsProperties);
```

```
        try {
            GetCINeighboursResponse response =
                getStub().getCINeighbours(request);
            Topology topology = response.getTopology();
        } catch (RemoteException e) {
            //handle exception
          } catch (UcmdbFaultException e) {
             //handle exception
          }

    }
```

```
//get Topology Map for chunked/non-chunked result

    private TopologyMap getTopologyMapResult(TopologyMap topologyMap, ChunkInfo
chunkInfo) {
        if(chunkInfo.getNumberOfChunks() == 0) {
            return topologyMap;
        } else {
```

```
            topologyMap = new TopologyMap();
            for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
                ChunkRequest chunkRequest = new ChunkRequest();
                chunkRequest.setChunkInfo(chunkInfo);
                chunkRequest.setChunkNumber(i);
                PullTopologyMapChunks req =
                    new PullTopologyMapChunks();
                req.setChunkRequest(chunkRequest);
                req.setCmdbContext(getContext());
                PullTopologyMapChunksResponse res = null;
```

```
                try  {
                    res = getStub().pullTopologyMapChunks(req);
                    TopologyMap map = res.getTopologyMap();
                    topologyMap = mergeMaps(topologyMap, map);
                } catch (RemoteException e) {
                    //handle exception
                } catch (UcmdbFaultException e) {
                    //handle exception
                }
            }
        }
        return topologyMap;

    }
```

```
    private TopologyMap getTopologyMapResultFromCIs(CIs cis, ChunkInfo chunkInfo)
{
        TopologyMap topologyMap = new TopologyMap();
        if(chunkInfo.getNumberOfChunks() == 0) {
            CINode ciNode = new CINode();
            ciNode.setLabel("");
            ciNode.setCIs(cis);
            CINodes ciNodes = new CINodes();
            ciNodes.addCINode(ciNode);
            topologyMap.setCINodes(ciNodes);
        } else {
```

```
            for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
                ChunkRequest chunkRequest =
                    new ChunkRequest();
                chunkRequest.setChunkInfo(chunkInfo);
                chunkRequest.setChunkNumber(i);
                PullTopologyMapChunks req =
                    new PullTopologyMapChunks();
                req.setChunkRequest(chunkRequest);
                req.setCmdbContext(getContext());
                PullTopologyMapChunksResponse res = null;
```

```
                try  {
                    res = getStub().pullTopologyMapChunks(req);
                } catch (RemoteException e) {
                    //handle exception
                } catch (UcmdbFaultException e) {
                    //handle exception
                }
                TopologyMap map = res.getTopologyMap();
                topologyMap = mergeMaps(topologyMap, map);
            }
```

```
            //release chunks
            ReleaseChunks req = new ReleaseChunks();
            req.setChunksKey(chunkInfo.getChunksKey());
            req.setCmdbContext(getContext());
```

```
            try {
                getStub().releaseChunks(req);
            } catch (RemoteException e) {
                //handle exception
            } catch (UcmdbFaultException e) {
                //handle exception
            }
        }
        return topologyMap;

    }
```

```
//=================================================
/*  WARNING merge will be correct only if a each node is given
    a unique name. This applies to both CI and Relation nodes .*/
//=================================================
    private TopologyMap mergeMaps(TopologyMap topologyMap, TopologyMap
newMap) {
        for(int i=0 ; i < newMap.getCINodes().sizeCINodeList() ; i++ ) {
            CINode ciNode = newMap.getCINodes().getCINode(i);
            boolean alreadyExist = false;
            if(topologyMap.getCINodes() == null) {
                topologyMap.setCINodes(new CINodes());
            }
```

```
        for(int j=0 ; j < topologyMap.getCINodes().sizeCINodeList() ; j++) {
            CINode ciNode2 = topologyMap.getCINodes().getCINode(j);
            if(ciNode2.getLabel().equals(ciNode.getLabel())){
```

```
                CIs cisTOAdd = ciNode.getCIs();
                CIs cis =
                    mergeCIsGroups
                    (topologyMap.getCINodes().getCINode(j).getCIs(),
                        cisTOAdd);
                topologyMap.getCINodes().getCINode(j).setCIs(cis);
                alreadyExist = true;
            }
        }
        if(!alreadyExist) {
            topologyMap.getCINodes().addCINode(ciNode);
        }

    }
```

```
    for(int i=0 ; i < newMap.getRelationNodes().sizeRelationNodeList() ; i++ ) {
        RelationNode relationNode =
            newMap.getRelationNodes().getRelationNode(i);
        boolean alreadyExist = false;
        if(topologyMap.getRelationNodes() == null) {
            topologyMap.setRelationNodes(new RelationNodes());
        }
```

```
            for(int j=0 ;
                    j < topologyMap.getRelationNodes().sizeRelationNodeList() ;
                    j++) {
                RelationNode relationNode2 =
                    topologyMap.getRelationNodes().getRelationNode(j);
                if(relationNode2.getLabel().equals(relationNode.getLabel())){
                    Relations relationsTOAdd = relationNode.getRelations();
                    Relations relations =
                        mergeRelationsGroups
                        (topologyMap.getRelationNodes().
                            getRelationNode(j).getRelations(),
                            relationsTOAdd);
                    topologyMap.getRelationNodes().
                        getRelationNode(j).setRelations(relations);
                    alreadyExist = true;
                }
            }
```

```
            if(!alreadyExist) {
                topologyMap.getRelationNodes().addRelationNode(relationNode);
            }
        }

        return topologyMap;

    }
```

```
    private Relations mergeRelationsGroups(Relations relations1, Relations relations2)
{
        for(int i=0 ; i < relations2.sizeRelationList() ; i++) {
            relations1.addRelation(relations2.getRelation(i));
        }
        return relations2;
    }
```

```
    private CIs mergeCIsGroups(CIs cis1, CIs cis2) {
        for(int i=0 ; i < cis2.sizeCIList() ; i++) {
            cis1.addCI(cis2.getCI(i));
        }
        return cis1;
    }


}
```

Exemple de mise à jour

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.update.AddCIsAndRelations;
import com.hp.ucmdb.generated.params.update.AddCIsAndRelationsResponse;
import com.hp.ucmdb.generated.params.update.UpdateCIsAndRelations;
import com.hp.ucmdb.generated.params.update.DeleteCIsAndRelations;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.update.CIsAndRelationsUpdates;
import com.hp.ucmdb.generated.types.update.ClientIDToCmdbID;

import java.rmi.RemoteException;

public class UpdateDemo extends Demo{
```

```
    public void getAddCIsAndRelationsDemo() {
        AddCIsAndRelations request = new AddCIsAndRelations();
        request.setCmdbContext(getContext());
        request.setUpdateExisting(true);
        CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        CI ci = new CI();
        ID id = new ID();
        id.setBase("temp1");
        id.setTemp(true);
```

```
        ci.setID(id);
        ci.setType("host");
```

```
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp = new StrProp();
        strProp.setName("host_key");
        String value = "blabla";
        strProp.setValue(value);
```

```
        strProps.addStrProp(strProp);
        props.setStrProps(strProps);
        ci.setProps(props);
        cis.addCI(ci);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
```

```
        try {
            AddCIsAndRelationsResponse response  =
                getStub().addCIsAndRelations(request);
            for(int i = 0  ;  i < response.sizeCreatedIDsMapList() ; i++) {
                ClientIDToCmdbID idsMap = response.getCreatedIDsMap(i);
                //do something
            }
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getUpdateCIsAndRelationsDemo() {
        UpdateCIsAndRelations request = new UpdateCIsAndRelations();
        request.setCmdbContext(getContext());
```

```
        CIsAndRelationsUpdates updates =
            new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        CI ci = new CI();
        ID id = new ID();
```

```
        id.setBase("temp1");
        id.setTemp(true);
        ci.setID(id);
        ci.setType("host");
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
```

```
        StrProp hostKeyProp = new StrProp();
        hostKeyProp.setName("host_key");
        String hostKeyValue = "blabla";
        hostKeyProp.setValue(hostKeyValue);
        strProps.addStrProp(hostKeyProp);

        StrProp hostOSProp = new StrProp();
        hostOSProp.setName("host_os");
        String hostOSValue = "winXP";
        hostOSProp.setValue(hostOSValue);
        strProps.addStrProp(hostOSProp);

        StrProp hostDNSProp = new StrProp();
        hostDNSProp.setName("host_dnsname");
        String hostDNSValue = "dnsname";
        hostDNSProp.setValue(hostDNSValue);
        strProps.addStrProp(hostDNSProp);

        props.setStrProps(strProps);
        ci.setProps(props);
        cis.addCI(ci);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);

        try {
            getStub().updateCIsAndRelations(request);
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getDeleteCIsAndRelationsDemo() {
        DeleteCIsAndRelations request =
            new DeleteCIsAndRelations();
        request.setCmdbContext(getContext());
        CIsAndRelationsUpdates updates =
            new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        CI ci = new CI();
        ID id = new ID();
        id.setBase("stam");
        id.setTemp(true);
        ci.setID(id);
        ci.setType("host");
```

```
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp1 = new StrProp();
        strProp1.setName("host_key");
        String value1 = "for_delete";
        strProp1.setValue(value1);
        strProps.addStrProp(strProp1);
        props.setStrProps(strProps);
        ci.setProps(props);
        cis.addCI(ci);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
```

```
        try {
            getStub().deleteCIsAndRelations(request);
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }


}
```

Exemple de modèle de classe

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.classmodel.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClassModelHierarchy;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClass;

import java.rmi.RemoteException;

public class ClassmodelDemo extends Demo{
```

```
    public void getClassAncestorsDemo() {
        GetClassAncestors request =
            new GetClassAncestors();
        request.setCmdbContext(getContext());
        request.setClassName("className");
```

```
        try {
            GetClassAncestorsResponse response  =
                getStub().getClassAncestors(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassHierarchy();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getAllClassesHierarchyDemo() {
        GetAllClassesHierarchy request =
            new GetAllClassesHierarchy();
        request.setCmdbContext(getContext());
        try {
            GetAllClassesHierarchyResponse response  =
                getStub().getAllClassesHierarchy(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassesHierarchy();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }
```

```
    public void getCmdbClassDefinitionDemo() {
        GetCmdbClassDefinition request =
            new GetCmdbClassDefinition();
        request.setCmdbContext(getContext());
        request.setClassName("className");
```

```
        try {
            GetCmdbClassDefinitionResponse response  =
            getStub().getCmdbClassDefinition(request);
            UcmdbClass ucmdbClass = response.getUcmdbClass();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }

    }


}
```

Exemple d'analyse de l'impact

```
package com.hp.ucmdb.demo;

import com.hp.ucmdb.generated.params.impact.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.impact.*;

import java.rmi.RemoteException;
```

```
/**
 * Date: Jul 17, 2007
 */
public class ImpactDemo extends Demo{

//Impact Rule Name : impactExample
//Impact Query:
//              Network
//                 |
//               Host
//                 |
//                IP
//Impact Action: network affect on ip ;severity 100% ; category: change
//
public void calculateImpactAndGetImpactPathDemo() {
    CalculateImpact request = new CalculateImpact();
    request.setCmdbContext(getContext());
    //set root cause ids
    IDs ids = new IDs();
    ID id = new ID();
    id.setBase("rootCauseCmdbID");
    ids.addID(id);
```

```
request.setIDs(ids);
//set impact category
request.setImpactCategory("change");
//set rule Names
ImpactRuleNames impactRuleNames = new ImpactRuleNames();
ImpactRuleName impactRuleName = new ImpactRuleName();
impactRuleName.setBase("impactExample");
impactRuleNames.addImpactRuleName(impactRuleName);
request.setImpactRuleNames(impactRuleNames);
//set severity
request.setSeverity(100);
CalculateImpactResponse response =
    new CalculateImpactResponse();
```

```
request.setIDs(ids);
//set impact category
request.setImpactCategory("change");
//set rule Names
ImpactRuleNames impactRuleNames = new ImpactRuleNames();
ImpactRuleName impactRuleName = new ImpactRuleName();
impactRuleName.setBase("impactExample");
impactRuleNames.addImpactRuleName(impactRuleName);
request.setImpactRuleNames(impactRuleNames);
//set severity
request.setSeverity(100);
CalculateImpactResponse response =
    new CalculateImpactResponse();
```

```
try {
    response  = getStub().calculateImpact(request);
} catch (RemoteException e) {
    //handle exception
```

```
    } catch (UcmdbFaultException e) {
        //handle exception
    }
    Identifier identifier= response.getIdentifier();
    Topology topology = response.getImpactTopology();
    Relation relation = topology.getRelations().getRelation(0);
    GetImpactPath request2 = new GetImpactPath();
    //set cmdb context
    request2.setCmdbContext(getContext());
    //set impact identifier
    request2.setIdentifier(identifier);
    //set shallowRelation
    ShallowRelation shallowRelation = new ShallowRelation();
    shallowRelation.setID(relation.getID());
    shallowRelation.setEnd1ID(relation.getEnd1ID());
    shallowRelation.setEnd2ID(relation.getEnd2ID());
    shallowRelation.setType(relation.getType());
    request2.setRelation(shallowRelation);
```

```
    try {
        GetImpactPathResponse response2 =
            getStub().getImpactPath(request2);
        ImpactTopology impactTopology =
            response2.getImpactPathTopology();
    } catch (RemoteException e) {
        //To change body of catch statement
        // use File | Settings | File Templates.
        e.printStackTrace();
    } catch (UcmdbFaultException e) {
        //To change body of catch statement
        // use File | Settings | File Templates.
        e.printStackTrace();
    }
}
```

```
public void getImpactRulesByGroupName() {
    GetImpactRulesByGroupName request =
        new GetImpactRulesByGroupName();
    //set cmdb context
    request.setCmdbContext(getContext());
    //set group names list
    request.addRuleGroupNameFilter("groupName1");
    request.addRuleGroupNameFilter("groupName2");
```

```
    try {
        GetImpactRulesByGroupNameResponse response  =
            getStub().getImpactRulesByGroupName(request);
        ImpactRules impactRules = response.getImpactRules();
    } catch (RemoteException e) {
        //handle exception
    } catch (UcmdbFaultException e) {
        //handle exception
    }

}
```

```
public void getImpactRulesByNamePrefix() {
        GetImpactRulesByNamePrefix request =
            new GetImpactRulesByNamePrefix();
        //set cmdb context
        request.setCmdbContext(getContext());
        //set prefixes list
        request.addRuleNamePrefixFilter("prefix1");
```

```
        try {
            GetImpactRulesByNamePrefixResponse response  =
                getStub().getImpactRulesByNamePrefix(request);
            ImpactRules impactRules = response.getImpactRules();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
}

}
```

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(cmdbContext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credentials");
            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", newCredsProperties);
        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,
CIProperties 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 GetDomainsNamesRequest(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=" +
isConnected);
            }
        }
    }
```

```
    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.AUTHENTIC
ATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }

        return discoveryService;

    }
} // End class
```

Paramètres généraux UCMDB

Cette section décrit les paramètres les plus courants des méthodes du service. Pour plus d'informations, reportez-vous à la documentation sur les schémas.

Cette section comprend les rubriques suivantes :

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" 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 312.

Types d'ID

Un élément ID peut contenir un ID réel, un ID temporaire, ou peut être vide.

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 vide signifie qu'aucune valeur n'est affectée.

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 la base CMDB peut identifier un élément de configuration de données existant en utilisant les propriétés clés du CI, ce dernier est utilisé selon le contexte comme s'il avait été identifiée avec un ID réel.

L'ID réel d'un CI est calculé par la base CMDB en fonction d'une combinaison du type et des propriétés clés du CI. L'ID réel d'une Relation est basé sur le type de la relation, l'ID des deux CI qui constituent cette relation et les propriétés clés de la relation. Par conséquent, les valeurs des attributs clés doivent être définies lors de la création du CI ou de la Relation. Si les valeurs des propriétés clés ne sont pas spécifiées à la création d'un CI, deux possibilités se présentent :

- Si le type de CI comprend un qualificatif RANDOM_GENERATED_ID, le serveur génère un ID unique.
- Si le type de CI ne comprend pas de qualificatif RANDOM_GENERATED_ID, une exception est levée.

Pour plus d'informations, voir "Gestionnaire des types de CI" dans le *Manuel de modélisation HP Universal CMDB*.

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 380 et "Relation", page 380.

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 de 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. En cas de recours à un ID temporaire, attribuez la valeur true à l'indicateur temp. Lors de la suppression d'un élément, l'ID peut être vide. Les Méthodes de requête UCMDB prennent des ID réels comme paramètres d'entrée et renvoient des ID réels dans les résultats des requêtes.

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" dans le *Manuel de modélisation HP Universal CMDB*.

L'élément props est une collection CIProperties. Pour plus d'informations, voir "CIProperties", page 379.

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 de Méthodes de mise à jour UCMDB pour mettre à jour une Relation, la valeur de l'ID de Relation peut être un ID CMDB réel ou un ID temporaire. Lors de la suppression d'un élément, l'ID peut être vide. Les Méthodes de requête UCMDB prennent des ID réels comme paramètres d'entrée et renvoie des 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 311.

Pour plus d'informations, voir "Gestionnaire des types de CI" 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 379.

Paramètres de sortie UCMDB

Cette section décrit les paramètres de sortie les plus courants des méthodes du service. Pour plus d'informations, reportez-vous à la documentation sur les schémas.

Cette section comprend les rubriques suivantes :

CIs

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 un label. 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 382).
- relationNodes. Il s'agit d'un ou de plusieurs RelationNode (voir "RelationNode", page 382).

Les label 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 305.

10

API HP Universal CMDB

Contenu de ce chapitre :

Concepts

Tâches

Références

Concepts

Conventions

Ce chapitre observe les conventions suivantes :

- **UCMDB** se rapporte à la base de données de gestion de la configuration universelle elle-même. **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.

Utilisation de l'API HP Universal CMDB

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 (CMDB). 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 ;
- 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 "TQL (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

Cette section comprend les rubriques suivantes :

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", page 391.

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, mais des applications qui se connectent à CMDB.

Pour plus d'informations, voir "Création d'un utilisateur d'intégration", page 388.

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 nouvelle définition de requête, le client :

1. extrait le service de requête du principal objet de service CMDB ;
2. extrait un objet d'usine de requêtes de l'objet de service ;
3. extrait une nouvelle définition de requête de l'usine.

```
UcmdbServiceProvider provider =
  UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
UcmdbService = provider.connect(provider.createCredentials(USERNAME,
  PASSWORD), provider.createClientContext("Test"));
TopologyQueryService queryService = ucmdbService.getTopologyQueryService();
TopologyQueryFactory factory = queryService.getFactory();
QueryDefinition queryDefinition = factory.createQueryDefinition("Test Query");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() + " hosts in uCMDB");
```

Les services accessibles depuis **UcmdbService** sont les suivants :

Méthodes de service	Utilisation
getClassModelService	Informations sur les types de CI et les relations
getDDMConfigurationService	Configuration du système de gestion des découvertes et des dépendances
getDDMManagementService	Analyse et affichage de la progression, des résultats et des erreurs du système de gestion des découvertes et des dépendances
getImpactAnalysisService	Exécution d'un scénario d'analyse d'impact (également appelée **corrélation**).

Méthodes de service	Utilisation
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.
getResourceBundleManagement Service	Identification de ressources (services de "regroupement"). Permet la création explicite de nouveaux indicateurs et le retrait d'indicateurs de toutes les ressources identifiées.
getSoftwareSignatureService	Définition de logiciels devant être découverts par le système de gestion des découvertes et des dépendances
getTopologyQueryService	Obtention d'informations sur l'Univers IT
getTopologyUpdateService	Modification d'informations dans l'Univers IT
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.
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.

Le client communique avec le serveur sur HTTP.

Tâches

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 entrant http://localhost:8080 dans un navigateur Web et en cliquant sur le lien **API Client Download**.

Placez le fichier jar dans le classpath avant de compiler ou d'exécuter votre application.

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 : Seul un utilisateur d'intégration peut se connecter à CMDB via ce jeu d'API. Toute tentative de connexion par d'autres types d'utilisateurs peut provoquer des erreurs, même lorsque la vérification LDAP est utilisée.

Pour créer un utilisateur d'intégration :

1. Lancez le navigateur Web, puis entrez l'adresse du serveur selon la syntaxe suivante :

   http://localhost:8080/jmx-console.

   Vous pouvez être amené à vous connecter avec un nom d'utilisateur et un mot de passe (les valeurs par défaut étant (sysadmin/sysadmin).

2. Sous UCMDB, cliquez sur **service=UCMDB Security Services** pour ouvrir la page JMX MBEAN View.

3. Recherchez l'opération **CreateIntegrationUser**. Cette méthode accepte les paramètres suivants :

   - **customerId**. ID du client.
   - **username**. Nom de l'utilisateur d'intégration.
   - **password**. Mot de passe de l'utilisateur d'intégration.
   - **dataStoreOrigin**. Nom du produit qui va faire appel à cet utilisateur d'intégration.

   Les opérations suivantes sont utiles pour la gestion de l'utilisateur d'intégration :

   - **DeleteIntegrationUser**. Supprime l'utilisateur d'intégration donné.
   - **ExportIntegrationUser**. Exporte l'utilisateur d'intégration vers un fichier XML dans le chemin indiqué (sur la machine serveur).
   - **getIntegrationUser**. Affiche les informations de l'utilisateur d'intégration.
   - **changeIntegrationUserPassword**. Change le mot de passe de l'utilisateur d'intégration.
   - **canUserAuthenticate**. **isIntegrationUser** a la valeur **true** : l'utilisateur d'intégration peut-il s'authentifier avec les informations d'identification données ?

4. Cliquez sur **Invoke**.

   Cliquez sur **Back to MBean View** pour créer d'autres utilisateurs ou fermez la console JMX.

5. Connectez-vous à UCMDB en tant qu'administateur.
6. Dans l'onglet **Administration**, exécutez le **Gestionnaire des composants applicatifs**.
7. Cliquez sur l'icône **Nouveau**.
8. Saisissez le nom du nouveau composant et cliquez sur **Suivant**.
9. Dans l'onglet Sélection des ressources, sous **Administration**, cliquez sur **Utilisateurs d'intégration**.
10. Sélectionnez un ou des utilisateurs que vous avez créés à l'aide de la console JMX.

11 Cliquez sur **Suivant** puis sur **Terminer**. Votre nouveau composant applicatif apparaît dans la liste Nom du composant applicatif, dans le Gestionnaire de composants applicatifs.

12 Déployez le composant applicatif pour les utilisateurs qui vont exécuter les applications API.

Pour plus d'informations, voir "Déployer un composant applicatif" dans le *Manuel d'administration HP Universal CMDB*.

Remarque :

Il y a un utilisateur d'intégration par 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** (**createSystemUser**, **removeSystemUser**, **showAllSystemUsers**, **changeSystemUserPassword**, **canSuperIntegrationUserAuthenticate**, etc.).

Il existe deux utilisateurs système prêts à l'emploi ; il est recommandé de modifier leurs mots de passe après l'installation à l'aide de la méthode **changeSystemUserPassword**.

- **sysadmin**/**sysadmin**
- **UISysadmin**/**UISysadmin** (cet utilisateur est également le superutilisateur d'intégration **SuperIntegrationUser**).

Si vous changez le mot de passe d'UISysadmin à l'aide de **changeSystemUserPassword**, vous devez exécuter la méthode suivante : dans la console JMX, recherchez le service **UCMDB-UI:name=UCMDB Integration**. Exécutez **setCMDBSuperIntegrationUser** avec le nom d'utilisateur et le nouveau mot de passe de l'utilisateur d'intégration.

Références

Référence des API HP Universal CMDB

Pour une documentation complète sur les API disponibles, voir "Introduction aux API", page 293.

Cas d'utilisation

Les cas d'utilisation suivants nécessitent deux systèmes :

- serveur HP Universal CMDB ;
- système tiers contenant un référentiel des éléments de configuration.

Cette section comprend les rubriques suivantes :

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 de 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

Cette section comprend les rubriques suivantes :

Exemple de point d'entrée

```
final String HOST_NAME = "localhost";
final int PORT = 8080;
UcmdbServiceProvider provider =
    UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
final String USERNAME = "integration_user";
final String PASSWORD = "integration_password";
Credentials credentials =
    provider.createCredentials(USERNAME, PASSWORD),
ClientContext clientContext = provider.createClientContext("Example");
UcmdbService ucmdbService = provider.connect(credentials, clientContext);
```

Exemples de requêtes

Les exemples suivants démontrent l'extraction d'une définition de classe unique et l'obtention d'une liste de toutes les définitions de type de CI et de leurs attributs.

Extraction d'une définition de classe

```
ClassModelService classModelService
    = ucmdbService.getClassModelService();
String typeName = "disk";
ClassDefinition def =
    classModelService.getClassDefinition(typeName);
System.out.println("Type " + typeName + " is derived from type "
    + def.getParentClassName());
System.out.println("Has " + def.getChildClasses().size() +
    " derived types");
System.out.println("Defined and inherited attributes:");
for (Attribute attr : def.getAllAttributes().values()) {
    System.out.println("Attribute " + attr.getName() +
    " of type " + attr.getType());
}
```

Extraction de la liste des définitions et attributs des types de CI

Cet exemple interroge les attributs d'un type de CI et imprime leurs noms et types.

```
ClassModelService classModelService =
     ucmdbService.getClassModelService();
for (ClassDefinition def : classModelService.getAllClasses()) {
    System.out.println("Type " + def.getName() +
        " (" + def.getDisplayName() + ") is derived from type "
        + def.getParentClassName());
    System.out.println
        ("Has " + def.getChildClasses().size() + " derived types");
    System.out.println
        ("Defined and inherited attributes:");
    for (Attribute attr : def.getAllAttributes().values()) {
        System.out.println
          ("Attribute " + attr.getName() +
          " of type " + attr.getType());
    }
}
```

Exemple de requête de topologie

```
TopologyQueryService queryService =
    ucmdbService.getTopologyQueryService();
TopologyQueryFactory queryFactory =
    queryService.getFactory();
QueryDefinition queryDefinition =
    queryFactory.createQueryDefinition
        ("Get hosts with more than one network interface");
String hostNodeName = "Host";
QueryNode hostNode =

queryDefinition.addNode(hostNodeName).ofType("host").queryProperty("display_label"
);
QueryNode ipNode =
    queryDefinition.addNode("IP").ofType("ip").queryProperty("ip_address");
hostNode.linkedTo(ipNode).withLinkOfType("contained").atLeast(2);
Topology topology = queryService.executeQuery(queryDefinition);
Collection<TopologyCI> hosts = topology.getCIsByName(hostNodeName);
for (TopologyCI host : hosts) {
    System.out.println("Host " + host.getPropertyValue("display_label"));
    for (TopologyRelation relation : host.getOutgoingRelations()) {
         System.out.println
        ("  has IP " + relation.getEnd2CI().getPropertyValue("ip_address"));
    }
}
```

Exemple de mise à jour de la topologie

```
TopologyUpdateService topologyUpdateService =
    ucmdbService.getTopologyUpdateService();
TopologyUpdateFactory topologyUpdateFactory =
    topologyUpdateService.getFactory();
TopologyModificationData topologyModificationData =
    topologyUpdateFactory.createTopologyModificationData();
CI host = topologyModificationData.addCI("host");
host.setPropertyValue("host_key", "test1");
CI ip = topologyModificationData.addCI("ip");
ip.setPropertyValue("ip_address", "127.0.0.10");
ip.setPropertyValue("ip_domain", "DefaultDomain");
topologyModificationData.addRelation("contained", host, ip);
topologyUpdateService.create
    (topologyModificationData, CreateMode.IGNORE_EXISTING);
```

Exemple d'analyse d'impact

```
ImpactAnalysisService impactAnalysisService =
    ucmdbService.getImpactAnalysisService();
ImpactAnalysisFactory impactFactory =
    impactAnalysisService.getFactory();
ImpactAnalysisDefinition definition =
    impactFactory.createImpactAnalysisDefinition();
definition.addTriggerCI(disk).withSeverity
    (impactFactory.getSeverityByName("Warning(2)"));
definition.useAllRules();
ImpactAnalysisResult impactResult =
    impactAnalysisService.analyze(definition);
AffectedTopology affectedCIs =
    impactResult.getAffectedCIs();
for (AffectedCI affectedCI : affectedCIs.getAllCIs()) {
     System.out.println("Affected " +
     affectedCI.getType() + " " + affectedCI.getId() +
     " - severity " + affectedCI.getSeverity());
}
```

Index

B

C

D

E

F

G

I

J

T

U

V