HP Universal CMDB

Für Windows und Red Hat Enterprise Linux Betriebsysteme

Softwareversion: 10.00

Entwicklerreferenzhandbuch

Datum der Dokumentveröffentlichung: Juni 2012

Datum des Software-Release: Juni 2012

Rechtliche Hinweise

Garantie

Die Garantiebedingungen für Produkte und Services von HP sind in der Garantieerklärung festgelegt, die diesen Produkten und Services beiliegt. Keine der folgenden Aussagen kann als zusätzliche Garantie interpretiert werden. HP haftet nicht für technische oder redaktionelle Fehler oder Auslassungen.

Die hierin enthaltenen Informationen können ohne vorherige Ankündigung geändert werden.

Eingeschränkte Rechte

Vertrauliche Computersoftware. Gültige Lizenz von HP für den Besitz, Gebrauch oder die Anfertigung von Kopien erforderlich. Entspricht FAR 12.211 und 12.212. Kommerzielle Computersoftware, Computersoftwaredokumentation und technische Daten für kommerzielle Komponenten werden an die U.S.-Regierung per Standardlizenz lizenziert.

Copyright-Hinweis

Markenhinweise

Adobe™ ist eine Marke von Adobe Systems Incorporated.

Microsoft® und Windows® sind in den USA eingetragene Marken der Microsoft Corporation.

UNIX® ist eine eingetragene Marke von The Open Group.

Aktualisierte Dokumentation

Auf der Titelseite dieses Dokuments befinden sich die folgenden identifizierenden Informationen:

- Software-Versionsnummer, die Auskunft über die Version der Software gibt.
- Datum der Dokumentveröffentlichung, das bei jeder Änderung des Dokuments ebenfalls aktualisiert wird.
- Datum des Software-Release, das angibt, wann diese Version der Software veröffentlicht wurde.

Unter der unten angegebenen Internetadresse können Sie überprüfen, ob neue Updates verfügbar sind, und sicherstellen, dass Sie mit der neuesten Version eines Dokuments arbeiten:

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

Für die Anmeldung an dieser Website benötigen Sie einen HP Passport. Hier können Sie sich für eine HP Passport-ID registrieren:

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

Alternativ können Sie auf den Link **New user registration** (Neue Benutzer registrieren) auf der HP Passport-Anmeldeseite klicken.

Wenn Sie sich beim Support-Service eines bestimmten Produkts registrieren, erhalten Sie ebenfalls aktualisierte Softwareversionen und überarbeitete Ausgaben der zugehörigen Dokumente. Weitere Informationen erhalten Sie bei Ihrem HP-Kundenbetreuer.

Support

Besuchen Sie die HP Software Support Online-Website von HP unter:

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

Auf dieser Website finden Sie Kontaktinformationen und Details zu Produkten, Services und Support-Leistungen von HP Software.

Der Online-Support von HP Software bietet Kunden mit Hilfe interaktiver technischer Support-Werkzeuge die Möglichkeit, ihre Probleme intern zu lösen. Als Valued Support Customer können Sie die Support-Website für folgende Aufgaben nutzen:

- Suchen nach interessanten Wissensdokumenten
- Absenden und Verfolgen von Support-Fällen und Erweiterungsanforderungen
- Herunterladen von Software-Patches
- Verwalten von Support-Verträgen
- Nachschlagen von HP-Support-Kontakten
- Einsehen von Informationen über verfügbare Services
- Führen von Diskussionen mit anderen Softwarekunden
- Suchen und Registrieren für Softwareschulungen

Für die meisten Support-Bereiche müssen Sie sich als Benutzer mit einem HP Passport registrieren und anmelden. In vielen Fällen ist zudem ein Support-Vertrag erforderlich. Hier können Sie sich für eine HP Passport-ID registrieren:

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

Weitere Informationen zu Zugriffsebenen finden Sie unter:

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

Haftungsausschluss für die PDF-Version der Online-Hilfe

Bei diesem Dokument handelt es sich um eine PDF-Version der Online-Hilfe. Diese PDF-Datei wird bereitgestellt, um Ihnen das Drucken mehrerer Themen der Hilfe oder das Lesen der Online-Hilfe im PDF-Format zu ermöglichen.

Hinweis: Bei einigen Themen werden die Formate nicht fehlerfrei in das PDF-Format konvertiert. Andere Teile der Online-Hilfe fehlen vollständig in der PDF-Datei. Diese fehlenden Themen können jedoch problemlos direkt aus der Online-Hilfe gedruckt werden.

Inhalt

Erstellen von Discovery- und Integrationsadaptern

Kapitel 1

Entwickeln und Schreiben von Adaptern

Dieses Kapitel umfasst folgende Themen:

Entwickeln und Schreiben von Adaptern – Übersicht

Bevor Sie mit der eigentlichen Planung für die Entwicklung neuer Adapter beginnen, ist es wichtig, dass Sie sich mit den Prozessen und Interaktionen vertraut machen, die im Allgemeinen mit der Entwicklung verbunden sind.

In den folgenden Abschnitten erfahren Sie alles Wissenswerte für die erfolgreiche Verwaltung und Ausführung eines Discovery-Entwicklungsprojekts.

Voraussetzungen und Themen:

- Es wird davon ausgegangen, dass Sie über praktische Kenntnisse der HP Universal CMDB verfügen und mit den Elementen des Systems grundlegend vertraut sind. Die folgenden Ausführungen sind nicht als detaillierter Leitfaden vorgesehen, sondern sollen Sie lediglich bei Ihrem Lernprozess unterstützen.
- Sie werden sich mit der Planung, Untersuchung und Implementierung neuer Discovery-Inhalte für HP Universal CMDB sowie mit den zu berücksichtigenden Richtlinien und Überlegungen beschäftigen.
- Sie erhalten Informationen zu wichtigen APIs des Datenflussverwaltungs-Frameworks. Die vollständige Dokumentation zu den verfügbaren APIs finden Sie in der *API-Referenz zur Datenflussverwaltung von HP Universal CMDB*. (Darüber hinaus gibt es noch andere, nicht formale APIs. Auch wenn diese bei Standardadaptern zum Einsatz kommen, bleiben Änderungen vorbehalten.)

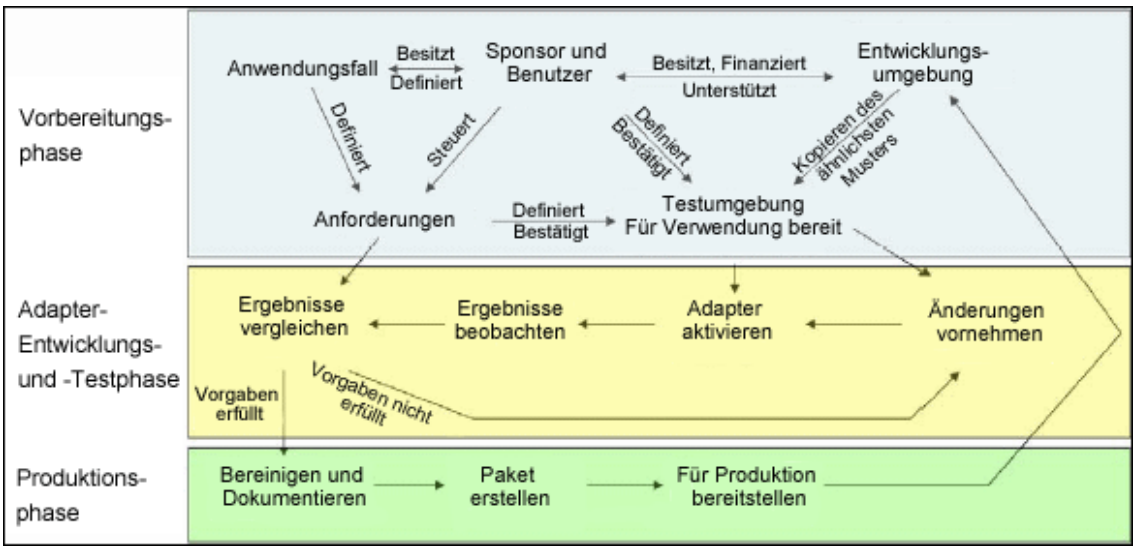
Erstellen von Inhalt

Dieser Abschnitt umfasst Folgendes:

- "Der Zyklus der Adapterentwicklung" oben
- "Datenflussverwaltung und Integration" auf Seite 18
- "Zuweisen eines Geschäftswerts zur Discovery-Entwicklung" auf Seite 19
- "Untersuchen der Integrationsanforderungen" auf Seite 20

Der Zyklus der Adapterentwicklung

Die folgende Abbildung zeigt ein Flussdiagramm für das Schreiben von Adaptern. Der mittlere Abschnitt, in dem es um das iterative Entwickeln und Testen geht, erfordert den größten Zeitaufwand.



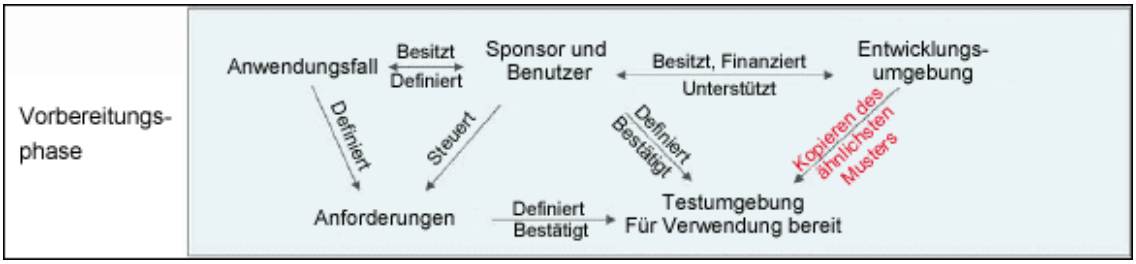
Die einzelnen Phasen der Adapterentwicklung bauen aufeinander auf.

Wenn Sie mit dem Aussehen und der Funktionsweise des Adapters zufrieden sind, können Sie ein Package erstellen. Verwenden Sie dazu entweder UCMDB Package Manager oder exportieren Sie die Komponenten manuell und erstellen Sie ein Package in Form einer ZIP-Datei. Als Best Practice wird empfohlen, das Package auf einem anderen UCMDB-System bereitzustellen und zu testen, bevor es für die Produktion freigegeben wird, um sicherzustellen, dass alle Komponenten vorhanden sind und erfolgreich verpackt wurden. Weitere Informationen zum Erstellen von Packages finden Sie unter "Package Manager" im *HP Universal CMDB – Verwaltungshandbuch*.

Die folgenden Abschnitte befassen sich mit den einzelnen Phasen der Adaptererstellung unter besonderer Berücksichtigung der wichtigsten Schritte und Best Practices:

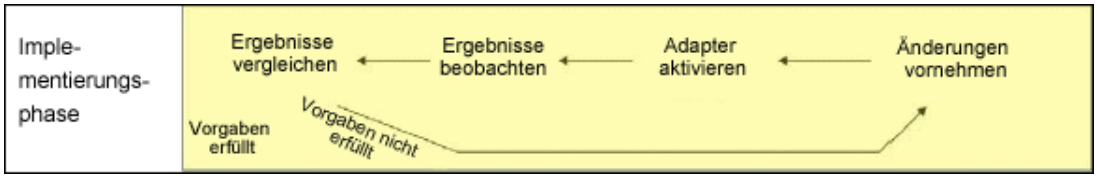
- "Untersuchungs- und Vorbereitungsphase" auf der nächsten Seite
- "Entwickeln und Testen des Adapters" auf der nächsten Seite
- "Verpacken und Produzieren des Adapters " auf Seite 17

Untersuchungs- und Vorbereitungsphase



Die Untersuchungs- und Vorbereitungsphase umfasst die ausschlaggebenden Geschäftsanforderungen und Anwendungsfälle. Darüber hinaus beschäftigt sie sich mit der Bereitstellung der erforderlichen Einrichtungen zum Entwickeln und Testen des Adapters.

1. Wenn Sie einen vorhandenen Adapter ändern möchten, besteht der erste technische Schritt darin, eine Sicherungskopie des Adapters zu erstellen, damit Sie den Adapter bei Bedarf in seinen ursprünglichen Zustand zurücksetzen können. Wenn Sie einen neuen Adapter erstellen möchten, kopieren Sie den Adapter, der die größte Ähnlichkeit mit dem neuen Adapter aufweist, und speichern Sie diesen unter einem neuen Namen. Weitere Informationen finden Sie unter "Ausschnitt "Ressourcen"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.
2. Überlegen Sie, wie der Adapter Daten erfassen soll:
   - Sollen externe Werkzeuge/Protokolle zum Abrufen der Daten verwendet werden?
   - Wie soll der Adapter auf Basis der Daten CIs erstellen?
   - Nun wissen Sie, wie ein ähnlicher Adapter aussehen muss.
3. Bestimmen Sie anhand der folgenden Kriterien, welcher Adapter die größte Ähnlichkeit aufweist:
   - Es werden die gleichen CIs erstellt.
   - Es werden die gleichen Protokolle verwendet (SNMP).
   - Die Ziele sind identisch (nach Betriebssystemtyp, -version usw.).
4. Kopieren Sie das gesamte Package.
5. Entpacken Sie das Package im Arbeitsbereich und benennen Sie die Adapterdateien (XML) und die Jython-Dateien (PY) um.



Entwickeln und Testen des Adapters

Die Entwicklungs- und Testphase ist ein hoch iterativer Prozess. Sobald der Adapter Form annimmt, beginnen Sie damit, ihn in Bezug auf die endgültigen Anwendungsfälle zu testen, nehmen Änderungen vor, testen erneut und wiederholen diesen Prozess, bis der Adapter den Anforderungen entspricht.

Starten und Vorbereiten der Kopie

- Ändern Sie die XML-Elemente des Adapters: Name (ID) in Zeile 1, erstellte CI-Typen und Name des aufgerufenen Jython-Skripts.
- Führen Sie die Kopie mit den gleichen Ergebnissen aus wie den Originaladapter.
- Kommentieren Sie den Großteil des Codes aus, insbesondere den kritischen Code, der das Ergebnis generiert.

Entwickeln und Testen

- Verwenden Sie einen anderen Beispielcode, um Änderungen zu entwickeln.
- Führen Sie den Adapter aus, um ihn zu testen.
- Verwenden Sie eine dedizierte Ansicht, um komplexe Ergebnisse zu prüfen. Prüfen Sie einfache Ergebnisse anhand einer Suche.

Verpacken und Produzieren des Adapters

Das Verpacken und Produzieren des Adapters stellt die letzte Phase der Entwicklung dar. Als Best Practice wird empfohlen, vor dem Verpacken in einem letzten Durchgang Debugging-Reste, Dokumente und Kommentare zu entfernen und Sicherheitsaspekte zu überprüfen. Sie sollten grundsätzlich mindestens ein Readme-Dokument erstellen, in dem der Aufbau des Adapters erläutert wird. Wenn jemand (vielleicht sogar Sie) zu einem späteren Zeitpunkt an diesem Adapter arbeiten muss, wird jede noch so allgemein gehaltene Dokumentation von großer Hilfe sein.

Bereinigen und Dokumentieren

- Entfernen Sie alle Debugging-Reste.
- Kommentieren Sie alle Funktionen und fügen Sie im Hauptabschnitt einige Eröffnungskommentare hinzu.
- Erstellen Sie eine Beispiel-TQL und eine Beispielansicht für den Benutzer zum Testen.

Erstellen des Package

- Exportieren Sie die Adapter, die TQL usw. mit Package Manager. Weitere Informationen finden Sie unter "Package Manager" im *HP Universal CMDB – Verwaltungshandbuch*.
- Prüfen Sie, welche Abhängigkeiten zwischen Ihrem Package und anderen Packages bestehen, z. B. ob die von anderen Packages erstellten CIs Eingabe-CIs Ihres Adapters sind.
- Erstellen Sie mit Package Manager ein Package in Form einer ZIP-Datei. Weitere Informationen finden Sie unter "Package Manager" im *HP Universal CMDB – Verwaltungshandbuch*.
- Testen Sie die Bereitstellung, indem Sie Teile des neuen Inhalts entfernen und erneut bereitstellen oder auf einem anderen Testsystem testen.

Datenflussverwaltung und Integration

Datenflussverwaltungsadapter lassen sich mit anderen Produkten integrieren. Es gelten die folgenden Definitionen:

- Die Datenflussverwaltung erfasst bestimmte Inhalte von mehreren Zielen.
- Die Integration erfasst mehrere Inhaltstypen von einem System.

Bei diesen Definitionen wird nicht zwischen den Erfassungsmethoden unterschieden. Die Datenflussverwaltung tut dies auch nicht. Der Prozess der Entwicklung eines neues Adapters ist identisch mit dem Prozess der Entwicklung einer neuen Integration. Sie führen die gleichen Untersuchungen durch, treffen die gleichen Entscheidungen in Bezug auf neue und vorhandene Adapter, führen die gleichen Schreibvorgänge durch usw. Es gibt lediglich ein paar Unterschiede:

- Die zeitliche Planung des endgültigen Adapters. Integrationsadapter werden möglicherweise häufiger ausgeführt als Discovery-Adapter, aber dies ist von den Anwendungsfällen abhängig.
- Eingabe-CIs:
  - Integration: Nicht-CI-Trigger werden ohne Eingabe ausgeführt: Ein Dateiname oder eine Quelle wird durch den Adapterparameter übergeben.
  - Discovery: Verwendet reguläre CMDB-CIs für die Eingabe.

Für Integrationsprojekte sollte in der Regel ein vorhandener Adapter verwendet werden. Die Integrationsrichtung (von HP Universal CMDB in ein anderes Produkt oder von einem anderen Produkt in HP Universal CMDB) kann Auswirkungen auf Ihren Entwicklungsansatz haben. Es werden Feld-Packages angeboten, die Sie unter Verwendung bewährter Verfahren für Ihre eigenen Anwendungen kopieren können.

Von HP Universal CMDB in ein anderes Projekt:

- Erstellen Sie eine TQL, die die zu exportierenden CIs und Beziehungen generiert.
- Verwenden Sie einen allgemeinen Wrapper-Adapter, um die TQL auszuführen, und schreiben Sie die Ergebnisse in eine XML-Datei, damit sie vom externen Produkt gelesen werden können.

> **Hinweis:** Beispiele für Feld-Packages erhalten Sie beim HP Software Support.

Für die Integration eines anderen Produkts in HP Universal CMDB ist die Funktionsweise des Integrationsadapters von der Art und Weise der Datenanzeige des anderen Produkts abhängig:

Integrationstyp	Wiederzuverwendendes Referenzbeispiel
Direkter Zugriff auf die Produktdatenbank	HP ED
Lesen in einer durch einen Export generierten CSV- oder XML-Datei	HP ServiceCenter
Zugriff auf die API eines Produkts	BMC Atrium/Remedy

Zuweisen eines Geschäftswerts zur Discovery-Entwicklung

Der Anwendungsfall für die Entwicklung neuer Discovery-Inhalte sollte auf einem Business Case und einem Plan zur Generierung eines Geschäftswerts beruhen. Das heißt, die Zielsetzung bei der Zuordnung von Systemkomponenten zu CIs und ihrem Hinzufügen zur CMDB besteht in der Realisierung eines Geschäftswerts.

Der Inhalt kann nicht immer für die Applikationszuordnung verwendet verwenden, obwohl dies ein üblicher Zwischenschritt für viele Anwendungsfälle ist. Ungeachtet der endgültigen Verwendung des Inhalts sollte der Plan Antworten auf folgenden Fragen enthalten:

- Wer ist der Benutzer? Wie soll der Benutzer die von den CIs (und den zwischen ihnen bestehenden Beziehungen) bereitgestellten Informationen verwenden? In welchem Geschäftskontext sollen die CIs und Beziehungen angezeigt werden? Werden die CIs von einer Person und/oder einem Produkt verwendet?
- Wenn die perfekte Kombination aus CIs und Beziehungen in der CMDB gespeichert ist: Wie plane ich ihre Verwendung, um einen Geschäftswert zu generieren?
- Wie soll die perfekte Zuordnung aussehen?
  - Mit welchem Schlagwort lassen sich die Beziehungen zwischen den einzelnen CIs am besten beschreiben?
  - Welche CI-Typen müssen auf jeden Fall berücksichtigt werden?
  - Für welchen Verwendungszweck ist die Karte vorgesehen und wer ist der Endbenutzer?
- Wie sieht das perfekte Report-Layout aus?

Nachdem Sie die geschäftliche Rechtfertigung festgelegt haben, besteht der nächste Schritt darin, den Geschäftswert in einem Dokument zu konkretisieren. Dies bedeutet, dass Sie die perfekte Karte mit einem Zeichenwerkzeug bildlich darstellen und sich entsprechend den Anwendungsfällen mit den Auswirkungen und Abhängigkeiten von CIs und Reports, mit der Art der Änderungsverfolgung sowie mit wichtigen Änderungen, Überwachungs- und Compliance-Fragen und zusätzlichen Geschäftswerten vertraut machen müssen.

Diese Zeichnung (bzw. dieses Modell) wird als **Blaupause** bezeichnet.

Wenn die Applikation beispielsweise wissen muss, wann eine bestimmte Konfigurationsdatei geändert wurde, sollte die Datei in der gezeichneten Karte dem entsprechenden CI (auf das sie sich bezieht) zugeordnet und mit diesem CI verknüpft werden.

Arbeiten Sie mit einem Fachexperten zusammen, der als Endbenutzer mit dem entwickelten Inhalt arbeiten wird. Dieser Experte sollte auf die kritischen Entitäten (CIs mit Attributen und Beziehungen) hinweisen, die in der CMDB vorhanden sein müssen, um einen Geschäftswert zu generieren.

Eine Möglichkeit besteht darin, dem Besitzer der Applikation (in diesem Fall auch dem Fachexperten) einen Fragebogen an die Hand zu geben. Der Besitzer sollte in der Lage sein, die oben aufgeführten Ziele und die Blaupause zu spezifizieren. Auf jeden Fall muss der Besitzer eine aktuelle Architektur der Applikation zur Verfügung stellen.

Sie sollten nur wichtige Daten zuordnen und auf unnötige Daten verzichten: Sie können den Adapter später jederzeit ergänzen. Das Ziel sollte darin bestehen, einen eingeschränkten Discovery-Adapter einzurichten, der funktionsfähig ist und einen Wert liefert. Die Zuordnung großer Datenmengen resultiert zwar in eindrucksvolleren Karten, kann aber im Hinblick auf die Entwicklung verwirrend und zu zeitaufwendig sein.

Sobald das Modell und der Geschäftswert feststehen, können Sie mit der nächsten Phase fortfahren. Sie können später noch einmal zu dieser Phase zurückkehren, nachdem Sie die nächsten Phasen durchgeführt und dabei konkrete Informationen erhalten haben.

Untersuchen der Integrationsanforderungen

Voraussetzung für diese Phase ist eine **Blaupause** der von der Datenflussverwaltung zu ermittelnden CIs und Beziehungen, die die Discovery-Attribute enthalten sollen. Weitere Informationen finden Sie unter "Entwickeln und Schreiben von Adaptern – Übersicht" auf Seite 14.

Dieser Abschnitt umfasst die folgenden Themen:

Ändern eines vorhandenen Adapters

Sie ändern einen vorhandenen Adapter, wenn zwar ein Standard- oder ein Feldadapter existiert, dieser jedoch folgende Defizite aufweist:

- Bestimmte erforderliche Attribute werden nicht erkannt.
- Ein bestimmter Zieltyp (Betriebssystem) wird nicht oder nicht richtig erkannt.
- Eine bestimmte Beziehung wird nicht erkannt oder nicht erstellt.

Wenn ein vorhandener Adapter die Anforderungen nur zum Teil erfüllt, sollte Ihr erster Ansatz darin bestehen, die vorhandenen Adapter zu evaluieren, um festzustellen, ob einer von ihnen annähernd für die vorgesehene Aufgabe geeignet ist. Wenn ja, können Sie den vorhandenen Adapter ändern.

Zudem sollten Sie evaluieren, ob ein vorhandener Feldadapter verfügbar ist. Feldadapter sind Discovery-Adapter, die zwar zur Verfügung stehen, aber keine Standardadapter sind. Eine aktuelle Liste der Feldadapter erhalten Sie beim HP Software Support.

Schreiben eines neuen Adapters

In folgenden Situationen muss ein neuer Adapter entwickelt werden:

- Wenn es schneller geht, einen Adapter zu schreiben als die Informationen manuell in die CMDB einzufügen (im Allgemeinen von ca. 50 bis 100 CIs und Beziehungen) oder der Zeitaufwand sehr hoch ist.
- Wenn die Notwendigkeit den Aufwand rechtfertigt.

- Wenn kein Standard- oder Feldadapter verfügbar ist.
- Wenn die Ergebnisse wiederverwendbar sind.
- Wenn die Zielumgebung oder ihre Daten verfügbar sind (Sie können eine Discovery nur für die Elemente durchführen, die Sie sehen).

Modelluntersuchung

- Durchsuchen Sie das UCMDB-Klassenmodell (CIT Manager) und ordnen Sie die Entitäten und Beziehungen von Ihrer **Blaupause** den vorhandenen CITs zu. Es wird dringend empfohlen, dass Sie sich an das aktuelle Modell halten, um mögliche Komplikationen während eines Versions-Upgrades zu vermeiden. Wenn Sie das Modell erweitern müssen, sollten Sie neue CITs erstellen, da die Standard-CITs durch ein Upgrade überschrieben werden können.
- Wenn beim aktuellen Modell einige Entitäten, Beziehungen oder Attribute fehlen, müssen Sie sie erstellen. Vorzugsweise sollte ein Package mit diesen CITs erstellt werden (in dem später auch die gesamte Discovery, alle Ansichten und sonstigen Elemente, die sich auf dieses Package beziehen, gespeichert werden), da Sie diese CITs bei jeder Installation von HP Universal CMDB bereitstellen müssen.

Technologieuntersuchung

Nachdem Sie sich vergewissert haben, dass alle relevanten CIs in der CMDB gespeichert sind, müssen Sie als Nächstes entscheiden, wie diese Daten von den jeweiligen Systemen abgerufen werden sollen.

Das Abrufen der Daten erfordert in der Regel die Verwendung eines Protokolls, um auf einen Verwaltungsbereich der Applikation, die eigentlichen Daten der Applikation oder die zu der Applikation gehörenden Konfigurationsdateien oder Datenbanken zuzugreifen. Jede Datenquelle, die Informationen zu einem System liefern kann, ist wertvoll. Die Technologieuntersuchung erfordert sowohl umfassende Kenntnisse des betreffenden Systems als auch ein gewisses Maß an Kreativität.

Für selbst entwickelte Applikationen kann es hilfreich sein, dem Besitzer der Applikation einen Fragebogen an die Hand zu geben. In diesem Fragebogen sollte der Besitzer alle Bereiche der Applikation aufführen, die Informationen für die Blaupause und die Geschäftswerte liefern können. Diese Informationen sollten unter anderem Angaben zu Managementdatenbanken, Konfigurationsdateien, Protokolldateien, Managementschnittstellen, Verwaltungsprogrammen, Webdiensten sowie gesendeten Nachrichten oder Ereignissen enthalten.

Für Standardprodukte sollten Sie sich auf die Dokumentation, auf Foren oder auf den Produkt-Support konzentrieren. Suchen Sie nach Administratorhandbüchern, Plugin- und Integrationshandbüchern, Managementhandbüchern usw. Wenn immer noch Daten von den Verwaltungsschnittstellen fehlen, informieren Sie sich über die Konfigurationsdateien der Applikation sowie über Registrierungseinträge, Protokolldateien, NT-Ereignisprotokolle und sonstige Elemente, die den ordnungsgemäßen Betrieb der Applikation steuern.

Richtlinien für die Auswahl von Datenzugriffsmöglichkeiten

Relevanz: Wählen Sie die Quellen oder eine Kombination von Quellen aus, die die meisten Daten liefern bzw. liefert. Wenn eine einzelne Quelle die meisten Informationen liefert und die übrigen Informationen auf mehrere Quellen verteilt oder nur schwer zugänglich sind, versuchen Sie den Wert der übrigen Informationen zu ermitteln, indem Sie ihren Nutzen dem mit ihrem Abruf verbundenen Aufwand bzw. Risiko gegenüberstellen. In einigen Fällen werden Sie sich

möglicherweise für eine Reduzierung der Blaupause entscheiden, wenn der Wert oder die Kosten den notwendigen Aufwand nicht rechtfertigen.

Wiederverwendung: Ein guter Grund für die Wiederverwendung ist die Tatsache, dass HP Universal CMDB bereits ein bestimmtes Verbindungsprotokoll unterstützt. Dies bedeutet, dass das Framework der Datenflussverwaltung in der Lage ist, einen einsatzbereiten Client und eine gebrauchsfertige Konfiguration für die Verbindung bereitzustellen. Andernfalls müssen Sie möglicherweise in die Infrastrukturentwicklung investieren. So können Sie die gegenwärtig unterstützten HP Universal CMDB-Verbindungsprotokolle anzeigen: **Datenflussverwaltung > Data Flow Probe einrichten > Ausschnitt "Domänen und Proben"**. Weitere Informationen finden Sie unter "Ausschnitt "Domänen und Proben"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Durch Hinzufügen neuer CIs zum Modell können Sie neue Protokolle hinzufügen. Weitere Informationen erhalten Sie beim HP Software Support.

> **Hinweis:** Für den Zugriff auf die Windows-Registrierungseinträge können Sie WMI oder NTCMD verwenden.

Sicherheit: Der Zugriff auf Informationen erfordert in der Regel die Eingabe von Anmeldeinformationen (Benutzername, Kennwort) in die CMDB. Die Anmeldeinformationen sind im Produkt vor unautorisierter Nutzung geschützt. Wenn das Hinzufügen von Sicherheit nicht mit anderen von Ihnen vorgenommenen Einstellungen in Widerspruch steht, sollten Sie nach Möglichkeit die am wenigsten sensitiven Anmeldeinformationen verwenden oder sich für ein Protokoll entscheiden, durch das die Zugriffsanforderungen noch erfüllt werden. Wenn die Informationen beispielsweise sowohl über JMX (Standardverwaltungsschnittstelle, eingeschränkt) als auch über Telnet verfügbar sind, empfiehlt sich die Verwendung von JMX, da es inhärent einen eingeschränkten Zugriff gewährt und (normalerweise) keinen Zugriff auf die zugrunde liegende Plattform gestattet.

Komfort: Einige Verwaltungsschnittstellen enthalten möglicherweise erweiterte Funktionen. So kann es beispielsweise einfacher sein, Abfragen (SQL, WMI) auszuführen als in Informationsstrukturen zu navigieren oder reguläre Ausdrücke für die Analyse zu erstellen.

Entwicklerzielgruppe: Die Personen, die schließlich die Adapter entwickeln, bevorzugen möglicherweise eine bestimmte Technologie. Dieser Aspekt kann auch berücksichtigt werden, wenn zwei Technologien nahezu identische Informationen zu gleichen Kosten bereitstellen.

Übersicht

Das Ergebnis dieser Phase ist ein Dokument, in dem die Zugriffsmethoden und die relevanten Informationen beschrieben werden, die aus jeder Methode extrahiert werden können. Das Dokument sollte außerdem eine Zuordnung von jeder Quelle zu den entsprechenden Daten der Blaupause enthalten.

Jede Zugriffsmethode sollte nach den obigen Anweisungen markiert werden. Sie sollten jetzt über einen Plan verfügen, aus dem ersichtlich ist, welche Quellen in die Discovery einbezogen und welche Informationen von jeder Quelle in das Blaupausenmodell extrahiert werden sollen (dieses sollte inzwischen auch schon dem entsprechenden UCMDB-Modell zugeordnet worden sein).

Entwickeln von Integrationsinhalt

Bevor Sie eine neue Integration erstellen, müssen Sie die Anforderungen kennen, die an die Integration gestellt werden:

- Soll die Integration Daten in die CMDB kopieren? Sollen die Daten von der Historie verfolgt werden? Ist die Quelle unzuverlässig?

  Auffüllung erforderlich.

- Soll die Integration Daten für Ansichten und TQL-Abfragen nach dem On-the-Fly-Prinzip föderieren? Ist die Genauigkeit von Datenänderungen ein wichtiger Faktor? Ist die Datenmenge zu groß, um sie in die CMDB zu kopieren, während die angeforderte Datenmenge in der Regel klein ist?

  Förderation erforderlich.

- Soll die Integration Daten per Push an Remote-Datenquellen übertragen?

  Datenpush erforderlich.

> **Hinweis:** Um ein Maximum an Flexibilität zu erreichen, können für eine Integration sowohl Föderations- als auch Auffüllungsflüsse konfiguriert werden.

Weitere Informationen zu den unterschiedlichen Integrationstypen finden Sie unter "Integration Studio" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Für die Erstellung von Integrationsadaptern stehen vier verschiedene Optionen zur Verfügung:

1. Jython-Adapter
   - Klassisches Discovery-Pattern
   - In Jython geschrieben
   - Wird zur Auffüllung verwendet

   Weitere Informationen finden Sie unter "Entwickeln von Jython-Adaptern" auf Seite 41.
2. Java-Adapter
   - Ein Adapter, der eine der Adapterschnittstellen in Federation Framework SDK implementiert.
   - Kann für Föderationen, Auffüllungen oder Datenpushes verwendet werden (je nach erforderlicher Implementierung).
   - Vollständig in Java geschrieben, sodass es möglich ist, Code zu erstellen, der mit jeder beliebigen Quelle oder jedem beliebigen Ziel verbunden werden kann.
   - Geeignet für Jobs, wenn jeder einzelne Job mit einer einzelnen Datenquelle oder einem einzelnen Datenziel verbunden wird.

   Weitere Informationen finden Sie unter "Entwickeln von Java-Adaptern" auf Seite 159.
3. Allgemeiner DB-Adapter

- Ein auf dem Java-Adapter basierender abstrakter Adapter, der Federation Framework SDK verwendet.
- Ermöglicht die Erstellung von Adaptern, die eine Verbindung zu externen Daten-Repositorys herstellen.
- Unterstützt sowohl Föderation als auch Auffüllung (mit einem für die Änderungsunterstützung implementierten Java-Plugin).
- Relativ einfach zu definieren, da er hauptsächlich auf XML- und Property-Konfigurationsdateien beruht.
- Die Hauptkonfiguration basiert auf einer Datei vom Typ **orm.xml**, die Zuordnungen zwischen UCMDB-Klassen und Datenbankspalten vornimmt.
- Geeignet für Jobs, wenn jeder einzelne Job mit einer einzelnen Datenquelle verbunden wird.

Weitere Informationen finden Sie unter "Entwickeln von allgemeinen Datenbankadaptern" auf Seite 86.

4. Generischer Push-Adapter
   - Ein auf dem Java-Adapter (Federation Framework SDK) und dem Jython-Adapter basierender abstrakter Adapter.
   - Ermöglicht die Erstellung von Adaptern, die Daten per Push an Remote-Ziele übertragen.
   - Relativ einfach zu definieren, da Sie lediglich die Zuordnung zwischen den UCMDB-Klassen und XML festlegen und ein Jython-Skript schreiben müssen, das den Datenpush zum Ziel durchführt.
   - Geeignet für Jobs, wenn jeder einzelne Job mit einem einzelnen Datenziel verbunden wird.
   - Wird für Datenpushes verwendet.

   Weitere Informationen finden Sie unter "Entwickeln von Push-Adaptern" auf Seite 190.

In der folgenden Tabelle sind die Funktionen der einzelnen Adapter aufgeführt:

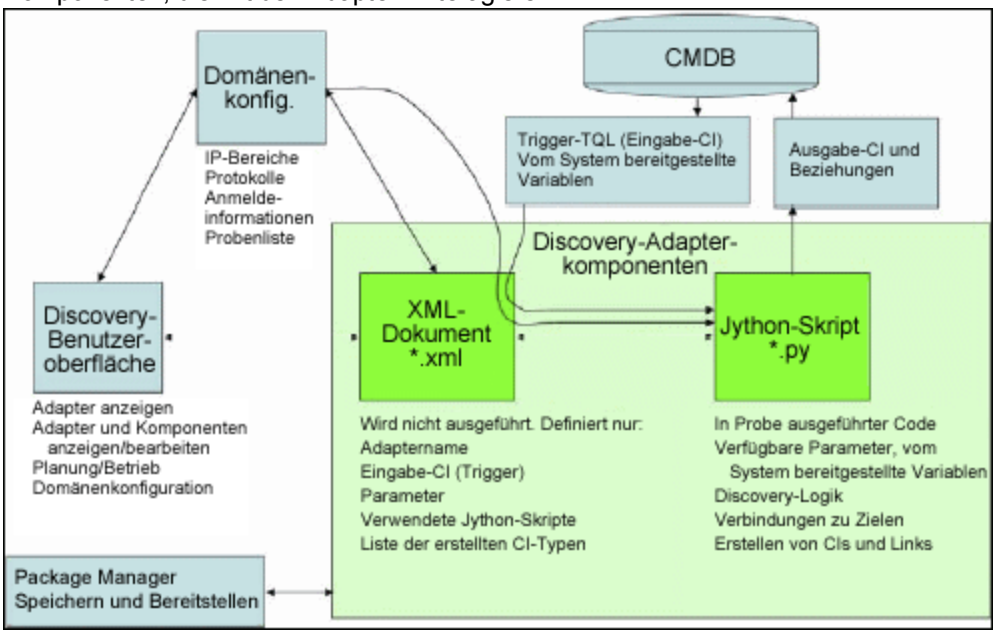
Fluss/Adapter	Jython-Adapter	Java-Adapter	GDB-Adapter	Push-Adapter
Auffüllung	X	X	X	
Föderation		X	X	
Datenpush		X		X

Entwickeln von Discovery-Inhalt

Dieser Abschnitt umfasst Folgendes:

Discovery-Adapter und zugehörige Komponenten

Die folgende Grafik zeigt die Komponenten eines Adapters zusammen mit den Komponenten, mit denen sie interagieren, um die Discovery-Funktion auszuführen. Die grün dargestellten Komponenten sind die eigentlichen Adapter und die blau dargestellten Komponenten sind die Komponenten, die mit den Adaptern interagieren.



Für einen Adapter sind mindestens zwei Dateien erforderlich: ein XML-Dokument und ein Jython-Skript. Discovery Framework, einschließlich der Eingabe-CIs, der Anmeldeinformationen und der vom Benutzer bereitgestellten Bibliotheken, wird zum Zeitpunkt der Ausführung für den Adapter verfügbar gemacht. Beide Komponenten des Discovery-Adapters werden über die Datenflussverwaltung verwaltet. Sie werden in der CMDB betriebsbereit gespeichert. Das externe Package bleibt zwar bestehen, aber für die Ausführung wird nicht darauf verwiesen. Package Manager ermöglicht die Beibehaltung der neuen Funktion für Discovery- und Integrationsinhalte.

Die Eingabe-CIs für den Adapter werden von einer TQL geliefert und in vom System bereitgestellten Variablen für den Adapter verfügbar gemacht. Die Adapterparameter werden ebenfalls als Zieldaten bereitgestellt, sodass Sie die Ausführung des Adapters entsprechend seiner spezifischen Funktion konfigurieren können.

Zum Erstellen und Testen neuer Adapter benötigen Sie die Datenflussverwaltungsapplikation. Während des Schreibvorgangs verwenden Sie die Seiten **Discovery-Systemsteuerung**, **Adapterverwaltung** und **Data Flow Probe einrichten**.

Adapter werden als Packages gespeichert und transportiert. Die Package Manager-Applikation und die JMX-Konsole werden zum Erstellen von Packages aus neu entwickelten Adaptern sowie zum Bereistellen der Adapter auf neuen Systemen verwendet.

Trennen von Adaptern

In technischer Hinsicht könnte eine gesamte Discovery als ein einzelner Adapter definiert werden. Ein gutes Design erfordert jedoch die Aufteilung eines komplexen Systems in einfachere

Komponenten, die besser zu verwalten sind.

Für die Aufteilung des Adapterprozesses gelten die folgenden Richtlinien und Best Practices:

- Die Discovery sollte in mehreren Stufen erfolgen. Dabei sollte jede Stufe von einem Adapter repräsentiert werden, der einen Bereich oder eine Schicht des Systems zuordnet. Damit die Discovery des Systems fortgesetzt werden kann, sollten die Adapter jeweils auf der vorherigen Stufe oder Schicht basieren. Beispiel: Adapter A wird durch das TQL-Ergebnis eines Applikationsservers ausgelöst und ordnet die Schicht des Applikationsservers zu. Im Rahmen dieser Zuordnung wird eine Komponente der JDBC-Verbindung zugeordnet. Adapter B registriert die Komponente der JDBC-Verbindung als Trigger-TQL. Er verwendet die Ergebnisse von Adapter A für den Zugriff auf die Datenbankschicht (z. B. durch das JDBC-URL-Attribut) und ordnet die Datenbankschicht zu.
- **Das Paradigma der zweistufigen Verbindung**: Die meisten Systeme erfordern die Eingabe von Anmeldeinformationen, bevor auf die Daten zugegriffen werden kann. Das heißt, es muss eine Kombination aus Benutzername und Kennwort bei diesen Systemen eingegeben werden. Der Administrator der Datenflussverwaltung stellt die Anmeldeinformationen in sicherer Form für das System bereit. Dabei kann er mehrere priorisierte Anmeldeinformationen angeben. Dies wird als **Protokoll-Dictionary** bezeichnet. Wenn das System aus irgendeinem Grund nicht zugänglich ist, ist es nicht sinnvoll, eine weitere Discovery durchzuführen. Wenn die Verbindung erfolgreich hergestellt wurde, muss es im Hinblick auf den künftigen Discovery-Zugriff eine Möglichkeit geben, festzustellen, welche Anmeldeinformationen verwendet wurden.

  Die beiden Stufen führen in folgenden Fällen zu einer Trennung der zwei Adapter:

  - **Verbindungsadapter**: Dieser Adapter akzeptiert einen ersten Trigger und prüft, ob bei diesem ein Remote-Agent vorhanden ist. Dazu probiert der Adapter alle Einträge im Protokoll-Dictionary aus, die mit dem jeweiligen Agententyp übereinstimmen. Sobald ein passender Eintrag gefunden wurde, liefert der Adapter als Ergebnis ein Remote-Agent-CI (SNMP, WMI usw.), das für künftige Verbindungen auf den korrekten Eintrag im Protokoll-Dictionary verweist. Dieses Agent-CI ist dann Bestandteil eines Triggers für den Inhaltsadapter.
  - **Inhaltsadapter**: Vorbedingung für diesen Adapter ist die erfolgreiche Verbindung des vorherigen Adapters (von den TQLs spezifizierte Vorbedingungen). Diese Adaptertypen müssen nicht mehr das gesamte Protokoll-Dictionary durchsuchen, da sie die korrekten Anmeldeinformationen vom Remote-Agent-CI erhalten und diese für die Anmeldung beim Discovery-System verwenden.
- Unterschiedliche Zeitplanungen können ebenfalls Einfluss auf die Discovery-Unterteilung haben. Beispielsweise kann es erforderlich sein, dass ein System nur außerhalb der Geschäftszeiten abgefragt wird. Auch wenn es sinnvoll wäre, den Adapter mit einem identischen Adapter zu verknüpfen, der zur Discovery eines anderen Systems eingesetzt wird, erfordern die unterschiedlichen Zeitpläne die Erstellung von zwei Adaptern.
- Für die Discovery verschiedener Verwaltungsschnittstellen oder Technologien innerhalb eines Systems sollten separate Adapter eingesetzt werden. Auf diese Weise können Sie die für jedes System oder jedes Unternehmen geeignete Zugriffsmethode aktivieren. Einige Unternehmen haben beispielsweise per WMI Zugriff auf ihre Computer, aber es sind keine SNMP-Agenten auf den Computern installiert.

Implementieren eines Discovery-Adapters

Das Ziel einer Datenflussverwaltungsaufgabe besteht darin, auf entfernte (oder lokale) Systeme zuzugreifen, extrahierte Daten als CIs zu modellieren und die CIs in der CMDB zu speichern. Die Aufgabe besteht aus folgenden Schritten:

1. **Erstellen eines Adapters.**

   Sie konfigurieren eine Adapterdatei, in der der Kontext, die Parameter und die Ergebnistypen gespeichert sind, indem Sie die Skripts auswählen, die Teil des Adapters sein sollen. Weitere Informationen finden Sie unter "Schritt 1: Erstellen eines Adapters" auf Seite 30.

2. **Erstellen eines Discovery-Jobs.**

   Sie konfigurieren einen Job mit Planungsinformationen und eine Trigger-Abfrage. Weitere Informationen finden Sie unter "Schritt 2: Zuweisen eines Jobs zum Adapter" auf Seite 37.

3. **Bearbeiten des Discovery-Codes.**

   Sie können den in den Adapterdateien enthaltenen Jython- oder Java-Code, der sich auf das Framework der Datenflussverwaltung bezieht, bearbeiten. Weitere Informationen finden Sie unter "Schritt 3: Erstellen von Jython-Code" auf Seite 38.

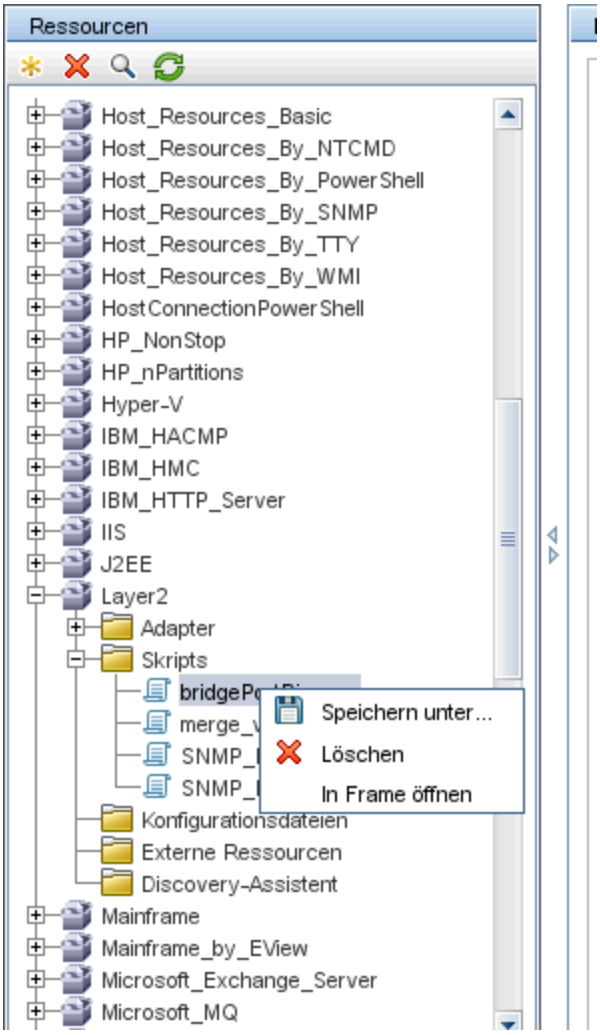
Um neue Adapter zu schreiben, müssen Sie alle oben aufgeführten Komponenten erstellen. Jede Komponente wird dann automatisch an die Komponente im vorherigen Schritt gebunden. Nachdem Sie zum Beispiel einen Job erstellt und den entsprechenden Adapter ausgewählt haben, wird die Adapterdatei an den Job gebunden.

Adaptercode

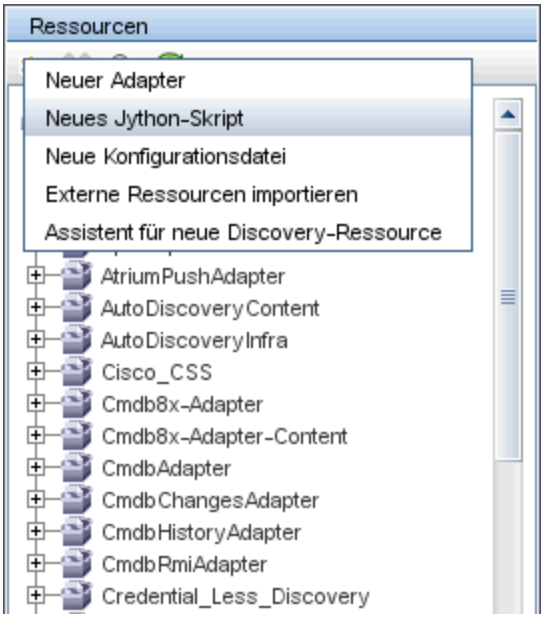
Die eigentliche Implementierung, d. h. das Herstellen einer Verbindung zum Remote-System, das Abfragen seiner Daten und deren Zuordnung als CMDB-Daten, wird mittels Jython-Code durchgeführt. Der Code enthält beispielsweise die Logik zum Herstellen einer Verbindung zu einer Datenbank und zum Extrahieren von Daten aus dieser Datenbank. In diesem Fall wartet der Code auf die Eingabe eines JDBC-URL, eines Benutzernamens, eines Kennworts, eines Ports usw. Diese Parameter sind spezifisch für jede Instanz der Datenbank, die die TQL-Abfrage beantwortet. Sie definieren diese Variablen im Adapter (in den Trigger-CI-Daten). Wenn der Job ausgeführt wird, werden diese spezifischen Details für die Ausführung an den Code übergeben.

Der Adapter kann über einen Java-Klassennamen oder einen Jython-Skriptnamen auf diesen Code verweisen. Im Folgenden werden wir uns mit dem Schreiben von Datenflussverwaltungscode in Form von Jython-Skripts befassen.

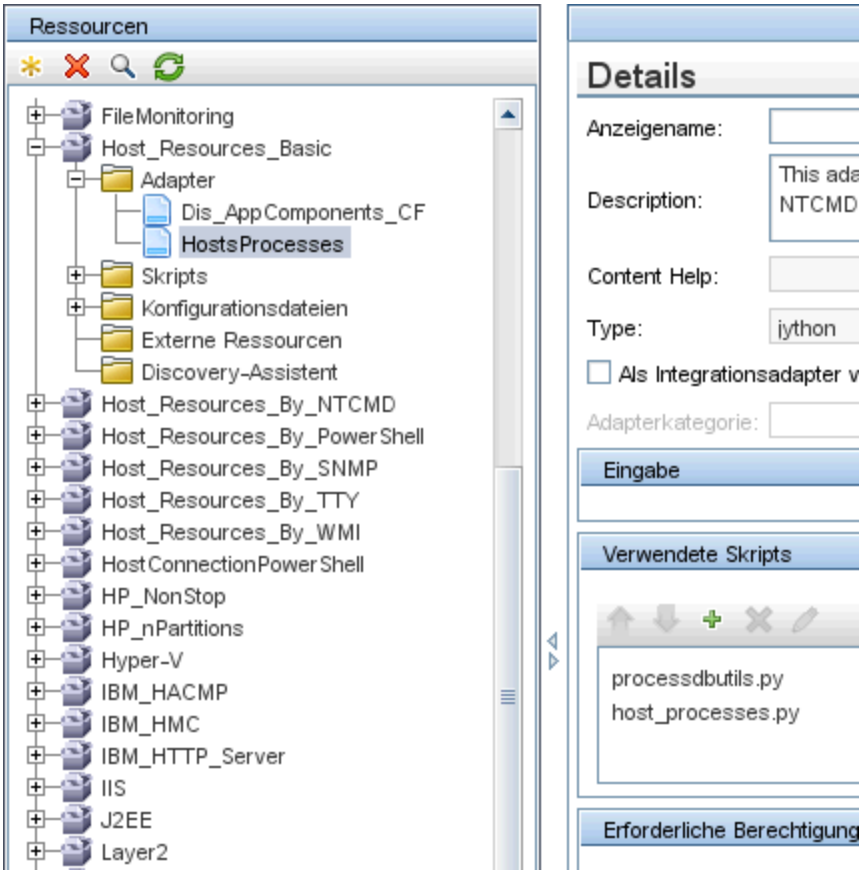
Ein Adapter kann eine Liste von Skripts enthalten, die bei der Discovery-Ausführung verwendet werden sollen. Bei der Erstellung eines neuen Adapters erstellen Sie in der Regel ein neues Skript, das Sie dann dem Adapter zuweisen. Ein neues Skript enthält Basisvorlagen. Sie können jedoch auch ein anderes Skript als Vorlage verwenden, indem Sie mit der rechten Maustaste auf das Skript klicken und die Option **Speichern unter** auswählen:



Weitere Informationen zum Schreiben neuer Jython-Skripts finden Sie unter "Schritt 3: Erstellen von Jython-Code" auf Seite 38. Sie fügen Skripts über den Ausschnitt **Ressourcen** hinzu:



Die Skripts werden nacheinander in der Reihenfolge, in der sie im Adapter definiert sind, ausgeführt:



Hinweis: Ein Skript muss auch dann angegeben werden, wenn es allein als Bibliothek von

einem anderen Skript verwendet wird. In diesem Fall muss das Bibliotheksskript vor dem Skript definiert werden, von dem es verwendet wird. In diesem Beispiel wird das Skript `processdbutils.py` als Bibliothek vom letzten `host_processes.py`-Skript verwendet. Bibliotheken unterscheiden sich insofern von normalen ausführbaren Skripts, als sie nicht über die Funktion `DiscoveryMain()` verfügen.

Schritt 1: Erstellen eines Adapters

Ein Adapter kann als Definition einer Funktion betrachtet werden. Diese Funktion definiert eine Eingabedefinition, führt die Logik bei der Eingabe aus, definiert die Ausgabe und liefert schließlich ein Ergebnis.

Jeder Adapter spezifiziert eine Eingabe und eine Ausgabe: Sowohl die Eingabe als auch die Ausgabe sind Trigger-CIs, die speziell im Adapter definiert werden. Der Adapter extrahiert Daten vom Eingabe-Trigger-CI und übergibt diese als Parameter an den Code. (Daten von zugehörigen CIs werden manchmal ebenfalls an den Code übergeben. Weitere Informationen hierzu finden Sie unter "Fenster "Zugehörige CIs"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*. Abgesehen von den spezifischen Parametern des Eingabe-Trigger-CI, die an den Code übergeben werden, ist der Code eines Adapters generisch.

Weitere Informationen zu Eingabekomponenten finden Sie unter "Universal Discovery - Konzepte" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Dieser Abschnitt umfasst die folgenden Themen:

1. ## Definieren der Adaptereingabe (Trigger-CIT und Eingabeabfrage)

   Mithilfe der Trigger-CIT- und Eingabeabfragekomponenten definieren Sie bestimmte CIs als Adaptereingabe:

   - Der Trigger-CIT definiert, welcher CIT als Eingabe für den Adapter verwendet wird. Für einen Adapter, der IPs ermitteln soll, lautet der Eingabe-CIT beispielsweise **Network**.
   - Bei der Eingabeabfrage handelt es sich um eine reguläre, bearbeitbare Abfrage, die die Abfrage bei der CMDB definiert. Die Eingabeabfrage definiert zusätzliche Einschränkungen des CIT (zum Beispiel, ob für die Aufgabe ein Attribut vom Typ `hostID` oder `application_ip` erforderlich ist). Bei Bedarf können weitere CI-Daten definiert werden.

     Wenn der Adapter zusätzliche Informationen von den zum Trigger-CI gehörenden CIs benötigt, können Sie weitere Knoten zur Eingabe-TQL hinzufügen. Weitere Informationen finden Sie im nachfolgenden Abschnitt "Beispiel für eine Eingabeabfragedefinition" auf der nächsten Seite sowie unter "Hinzufügen von Abfrageknoten und Beziehungen zu einer TQL-Abfrage" im *HP Universal CMDB – Modellierungshandbuch*.

- Die Trigger-CI-Daten enthalten alle erforderlichen Informationen zum Trigger-CI sowie Informationen von den anderen Knoten im Eingabe-TQL, sofern diese definiert wurden. Die Datenflussverwaltung verwendet Variablen, um Daten von den CIs abzurufen. Wenn die Aufgabe zur Probe heruntergeladen wird, werden die Variablen der Trigger-CI-Daten durch die tatsächlichen Werte der Attribute für reale CI-Instanzen ersetzt.

Beispiel für eine Trigger-CIT-Definition:

In diesem Beispiel definiert ein Trigger-CIT, dass IP-CIs im Adapter zulässig sind.

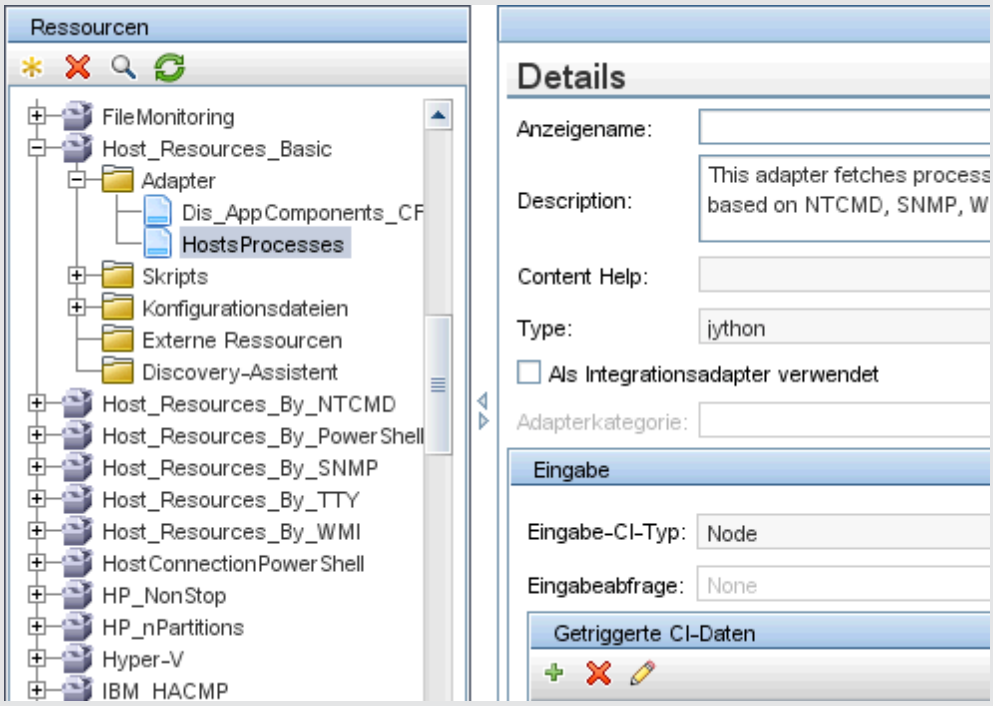
a. Klicken Sie auf **Datenflussverwaltung > Adapterverwaltung**. Wählen Sie den Adapter **HostProcesses** aus (**Packages > Host_Resources_Basic > Adapter > HostProcesses**).

b. Suchen Sie das Feld **Eingabe-CI-Typ**. Weitere Informationen finden Sie unter "Registerkarte "Adapterdefinition"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

c. Klicken Sie auf die Schaltfläche, um das Dialogfeld **Discovery-Klasse auswählen** zu öffnen. Weitere Informationen finden Sie unter "Dialogfeld "Discovery-Klasse auswählen"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*

d. Wählen Sie den CIT aus.

In diesem Beispiel ist das IP-CI (Host) im Adapter zulässig:



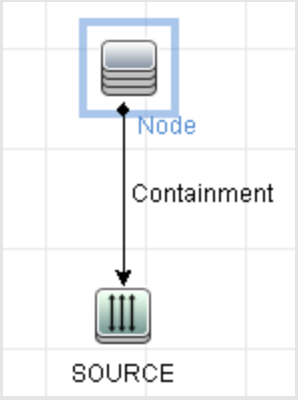
Beispiel für eine Eingabeabfragedefinition

In diesem Beispiel definiert die Eingabe-TQL-Abfrage, dass das `IpAddress`-CI (das im vorherigen Beispiel als Trigger-CIT konfiguriert wurde) mit einem `Node`-CI verbunden werden muss.

a. Klicken Sie auf **Datenflussverwaltung > Adapterverwaltung**. Suchen Sie das Feld **Eingabeabfrage**. Klicken Sie auf die Schaltfläche **Eingabeabfrage bearbeiten**, um den Eingabe-Abfrageeditor zu öffnen. Weitere Informationen finden Sie unter "Fenster "Eingabe-Abfrageeditor"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

b. Geben Sie im Eingabe-Abfrageeditor für den Trigger-CI-Knoten den Namen **SOURCE** ein: Klicken Sie mit der rechten Maustaste auf den Knoten und wählen Sie **Abfrageknoteneigenschaften** aus. Ändern Sie im Feld **Elementname** den Namen in **SOURCE**.

c. Fügen Sie dem `IpAddress`-CI ein `Node`-CI und eine `Containment`-Beziehung hinzu. Weitere Informationen zum Arbeiten mit dem Eingabe-Abfrageeditor finden Sie unter "Fenster "Eingabe-Abfrageeditor"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.



Das **IpAddress**-CI ist mit einem **Node**-CI verbunden. Die Eingabe-TQL besteht aus zwei Knoten, **Node** und **IpAddress**, die miteinander verknüpft sind. Das **IpAddress**-CI hat den Namen **SOURCE**.

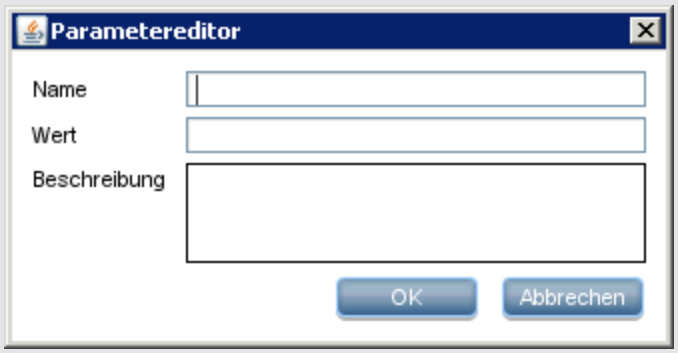
Beispiel für das Hinzufügen von Variablen zur Eingabe-TQL-Abfrage:

In diesem Beispiel fügen Sie die Variablen DIRECTORY und CONFIGURATION_FILE zu der im vorherigen Beispiel erstellten Eingabe-TQL-Abfrage hinzu. Mithilfe dieser Variablen wird definiert, was per Discovery ermittelt werden muss. In diesem Fall geht es darum, die Konfigurationsdateien auf den Hosts zu lokalisieren, die mit den zu ermittelnden IPs verknüpft sind.

a. Zeigen Sie die im vorherigen Beispiel erstellte Eingabe-TQL an.

   Klicken Sie auf **Datenflussverwaltung > Adapterverwaltung**. Suchen Sie den Ausschnitt **Getriggerte CI-Daten**. Weitere Informationen finden Sie unter "Registerkarte "Adapterdefinition"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

b. Fügen Sie Variablen zur Eingabe-TQL hinzu. Klicken Sie für weitere Informationen auf **Datenflussverwaltung > Adapterverwaltung**. Suchen Sie den Ausschnitt **Getriggerte CI-Daten**. Weitere Informationen finden Sie unter "Registerkarte "Adapterdefinition"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung* im Abschnitt zum Variablenfeld.



Beispiel für das Ersetzen der Variablen durch tatsächliche Daten:

In diesem Beispiel ersetzen die Variablen die **IpAddress**-CI-Daten durch tatsächliche Werte von realen **IpAddress**-CI-Instanzen Ihres Systems.

Die getriggerten CI-Daten für das **IpAddress**-CI enthalten eine Variable vom Typ `fileName`. Diese Variable ermöglicht den Austausch des Knotens **CONFIGURATION_DOCUMENT** im Eingabe-TQL durch die tatsächlichen Werte der auf einem Host gespeicherten Konfigurationsdatei:

Getriggerte CI-Daten

Name	Wert
Protocol	${SOURCE.credentials_id}
credentialsId	${SOURCE.credentials_id}
fileName	${CONFIGURATION_FILE.data_name}
hostID	${HOST.root_id}
ip_address	${SOURCE.application_ip}
path	${CONFIGURATION_FILE.document_path}

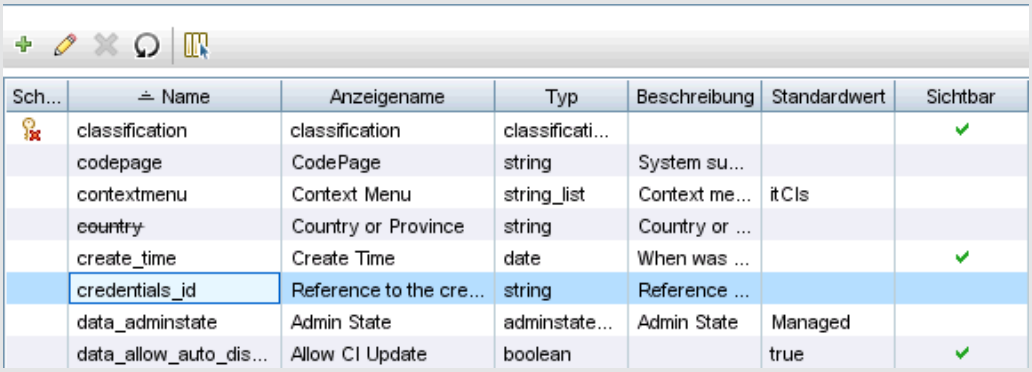
Die Trigger-CI-Daten werden zur Probe hochgeladen, wobei alle Variablen durch tatsächliche Werte ersetzt worden sind. Das Adapterskript enthält einen Befehl, der angibt, dass DFM Framework zum Abrufen der tatsächlichen Werte der definierten Variablen verwendet werden soll:

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

Die Variablen `fileName` und `path` verwenden die Attribute `data_name` und `document_path` vom Knoten mit der Bezeichnung **CONFIGURATION_DOCUMENT** (der in der Eingabeabfrageeditor definiert wurde – siehe vorheriges Beispiel).

Klicken Sie hier für eine Abbildung.

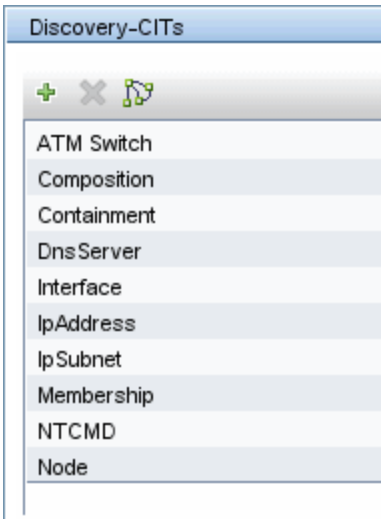
Die Variablen `Protocol`, `credentialsId` und `ip_address` verwenden die Attribute `root_class`, `credentials_id` und `application_ip`:



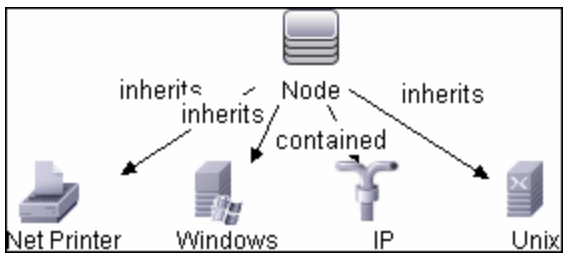
Sch...	Name	Anzeigename	Typ	Beschreibung	Standardwert	Sichtbar
	classification	classification	classificati...			✔
	codepage	CodePage	string	System su...		
	contextmenu	Context Menu	string_list	Context me...	itCIs	
	~~country~~	Country or Province	string	Country or ...		
	create_time	Create Time	date	When was ...		✔
	credentials_id	Reference to the cre...	string	Reference ...		
	data_adminstate	Admin State	adminstate...	Admin State	Managed	
	data_allow_auto_dis...	Allow CI Update	boolean		true	✔

2. Definieren der Adapterausgabe

Die Ausgabe des Adapters ist eine Liste der Discovery-CIs (**Datenflussverwaltung > Adapterverwaltung > Discovery-CITs** auf der Registerkarte **> Adapterdefinition**) und der zwischen ihnen bestehenden Verknüpfungen:



Sie können die CITs auch als Topologiekarte anzeigen, in der die Komponenten und ihre Verknüpfungen dargestellt werden (klicken Sie dazu auf die Schaltfläche **Discovery-CIT als Karte anzeigen**):



Die Discovery-CIs werden vom Datenflussverwaltungscode (d. h. dem Jython-Skript) im UCMDB-Format `ObjectStateHolderVector` zurückgegeben. Weitere Informationen finden Sie unter "Ergebnisgenerierung durch das Jython-Skript" auf Seite 46.

Beispiel für die Adapterausgabe:

In diesem Beispiel legen Sie fest, welche CITs bei der IP-CI-Ausgabe berücksichtigt werden sollen.

a. Klicken Sie auf **Datenflussverwaltung > Adapterverwaltung**.

b. Klicken Sie im Ausschnitt **Ressourcen** auf **Network > Adapter > NSLOOKUP_on_Probe**.

c. Suchen Sie auf der Registerkarte **Adapterdefinition** den Ausschnitt **Discovery-CITs**,

d. Hier sind die CITs aufgelistet, die in die Adapterausgabe einbezogen werden sollen. Sie können CITs zur Liste hinzufügen bzw. aus der Liste entfernen. Weitere Informationen finden Sie unter "Registerkarte "Adapterdefinition"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

3. Überschreiben der Adapterparameter

Sie können einen Adapter für mehrere Jobs konfigurieren, indem Sie die Adapterparameter überschreiben. Der Adapter `SQL_NET_Dis_Connection` wird beispielsweise sowohl vom Job `MSSQL Connection by SQL` als auch vom Job `Oracle Connection by SQL` verwendet.

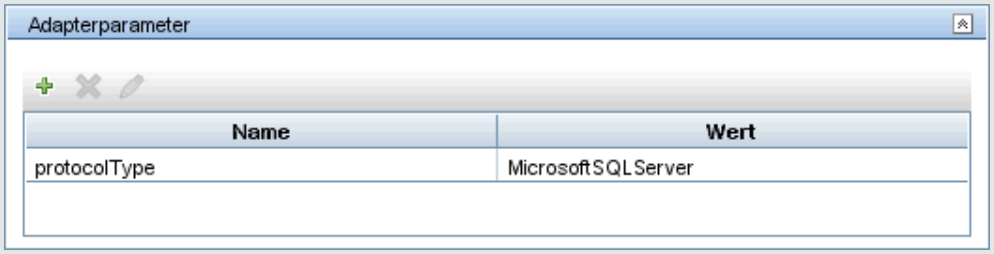
Beispiel für das Überschreiben eines Adapterparameters:

In diesem Beispiel geht es darum, einen Adapterparameter zu überschreiben, um mit einem einzigen Adapter sowohl die Microsoft SQL Server- als auch die Oracle-Datenbank per Discovery zu ermitteln.

a. Klicken Sie auf **Datenflussverwaltung > Adapterverwaltung**.

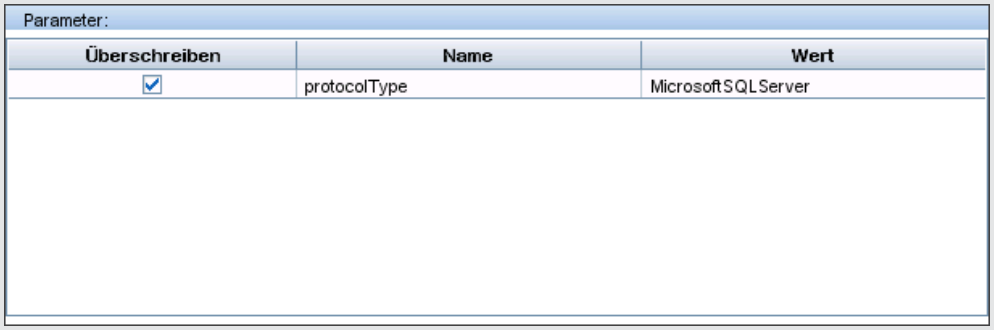
b. Wählen Sie im Ausschnitt **Ressourcen** die Optionen **Database_Basic > Adapter > SQL_NET_Dis_Connection** aus.

c. Suchen Sie auf der Registerkarte **Adapterdefinition** den Ausschnitt **Adapterparameter**. Der Parameter `protocolType` hat den Wert **all**:



d. Klicken Sie mit der rechten Maustaste auf den Adapter **SQL_NET_Dis_Connection_MsSql** und wählen Sie **Gehe zu Discovery-Job > MSSQL Connection by SQL** aus.

e. Öffnen Sie die Registerkarte **Eigenschaften**. Suchen Sie den Ausschnitt **Parameter**:



Der Wert `all` wird durch den Wert `MicrosoftSQLServer` überschrieben.

Hinweis: Der Job **Oracle Connection by SQL** enthält den gleichen Parameter. Der Wert wird jedoch durch einen Oracle-Wert überschrieben.

Detaillierte Informationen zum Hinzufügen, Löschen oder Bearbeiten von Parametern finden Sie unter "Registerkarte "Adapterdefinition"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Entsprechend diesem Parameter beginnt die Datenflussverwaltung mit der Suche nach Microsoft SQL Server-Instanzen.

4. ## Überschreiben der Auswahl der Probe – optional

Der UCMDB-Server verfügt über einen Verteilungsmechanismus, der die von der UCMDB empfangenen Trigger-CIs nimmt und entsprechend einer der folgenden Optionen automatisch auswählt, welche Probe den Job für jede Tripper-CI ausführen soll.

- **Für den IP-Adress-CI-Typ:** Nehmen Sie die Probe, die für diese IP definiert ist.
- **Für den ausgeführten Software-CI-Typ:**Wählen Sie die Attribute **application_ip** und **application_ip_domain** aus und wählen Sie die Probe aus, die für die IP in der relevanten Domäne definiert ist.
- **Für andere CI-Typen:** Nehmen Sie die IP des Knotens gemäß dem zugehörigen Knoten des CI (sofern dieser vorhanden ist).

Die automatische Probenauswahl erfolgt entsprechen dem zugehörigen Knoten des CI. Nachdem der zugehörige Knoten des CI abgerufen wurde, wählt der Verteilungsmechanismus eine der IPs des Knotens und die Probe entsprechend den Definitionen des Netzwerkbereichs für die Probe aus.

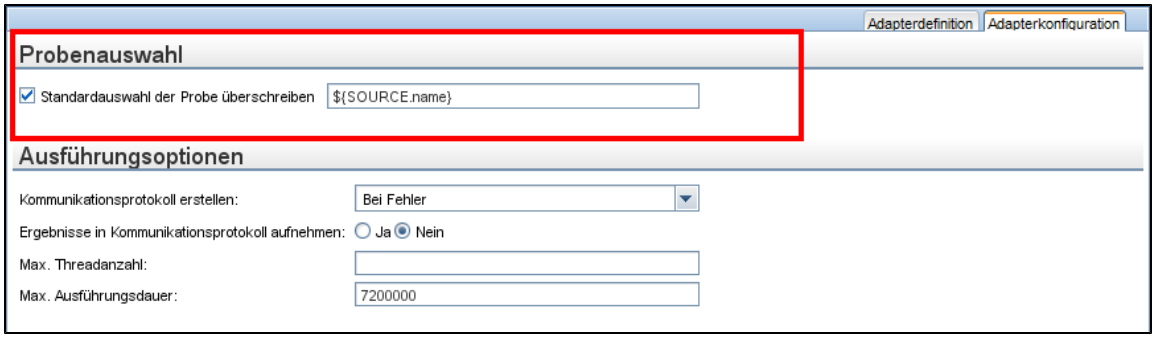
In den folgenden Fällen müssen Sie die Probe manuell angeben und können nicht den automatischen Verteilungsmechanismus verwenden:

- Sie wissen bereits, welche Probe für den Adapter ausgeführt werden soll, und Sie benötigen den automatischen Verteilungsmechanismus nicht für die Auswahl der Probe (beispielsweise wenn es sich beim Trigger-CI um das Probe Gateway handelt).
- Die automatische Probenauswahl schlägt möglicherweise fehl. Dies kann in folgenden Situationen geschehen:

- Ein Trigger-CI weist keinen zugehörigen Knoten auf (wie der Netzwerk-CIT).
- Der Knoten eines Trigger-CI weist mehrere IPs auf und jede gehört zu einer anderen Probe.

Um diese Probleme zu lösen, können Sie wie folgt angeben, welche Probe mit dem Adapter verwendet werden soll:

a. Wählen Sie im Abschnitt **Probenauswahl** die Option **Standardauswahl der Probe überschreiben** aus, wie unten gezeigt.



b. Geben Sie in das Feld **Probe** die Probe für die Aufgabe ein.

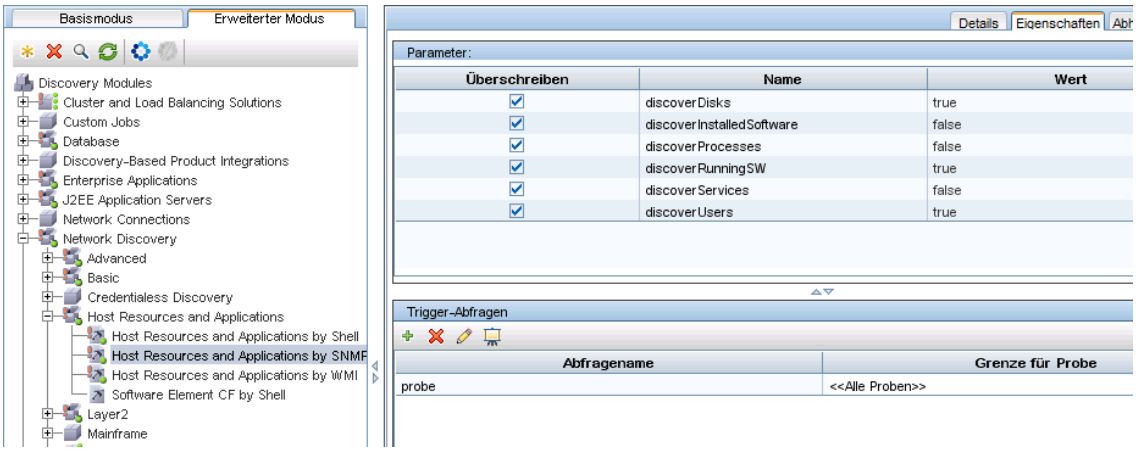
5. **Konfigurieren eines Klassenpfads für einen Remoteprozess – optional**

Weitere Informationen finden Sie unter "Konfigurieren der Remoteprozess-Ausführung" auf Seite 39.

Schritt 2: Zuweisen eines Jobs zum Adapter

Jedem Adapter ist mindestens ein Job zugeordnet, der die Ausführungsrichtlinie definiert. Mithilfe von Jobs können für einen Adapter unterschiedliche Planungen für verschiedene Sätze von getriggerten CIs festgelegt werden. Außerdem können für jeden Satz andere Parameter bereitgestellt werden.

Die Jobs werden, wie in der Abbildung unten dargestellt, in der Strukturansicht unter **Discovery-Module** angezeigt. Diese Entität wird vom Benutzer aktiviert.



Auswählen einer Trigger-TQL

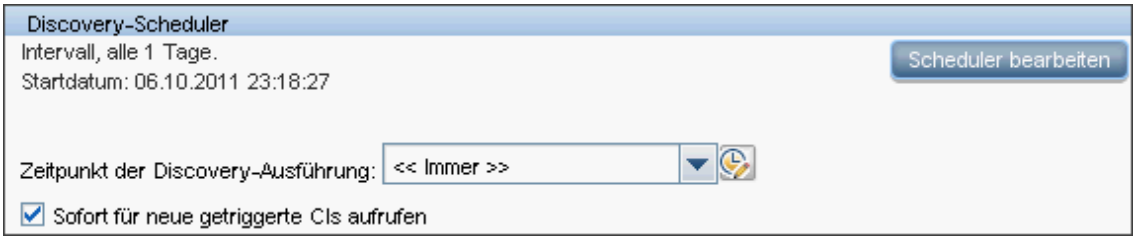
Jedem Job sind Trigger-TQLs zugeordnet. Diese Trigger-TQLs veröffentlichen Ergebnisse, die als Eingabe-Trigger-CIs für den Adapter dieses Jobs verwendet werden.

Eine Trigger-TQL kann Einschränkungen zu einer Eingabe-TQL hinzufügen. Wenn es sich beispielsweise bei den Ergebnissen einer Eingabe-TQL um IPs handelt, die mit SNMP verbunden sind, können die Ergebnisse einer Trigger-TQL die mit SNMP verbundenen IPs im Bereich von 195.0.0.0 bis 195.0.0.10 sein.

Hinweis: Eine Trigger-TQL muss auf die gleichen Objekte verweisen wie die Eingabe-TQL. Beispiel: Wenn eine Eingabe-TQL Abfragen nach IPs durchführt, die mit SNMP ausgeführt werden, ist es nicht möglich, für den gleichen Job eine Trigger-TQL zu definieren, um Abfragen nach den mit einem Host verbundenen IPs durchzuführen, da einige der IPs möglicherweise nicht mit einem SNMP-Objekt verbunden sind (wie von der Eingabe-TQL vorgegeben wurde).

Festlegen von Planungsinformationen

Die Planungsinformationen für die Probe geben an, wann der Code bei den Trigger-CIs ausgeführt werden soll. Wenn das Kontrollkästchen **Sofort für neue getriggerte CIs aufrufen** aktiviert ist, wird der Code bei Erreichen der Probe - ungeachtet der Planungseinstellungen - auch einmal bei jedem Trigger-CI ausgeführt.



Für jedes geplante Vorkommen der einzelnen Jobs führt die Probe den Code für alle Trigger-CIs aus, die für den jeweiligen Job akkumuliert wurden. Weitere Informationen finden Sie unter "Dialogfeld "Discovery-Scheduler"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Überschreiben der Adapterparameter

Bei der Konfiguration eines Jobs können Sie die Adapterparameter überschreiben. Weitere Informationen finden Sie unter "Überschreiben der Adapterparameter" auf Seite 35.

Schritt 3: Erstellen von Jython-Code

HP Universal CMDB verwendet zum Schreiben von Adaptern Jython-Skripts. Das Skript `SNMP_Connection.py` beispielsweise wird vom Adapter `SNMP_NET_Dis_Connection` verwendet, um unter Verwendung von SNMP eine Verbindung zu Computern herzustellen. Jython ist eine auf Python basierende Sprache, die von Java unterstützt wird.

Weitere Informationen zur Arbeit mit Jython finden Sie auf folgenden Websites:

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

Weitere Informationen finden Sie unter "Erstellen von Jython-Code" auf Seite 41.

Konfigurieren der Remoteprozess-Ausführung

Das Ausführen der Discovery für einen Discovery-Job kann in einem vom Data Flow Probe-Prozess getrennten Prozess erfolgen.

Beispielsweise können Sie den Job in einem separaten Remoteprozess ausführen, wenn er JAR-Bibliotheken verwendet, die eine andere Version aufweisen als die Bibliotheken der Probe oder die nicht mit den Bibliotheken der Probe kompatibel sind.

Ferner können Sie den Job in einem separaten Remoteprozess ausführen, wenn er voraussichtlich viel Speicher verbraucht (weil eine große Datenmenge vorhanden ist) und Sie die Probe von potenziellen Problemen wegen zu geringer Speicherkapazität isolieren möchten.

Um einen Job für die Ausführung als Remoteprozess zu konfigurieren, definieren Sie die folgenden Parameter in der Konfigurationsdatei des Adapters:

Parameter	Beschreibung
remoteJVMArgs	JVM-Parameter für den Java-Remoteprozess.
runInSeparateProcess	Die Einstellung **true** bewirkt, dass der Discovery-Job in einem separaten Prozess ausgeführt wird.
remoteJVMClasspath	(Optional) Ermöglicht die Anpassung des Klassenpfads für den Remoteprozess. Die Einstellung, die Sie hier vornehmen, überschreibt den Standardklassenpfad der Probe. Dies ist sinnvoll bei einer möglichen Versionsinkompatibilität zwischen den JAR-Dateien der Probe und den benutzerdefinierten JAR-Dateien, die für die benutzerdefinierte Discovery erforderlich sind. Ist der Parameter **remoteJVMClasspath** nicht definiert oder leer, wird der Standardklassenpfad der Probe verwendet. Wenn Sie einen neuen Discovery-Job entwickeln und sicherstellen möchten, dass die Version der JAR-Bibliothek der Probe nicht mit den JAR-Bibliotheken des Jobs kollidiert, müssen Sie mindestens den minimalen Klassenpfad verwenden, der für die Ausführung der Basis-Discovery erforderlich ist. Der minimale Klassenpfad ist in der Datei **DataFlowProbe.properties** im Parameter **basic_discovery_minimal_classpath** definiert. Beispiele für die Anpassung von **remoteJVMClasspath**: • Um dem Standardklassenpfad der Probe benutzerdefinierte JAR-Dateien voranzustellen oder anzuhängen, passen Sie den Parameter **remoteJVMClasspath** wie folgt an: `custom1.jar;%classpath%;custom2.jar -` In diesem Fall wird **custom1.jar** vor dem Standardklassenpfad der Probe platziert und **custom2.jar** an den Klassenpfad der Probe angehängt. • Um den minimalen Klassenpfad zu verwenden, passen Sie den

Parameter	Beschreibung
	Parameter **remoteJVMClasspath** wie folgt an: `custom1.jar;%minimal_classpath%;custom2.jar`

Kapitel 2

Entwickeln von Jython-Adaptern

Dieses Kapitel umfasst folgende Themen:

API-Referenz zur Datenflussverwaltung von HP

Die vollständige Dokumentation zu den verfügbaren APIs finden Sie in der *API-Referenz zur Datenflussverwaltung von HP Universal CMDB*. Die Dateien befinden sich im folgenden Ordner:

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

Erstellen von Jython-Code

HP Universal CMDB verwendet zum Schreiben von Adaptern Jython-Skripts. Das Skript `SNMP_Connection.py` beispielsweise wird vom Adapter `SNMP_NET_Dis_Connection` verwendet, um unter Verwendung von SNMP eine Verbindung zu Computern herzustellen. Jython ist eine auf Python basierende Sprache, die von Java unterstützt wird.

Weitere Informationen zur Arbeit mit Jython finden Sie auf folgenden Websites:

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

Im folgenden Abschnitt wird das Schreiben von Jython-Code innerhalb des Datenflussverwaltungs-Frameworks beschrieben. Dabei wird insbesondere auf die Kontaktpunkte zwischen dem Jython-Skript und dem von ihm aufgerufenen Framework eingegangen. Darüber hinaus werden die Jython-Bibliotheken und Dienstprogramme beschrieben, die nach Möglichkeit verwendet werden sollten.

Hinweis:

- Skripts, die für die Datenflussverwaltung geschrieben werden, sollten mit der Jython-Version 2.1 kompatibel sein.

- Die vollständige Dokumentation zu den verfügbaren APIs finden Sie in der *API-Referenz zur Datenflussverwaltung von HP Universal CMDB*.

Dieser Abschnitt umfasst die folgenden Themen:

Verwenden externer JAR-Dateien in Jython-Skripts

Bei der Entwicklung neuer Jython-Skripts sind gelegentlich externe Java-Bibliotheksdateien (JAR-Dateien) oder ausführbare Dateien von Drittanbietern als Java-Dienstprogrammarchive, Verbindungsarchive (z. B. JDBC-Treiber-JAR-Dateien) oder ausführbare Dateien erforderlich (**nmap.exe** wird beispielsweise für die Durchführung einer Discovery ohne Anmeldeinformationen verwendet).

Diese Ressourcen sollten in einem Package im Ordner **Externe Ressourcen** gebündelt werden. Jede Ressource, die sich in diesem Ordner befindet, wird automatisch an jede Probe gesendet, die eine Verbindung zu Ihrem HP Universal CMDB-Server herstellt.

Außerdem wird beim Starten der Discovery jede JAR-Dateiressource in den Jython-Klassenpfad geladen, sodass alle darin enthaltenen Klassen für den Import und die Verwendung zur Verfügung stehen.

Ausführen des Codes

Nachdem ein Job aktiviert wurde, wird eine Aufgabe mit allen erforderlichen Informationen zur Probe heruntergeladen.

Anhand der in der Aufgabe angegebenen Informationen beginnt die Probe mit der Ausführung des Datenflussverwaltungscodes.

Der Jython-Codefluss beginnt mit der Ausführung bei einem Haupteintrag im Skript, führt den Code für die Discovery der CIs aus und liefert die Ergebnisse eines Vektors der Discovery-CIs.

Ändern von Standardskripts

An Standardskripts sollten nur minimale Änderungen vorgenommen werden. Eventuell erforderliche Methoden sollten in einem externen Skript gespeichert werden. Auf diese Weise können Sie

Änderungen effizienter verfolgen und wenn Sie auf eine neuere HP Universal CMDB-Version aktualisieren, wird Ihr Code nicht überschrieben.

Die folgende Codezeile eines Standardskripts ruft beispielsweise eine Methode auf, die den Namen eines Webservers anwendungsspezifisch berechnet:

```
serverName = iplanet_cspecific.PlugInProcessing(serverName,
transportHN, mam_utils)
```

Die komplexere Logik, die entscheidet, wie dieser Name berechnet wird, befindet sich in einem externen Skript:

```
# 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
```

Speichern Sie das externe Skript im Ordner **Externe Ressourcen**. Weitere Informationen finden Sie unter "Ausschnitt "Ressourcen"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*. Wenn Sie das Skript zu einem Package hinzufügen, können Sie es auch für andere Jobs verwenden. Weitere Informationen zum Arbeiten mit Package Manager finden Sie unter "Package Manager" im *HP Universal CMDB – Verwaltungshandbuch*.

Während der Aktualisierung werden Änderungen, die Sie an einer Codezeile vorgenommen haben, durch die neue Version des Standardskripts überschrieben, sodass Sie die Zeile ersetzen müssen. Das externe Skript wird jedoch nicht überschrieben.

Struktur der Jython-Datei

Die Jython-Datei besteht aus drei Teilen, die in einer bestimmten Reihenfolge angeordnet sind:

1. Importe
2. Hauptfunktion - DiscoveryMain
3. Funktionsdefinitionen (optional)

Im Folgenden sehen Sie ein Beispiel für ein Jython-Skript:

```
#   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
```

Importe

Jython-Klassen erstrecken sich über hierarchische Namenspaces. Ab Version 7.0 gibt es im Gegensatz zu früheren Versionen keine impliziten Importe mehr. Daher muss jede Klasse, die Sie verwenden, explizit importiert werden. (Die Änderung wurde vorgenommen, um die Leistung zu verbessern und das Verständnis des Jython-Skripts dadurch zu erleichtern, dass keine notwendigen Details mehr ausgeblendet werden.)

- So importieren Sie ein Jython-Skript:

  ```
  import logger
  ```

- So importieren Sie eine Java-Klasse:

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

Hauptfunktion – DiscoveryMain

Jede ausführbare Jython-Skriptdatei enthält eine Hauptfunktion: DiscoveryMain.

Die `DiscoveryMain`-Funktion ist der Haupteintrag im Skript; es ist die erste Funktion, die ausgeführt wird. Die Hauptfunktion kann andere Funktionen aufrufen, die in den Skripts definiert sind:

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

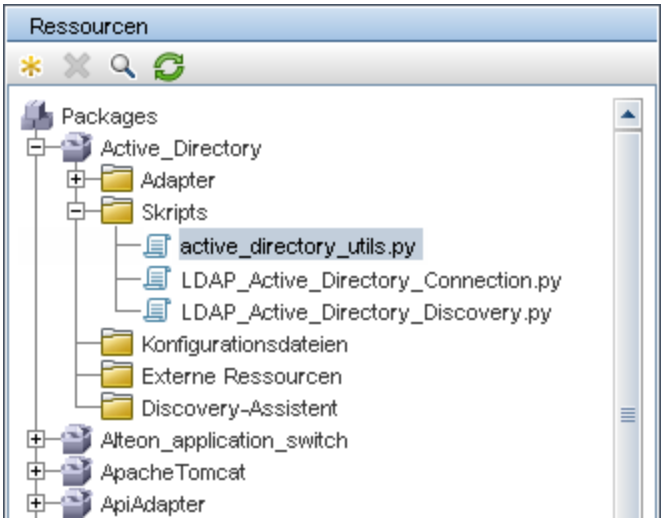
Das Framework-Argument muss in der Definition der Hauptfunktion angegeben werden. Dieses Argument wird von der Hauptfunktion zum Abrufen von Informationen verwendet, die für die Ausführung der Skripts erforderlich sind (z. B. Informationen zum Trigger-CI und den Parametern). Zudem kann es zum Melden von Fehlern verwendet werden, die bei der Skriptausführung auftreten.

Sie können ein Jython-Skript auch ohne Hauptmethode erstellen. Diese Skripts werden als Bibliotheksskripts verwendet, die von anderen Skripts aufgerufen werden.

Funktionsdefinition

Jedes Skript kann zusätzliche Funktionen enthalten, die vom Hauptcode aufgerufen werden. Jede dieser Funktionen kann wiederum eine andere Funktion aus dem aktuellen Skript oder einem

anderen Skript (über die Anweisung `import`) aufrufen. Um ein anderes Skript verwenden zu können, müssen Sie es zum Skriptordner des Packages hinzufügen:



Beispiel für eine Funktion, die eine andere Funktion aufruft:

Im folgenden Beispiel ruft der Hauptcode die Methode `doQueryOSUsers(..)` auf, die wiederum die interne Methode `doOSUserOSH(..)` aufruft:

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

```
        client.close()
    return OSHVResult
```

Wenn dieses Skript eine globale Bibliothek ist, die für mehrere Adapter relevant ist, können Sie es zur Skriptliste in der Konfigurationsdatei `jythonGlobalLibs.xml` hinzufügen, anstatt es zu jedem einzelnen Adapter hinzuzufügen (**Adapterverwaltung > Ausschnitt 'Ressourcen' > AutoDiscoveryContent > Konfigurationsdateien**).

Ergebnisgenerierung durch das Jython-Skript

Jedes Jython-Skript wird bei einem bestimmten Trigger-CI ausgeführt und endet mit den Ergebnissen, die vom Rückgabewert der Funktion `DiscoveryMain` zurückgegeben werden.

Bei dem Skriptergebnis handelt es sich eigentlich um eine Gruppe von CIs und Links, die in die CMDB eingefügt oder dort aktualisiert werden. Das Skript gibt diese Gruppe von CIs und Links im Format `ObjectStateHolderVector` zurück.

Die Klasse `ObjectStateHolder` ist eine Möglichkeit, ein Objekt oder einen Link, das bzw. der in der CMDB definiert ist, darzustellen. Das Objekt `ObjectStateHolder` enthält den CIT-Namen zusammen mit einer Liste der Attribute und ihrer Werte. `ObjectStateHolderVector` ist ein Vektor der `ObjectStateHolder`-Instanzen.

Die ObjectStateHolder-Syntax

In diesem Abschnitt wird beschrieben, wie die Datenflussverwaltungsergebnisse in ein UCMDB-Modell eingebaut werden.

Beispiel für das Setzen von Attributen bei den CIs:

Die ObjectStateHolder-Klasse beschreibt das Ergebnisdiagramm der Datenflussverwaltung. Alle CIs und Links (Beziehungen) werden wie im folgenden Jython-Codebeispiel in einer Instanz der `ObjectStateHolder`-Klasse gespeichert:
siebel application server 1 appServerOSH = ObjectStateHolder('siebelappserver') 2 appServerOSH.setStringAttribute('data_name', sblsvrName) 3 appServerOSH.setStringAttribute ('application_ip', ip) 4 appServerOSH.setContainer (appServerHostOSH)

- Zeile 1 erstellt ein CI des Typs **siebelappserver**.
- Zeile 2 erstellt ein Attribut namens **data_name** mit dem Wert **sblsvrName**. Dabei handelt es sich um eine Jython-Variable, die mit dem für den Servernamen ermittelten Wert gesetzt wird.
- Zeile 3 setzt ein Nichtschlüsselattribut, das in der CMDB aktualisiert wird.
- Zeile 4 ist die Erstellung eines Containment (das Ergebnis ist ein Diagramm). Sie gibt an, dass dieser Applikationsserver in einem Host enthalten ist (eine andere `ObjectStateHolder`-Klasse im Gültigkeitsbereich).

Hinweis: Jedes vom Jython-Skript gemeldete CI muss Werte für alle Schlüsselattribute des zugehörigen CI-Typs enthalten.

Beispiel für Beziehungen (Links):

Das folgende Link-Beispiel erklärt die Darstellung des Diagramms:
1 linkOSH = ObjectStateHolder('route') 2 linkOSH.setAttribute('link_end1', gatewayOSH) 3 linkOSH.setAttribute('link_end2', appServerOSH)

- Zeile 1 erstellt den Link (ebenfalls von der Klasse `ObjectStateHolder`. Der einzige Unterschied ist, dass `route` ein Link-CI-Typ ist).
- Die Zeilen 2 und 3 geben die Knoten am Ende jedes Links an. Dies geschieht mithilfe der Attribute **end 1** und **end 2** des Links, die angegeben werden müssen (da sie die Mindestschlüsselattribute jedes Links sind). Die Attributwerte sind `ObjectStateHolder`-Instanzen. Weitere Informationen zu den Attributen **end 1** und **end 2** finden Sie unter "Link" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Achtung: Ein Link ist richtungsgebunden. Sie sollten daher sicherstellen, dass die Knoten **end 1** und **end 2** an jedem Ende gültigen CITs entsprechen. Wenn die Knoten ungültig sind, schlägt die Prüfung des Ergebnisobjekts fehl, sodass das Objekt nicht korrekt gemeldet wird. Weitere Informationen finden Sie unter "CIT-Beziehungen" im *HP Universal CMDB – Modellierungshandbuch*.

Vektorbeispiel (Sammeln von CIs):

Nachdem Sie Objekte mit Attributen und Links mit Objekten an den Enden erstellt haben, müssen Sie die Objekte und Links noch gruppieren. Zu diesem Zweck müssen Sie sie wie folgt zu einer `ObjectStateHolderVector`-Instanz hinzufügen:

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

Weitere Informationen darüber, wie Sie dieses Gesamtergebnis an das Framework übertragen, damit es an den CMDB-Server gesendet werden kann, finden Sie in der Beschreibung zur sendObjects-Methode.

Nachdem das Ergebnisdiagramm in einer `ObjectStateHolderVector`-Instanz erstellt wurde, muss es an das Datenflussverwaltungs-Framework zurückgegeben werden, damit es in die CMDB eingefügt werden kann. Zu diesem Zweck muss die `ObjectStateHolderVector`-Instanz als Ergebnis der `DiscoveryMain()`-Funktion zurückgegeben werden.

Hinweis: Weitere Informationen zum Erstellen des **OSH** für allgemeine CITs finden Sie unter "modeling.py" im Abschnitt "Jython-Bibliotheken und Dienstprogramme" auf Seite 77.

Die Framework-Instanz

Die Framework-Instanz ist das einzige Argument, das in der Hauptfunktion im Jython-Skript bereitgestellt wird. Es handelt sich hierbei um eine Schnittstelle, die zum Abrufen der für die Ausführung des Skripts erforderlichen Informationen verwendet werden kann (z. B. Informationen zu Trigger-CIs und Adapterparametern). Zudem wird sie zum Melden von Fehlern verwendet, die bei der Skriptausführung auftreten. Weitere Informationen finden Sie unter "API-Referenz zur Datenflussverwaltung von HP" auf Seite 41.

Die korrekte Verwendung der Framework-Instanz besteht darin, sie als Argument an jede Methode zu übergeben, die sie verwendet.

Beispiel:

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

In diesem Abschnitt werden die wichtigsten Framework-Verwendungen beschrieben:

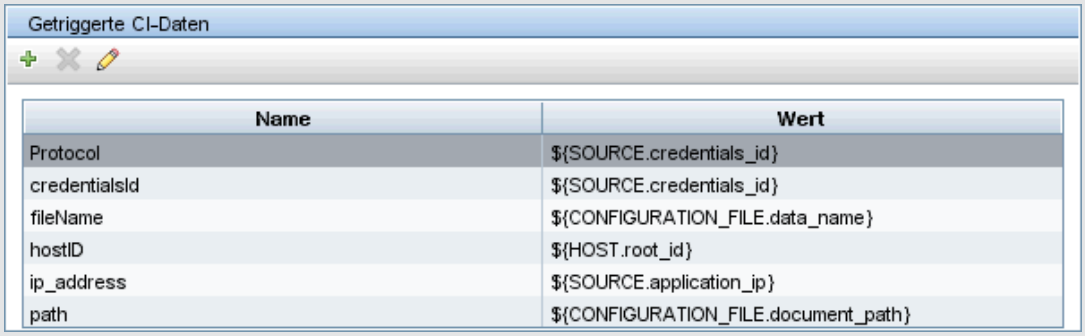
- "Framework.getTriggerCIData(String attributeName)" oben
- "Framework.createClient(credentialsId, props)" oben
- "Framework.getParameter (parameterName-Zeichenkette)" auf Seite 50
- "Framework.reportError(Zeichenfolgennachricht) und Framework.reportWarning (Zeichenfolgennachricht)" auf Seite 50

Framework.getTriggerCIData(String attributeName)

Diese API liefert den Zwischenschritt zwischen den im Adapter definierten Trigger-CI-Daten und dem Skript.

Beispiel für das Abrufen von Anmeldeinformationen:

Sie fordern die folgenden Trigger-CI-Daten an:



Name	Wert
Protocol	${SOURCE.credentials_id}
credentialsId	${SOURCE.credentials_id}
fileName	${CONFIGURATION_FILE.data_name}
hostID	${HOST.root_id}
ip_address	${SOURCE.application_ip}
path	${CONFIGURATION_FILE.document_path}

Verwenden Sie diese API, um die Anmeldeinformationen von der Aufgabe abzurufen:

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

Framework.createClient(credentialsId, props)

Sie stellen eine Verbindung zu einem Remote-Computer her, indem Sie ein Client-Objekt erstellen und bei diesem Client Befehle ausführen. Um einen Client zu erstellen, müssen Sie die Klasse `ClientFactory` abrufen. Die Methode getClientFactory() empfängt den Typ des angeforderten

Client-Protokolls. Die Protokollkonstanten sind in der Klasse ClientsConsts definiert. Weitere Informationen zu Anmeldeinformationen und unterstützten Protokollen finden Sie im *HP Universal CMDB Discovery and Integration Content Guide*.

Beispiel für das Erstellen einer Client-Instanz für die Anmeldeinformationen-ID:

So erstellen Sie eine `Client`-Instanz für die Anmeldeinformationen-ID:

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

Sie können nun über die `Client`-Instanz eine Verbindung zum relevanten Computer oder zu der relevanten Applikation herstellen.

Beispiel für das Erstellen eines WMI-Client und das Ausführen einer WMI-Abfrage:

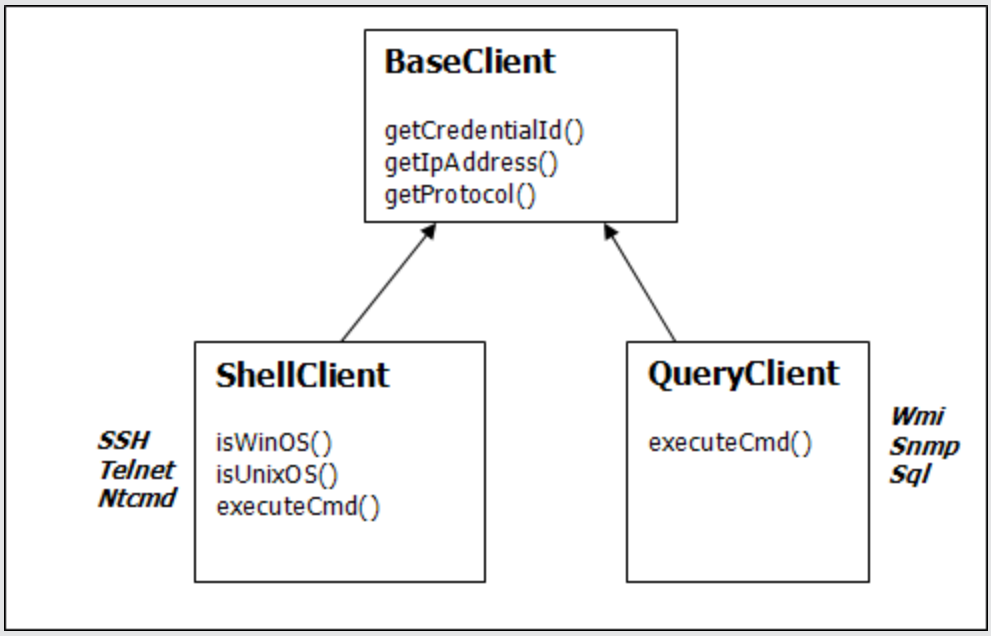
So können Sie einen WMI-Client erstellen und mit diesem eine WMI-Abfrage ausführen:

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

FROM    Win32_LogicalMemoryConfiguration")
```

Hinweis: Damit die `createClient()`-API funktioniert, müssen Sie im Ausschnitt **Getriggerte CI-Daten** den folgenden Parameter zu den Parametern der Trigger-CI-Daten hinzufügen: **credentialsId = ${SOURCE.credentials_id}**. Sie können auch die folgende Funktion aufrufen und die Anmeldeinformationen-ID manuell hinzufügen: **wmiClient = clientFactory().createClient(credentials_id)**.

Das folgende Diagramm zeigt die Hierarchie der Clients sowie die von jedem Client üblicherweise unterstützten APIs:

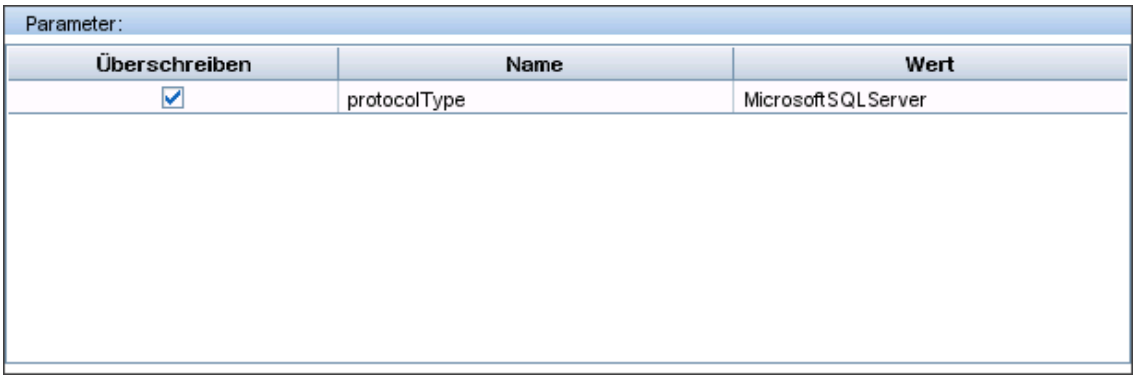


Weitere Informationen zu den Clients und ihren unterstützten APIs finden Sie unter BaseClient, ShellClient und QueryClient in der *Schemareferenz zu HP Discovery and Dependency Mapping*. Die Dateien befinden sich im folgenden Ordner:

<UCMDB-Stammverzeichnis>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

Framework.getParameter (parameterName-Zeichenkette)

Zusätzlich zum Abrufen von Informationen zum Trigger-CI müssen Sie häufig auch den Wert eines Adapterparameters abrufen. Beispiel:



Beispiel für das Abrufen des Wertes des protocolType-Parameters:

Verwenden Sie die folgende API, um den Wert des `protocolType`-Parameters vom Jython-Skript abzurufen:

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

Framework.reportError(Zeichenfolgennachricht) und Framework.reportWarning(Zeichenfolgennachricht)

Während der Skriptausführung können einige Fehler (z. B. Verbindungsfehler, Hardwareprobleme, Zeitüberschreitungen) auftreten. Werden derartige Fehler erkannt, kann Framework das Problem melden. Die Meldung wird an den Server weitergeleitet und dem Benutzer angezeigt.

Beispiel für einen Report-Fehler und eine Report-Meldung:

Das folgende Beispiel zeigt die Verwendung der `reportError(<Fehlermeldung>)`-API:

```
try:
        client = Framework.getClientFactory(ClientsConsts.SNMP_
PROTOCOL_NAME)
        createClient()
except:
        strException = str(sys.exc_info()[1]).strip()
        Framework. reportError ('Connection failed: %s' %
strException)
```

Sie können beide APIs - `Framework.reportError(Zeichenfolgennachricht)`, `Framework.reportWarning(Zeichenfolgennachricht)` - zum Melden eines Problems verwenden. Der Unterschied zwischen den beiden APIs besteht darin, dass die Probe beim Melden eines Fehlers eine Kommunikationsprotokolldatei mit den Parametern der gesamten Sitzung im Dateisystem speichert. Auf diese Weise können Sie die Sitzung verfolgen und den Fehler besser nachvollziehen.

Weitere Informationen zu Fehlermeldungen finden Sie unter "Fehlermeldungen" auf Seite 81.

Suchen nach den richtigen Anmeldeinformationen (für Verbindungsadapter)

Ein Adapter, der versucht, eine Verbindung zu einem Remote-System herzustellen, muss alle möglichen Anmeldeinformationen ausprobieren. Einer der Parameter, die beim Erstellen eines Client (über ClientFactory) erforderlich sind, ist die Anmeldeinformationen-ID. Das Verbindungsskript erhält Zugriff auf die möglichen Sätze mit Anmeldeinformationen und probiert diese unter Verwendung der Methode `clientFactory.getAvailableProtocols()` der Reihe nach aus. Wird ein passender Satz gefunden, meldet der Adapter ein CI-Verbindungsobjekt bei dem Host dieses Trigger-CI (dessen Anmeldeinformationen-ID mit der IP übereinstimmt) an die CMDB. Nachfolgende Adapter können dieses Verbindungsobjekt-CI direkt für die Verbindung zu dem Satz mit den Anmeldeinformationen verwenden (d. h., die Adapter müssen nicht mehr alle möglichen Anmeldeinformationen ausprobieren).

Das folgenden Beispiel zeigt, wie alle Einträge des SNMP-Protokolls abgerufen werden. Beachten Sie, dass die IP hier von den Trigger-CI-Daten abgerufen wird (`# Get the Trigger CI data values`).

Das Verbindungsskript fordert alle möglichen Protokollanmeldeinformationen an (`# Go over all the protocol credentials`) und probiert sie der Reihe nach aus, bis ein passender Satz gefunden wird (`resultVector`). Weitere Informationen finden Sie im Abschnitt **Das Paradigma der zweistufigen Verbindung** unter "Trennen von Adaptern" auf Seite 25.

```
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
```

Behandeln von Java-Ausnahmen

Einige Java-Klassen lösen bei einem Fehler eine Ausnahme aus. Es wird empfohlen, die Ausnahme aufzufangen und zu verarbeiten, da es andernfalls zu einer unerwarteten Beendigung des Adapters kommen kann.

Beim Auffangen einer bekannten Ausnahme sollten Sie nach Möglichkeit die Stapelablaufverfolgung in das Protokoll drucken und eine entsprechende Meldung für die Benutzeroberfläche ausgeben. Beispiel:

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

Wenn die Ausnahme nicht schwerwiegend ist und das Skript fortgesetzt werden kann, sollten Sie den Aufruf der Methode `reportError()` unterdrücken und mit der Ausführung des Skripts fortfahren.

Lokalisierungsunterstützung in Jython-Adaptern

Durch die mehrsprachige Gebietsschemafunktion kann die Datenflussverwaltung bei verschiedenen Betriebssystemsprachen eingesetzt und zum Zeitpunkt der Ausführung entsprechend benutzerspezifisch angepasst werden.

Vor Content Pack 3.00 verwendete die Datenflussverwaltung eine statisch spezifizierte Codierung für die Ausgabe von allen Netzwerkzielen. Dieser Ansatz eignet sich jedoch nicht für ein mehrsprachiges IT-Netzwerk: Um Hosts mit unterschiedlichen Betriebssystemsprachen per Discovery zu ermitteln, mussten die Probe-Administratoren die Datenflussverwaltungjobs mehrmals manuell ausführen und für jede Ausführung die Job-Parameter ändern. Dieses Verfahren führte zu einem erheblichen Overhead im Netzwerk und ließ mehrere Schlüsselfunktionen der

Datenflussverwaltung außer Acht, wozu unter anderem der sofortige Job-Aufruf bei einem Trigger-CI oder die automatische Datenaktualisierung in UCMDB durch den Schedule Manager gehörten.

Standardmäßig werden die folgenden Gebietsschemata unterstützt: Japanisch, Russisch und Deutsch. Das Standardgebietsschema ist Englisch.

Dieser Abschnitt umfasst Folgendes:

Hinzufügen von Unterstützung für eine neue Sprache

Diese Aufgabe beschreibt, wie Sie Unterstützung für eine neue Sprache hinzufügen.

Diese Aufgabe umfasst folgende Schritte:

1. ### Hinzufügen eines Ressourcen-Bundle (*.properties-Dateien)

   Fügen Sie je nach auszuführendem Job das passende Ressourcen-Bundle hinzu. In der folgenden Tabelle sind die Datenflussverwaltungsjobs mit den jeweiligen Ressourcen-Bundles aufgeführt:

Job	Basisname des Ressourcen-Bundle
File Monitor by Shell	langFileMonitoring
Host Resources and Applications by Shell	langHost_Resources_By_TTY, langTCP
Hosts by Shell using NSLOOKUP in DNS Server	langNetwork
Host Connection by Shell	langNetwork
Collect Network Data by Shell or SNMP	langTCP
Host Resources and Applications by SNMP	langTCP
Microsoft Exchange Connection by NTCMD, Microsoft Exchange Topology by NTCMD	msExchange
MS Cluster by NTCMD	langMsCluster

Weitere Informationen zu Bundles finden Sie unter "Arbeiten mit Ressourcen-Bundles" auf Seite 56.

2. **Deklarieren und Registrieren des Sprachobjekts**

   Um eine neue Sprache zu definieren, fügen Sie die folgenden zwei Codezeilen zum Skript **shellutils.py** hinzu, das momentan die Liste aller unterstützten Sprachen enthält. Das Skript befindet sich im Package `AutoDiscoveryContent`. Sie können über das Fenster **Adapterverwaltung** auf das Skript zugreifen. Weitere Informationen finden Sie unter "Fenster "Adapterverwaltung"" im HP Universal CMDB – Handbuch zur Datenflussverwaltung.

   a. Deklarieren Sie die Sprache wie folgt:

   ```
   LANG_RUSSIAN = Language(LOCALE_RUSSIAN, 'rus', ('Cp866',
   'Cp1251'), (1049,), 866)
   ```

   Weitere Informationen zur Klassensprache finden Sie unter "API-Referenz" auf Seite 58. Weitere Informationen zum Gebietsschemaobjekt der Klasse finden Sie unter http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html. Sie können ein vorhandenes Gebietsschema verwenden oder ein neues Gebietsschema definieren.

   b. Registrieren Sie die Sprache, indem Sie sie zur folgenden Auflistung hinzufügen:

   ```
   LANGUAGES = (LANG_ENGLISH, LANG_GERMAN, LANG_SPANISH, LANG_
   RUSSIAN, LANG_JAPANESE)
   ```

Ändern der Standardsprache

Wenn die Betriebssystemsprache nicht bestimmt werden kann, wird die Standardsprache verwendet. Die Standardsprache ist in der Datei **shellutils.py** angegeben.

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

Um die Standardsprache zu ändern, initialisieren Sie die Variable DEFAULT_LANGUAGE mit einer anderen Sprache. Weitere Informationen finden Sie unter "Hinzufügen von Unterstützung für eine neue Sprache" auf der vorherigen Seite.

Festlegen des Zeichensatzes für die Codierung

Der geeignete Zeichensatz für die Decodierung der Befehlsausgabe wird zum Zeitpunkt der Ausführung bestimmt. Die mehrsprachige Lösung basiert auf folgenden Fakten und Annahmen:

1. Es ist möglich, die Betriebssystemsprache unabhängig vom Gebietsschema zu bestimmen, beispielsweise durch Ausführen des Befehls **chcp** (Windows) bzw. des Befehls **locale** (Linux).
2. Die Sprachcodierung ist bekannt und kann statisch definiert werden. Die russische Sprache beispielsweise hat zwei der am häufigsten verwendeten Codierungen: `Cp866` und `Windows-1251`.
3. Für jede Sprache wird ein Zeichensatz bevorzugt. Der bevorzugte Zeichensatz für die russische Sprache beispielsweise ist `Cp866`. Dies bedeutet, dass die meisten Befehle eine Ausgabe in dieser Codierung erzeugen.
4. Die Codierung der nächsten Befehlsausgabe ist nicht vorhersagbar, aber es ist eine der

möglichen Codierungen für eine bestimmte Sprache. Wenn Sie beispielsweise mit einem Windows-Computer und russischem Gebietsschema arbeiten, liefert das System die Ausgabe des Befehls **ver** in Cp866 und die Ausgabe des Befehls **ipconfig** in Windows-1251.

5. Ein bekannter Befehl generiert in seiner Ausgabe bekannte Schlüsselwörter. Der Befehl **ipconfig** enthält zum Beispiel die übersetzten Formen der Zeichenkette **IP-Adresse**. Folglich enthält die Ausgabe des Befehls **ipconfig** die Zeichenkette **IP-Address** für das englische Betriebssystem, IP-Адрес für das russische Betriebssystem, **IP-Adresse** für das deutsche Betriebssystem usw.

Wenn feststeht, in welcher Sprache die Befehlsausgabe generiert wird (Schritt 1), ist die Anzahl der möglichen Zeichensätze auf einen oder zwei begrenzt (Schritt 2). Darüber hinaus ist bekannt, welche Schlüsselwörter in der Ausgabe enthalten sind (Schritt 5).

Die Lösung besteht daher darin, die Befehlsausgabe mit einer der möglichen Codierungen zu decodieren, indem im Ergebnis nach einem Schlüsselwort gesucht wird. Wird das Schlüsselwort gefunden, geht das System davon aus, dass der aktuelle Zeichensatz der richtige ist.

Definieren eines neuen Jobs für die Ausführung mit lokalisierten Daten

Diese Aufgabe beschreibt, wie Sie einen neuen Job schreiben, der mit lokalisierten Daten ausgeführt werden kann.

Jython-Skripts führen in der Regel Befehle aus und analysieren die Ausgabe. Um eine korrekte Decodierung dieser Befehlsausgabe sicherzustellen, verwenden Sie die API für die Klasse **ShellUtils**. Weitere Informationen finden Sie unter "HP Universal CMDB-Webservice-API – Übersicht" auf Seite 238.

Dieser Code ist normalerweise wie folgt aufgebaut:

```
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. Erstellen Sie einen Client:

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

2. Erstellen Sie eine Instanz der Klasse **ShellUtils** und fügen Sie die Betriebssystemsprache zu der Instanz hinzu. Wenn keine Sprache hinzugefügt wird, wird die Standardsprache (in der Regel Englisch) verwendet:

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

   Während der Objektinitialisierung erkennt die Datenflussverwaltung automatisch die Sprache des Computers und übernimmt die bevorzugte Codierung vom vordefinierten `Language`-

Objekt. Die bevorzugte Codierung ist die Codierung, die an erster Stelle in der Codierungsliste erscheint.

3. Rufen Sie das entsprechende Ressourcen-Bundle mit der Methode **getLanguageBundle** vom **shellclient** ab:

```
languageBundle = shellutils.getLanguageBundle (’langNetwork’,
shellUtils.osLanguage, Framework)
```

4. Rufen Sie ein Schlüsselwort vom Ressourcen-Bundle ab, das für einen bestimmten Befehl geeignet ist:

```
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_
str_ip_address')
```

5. Rufen Sie die Methode **executeCommandAndDecode** auf und leiten Sie das zugehörige Schlüsselwort über das Objekt **ShellUtils** weiter:

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

Das ShellUtils object wird außerdem benötigt, um einen Benutzer mit der API-Referenz zu verknüpfen (wo diese Methode ausführlich beschrieben wird).

6. Analysieren Sie die Ausgabe wie gewohnt.

Decodieren von Befehlen ohne Schlüsselwort

Der aktuelle Lokalisierungsansatz verwendet ein Schlüsselwort zum Decodieren der gesamten Befehlsausgabe. Weitere Informationen finden Sie im Schritt "Rufen Sie ein Schlüsselwort vom Ressourcen-Bundle ab, das für einen bestimmten Befehl geeignet ist:" unter "Definieren eines neuen Jobs für die Ausführung mit lokalisierten Daten" auf der vorherigen Seite.

Bei einem anderen Ansatz wird jedoch nur die erste Befehlsausgabe mithilfe eines Schlüsselworts decodiert. Alle weiteren Befehle werden mit dem Zeichensatz decodiert, der zum Decodieren des ersten Befehls verwendet wurde. Sie verwenden hierzu die Methoden **getCharsetName** und **useCharset** des Objekts **ShellUtils**.

Der reguläre Anwendungsfall sieht folgendermaßen aus:

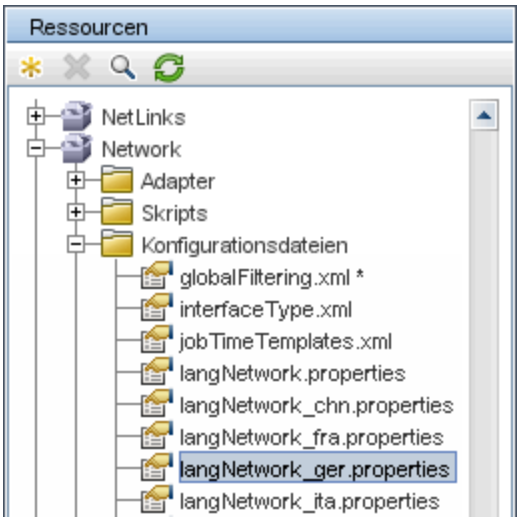
1. Rufen Sie einmal die Methode **executeCommandAndDecode** auf.
2. Rufen Sie über die Methode **getCharsetName** den Namen des zuletzt verwendeten Zeichensatzes ab.
3. Legen Sie fest, dass dieser Zeichensatz standardmäßig von **shellUtils** verwendet werden soll, indem Sie beim Objekt **ShellUtils** die Methode **useCharset** aufrufen.
4. Rufen Sie einmal oder mehrmals die Methode **execCmd** des Objekts **ShellUtils** auf. Die Ausgabe wird mit dem im vorherigen Schritt angegebenen Zeichensatz zurückgegeben. Weitere Decodierungsvorgänge finden nicht statt.

Arbeiten mit Ressourcen-Bundles

Ein Ressourcen-Bundle ist eine Eigenschaftendatei mit der Erweiterung ***.properties**. Eine Eigenschaftendatei kann als eine Art Wörterbuch betrachtet werden, in dem Daten im Format

`Schlüssel = Wert` gespeichert sind. Jede Zeile in einer Eigenschaftendatei enthält eine Zuordnung des Typs `Schlüssel = Wert`. Die Hauptfunktionalität eines Ressourcen-Bundle besteht in der Rückgabe eines Wertes anhand seines Schlüssels.

Ressourcen-Bundles befinden sich auf dem Probe-Computer: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles**. Sie werden wie jede andere Konfigurationsdatei vom UCMDB-Server heruntergeladen. Ressourcen-Bundles können im Ressourcen-Fenster bearbeitet, hinzugefügt oder entfernt werden. Weitere Informationen finden Sie unter "Ausschnitt "Konfigurationsdateien"" im HP Universal CMDB – Handbuch zur Datenflussverwaltung.



Wenn ein Ziel ermittelt wird, muss die Datenflussverwaltung normalerweise Text von der Befehlsausgabe oder vom Dateiinhalt analysieren. Diese Analyse basiert häufig auf einem regulären Ausdruck. Unterschiedliche Sprachen erfordern die Verwendung unterschiedlicher regulärer Ausdrücke für die Analyse. Um Code zu schreiben, der für alle Sprachen gilt, müssen sämtliche sprachenspezifischen Daten in die Ressourcen-Bundles extrahiert werden. Für jede Sprache gibt es ein Ressourcen-Bundle. (Obwohl es möglich ist, dass ein Ressourcen-Bundle Daten für verschiedene Sprachen enthält, sind in der Datenflussverwaltung immer nur die Daten für eine Sprache in einem Ressourcen-Bundle gespeichert.)

Das Jython-Skript enthält keine hartcodierten, sprachenspezifischen Daten (z. B. sprachenspezifische reguläre Ausdrücke). Das Skript bestimmt die Sprache des Remote-Systems, lädt das entsprechende Ressourcen-Bundle und ruft alle sprachenspezifischen Daten nach einem bestimmten Schlüssel ab.

In der Datenflussverwaltung haben Ressourcen-Bundles ein bestimmtes Namensformat: `<Basisname>_<Sprachen_ID>.properties`, z. B. `langNetwork_spa.properties`. (Das Standard-Ressourcen-Bundle hat das folgende Format: `<Basisname>.properties`, z. B. `langNetwork.properties`.)

Das Format `Basisname` spiegelt den beabsichtigten Zweck dieses Bundle wider. So bedeutet beispielsweise **langMsCluster**, dass das Ressourcen-Bundle sprachenspezifische Ressourcen enthält, die von MS Cluster-Jobs verwendet werden.

Das Format `Sprachen_ID` ist eine aus 3 Buchstaben bestehende Abkürzung, die die Sprache angibt. Beispielsweise steht die Abkürzung `rus` für die russische Sprache und die Abkürzung `ger`

für die deutsche Sprache. Diese Sprachen-ID ist in der Deklaration des Objekts `Language` enthalten.

API-Referenz

Dieser Abschnitt umfasst Folgendes:

Die Sprachenklasse

Diese Klasse enthält Informationen über die Sprache, wie beispielsweise das Postfix des Ressourcen-Bundle, die mögliche Codierung usw.

Felder

Name	Beschreibung
locale	Java-Objekt, das das Gebietsschema repräsentiert.
bundlePostfix	Postfix des Ressourcen-Bundle. Dieses Postfix wird in Dateinamen von Ressourcen-Bundles zur Identifizierung der Sprache verwendet. Das Bundle **langNetwork_ger.properties** enthält beispielsweise das Postfix **ger**.
charsets	Die zur Codierung der Sprache verwendeten Zeichensätze. Jede Sprache kann mehrere Zeichensätze haben. Für die russische Sprache werden normalerweise die Codierungen `Cp866` und `Windows-1251` verwendet.
wmiCodes	Die Liste der WMI-Codes, die vom Microsoft Windows-Betriebssystem zur Identifizierung der Sprache verwendet werden. Die möglichen Codes sind unter http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (im Abschnitt "OSLanguage") aufgelistet. Eine der Methoden zur Identifizierung der Betriebssystemsprache besteht darin, die WMI-Klasse für die Eigenschaft `OSLanguage` abzufragen.
codepage	Die mit einer bestimmten Sprache verwendete Codeseite. Für russische Computer wird beispielsweise `866` und für englische Computer `437` verwendet. Eine der Methoden zur Identifizierung der Betriebssystemsprache besteht darin, die Standardcodeseite abzufragen (z. B. mit dem Befehl `chcp`).

Die executeCommandAndDecode-Methode

Diese Methode ist für Jython-Skripts zur Business-Logik vorgesehen. Sie enthält den Decodierungsvorgang und gibt eine decodierte Befehlsausgabe zurück.

Argumente

Name	Beschreibung
cmd	Der eigentliche Befehl, der ausgeführt wird.
keyword	Das für den Decodierungsvorgang zu verwendende Schlüsselwort.
framework	Das Framework-Objekt, das an jedes ausführbare Jython-Skript in der Datenflussverwaltung übergeben wird.
timeout	Der Befehl für eine Zeitüberschreitung.
waitForTimeout	Gibt an, ob der Client nach Ablauf der Zeitüberschreitung warten soll.
useSudo	Gibt an, ob `sudo` verwendet werden soll (nur relevant für UNIX-Clients).
language	Ermöglicht die direkte Angabe der Sprache (anstelle der automatischen Spracherkennung).

Die getCharsetName-Methode

Diese Methode gibt den Namen des zuletzt verwendeten Zeichensatzes zurück.

Die useCharset-Methode

Diese Methode legt den Zeichensatz bei der Instanz `ShellUtils` fest, die diesen Zeichensatz für die erste Datendecodierung verwendet.

Argumente

Name	Beschreibung
charsetName	Der Name des Zeichensatzes, z. B. `windows-1251` oder `UTF-8`.

Siehe auch "Die getCharsetName-Methode" unten.

Die getLanguageBundle-Methode

Diese Methode sollte zum Abrufen des korrekten Ressourcen-Bundle verwendet werden. Sie ersetzt die folgende API:

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

Argumente

Name	Beschreibung
baseName	Der Name des Bundle ohne Sprachsuffix, z. B. `langNetwork`.
language	Das Sprachobjekt. Hier sollte `ShellUtils.osLanguage` übergeben werden.

Name	Beschreibung
framework	Das Framework, das gemeinsame Objekt, das an jedes ausführbare Jython-Skript in der Datenflussverwaltung übergeben wird.

Das osLanguage-Feld

Dieses Feld enthält ein Objekt, das die Sprache repräsentiert.

Arbeiten mit Discovery Analyzer

Das Discovery Analyzer-Werkzeug ist zum Debuggen beim Entwickeln von Packages, Skripts und anderen Inhalten vorgesehen. Das Werkzeug führt einen Job für ein Remote-Ziel aus und gibt Protokolle zurück, die Informationen, Warn- und Fehlermeldungen sowie Ergebnisse zu Discovery-CIs enthalten.

Beachten Sie jedoch, dass die Ergebnisse nicht immer an die Benutzeroberfläche gemeldet werden. Dies liegt daran, dass die Ergebnisse auf zwei Arten gemeldet werden, von denen nur eine unterstützt wird. Außerdem wird das Kommunikationsprotokoll nicht von Eclipse unterstützt.

Wenn Sie das Werkzeug über Eclipse ausführen, muss in der Datei **DataFlowProbe.properties** (**C:\hp\UCMDB\DataFlowProbe\conf\DataFlowProbe.properties**) der folgende Parameter auf **true** gesetzt sein:

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

Weitere Informationen finden Sie unter "Ausführen von Discovery Analyzer über Eclipse" auf Seite 66.

In allen anderen Fällen (wenn das Werkzeug während der Ausführung der Probe oder über die Datei **cmd** ausgeführt wird), muss dieses Kennzeichen auf **false** gesetzt werden:

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

Aufgaben und Datensätze

Eine Aufgabendatei enthält die Daten in Bezug auf eine auszuführende Aufgabe. Die Aufgabe besteht aus Informationen wie dem Namen des Jobs und den erforderlichen Parametern, die das Trigger-CI, wie beispielsweise die Adresse des Remote-Ziels, definieren.

Eine Datensatzdatei enthält Aufgabeninformationen sowie die Ergebnisse einer bestimmten Ausführung, d. h. die detaillierte Kommunikation (einschließlich einer Antwort) zwischen der Probe oder dem Discovery Analyzer (je nachdem, welches Modul die Aufgabe ausgeführt hat) und dem Remote-Ziel.

Eine durch eine Aufgabendatei definierte Aufgabe kann für ein Remote-Ziel ausgeführt werden. Eine Aufgabe, die durch eine Datensatzdatei (die zusätzliche Daten in Bezug auf eine bestimmte Ausführung enthält) definiert ist, kann hingegen ausgeführt und wiedergegeben werden (d. h., die in der Datensatzdatei dokumentierte Ausführung kann reproduziert werden).

Protokolle

Protokolle liefern Informationen über die letzte Ausführung. Dabei wird zwischen folgenden Protokollen unterschieden:

- **General Log**. Das allgemeine Protokoll enthält alle informativen Daten, Fehler und Warnungen, die während der Ausführung erfasst wurden.
- **Communication Log**. Das Kommunikationsprotokoll enthält die detaillierte Kommunikation zwischen dem Discovery Analyzer und dem Remote-Ziel (einschließlich seiner Antwort). Nach der Ausführung kann das Protokoll als Datensatzdatei gespeichert werden.
- **Results Log**. Das Ergebnisprotokoll zeigt eine Liste der Discovery-CIs an. Die Darstellungszeit jedes CI hängt vom Aufbau der Adapter und Skripts ab.

Sie können alle Protokolle zusammen oder jedes Protokoll einzeln speichern. Wenn Sie alle Protokolle speichern, werden sie zusammen unter einem Namen gespeichert.

Wenn Sie eine Datensatzdatei wiedergeben, werden die gleichen Daten im Kommunikationsprotokoll angezeigt. Der einzige Unterschied besteht in der Ausführungszeit.

Einschränkung: Das Kommunikations- und das Ergebnisprotokoll sind nicht verfügbar, wenn Sie Discovery Analyzer über Eclipse ausführen.

Dieser Abschnitt umfasst folgende Schritte:

1. ## Voraussetzungen

   - Die Probe muss installiert sein. (Discovery Analyzer wird bei der Probe-Installation installiert und nutzt Ressourcen gemeinsam mit der Probe.)

- Die Probe muss nicht ausgeführt werden, wenn Sie mit Discovery Analyzer arbeiten.

  Wurde die Probe jedoch bereits für einen UCMDB -Server ausgeführt, wurden auch schon alle erforderlichen Dateien zum Dateisystem heruntergeladen. Wurde die Probe nicht ausgeführt, können Sie die von Discovery Analyzer benötigten Ressourcen über das Menü **Einstellungen** hochladen. Weitere Informationen finden Sie unter "Import Settings" auf Seite 65.

- Der CMDB-Server muss nicht installiert sein.

2. ## Zugreifen auf Discovery Analyzer

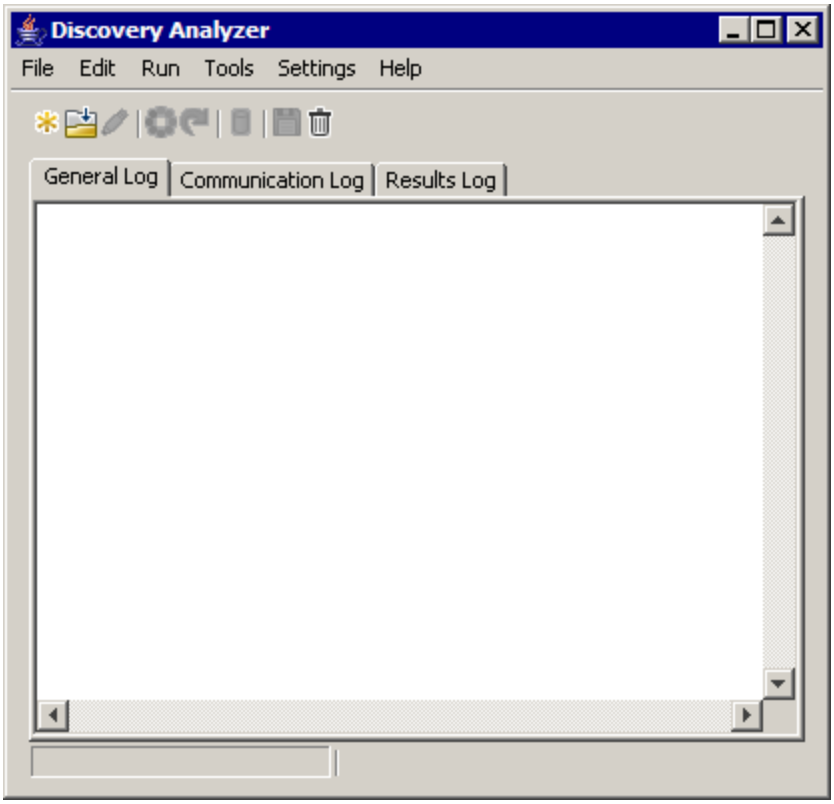
Für den Zugriff auf Discovery Analyzer gibt es die folgenden Möglichkeiten:

- Über Eclipse.

  Die Probe-Installation verfügt standardmäßig über einen Eclipse-Arbeitsbereich, der sich unter **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace** befindet. Dieser Arbeitsbereich enthält ein Jython-Skript zum Starten von Discovery Analyzer (**startDiscoveryAnalyzerScript.py**) sowie einen Link zu allen Skripts der Datenflussverwaltung. Wenn Sie das Werkzeug auf diese Weise starten, können Sie Haltepunkte zu Debugging-Zwecken in den Jython-Skripts festlegen.

- Direkt durch Doppelklicken auf die Datei im folgenden Ordner: **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzer.cmd**. Weitere Informationen finden Sie im folgenden Abschnitt.

Das Discovery Analyzer-Fenster wird geöffnet:



3. **Definieren einer Aufgabe**

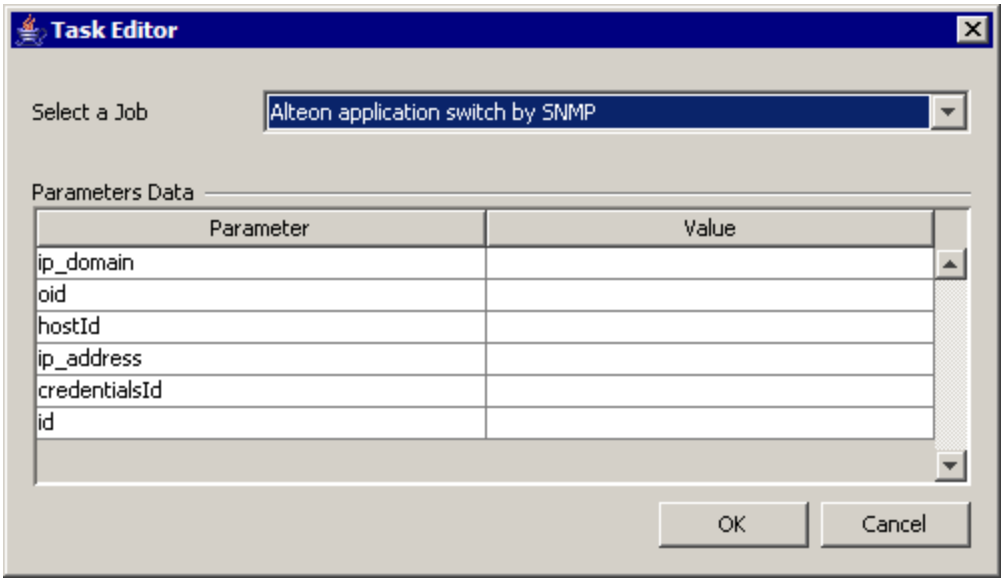
   Sie definieren eine Aufgabe mittels einer der folgenden Methoden:

   - Durch Definieren einer neuen Aufgabe. Weitere Informationen finden Sie unter "Definieren einer neuen Aufgabe" oben.
   - Durch Importieren einer Aufgabe aus einer Datensatzdatei. Weitere Informationen finden Sie unter "Abrufen eines Datensatzes" auf der nächsten Seite.
   - Durch Importieren einer gespeicherten Aufgabe aus einer Aufgabendatei. Weitere Informationen finden Sie unter "Öffnen einer Aufgabendatei" auf der nächsten Seite.
   - Durch Abrufen eines Jobs aus der internen Datenbank der Probe. Weitere Informationen finden Sie unter "Importieren einer Aufgabe aus der Datenbank" auf der nächsten Seite.

4. **Definieren einer neuen Aufgabe**

   a. Öffnen Sie den Task Editor: Klicken Sie auf die Schaltfläche **New Task** .

   Der Task Editor zeigt eine Liste der momentan im Dateisystem gespeicherten Jobs an. Diese Liste wird jedes Mal aktualisiert, wenn die Probe Aufgaben vom Server empfängt oder Packages manuell über das Menü **Settings** bereitgestellt werden.



   b. Wählen Sie einen Job aus.

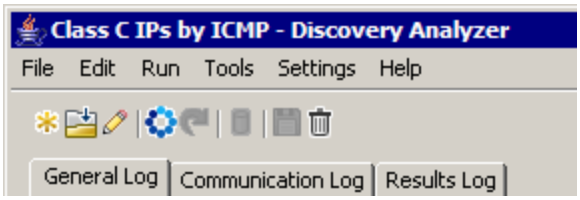
   c. Geben Sie Werte für alle Parameter ein.

   Bei den hier angezeigten Parametern handelt es sich um Adapterparameter der Datenflussverwaltung. Die Parameter werden auf der Registerkarte **Adapterdefinition** im Ausschnitt **Adapterparameter** angezeigt. Weitere Informationen finden Sie unter "Registerkarte "Adapterdefinition"" auf Seite 1 im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

   Alle Felder sind Pflichtfelder (es sei denn, das Skript eines Jobs verlangt, dass das Feld leer bleibt).

Für Parameter, die einen Eingabewert für die ID oder die Anmeldeinformationen-ID erfordern, können Sie nach dem Zufallsprinzip erstellte IDs verwenden: Klicken Sie mit der rechten Maustaste in das Wertefeld und wählen Sie **Generate random CMDB ID** oder **Credential Chooser** aus.

Die Aufgabe ist nun aktiv und der Name der offenen Aufgabe wird in der Titelleiste angezeigt:



d. Fahren Sie mit der Aufgabendefinition fort. Weitere Informationen finden Sie unter "Speichern der Aufgabe und der Protokolle" auf der nächsten Seite.

5. ## Abrufen eines Datensatzes

Sie können eine Aufgabe definieren, indem Sie eine Datensatzdatei öffnen, die die Daten in Bezug auf eine bestimmte Ausführung enthält. Wenn eine Aufgabe auf diese Weise definiert wird, können Sie die jeweilige Ausführung mithilfe der Wiedergabe-Option reproduzieren. (Wenn eine Aufgabe wiedergegeben wird, werden die Antworten nicht vom Remote-Ziel, sondern von den in der Datensatzdatei gespeicherten Daten empfangen.)

Wählen Sie **File > Open Record**. Suchen Sie den Ordner, in dem Sie den Datensatz gespeichert haben. Der Datensatz ist nun aktiv und der Name der offenen Aufgabe wird in der Titelleiste angezeigt.

Weitere Informationen zum Erfassen einer Datensatzdatei finden Sie unter "Aufzeichnen von Datenflussverwaltungscode" auf Seite 76.

6. ## Öffnen einer Aufgabendatei

Sie können eine Aufgabe von einer Aufgabendatei definieren: Wählen Sie **File > Open Task**.

7. ## Importieren einer Aufgabe aus der Datenbank

Falls die Probe bereits ausgeführt wurde und aktive Aufgaben in ihrer internen Datenbank hat, können Sie eine Aufgabe aus der Probe-Datenbank abrufen. Sie können die Parameterwerte verwenden, um die Aufgabe zu definieren.

a. Wählen Sie **File > Import Task from Probe Database**.

b. Wählen Sie im angezeigten Dialogfeld die Aufgabe aus, die ausgeführt werden soll, und klicken Sie auf **OK**.

c. Fahren Sie mit der Aufgabendefinition fort. Weitere Informationen finden Sie unter "Speichern der Aufgabe und der Protokolle" auf der nächsten Seite.

8. ## Bearbeiten einer Aufgabe

Nachdem eine Aufgabe definiert wurde, wird der Name der Aufgabe (bzw. der Datei) in der Titelleiste angezeigt. Nun können Sie die Datei bearbeiten.

a. Wählen Sie **Edit > Edit Task**.

b. Nehmen Sie die gewünschten Änderungen an der Aufgabe vor und klicken Sie auf **OK**.

9. ## Speichern der Aufgabe und der Protokolle

Sie können die Aufgabenparameter wie folgt speichern: Wählen Sie **File > Save Task**.

Folgende Optionen stehen erst nach der Ausführung einer Aufgabe zur Verfügung:

- Speichern eines Datensatzes der Aufgabe. Sie können die Aufgabenparameter und die Ergebnisse der Aufgabenausführung speichern: Wählen Sie **File > Save Record**.
- Speichern eines Protokolls der Aufgabe: Wählen Sie **File > Save General Log**.
- Speichern der Ergebnisse: Wählen Sie **File > Save Results**.

10. ## Ausführen der Aufgabe

Der nächste Schritt besteht in der Ausführung der erstellten Aufgabe.

a. Importieren Sie die Konfigurationsdatei mit den Anmeldeinformationen/Bereichen. Weitere Informationen finden Sie unter "Import Settings" oben.

b. Um die Aufgabe nur für ein Remote-Ziel auszuführen, klicken Sie auf die Schaltfläche **Run Task**.

Discovery Analyzer führt den Job aus und zeigt die Informationen in den drei Protokolldateien an: **General**, **Communication** und **Results**.

c. Sie können die Protokolldateien entweder zusammen oder einzeln speichern: Wählen Sie **File > Save General Log**, **Save Record**, **Save Results** oder **Save All Logs**. Weitere Informationen zu den Protokolldateien finden Sie unter "Protokolle" auf Seite 61.

d. Wenn eine Aufgabe von einer Datensatzdatei abgerufen wird, kann die in dieser Datei dokumentierte Ausführung durch Klicken auf die Schaltfläche **Playback** reproduziert werden. Es wird das gleiche Kommunikationsprotokoll angezeigt, die Ausführungszeit wird jedoch aktualisiert.

11. ## Senden eines Aufgabenergebnisses an den Server

Wenn die Ausführung einer Aufgabe mit Ergebnissen endet (d. h. auf der Registerkarte mit dem Ergebnisprotokoll wird eine Liste der Discovery-CIs angezeigt), können Sie die Ergebnisse an den UCMDB-Server senden. Dies ist beispielsweise nützlich, wenn Sie zuvor ein Skript getestet haben und der Server nicht verfügbar war.

Hinweis: Sie können die Ergebnisse nur an einen UCMDB-Server senden, der Aufgaben von der Probe empfängt, die auf demselben Computer installiert ist wie Discovery Analyzer.

12. ## Import Settings

Um Aufgaben auszuführen oder die Datensatzdatei wiederzugeben, müssen Sie die Datei **domainScopeDocument.bin** importieren. Während des Importvorgangs werden Sie zur Eingabe eines Kennworts aufgefordert.

a. Öffnen Sie einen Webbrowser und geben Sie den folgenden URL ein: **http://localhost:8080/jmx-console**. Eventuell müssen Sie sich mit einem Benutzernamen und einem Kennwort anmelden.

b. Klicken Sie auf **UCMDB:service=DiscoveryManager**, um die Seite **JMX MBEAN View** zu öffnen.

c. Suchen Sie den Vorgang **exportCredentialsAndRangesInformation**. Führen Sie folgende Aktion aus:

- Geben Sie die Kunden-ID ein (die Standard-ID lautet **1**).
- Geben Sie einen Namen für die exportierte Datei ein.
- Geben Sie das Kennwort ein.
- Setzen Sie **isEncrypted** auf **False**.

d. Klicken Sie auf **Invoke**, um die Datei **domainScopeDocument.bin** zu exportieren.

Wenn der Exportvorgang erfolgreich abgeschlossen wurde, befindet sich die Datei im folgenden Ordner: **C:\hp\UCMDB\UCMDBServer\conf\discovery\<customer_dir>**.

e. Kopieren Sie die Datei **domainScopeDocument.bin** in das Data Flow Probe-Dateisystem und importieren Sie sie, indem Sie folgende Optionen auswählen: **Settings > Import domainScopeDocument**.

> **Hinweis:** Während des Imports der Datei **domainScopeDocument** werden Sie zur Eingabe eines Kennworts aufgefordert. Diese Aufforderung wird auch nach jedem Neustart von Discovery Analyzer sowie vor Ausführung der ersten Aufgabe bzw. des ersten Datensatzes angezeigt.

13. ### Haltepunkte

Wenn Sie Discovery Analyzer über ein Python-Skript ausführen, können Sie Haltepunkte zum Skript hinzufügen.

14. ### Konfigurieren von Eclipse

Weitere Informationen zum Ausführen Ihrer Jython-Skripts im Debug-Modus finden Sie unter "Ausführen von Discovery Analyzer über Eclipse" oben.

Ausführen von Discovery Analyzer über Eclipse

Diese Aufgabe beschreibt, wie Sie Eclipse konfigurieren müssen, um Ihre Jython-Skripts im Debug-Modus auszuführen, damit Sie einen besseren Einblick in Job-Threads, Trigger-CIs und Ergebnisse erhalten.

Dieser Abschnitt umfasst folgende Schritte:

- "Konfigurieren des Discovery Analyzer-Arbeitsbereichs" auf Seite 69
- "Konfigurieren des Klassenpfads und des Interpreters" auf Seite 73
- "Ausführen von Discovery Analyzer" auf Seite 75

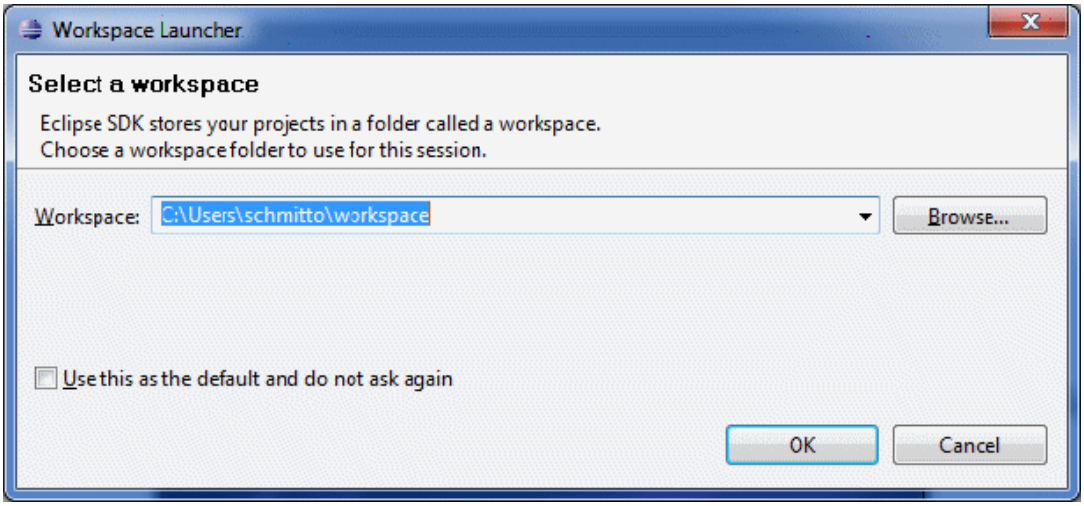
1. ## Voraussetzungen

   - Installieren Sie die neueste Eclipse-Version auf Ihrem Computer. Die Applikation steht unter www.eclipse.org zum Download bereit.
   - Stellen Sie sicher, dass die Data Flow Probe auf dem gleichen Computer installiert ist.
   - Vergewissern Sie sich, dass der Parameter **appilog.agent.local.discoveryAnalyzerFromEclipse** in der Datei **DataFlowProbe.properties** auf **true** gesetzt ist.

2. Entpacken und Starten von Eclipse

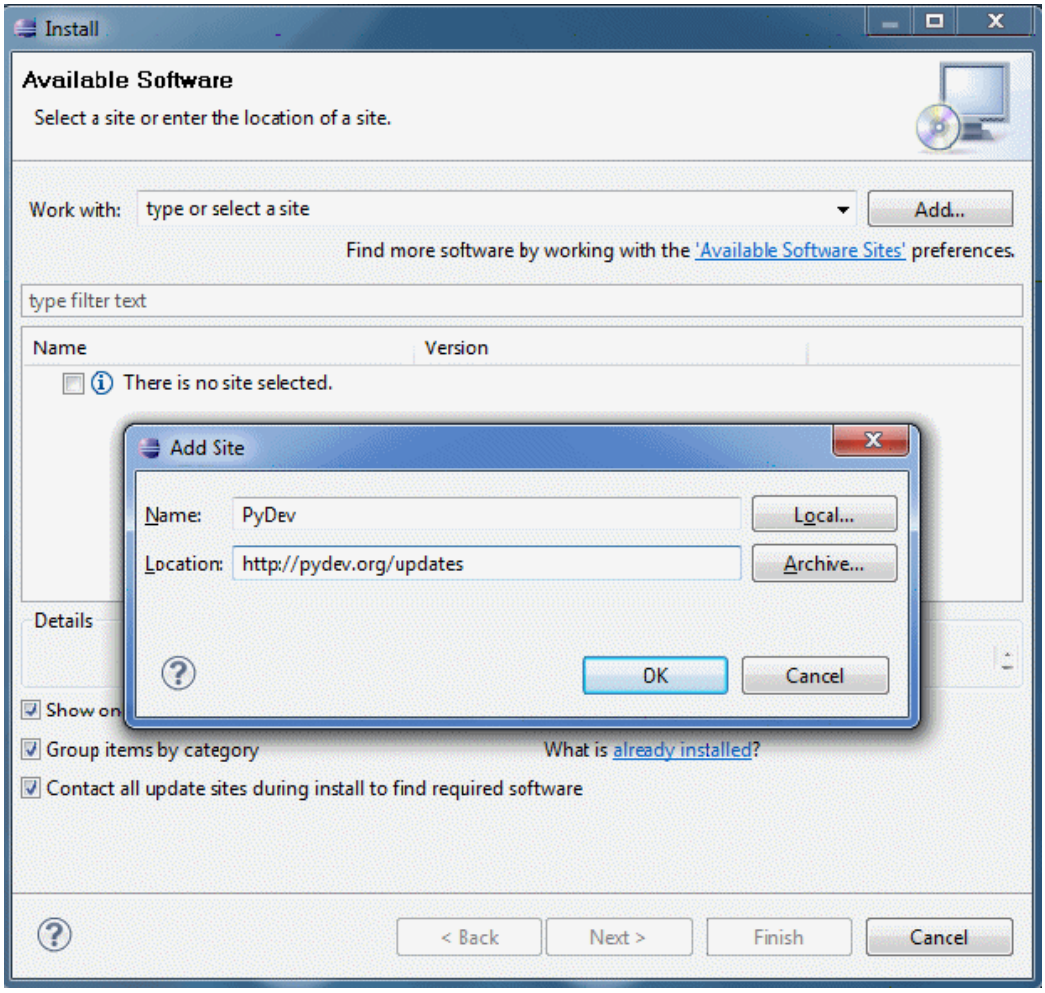
3. ## Konfigurieren des Standardarbeitsbereichs

   Konfigurieren Sie den Standardarbeitsbereich, in dem Eclipse alle Projekte und die zugehörigen Daten speichern soll.

4. ## Konfigurieren der PyDev-Erweiterungen

   a. Klicken Sie auf **Help > Install New Software**, wählen Sie **Add**, geben Sie einen Namen für das PyDev-Plugin ein und geben Sie im Feld **Location** den URL der Website an, von der **pydev** heruntergeladen werden kann: **http://pydev.org/updates**. Klicken Sie auf **OK**.

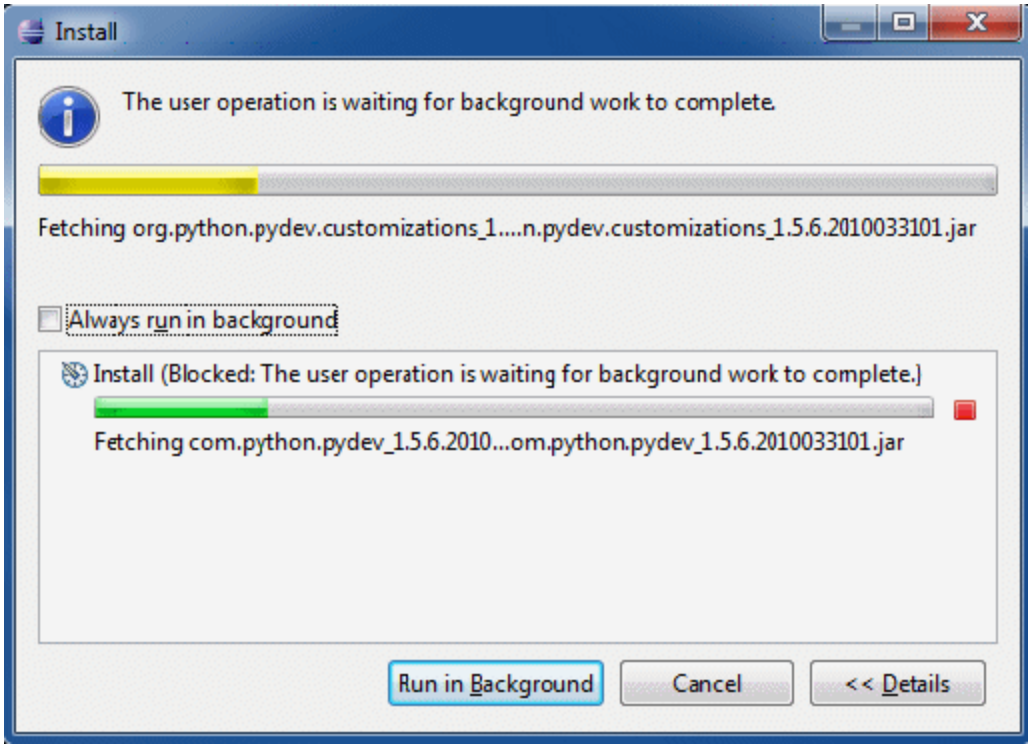


Hinweis: PyDev und PyDev-Erweiterungen sind jetzt in einem Plugin zusammengefasst, da PyDev-Erweiterungen Open Source sind. Weitere Informationen finden Sie im Internet unter http://pydev.org.

b. Wählen Sie im angezeigten Fenster die Option **Pydev** aus. Das zweite Plugin ist für aufgabenfokusierte Benutzeroberflächen vorgesehen. Klicken Sie auf **Next**, prüfen Sie die Angaben zur Installation und klicken Sie erneut auf **Next**.

c. Stimmen Sie der Lizenzvereinbarung zu und klicken Sie auf **Next**.

d. Pydev ist nun installiert. Wenn Sie gefragt werden, ob Sie unsignierte Inhalte installieren möchten, klicken Sie auf **OK**.

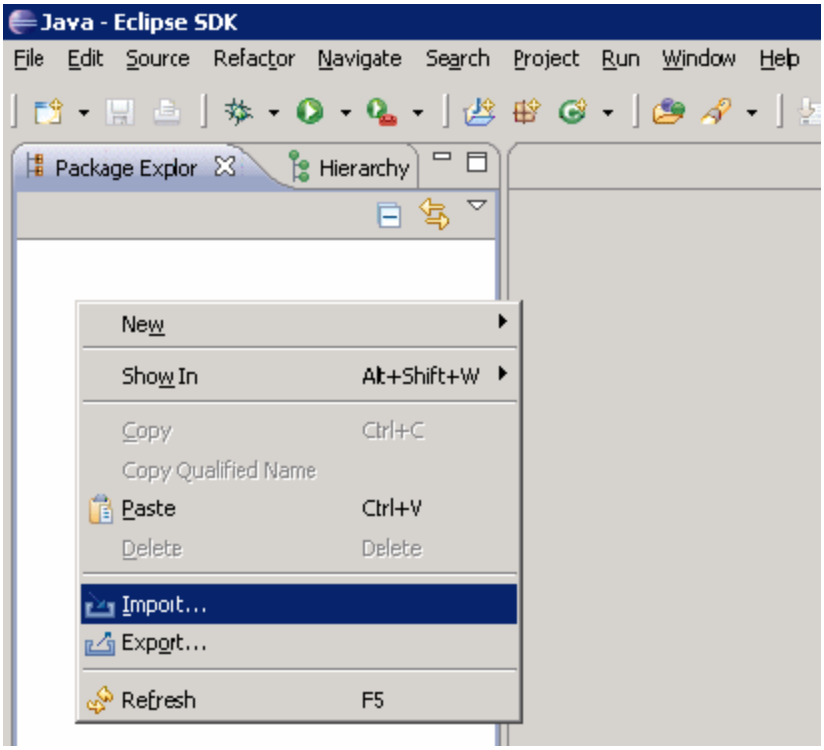


e. Starten Sie Eclipse neu.

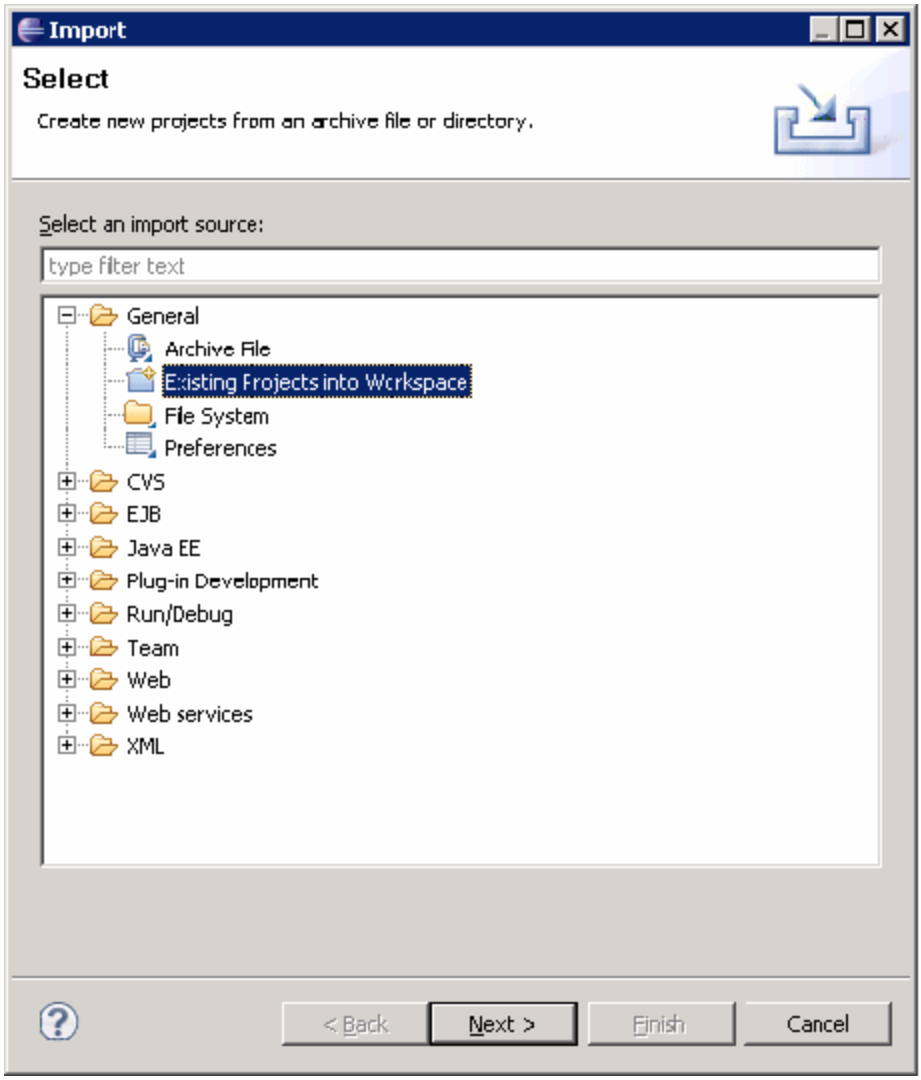
PyDev ist nun in Ihrer Eclipse-IDE installiert. Eclipse bietet Ihnen neue Blickwinkel und die IDE ist in der Lage, Python-Skripts zu interpretieren (Texthervorhebungen, zusätzliche Konfigurationsoptionen usw.).

5. Konfigurieren des Discovery Analyzer-Arbeitsbereichs

a. Importieren Sie die erforderlichen Dateien: Klicken Sie mit der rechten Maustaste in den weißen Bereich des Package Explorer. Klicken Sie dann auf **Import**, um den vorkonfigurierten Arbeitsbereich **discoveryAnalyzerWorkspace** zu importieren, der in der Probe-Installation enthalten ist.



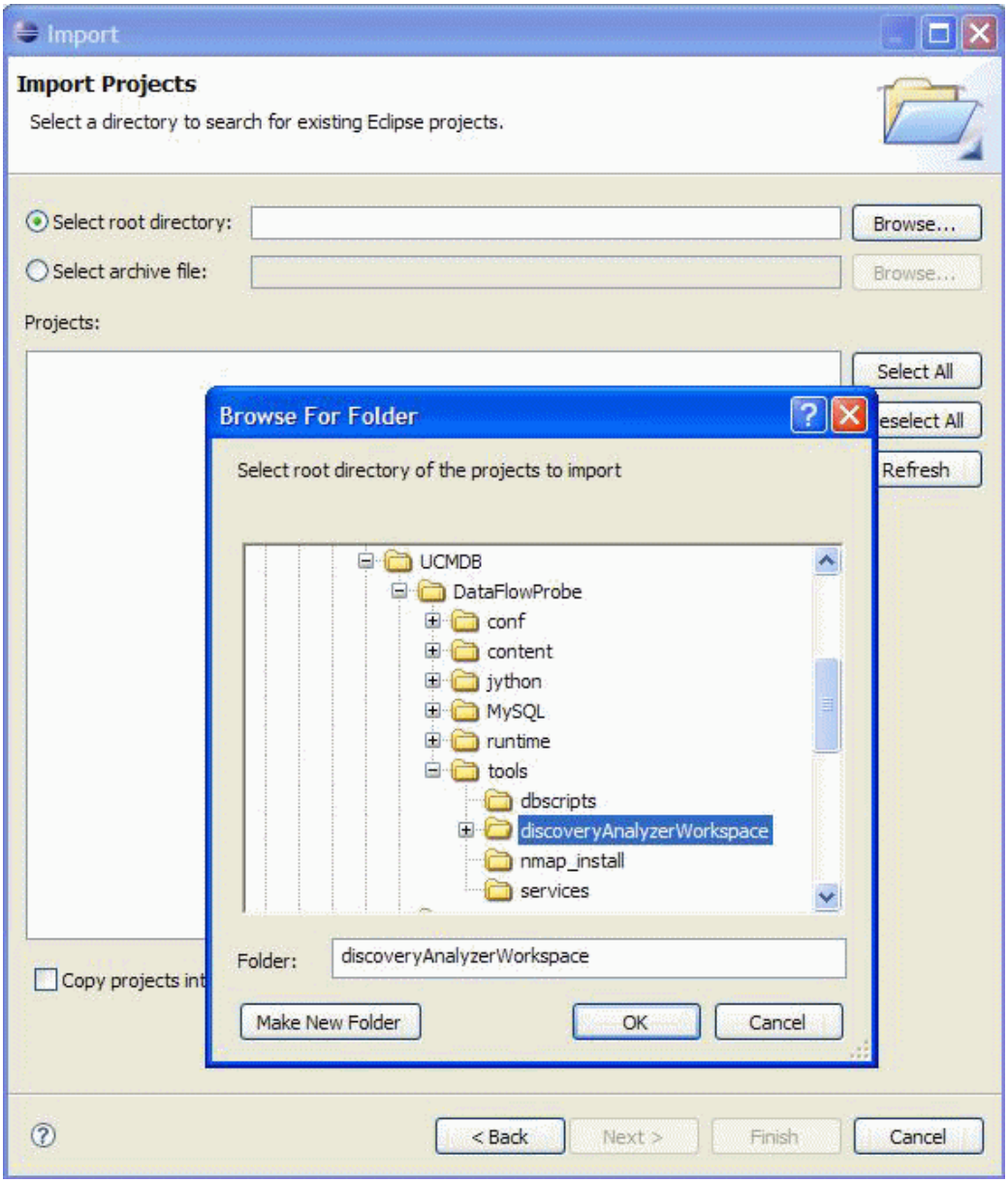
b. Klicken Sie unter **General** auf **Existing Projects into Workspace**, um das Projekt in den Eclipse-Arbeitsbereich zu importieren.



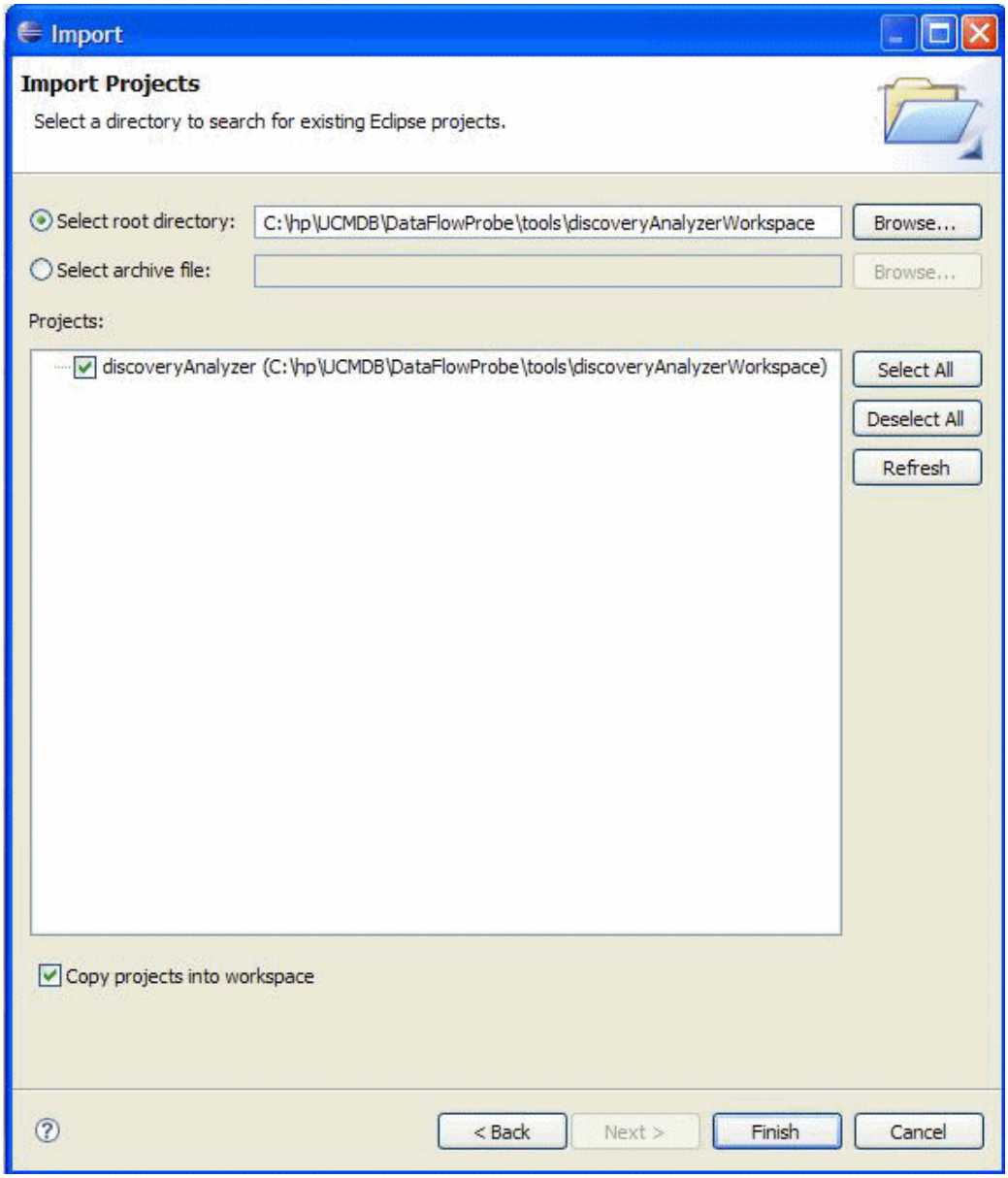
c. Wählen Sie unter **Select root directory** den Analyzer-Arbeitsbereich aus. Dieser befindet sich in der Regel unter

C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace.

d. Wählen Sie **Copy projects into workspace** aus, um eine Kopie des vorhandenen Arbeitsbereichs zu erstellen. Dieser Schritt ist wichtig, damit Sie bei einem Fehler den ursprünglichen **discoveryAnalyserWorkspace** erneut importieren können.



e. Klicken Sie auf **Finish**, um den Importvorgang zu starten.



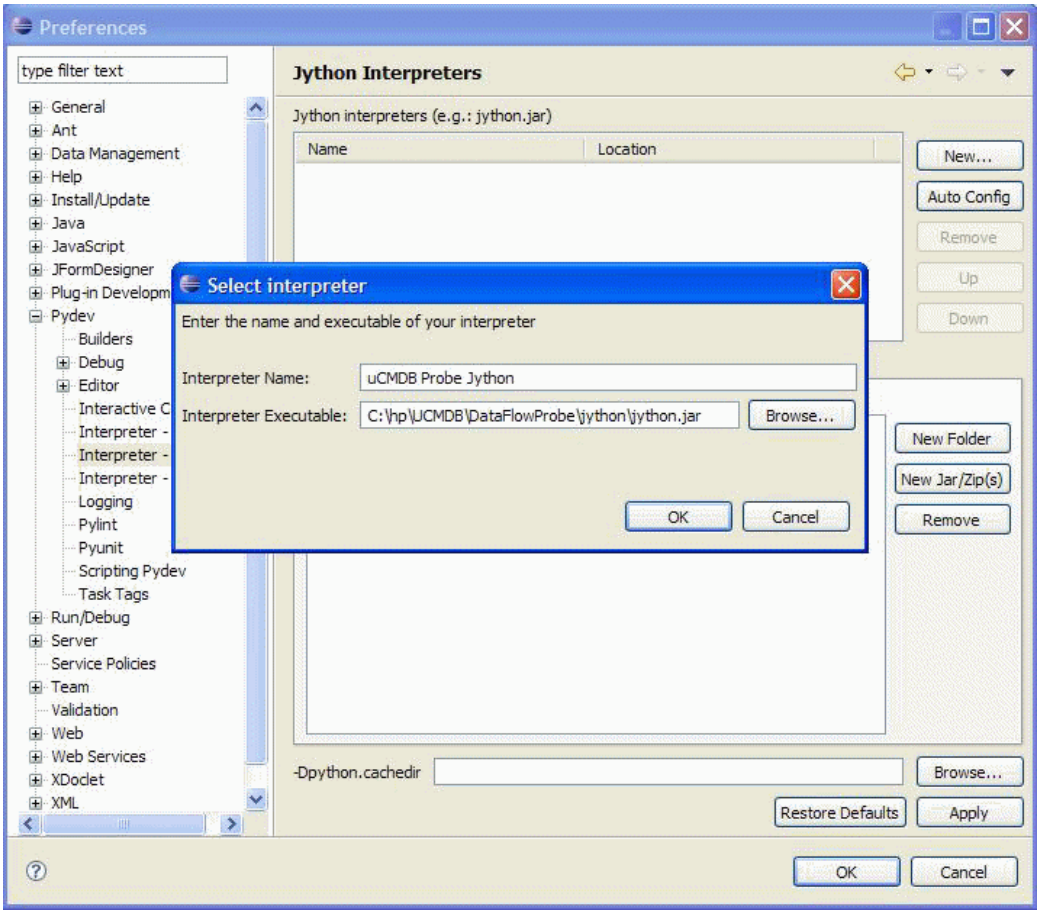
6. ## Konfigurieren des Klassenpfads und des Interpreters

   a. Klicken Sie mit der rechten Maustaste auf **discoveryAnalyzerWorkspace** und wählen Sie **Properties** aus, um die projektspezifischen Einstellungen anzuzeigen.

   b. Klicken Sie auf **Pydev > Interpreter/Grammar** und wählen Sie die Option **Please configure an interpreter in the related preferences before proceeding** aus.

   Durch diesen Schritt wird der gleiche Jython-Interpreter konfiguriert, der auch von der Probe verwendet wird, um sicherzustellen, dass die Skripts nicht von unterschiedlichen Jython-Versionen interpretiert werden.

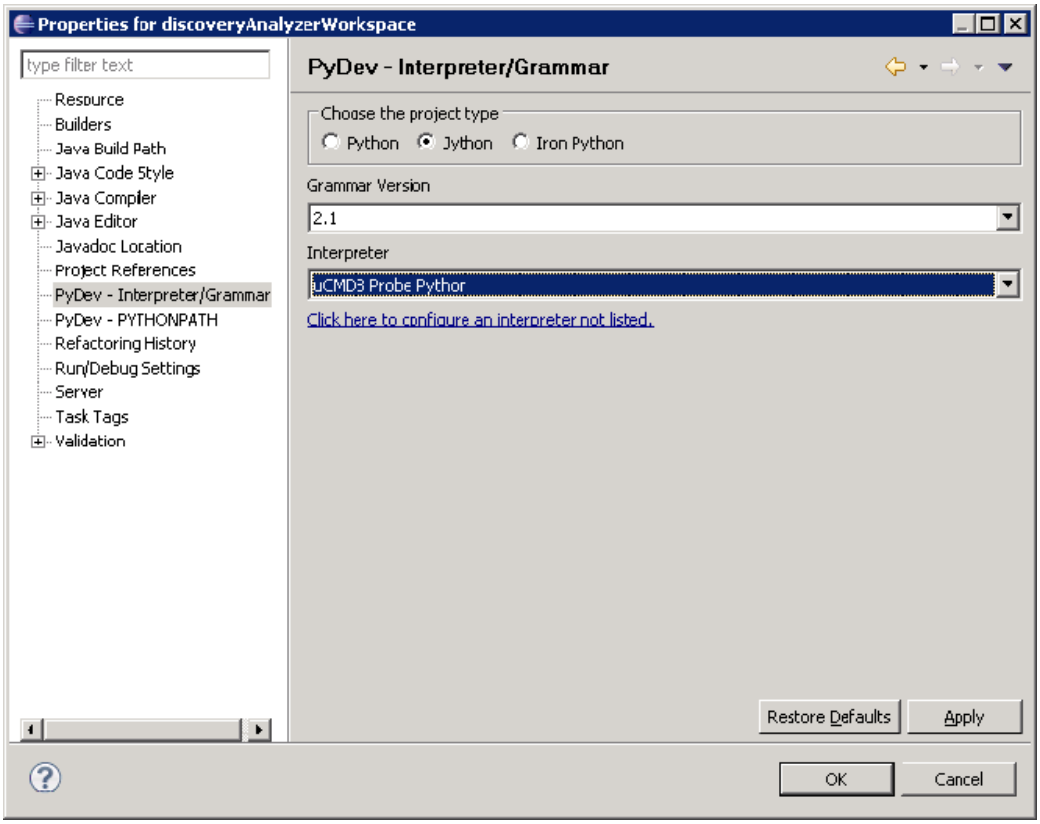
   c. Klicken Sie auf **New**, geben Sie einen Namen für den Interpreter ein und wählen Sie die Datei aus dem folgenden Ordner aus: **C:\hp\UCMDB\DataFlowProbe\jython\jython.jar**.



d. Klicken Sie auf **OK**. Wenn ein Fenster angezeigt wird, in dem Sie aufgefordert werden, die in Ihren Python-Systempfad zu importierenden Ordner auszuwählen, übernehmen Sie die Voreinstellungen (dies sollten die Ordner **C:\hp\UCMDB\DataFlowProbe\jython** und **C:\hp\UCMDB\DataFlowProbe\jython\lib** sein) und klicken Sie auf **OK**.

e. Klicken Sie auf **Apply** und anschließend auf **OK**.

f. Klicken Sie auf **Interpreter** und wählen Sie den soeben erstellten Interpreter aus.



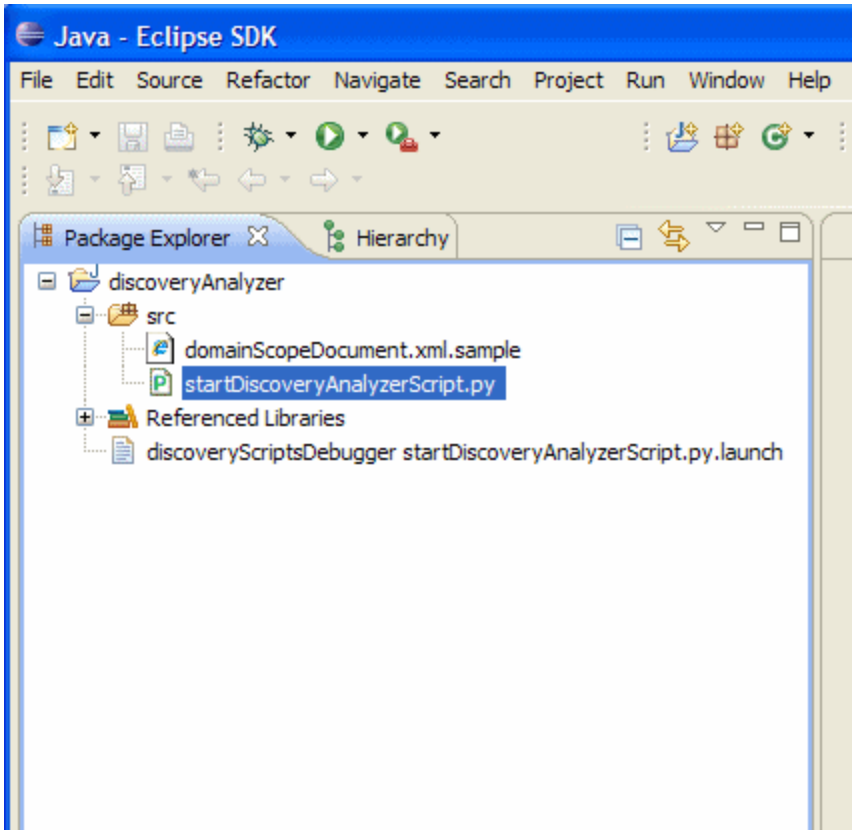
g. Klicken Sie auf **Apply** und anschließend auf **OK**.

Der Jython-Interpreter ist nun identisch mit dem von der Probe verwendeten Interpreter.

7. Ausführen von Discovery Analyzer

a. Fügen Sie in das zu debuggende Jython-Skript einen Haltepunkt ein.

b. Klicken Sie zum Starten von Discovery Analyzer im Projekt **discoveryAnalyzerWorkspace\src** auf **startDiscoveryAnalyzerScript.py**. Klicken Sie mit der rechten Maustaste auf die Datei und wählen Sie **Debug as > Jython run**.

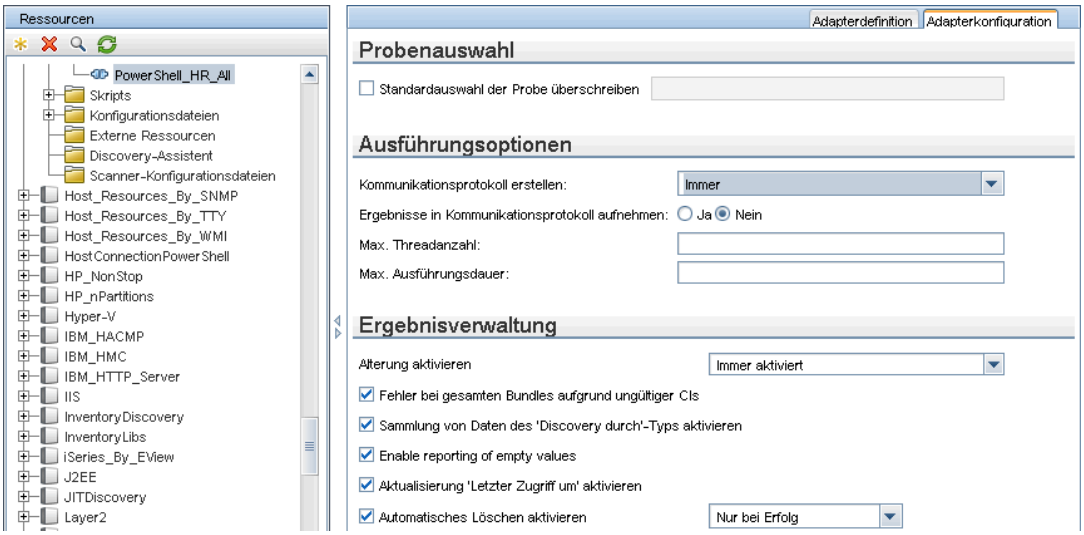


Aufzeichnen von Datenflussverwaltungscode

Beim Debuggen und Testen von Code kann es sehr nützlich sein, eine vollständige Ausführung, einschließlich aller Parameter, aufzuzeichnen. Diese Aufgabe beschreibt, wie Sie eine vollständige Ausführung mit allen relevanten Variablen aufzeichnen. Darüber hinaus können Sie auf diese Weise zusätzliche Debug-Informationen anzeigen, die normalerweise selbst auf der Debug-Ebene nicht in die Protokolldateien geschrieben werden.

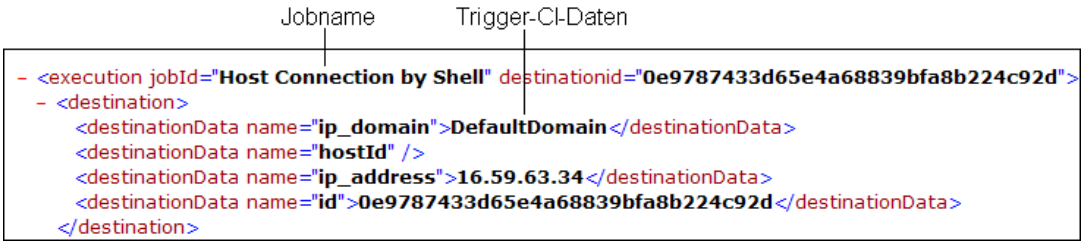
So zeichnen Sie Datenflussverwaltungscode auf:

1. Klicken Sie auf **Datenflussverwaltung > Discovery-Systemsteuerung**. Klicken Sie mit der rechten Maustaste auf den Job, dessen Ausführung protokolliert werden soll, und wählen Sie **Zum Adapter wechseln** aus, um die Applikation **Adapterverwaltung** zu öffnen.
2. Suchen Sie auf der Registerkarte **Adapterkonfiguration** den Ausschnitt **Ausführungsoptionen**.

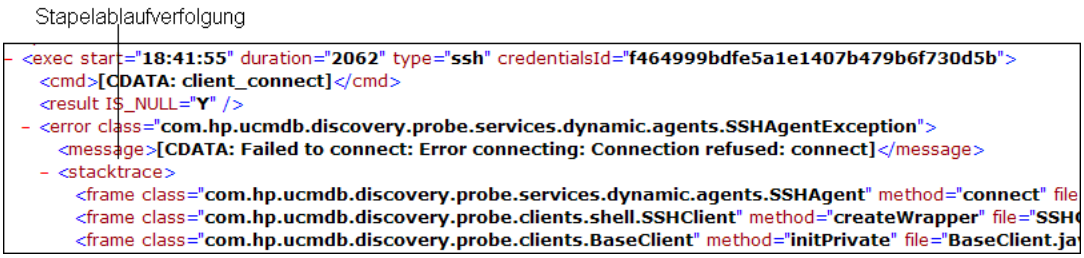


3. Ändern Sie die Einstellung des Felds **Kommunikationsprotokoll erstellen** auf **Immer**. Weitere Informationen zum Einstellen der Protokollierungsoptionen finden Sie unter "Ausschnitt "Ausführungsoptionen"" im HP Universal CMDB – Handbuch zur Datenflussverwaltung.

   Das folgende Beispiel zeigt die XML-Protokolldatei, die erstellt wird, wenn der Job `Host Connection by Shell` ausgeführt wird und das Feld **Kommunikationsprotokoll erstellen** auf **Immer** oder **Bei Fehler** gesetzt ist:



   Das folgende Beispiel zeigt die Meldung und die Parameter der Stapelablaufverfolgung:



Jython-Bibliotheken und Dienstprogramme

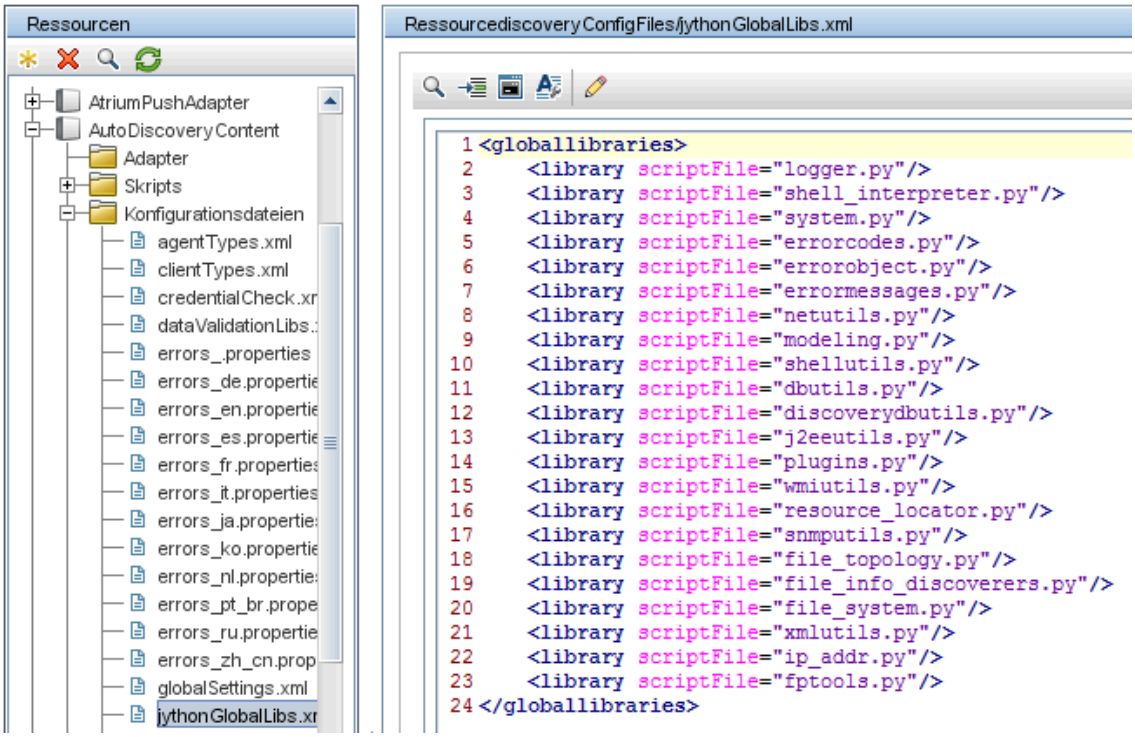
In Adaptern werden häufig mehrere Dienstprogrammskripts verwendet. Diese Skripts sind Teil des `AutoDiscovery`-Package, das in folgendem Ordner gespeichert ist: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts**. In diesem Ordner befinden sich auch die anderen Skripts, die zur Probe heruntergeladen wurden.

Hinweis: Der Ordner `discoveryScript` wird dynamisch erstellt, wenn die Probe zu arbeiten beginnt.

Um eines der Dienstprogrammskripts zu verwenden, fügen Sie die folgende Importzeile zum Importabschnitt des Skripts hinzu:

```
import <Skriptname>
```

Die AutoDiscovery Python-Bibliothek enthält Jython-Dienstprogrammskripts. Diese Bibliotheksskripts werden als externe Bibliothek der Datenflussverwaltung betrachtet. Sie sind in der Datei `jythonGlobalLibs.xml` (im Ordner **Konfigurationsdateien**) definiert.



Standardmäßig wird jedes Skript, das in der Datei `jythonGlobalLibs.xml` gespeichert ist, beim Starten der Probe geladen. Es ist daher nicht erforderlich, die Skripts explizit in der Adapterdefinition zu verwenden.

Dieser Abschnitt umfasst die folgenden Themen:

logger.py

Das Skript **logger.py** enthält die Protokolldienstprogramme und Hilfefunktionen für die Meldung von Fehlern. Sie können die zugehörigen Debug-, Informations- und Fehler-APIs aufrufen, damit die entsprechenden Daten in die Protokolldateien aufgenommen werden. Protokollmeldungen werden

unter **C:\hp\UCMDB\DataFlowProbe\runtime\log** aufgezeichnet.

Meldungen werden entsprechend der Debug-Ebene, die in der Datei **C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties** für den PATTERNS_DEBUG-Appender definiert wurde, in die Protokolldatei eingetragen. (Die Standardebene ist `DEBUG`.) Weitere Informationen finden Sie unter "Fehlerschweregrade" auf Seite 84.

```
#####################################################################
################         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/probeMgr-
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
```

Die Informations- und Fehlermeldungen werden auch in der Konsole (Eingabeaufforderung) angezeigt.

Es gibt zwei API-Sätze:

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

Der erste Satz gibt die Verkettung aller Zeichenkettenargumente auf der entsprechenden Protokollebene aus, während der zweite Satz zusätzlich zur Verkettung auch die Stapelablaufverfolgung der zuletzt ausgelösten Ausnahme ausgibt, um mehr Informationen zu liefern. Beispiel:

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

modeling.py

Das Skript **modeling.py** enthält APIs zum Erstellen von Hosts, IPs, Prozess-CIs usw. Diese APIs ermöglichen die Erstellung gemeinsamer Objekte und verbessern die Lesbarkeit des Codes. Beispiel:

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

netutils.py

Die Bibliothek **netutils.py** wird zum Abrufen von Netzwerk- und TCP-Informationen verwendet, wie beispielsweise zum Abrufen von Betriebssystemnamen, zum Überprüfen der Gültigkeit einer MAC-Adresse oder IP-Adresse usw. Beispiel:

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

shellutils.py

Die Bibliothek **shellutils.py** stellt eine API zum Ausführen der Shell-Befehle und Abrufen des Endstatus eines ausgeführten Befehls zur Verfügung und ermöglicht auf Basis des Endstatus die Ausführung mehrerer Befehle. Die Bibliothek wird mit einem Shell-Client initialisiert und verwendet den Client zum Ausführen von Befehlen und Abrufen von Ergebnissen. Beispiel:

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

Kapitel 3

Fehlermeldungen

Dieses Kapitel umfasst folgende Themen:

Fehlermeldungen – Übersicht

Während der Discovery können viele Fehler, beispielsweise Verbindungsfehler, Hardwareprobleme, Ausnahmen, Zeitüberschreitungen usw., aufgedeckt werden. Die Datenflussverwaltung zeigt diese Fehler immer dann in der Discovery-Systemsteuerung (Basismodus und Erweiterter Modus) an, wenn der reguläre Discovery-Fluss nicht erfolgreich ist. Sie können von dem Trigger-CI, das das Problem verursacht hat, einen Drilldown durchführen, um die Fehlermeldung anzuzeigen.

Die Datenflussverwaltung unterscheidet zwischen Fehlern, die gelegentlich ignoriert werden können (z. B. ein nicht erreichbarer Host) und Fehlern, die behoben werden müssen (z. B. Probleme mit Anmeldeinformationen oder fehlende Konfigurations- oder DLL-Dateien). Jeder Fehler wird von der Datenflussverwaltung nur einmal gemeldet. Dies gilt auch dann, wenn der gleiche Fehler bei mehreren aufeinanderfolgenden Ausführungen auftritt. Auch Fehler, die nur einmal auftreten, werden gemeldet.

Beim Erstellen eines Package können Sie die entsprechenden Meldungen als Ressourcen zum Package hinzufügen. Während der Package-Bereitstellung werden auch die Meldungen am richtigen Standort bereitgestellt. Meldungen unterliegen bestimmten Konventionen. Siehe hierzu "Konventionen für das Schreiben von Fehlermeldungen" oben.

Die Datenflussverwaltung unterstützt mehrsprachige Fehlermeldungen. Sie können die Meldungen, die Sie schreiben, lokalisieren, damit sie in der Landessprache angezeigt werden.

Weitere Informationen zur Fehlersuche finden Sie unter "Ausschnitt "Discovery-Status"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Weitere Informationen zum Einstellen der Kommunikationsprotokolle finden Sie unter "Ausschnitt "Ausführungsoptionen"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Konventionen für das Schreiben von Fehlermeldungen

- Jeder Fehler wird durch einen Meldungscode und ein Array von Argumenten (**int, String[]**) identifiziert. Die Kombination aus einem Meldungscode und einem Array von Argumenten definiert einen bestimmten Fehler. Das Array der Parameter kann Null sein.

- Jeder Fehlercode wird einer **Kurzmeldung**, d. h. einer festen Zeichenkette, und einer **detaillierten Meldung**, d. h. einer Vorlagenzeichenkette mit null oder mehr Argumenten, zugeordnet. Die Zuordnung wird zwischen der Anzahl der Argumente in der Vorlage und der tatsächlichen Anzahl der Parameter angenommen.

Beispiel für Meldungscode:

10234 kann einen Fehler mit der folgenden Kurzmeldung kennzeichnen:

```
Connection Error
```

und mit der folgenden detaillierten Meldung:

```
Verbindung über {0}-Protokoll konnte wegen einer
Zeitüberschreitung von {1} ms nicht hergestellt werden
```

Dabei gilt:

{0} = Erstes Argument: Protokollname

{1} = Zweites Argument: Dauer der Zeitüberschreitung in Millisekunden

Dieser Abschnitt umfasst außerdem die folgenden Themen:

- "Inhalt der Eigenschaftendatei" oben
- "Eigenschaftendatei für Fehlermeldungen" oben
- "Benennungskonventionen für Gebietsschemata" oben
- "Fehlermeldungscodes" auf der nächsten Seite
- "Nicht klassifizierte Inhaltsfehler" auf der nächsten Seite
- "Änderungen in Framework" auf Seite 84

Inhalt der Eigenschaftendatei

Eine Eigenschaftendatei sollte zwei Schlüssel für jeden Meldungscode enthalten. Zum Beispiel für Fehler 45:

- **DDM_ERROR_MESSAGE_SHORT_45**. Kurze Fehlerbeschreibung.
- **DDM_ERROR_MESSAGE_LONG_45**. Lange Fehlerbeschreibung (kann Parameter enthalten, z. B. {**0**},{**1**}).

Eigenschaftendatei für Fehlermeldungen

Eine Eigenschaftendatei enthält eine Zuordnung zwischen einem Meldungscode und zwei Meldungen (Kurzmeldung und detaillierte Meldung).

Nachdem eine Eigenschaftendatei bereitgestellt wurde, werden ihre Daten mit vorhandenen Daten zusammengeführt, d. h., neue Meldungscodes werden hinzugefügt, während alte Meldungscodes überschrieben werden.

Infrastrukturbezogene Eigenschaftendateien sind Teil des **AutoDiscoveryInfra**-Package.

Benennungskonventionen für Gebietsschemata

- Für das Standardgebietsschema: **<Dateiname>.properties.errors**

- Für ein bestimmtes Gebietsschema: **<Dateiname>_xx.properties.errors**

  Dabei gilt: **xx** ist das Gebietsschema (z. B. **infraerr_fr.properties.errors** oder **infraerr_en_us.properties.errors**).

Fehlermeldungscodes

Standardmäßig sind die folgenden Fehlercodes in HP Universal CMDB enthalten. Sie können eigene Fehlermeldungen zu dieser Liste hinzufügen.

Fehlername	Fehlercode	Beschreibung
Intern	100-199	Meistens abgeleitet von Ausnahmen, die während der Ausführung eines Jython-Skripts ausgelöst wurden.
Verbindung	200-299	Fehler beim Herstellen einer Verbindung, kein Agent auf dem Zielcomputer, Ziel nicht erreichbar usw.
Bezieht sich auf Anmeldeinformationen	300-399	Zugriff verweigert, Verbindungsversuch wurde aufgrund fehlender Anmeldeinformationen blockiert
Zeitüberschreitung	400-499	Zeitüberschreitung bei Verbindungsaufbau/Befehlsausführung
Unerwartetes oder ungültiges Verhalten	500-599	Fehlende Konfigurationsdateien, unerwartete Unterbrechungen usw.
Informationsabruf	600-699	Fehlende Informationen auf Zielcomputern, Fehler beim Abfragen von Informationen beim Agenten usw.
Ressourcenbezogen	700-799	Fehler in Bezug auf zu wenig Speicher oder nicht ordnungsgemäß freigegebene Clients
Analyse	800-899	Bei der Analyse festgestellte Fehler
Codierung	900	Fehler bei der Eingabe, nicht unterstützte Codierung
SQL-bezogen	901-903, 924	Fehler, die von SQL-Operationen empfangen wurden
HTTP-bezogen	904-909	Fehler, die bei HTTP-Verbindungen generiert oder anhand von HTTP-Fehlercodes analysiert wurden.
Bestimmte Applikation	910-923	Fehler, die aufgrund von applikationsspezifischen Problemen gemeldet wurden, z. B. falsche LSOF-Version, Warteschlangenmanager wurden nicht gefunden usw.

Nicht klassifizierte Inhaltsfehler

Um alte Inhalte zu unterstützen, ohne eine Regression zu verursachen, werden Fehler des Meldungscodes 100 (d. h. nicht klassifizierte Skriptfehler) von der Applikation und den SDK-relevanten Methoden unterschiedlich behandelt.

Diese Fehler werden nicht nach ihrem Meldungscode gruppiert (d. h., sie werden nicht als Fehler des gleichen Typs betrachtet), sondern nach dem Inhalt der Meldung. Wenn also ein Skript einen

Fehler nach den alten, verworfenen Methoden meldet (mit einer Meldungszeichenkette und ohne Fehlercode), erhalten alle Meldungen den gleichen Fehlercode. In der Applikation oder den SDK-relevanten Methoden werden unterschiedliche Meldungen jedoch als unterschiedliche Fehler angezeigt.

Änderungen in Framework

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

Die folgenden Methoden wurden zur Schnittstelle hinzugefügt:

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

Die folgenden alten Methoden werden aus Gründen der Abwärtskompatibilität weiterhin unterstützt, aber als veraltet markiert:

- `void reportError(Zeichenfolgennachricht);`
- `void reportWarning (Zeichenfolgennachricht);`
- `void reportFatal (Zeichenfolgennachricht);`

Fehlerschweregrade

Wenn die Ausführung eines Adapters für ein Trigger-CI beendet ist, gibt der Adapter einen Status zurück. Wenn keine Fehler- oder Warnmeldung vorliegt, lautet der Status **Erfolg**.

Im Folgenden sind die Schweregrade von der höchsten zur niedrigsten Stufe aufgeführt:

Abbruchfehler

Dieser Schweregrad meldet schwerwiegende Fehler, wie beispielsweise ein Problem mit der Infrastruktur, fehlende DLL-Dateien oder Ausnahmen:

- Fehler beim Generieren der Aufgabe (Probe wurde nicht gefunden, Variablen wurden nicht gefunden usw.)
- Skript kann nicht ausgeführt werden
- Verarbeitung der Ergebnisse auf dem Server fehlgeschlagen und es werden keine Daten in die CMDB geschrieben

Fehler

Dieser Schweregrad meldet Probleme, die dazu führen, dass die Datenflussverwaltung keine Daten abrufen kann. Schauen Sie sich diese Fehler an, denn sie erfordern in der Regel ein Eingreifen seitens des Benutzers (z. B. Erhöhen des Werts für die Zeitüberschreitung, Ändern eines Bereichs oder Parameters, Hinzufügen von Anmeldeinformationen usw.).

- In den Fällen, in denen ein Eingreifen des Benutzers helfen kann, wird ein Fehler gemeldet. Hierzu zählen beispielsweise Probleme in Bezug auf Anmeldeinformationen oder das Netzwerk, die möglicherweise eine weitere Untersuchung erfordern. (Dies sind keine Discovery-Fehler, sondern Konfigurationsfehler.)

- Interne Fehler, normalerweise aufgrund eines unerwarteten Verhaltens vom Discovery-Computer oder von der Discovery-Applikation, z. B. fehlende Konfigurationsdateien usw.

Warnungen

Wenn eine Ausführung erfolgreich ist, aber leichte Probleme vorliegen, über die Sie Bescheid wissen sollten, verwendet die Datenflussverwaltung den Schweregrad **Warnung**. Sie sollten sich diese CIs anschauen, um festzustellen, ob Daten fehlen, bevor Sie mit einer ausführlicheren Debug-Sitzung beginnen. Eine **Warnung** kann auf einen nicht installierten Agenten bei einem Remote-Host hinweisen oder darüber informieren, dass ein Attribut aufgrund ungültiger Daten nicht richtig berechnet werden konnte.

- Fehlender Verbindungsagent (SNMP, WMI)
- Discovery wurde erfolgreich durchgeführt, aber es wurden nicht alle verfügbaren Informationen ermittelt

Kapitel 4

Entwickeln von allgemeinen Datenbankadaptern

Dieses Kapitel umfasst folgende Themen:

Allgemeiner Datenbankadapter – Übersicht

Die allgemeine Datenbankadapter-Plattform dient zum Erstellen von Adaptern, die sich in relationale Datenbankmanagementsysteme (RDBMS) integrieren lassen sowie TQL-Abfragen und Auffüllungsjobs für die Datenbank ausführen können. Der allgemeine Datenbankadapter unterstützt folgende RDBMS: Oracle, Microsoft SQL Server und MySQL.

Diese Version der Datenbankadapter-Implementierung basiert auf einer JPA (Java Persistence API), bei der die ORM-Bibliothek Hibernate als Persistenz-Provider zum Einsatz kommt.

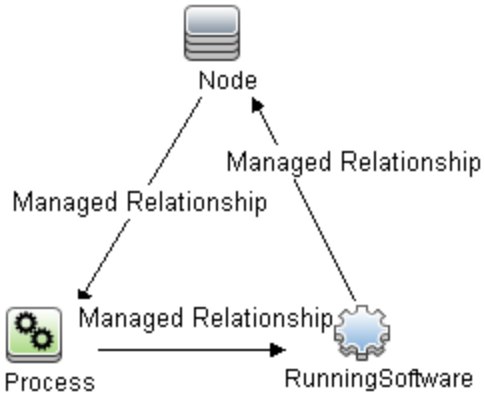
TQL-Abfragen für allgemeine Datenbankadapter

Für Auffüllungsjobs muss jedes erforderliche CI-Layout im Dialogfeld **Layouteinstellungen** von Modeling Studio geprüft werden. Weitere Informationen finden Sie unter "Dialogfeld "Abfrageknoteneigenschaften/Beziehungseigenschaften"" im *HP Universal CMDB – Modellierungshandbuch*. Es ist unbedingt zu beachten, dass ein CI möglicherweise ein Attribut erfordert, das identifiziert werden muss, und CIs ohne diese Attribute nicht zu UCMDB hinzugefügt werden können.

Die folgenden Einschränkungen gelten für TQL-Abfragen, die nur vom allgemeinen Datenbankadapter berechnet werden:

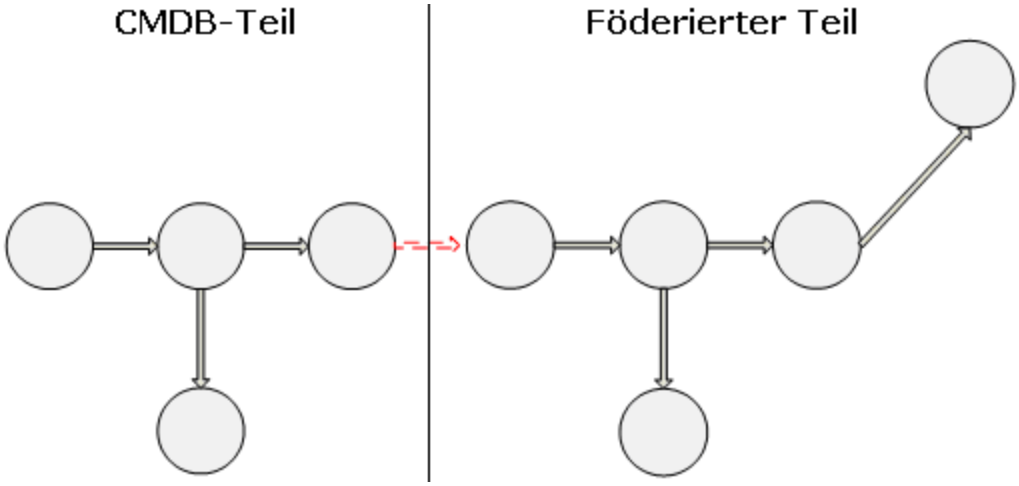
- Unterdiagramme werden nicht unterstützt.
- Verbundbeziehungen werden nicht unterstützt.
- Zyklen oder Zyklusteile werden nicht unterstützt.

  Die folgende TQL-Abfrage ist ein Beispiel für einen Zyklus:



- Funktionslayout wird nicht unterstützt.
- 0..0-Kardinalität wird nicht unterstützt.
- Die `Join`-Beziehung wird nicht unterstützt.
- Qualifiziererbedingungen werden nicht unterstützt.
- Um eine Verbindung zwischen zwei CIs herzustellen, muss in der externen Datenbankquelle

eine Beziehung in Form einer Tabelle oder eines Fremdschlüssels vorhanden sein.



Abstimmung

Die Abstimmung wird im Rahmen der TQL-Berechnung auf der Adapterseite durchgeführt. Damit eine Abstimmung erfolgen kann, wird die CMDB-Seite einer föderierten Entität mit der Bezeichnung "reconciliation-CIT" zugeordnet.

Zuordnung. Jedes Attribut in der CMDB wird einer Spalte in der Datenquelle zugeordnet.

Die Zuordnung erfolgt zwar direkt, aber es werden auch Transformationsfunktionen bei den Zuordnungsdaten unterstützt. Sie können neue Funktionen über Java-Code hinzufügen (z. B. Kleinschreibung, Großschreibung). Der Zweck dieser Funktionen besteht darin, Wertkonvertierungen zu ermöglichen (Werte, die in einem Format in der CMDB und in einem anderen Format in der föderierten Datenbank gespeichert sind).

Hinweis:

- Um eine Verbindung zwischen der CMDB und der externen Datenbankquelle herzustellen, muss eine entsprechende Zuweisung in der Datenbank vorhanden sein. Weitere Informationen finden Sie unter "Voraussetzungen" auf Seite 91.
- Die Abstimmung mit der CMDB-ID wird auch unterstützt.
- Die Abstimmung mit der globalen ID wird ebenfalls unterstützt.

Hibernate als JPA-Provider

Hibernate ist ein objektrelationales (OR) Zuordnungswerkzeug, das die Zuordnung von Java-Klassen zu Tabellen über mehrere Arten von relationalen Datenbanken (z. B. Oracle und Microsoft SQL Server) ermöglicht. Weitere Informationen finden Sie unter "Funktionale Einschränkungen" auf Seite 158.

In einer elementaren Zuordnung wird jede Java-Klasse einer einzelnen Tabelle zugeordnet. Eine erweiterte Zuordnung ermöglicht die Vererbungszuordnung (wie sie in der CMDB-Datenbank erfolgen kann).

Zu den weiteren unterstützten Funktionen zählen die Zuordnung einer Klasse zu mehreren Tabellen, die Unterstützung von Sammlungen sowie Zuweisungen vom Typ 1:1, 1:n und n:1. Weitere Informationen finden Sie unten unter "Zuweisungen" auf der nächsten Seite.

Für unsere Zwecke ist die Erstellung von Java-Klassen nicht erforderlich. Die Zuordnung wird von den CITs des CMDB-Klassenmodells zu den Datenbanktabellen definiert.

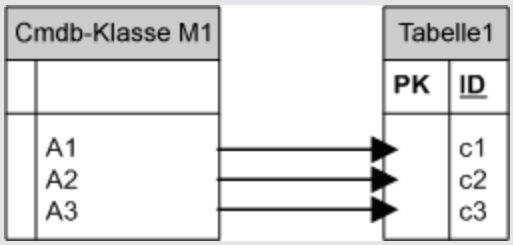
Dieser Abschnitt umfasst außerdem die folgenden Themen:

Beispiele für eine objektrelationale Zuordnung

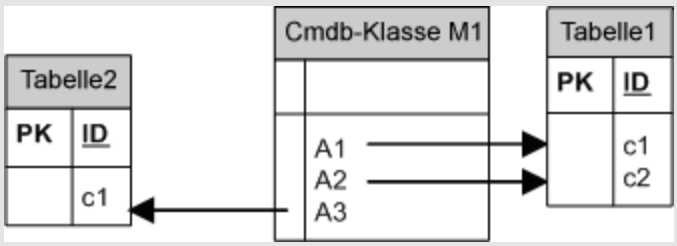
Die folgenden Beispiele beschreiben eine objektrelationale Zuordnung:

Beispiel für die Zuordnung einer CMDB-Klasse zu einer Datenbanktabelle:

Die Klasse `M1` mit den Attributen `A1`, `A2` und `A3` wird den Spalten `c1`, `c2` und `c3` von Tabelle `1` zugeordnet. Dies bedeutet, dass jede `M1`-Instanz eine übereinstimmende Zeile in Tabelle `1` hat.

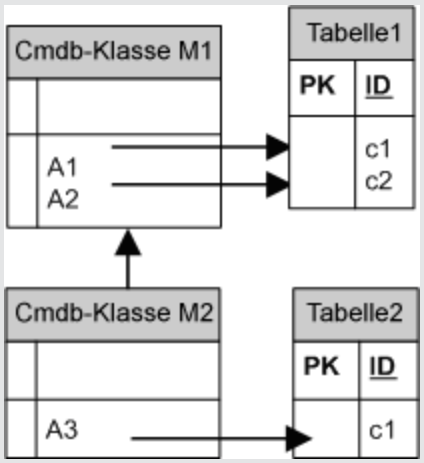


Beispiel für die Zuordnung einer CMDB-Klasse zu zwei Datenbanktabellen:



Beispiel für Vererbung:

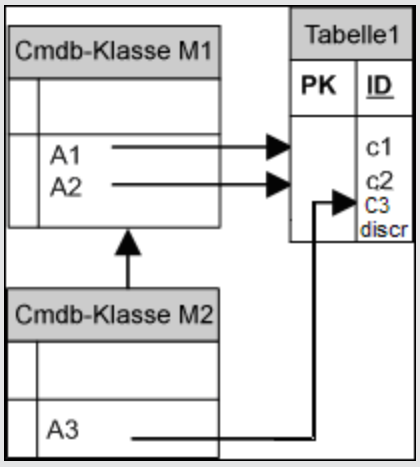
Dieser Fall wird in der CMDB verwendet, wenn jede Klasse ihre eigene Datenbanktabelle hat.



Beispiel für die Vererbung an eine einzelne Tabelle mit Diskriminator:

Eine vollständige Klassenhierarchie wird einer einzelnen Datenbanktabelle zugeordnet, deren Spalten aus einer Obermenge aller Attribute der zugeordneten Klassen bestehen. Die Tabelle enthält außerdem eine zusätzliche Spalte (`Diskriminator`), deren Wert angibt, welche Klasse diesem Eintrag zugeordnet werden soll.

Wenn Sie die Diskriminator-Funktionen nutzen, ist es nicht möglich, eine Klasse in der Hierarchie zu überspringen, d. h., da C3 von C2 und C2 von C1 erbt, können Sie nicht nur C1 und C3 definieren, sondern müssen alle drei Klassen definieren.



Zuweisungen

Es gibt drei Arten von Zuweisungen: 1:n, n:1 und n:n. Um eine Verbindung zwischen den verschiedenen Datenbankobjekten herzustellen, muss eine dieser Zuweisungen unter Verwendung einer Fremdschlüsselspalte (für den Fall 1:n) oder einer Zuordnungstabelle (für den Fall n:n) definiert werden.

Benutzerfreundlichkeit

Da das JPA-Schema sehr umfangreich ist, steht eine optimierte XML-Datei zur Verfügung, um Ihnen die Definitionen zu erleichtern.

Für die Verwendung dieser XML-Datei gilt der folgende Anwendungsfall: Föderierte Daten werden in eine föderierte Klasse modelliert. Diese Klasse hat n:1-Beziehungen zu einer nicht föderierten CMDB-Klasse. Darüber hinaus gibt es nur einen möglichen Beziehungstyp zwischen der föderierten und der nicht föderierten Klasse.

Vorbereitungen für die Adaptererstellung

Diese Aufgabe beschreibt die für die Erstellung eines Adapters erforderlichen Vorbereitungen.

Hinweis: Sie können Beispiele für den allgemeinen DB-Adapter in der UCMDB-API anzeigen. Speziell das Beispiel mit dem DDMi-Adapter enthält eine komplizierte **orm.xml**-Datei sowie Implementierungen für einige Plugin-Schnittstellen.

Diese Aufgabe umfasst folgende Schritte:

- "Voraussetzungen" oben
- "Erstellen eines CI-Typs" auf Seite 93
- "Erstellen einer Beziehung" auf Seite 93

1. ## Voraussetzungen

   Um sicherzustellen, dass Sie den Datenbankadapter mit Ihrer Datenbank verwenden können, prüfen Sie bitte Folgendes:

   - Die Abstimmungsklassen und ihre Attribute (auch als Multiknoten bezeichnet) sind in der Datenbank vorhanden. Wenn die Abstimmung beispielsweise auf Basis des Knotennamens ausgeführt wird, muss eine Tabelle vorhanden sein, die eine Spalte mit Knotennamen enthält. Wenn die Abstimmung entsprechend dem Knoten `cmdb_id` ausgeführt wird, muss eine Spalte mit CMDB-IDs vorhanden sein, die mit den CMDB-IDs der Knoten in der CMDB übereinstimmen. Weitere Informationen zur Abstimmung finden Sie unter "Abstimmung " auf Seite 88.

ID	NAME	IP-ADRESSE
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

   - Um zwei CITs einer Beziehung zuzuordnen, müssen Korrelationsdaten zwischen den CIT-Tabellen vorhanden sein. Die Korrelation kann entweder durch eine Fremdschlüsselspalte

oder durch eine Zuordnungstabelle hergestellt werden. Um beispielsweise eine Korrelation zwischen einem Knoten und einem Ticket herzustellen, muss in der Tickettabelle eine Spalte mit der Knoten-ID und in der Knotentabelle eine Spalte mit der Ticket-ID vorhanden sein oder es muss eine Zuordnungstabelle existieren, deren `Ende 1` die Knoten-ID und deren `Ende 2` die Ticket-ID ist. Weitere Informationen zu Korrelationsdaten finden Sie unter "Hibernate als JPA-Provider" auf Seite 88.

Die folgende Tabelle zeigt die Fremdschlüsselspalte `NODE_ID`:

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	Serieller Buscontroller	Intel 82801EB USB Universal Host Controller
3581	2	System	Intel 631xESB/6321ESB/3100 Chipset LPC
3581	3	Anzeige	ATI ES1000
3581	4	Peripheriegerät für Basissystem	HP ProLiant iLO 2 Legacy Support-Funktion

- Jeder CIT kann einer oder mehreren Tabellen zugeordnet werden. Um einen CIT mehreren Tabellen zuordnen zu können, müssen sowohl eine primäre Tabelle, deren Primärschlüssel in den anderen Tabellen enthalten ist, als auch eine eindeutige Wertespalte vorhanden sein.

  Beispiel: Ein Ticket wird zwei Tabellen zugeordnet: `Ticket1` und `Ticket2`. Die erste Tabelle hat die Spalten `c1` und `c2`. Die zweite Tabelle hat die Spalten `c3` und `c4`. Damit die Tabellen als eine Tabelle betrachtet werden, müssen sie den gleichen Primärschlüssel haben. Alternativ kann der Primärschlüssel der ersten Tabelle auch eine Spalte der zweiten Tabelle sein.

  Im folgenden Beispiel haben die Tabellen den gleichen Primärschlüssel mit der Bezeichnung `CARD_ID`:

CARD_ID	CARD_TYPE	CARD_NAME
1	Serieller Buscontroller	Intel 82801EB USB Universal Host Controller
2	System	Intel 631xESB/6321ESB/3100 Chipset LPC
3	Anzeige	ATI ES1000
4	Peripheriegerät für Basissystem	HP ProLiant iLO 2 Legacy Support-Funktion

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(Standard-USB-Hostcontroller)
3	Hewlett-Packard Company
4	(Standardsystemgeräte)
5	Hewlett-Packard Company

2. ## Erstellen eines CI-Typs

   In diesem Schritt erstellen Sie einen CIT, der für die Daten im RDBMS (externe Datenquelle) steht.

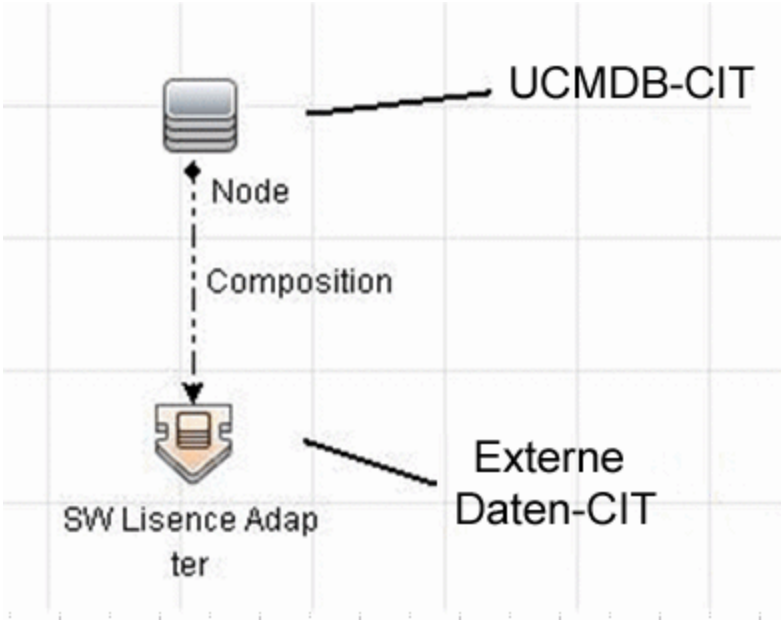
   a. Greifen Sie in UCMDB auf CIT Manager zu und erstellen Sie einen neuen CI-Typ. Weitere Informationen finden Sie unter "Erstellen eines CI-Typs" im *HP Universal CMDB – Modellierungshandbuch*.

   b. Fügen Sie die erforderlichen Attribute zum CIT hinzu, wie z. B. Zeitpunkt des letzten Zugriffs, Anbieter usw. Dies sind die Attribute, die der Adapter von der externen Datenquelle abruft und in den CMDB-Ansichten wiedergibt.

3. ## Erstellen einer Beziehung

   In diesem Schritt fügen Sie eine Beziehung zwischen dem UCMDB-CIT und dem neuen CIT hinzu, der die Daten der externen Datenquelle darstellt.

   Fügen Sie passende, gültige Beziehungen zum neuen CIT hinzu. Weitere Informationen finden Sie unter "Dialogfeld "Beziehung hinzufügen/entfernen"" im *HP Universal CMDB – Modellierungshandbuch*.

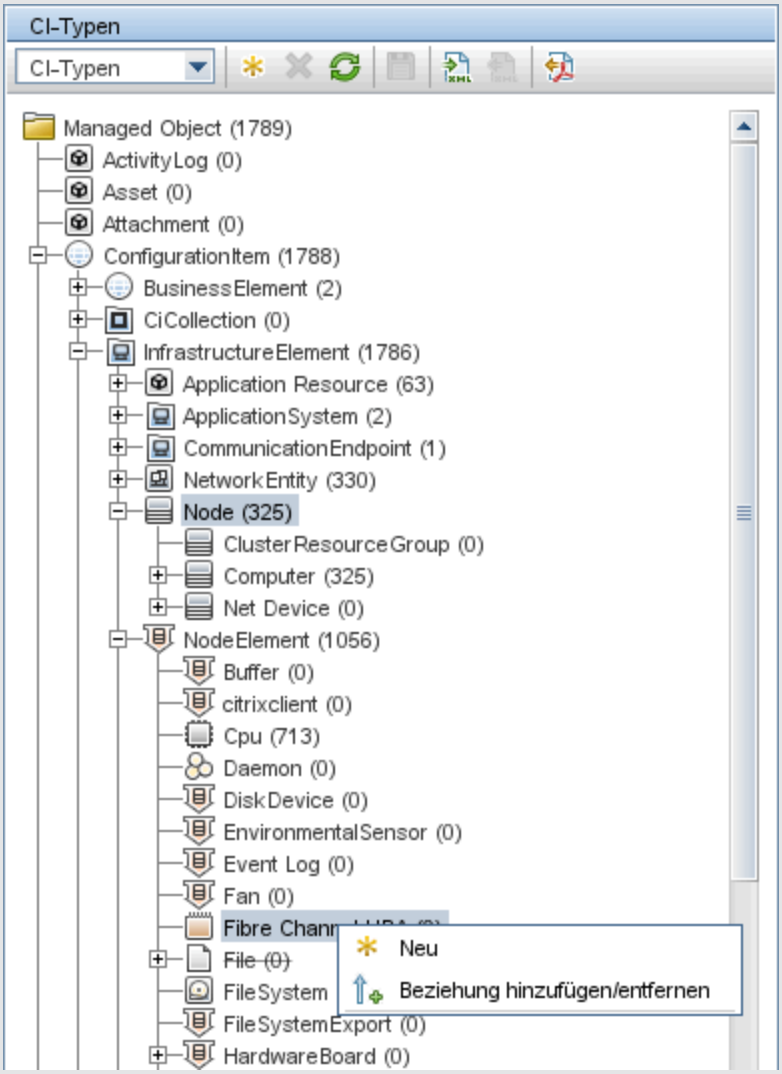


Hinweis: In diesem Stadium können Sie die föderierten Daten noch nicht sehen und die

externen Daten noch nicht auffüllen, da Sie noch keine Methode zum Einbinden der Daten definiert haben.

Beispiel für das Erstellen einer Containment-Beziehung:

a. Wählen Sie in CIT Manager die beiden CITs aus:



b. Erstellen Sie eine **Containment**-Beziehung zwischen den beiden CITs:

Vorbereiten des Adapter-Packages

In diesem Schritt suchen Sie das Package mit dem allgemeinen DB-Adapter und konfigurieren den Adapter.

1. Suchen Sie das Package **db-adapter.zip** im Ordner **C:\hp\UCMDB\UCMDBServer\content\adapters**
   .
2. Extrahieren Sie das Package in ein lokales temporäres Verzeichnis.
3. Bearbeiten Sie die XML-Datei des Adapters:
   - Öffnen Sie die Datei **discoveryPatterns\db_adapter.xml** in einem Texteditor.
   - Suchen Sie das Attribut **adapter id** und ersetzen Sie den Namen:

```
<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">
```

   Wenn der Adapter die Auffüllung unterstützt, sollte die folgende Funktion zum Element **<adapter-capabilities>** hinzugefügt werden:

```
<support-replicatioin-data>
    <source>
        <changes-source>
```

```
            </source>
        </support-replicatioin-data>
```

Das Anzeige-Label oder die ID wird in der Adapterliste im Ausschnitt **Integrationspunkt** in HP Universal CMDB angezeigt.

Beim Erstellen eines allgemeinen DB-Adapters muss das Tag **changes-source** im Tag **support-replication-data** nicht bearbeitet werden. Wenn das Plugin **FcmdbPluginForSyncGetChangesTopology** implementiert ist, wird die geänderte Topologie von der letzten Ausführung zurückgegeben. Bei nicht implementiertem Plugin wird die vollständige Topologie zurückgegeben und entsprechend den zurückgegebenen CIs ein automatischer Löschvorgang durchgeführt.

Weitere Informationen zum Auffüllen der CMDB mit Daten finden Sie unter "Seite "Integration Studio" " im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

- Falls der Adapter die Zuordnungs-Engine der Version 8.x verwendet (und nicht die neue Abstimmungszuordnungs-Engine), ersetzen Sie das Element:

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

  durch den folgenden Code:

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

  To revert to the new mapping engine, return the element to the following value:

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

- Suchen Sie die **category**-Definition:

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

  Ändern Sie den Kategoriennamen **Generic** in einen Namen Ihrer Wahl.

  > **Hinweis:** Adapter mit dem Kategoriennamen **Generic** werden bei der Erstellung eines neuen Integrationspunkts nicht in Integration Studio aufgeführt.

- Die Verbindung zur Datenbank kann mit einem Benutzernamen (Schema), einem Kennwort, einem Datenbanktyp, dem Namen eines Datenbankhostcomputers und dem Datenbanknamen oder der SID beschrieben werden.

  Für diese Art der Verbindung weisen die Parameter die folgenden Elemente im Abschnitt **parameter** der XML-Datei des Adapters auf:

  ```
  <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.-->
      <!--Display name and description may be overwritten by I18N
  values-->
  ```

```
        <parameter name="host" display-name="Hostname/IP"
type="string" description="The host name or IP address of the
remote machine" mandatory="false" order-index="10" />
        <parameter name="port" display-name="Port" type="integer"
description="The remote machine's connection port"
mandatory="false" order-index="11" />
        <parameter name="dbtype" display-name="DB Type"
type="string" description="The type of database" valid-
values="Oracle;SQLServer;MySQL;BO" mandatory="false" order-
index="13">Oracle</parameter>
        <parameter name="dbname" display-name="DB Name/SID"
type="string" description="The name of the database or its SID
(in case of Oracle)"  mandatory="false" order-index="13" />
        <parameter name="credentialsId" display-name="Credentials
ID" type="integer" description="The credentials to be used"
mandatory="true" order-index="12" />
</parameters>
```

Hinweis: Dies ist die Standardkonfiguration. Daher enthält die Datei **db_adapter.xml** bereits diese Definition.

In einigen Fällen kann die Verbindung zur Datenbank nicht auf diese Art konfiguriert werden. Dies ist beispielsweise der Fall beim Verbinden mit Oracle RAC oder beim Verbinden mithilfe eines anderen Datenbanktreibers als des mit der CMDB bereitgestellten.

In diesen Situationen können Sie die Verbindung mithilfe eines Benutzernamens (Schema), Kennworts und einer Verbindungs-URL-Zeichenfolge beschreiben.

Bearbeiten Sie hierzu den Abschnitt mit den XML-Parametern des Adapters wie folgt:

```
<parameters>
    <!--The description attribute may be written in simple text or
HTML.-->
    <!--The host attribute is treated as a special case by
CMDBRTSM-->
    <!--and will automatically select the probe name (if possible)
-->
    <!--according to this attribute's value.-->
    <!--Display name and description may be overwritten by I18N
values-->
        <parameter name="url" display-name="Connection String"
type="string" description="The connection string to connect to the
database" mandatory="true" order-index="10" />
        <parameter name="credentialsId" display-name="Credentials
ID" type="integer" description="The credentials to be used"
mandatory="true" order-index="12" />
</parameters>
```

Im Folgenden ein Beispiel für einen URL, der eine Verbindung mit Oracle RAC mithilfe eines vordefinierten DataDirect-Treibers herstellt:
jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true.

4. Öffnen Sie im temporären Verzeichnis den Ordner **adapterCode** und ändern Sie den Eintrag **GenericDBAdapter** in den im vorangegangenen Schritt verwendeten Wert des Attributs **adapter id**.

   Dieser Ordner enthält die Konfiguration des Adapters, wie beispielsweise den Adapternamen, die Abfragen und Klassen in der CMDB sowie die vom Adapter unterstützten Felder im RDBMS.

5. Konfigurieren Sie den Adapter wie erforderlich. Weitere Informationen finden Sie unter "Konfigurieren des Adapters – Minimale Methode" oben.

6. Erstellen Sie eine `ZIP`-Datei mit dem gleichen Namen, den Sie dem Attribut **adapter id** gegeben haben (siehe Schritt "Bearbeiten Sie die XML-Datei des Adapters:" auf Seite 95).

   Hinweis: Die Datei **descriptor.xml** ist standardmäßig in jedem Package enthalten.

7. Speichern Sie das neue Package, das Sie im vorherigen Schritt erstellt haben. Das Standardverzeichnis für Adapter lautet: **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Konfigurieren des Adapters – Minimale Methode

Das folgende Verfahren beschreibt eine Methode zur Zuordnung des Klassenmodells in der CMDB zu einem RDBMS.

Die Konfigurationsdateien befinden sich im Package **db-adapter.zip** im Ordner **C:\hp\UCMDB\UCMDBServer\content\adapters**, den Sie im Schritt "Extrahieren Sie das Package in ein lokales temporäres Verzeichnis." auf Seite 95 unter "Vorbereiten des Adapter-Packages" auf Seite 95 extrahiert haben.

Hinweis: Die Datei **orm.xml**, die infolge der Ausführung dieser Methode automatisch generiert wird, ist ein gutes Beispiel, das Sie beim Arbeiten mit der erweiterten Methode verwenden können.

In folgenden Fällen empfiehlt sich die Verwendung der minimalen Methode:

- Sie müssen einen einzelnen Knoten föderieren/auffüllen (z. B. ein Knotenattribut).
- Sie müssen die Funktionen des allgemeinen Datenbankadapters demonstrieren.

Diese Methode:

- Unterstützt nur die Föderation/Auffüllung eines Knotens
- Unterstützt nur virtuelle n:1-Beziehungen

Diese Aufgabe umfasst folgende Schritte:

- "Konfigurieren der Datei "adapter.conf"" oben
- "Konfigurieren der Datei "simplifiedConfiguration.xml"" auf der nächsten Seite

Konfigurieren der Datei "adapter.conf"

In diesem Schritt ändern Sie die Einstellungen in der Datei `adapter.conf` so, dass der Adapter die vereinfachte Konfigurationsmethode verwendet.

1. Öffnen Sie die Datei **adapter.conf** in einem Texteditor.
2. Suchen Sie nach der folgenden Zeile: **use.simplified.xml.config=<true/false>**.
3. Ändern Sie die Zeile wie folgt: **use.simplified.xml.config=true**.

Konfigurieren der Datei "simplifiedConfiguration.xml"

In diesem Schritt konfigurieren Sie die Datei **simplifiedConfiguration.xml**, indem Sie den CIT in der CMDB den Feldern in der RDBMS-Tabelle zuordnen.

1. Öffnen Sie die Datei **simplifiedConfiguration.xml** in einem Texteditor.

   Diese Datei enthält eine Vorlage, die Sie für jede zuzuordnende Entität verwenden können.

   Hinweis: Bearbeiten Sie die Datei **simplifiedConfiguration.xml** nicht mit dem Editor der Microsoft Corporation. Verwenden Sie Notepad++, UltraEdit oder einen Texteditor eines anderen Drittanbieters.

2. Ändern Sie die folgenden Attribute:

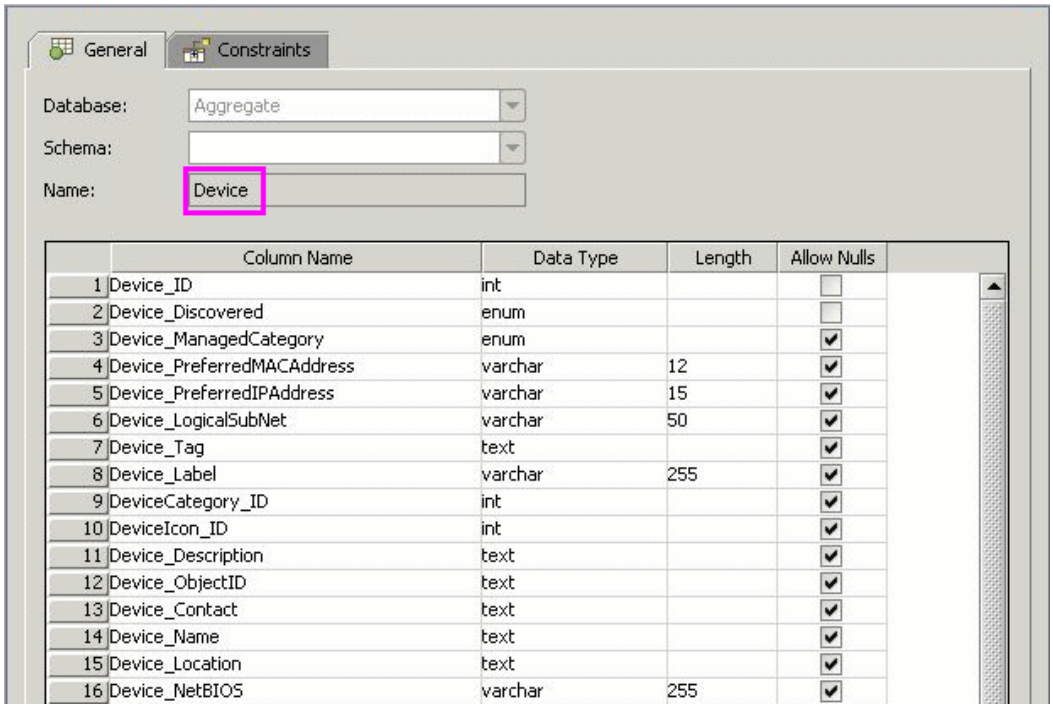
   - Den CIT-Namen in UCMDB (cmdb-class-name) und den entsprechenden Tabellennamen im RDBMS (default-table-name):

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

     Das Attribut `cmdb-class-name` wird vom Node-CIT übernommen:

     Das Attribut `default-table-name` wird von der Device-Tabelle übernommen:

- Die eindeutige ID im RDBMS:

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

  Hinweis: Der Primärschlüssel entspricht der Entitäts-ID in der Datei **orm.xml**.

- Die Abstimmungsregel (reconciliation-by-two-nodes):

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

- Das Abstimmungsattribut in UCMDB (`cmdb-attribute-name`) und im RDBMS (`column-name`):

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

- Den Namen des CIT (`cmdb-class-name`) und den Namen der entsprechenden Tabelle im RDBMS (`default-table-name`). Außerdem die CMDB-Beziehung (`connected-cmdb-class-name`) und die CIT-Beziehung (`link-class-name`):

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

- Den Primärschlüssel und den Fremdschlüssel:

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

- Die eindeutige ID im RDBMS:

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

- Die Zuordnung zwischen dem CMDB-Attribut (`cmdb-attribute-name`) und dem Spaltennamen im RDBMS (`column-name`):

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

3. Speichern Sie die Datei.

Beispiel: Auffüllen eines Knotens und einer IP-Adresse mit der vereinfachten Methode

In diesem Beispiel wird gezeigt, wie Sie einen **Knoten** auffüllen, der über einen Containment-Link mit einer **IP-Adresse** in UCMDB verknüpft ist. Das RDBMS weist eine Tabelle mit der Bezeichnung **simpleNode** auf, die Daten zum Computernamen, zum Computerknoten und zur IP-Adresse des Computers enthält.

Der Inhalt der Tabelle **simpleNode** ist unten abgebildet:

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

Die Auffüllung wird in drei Stufen durchgeführt:

- "Erstellen der Datei "simplifiedConfiguration.xml"" oben
- " Erstellen der TQL" auf der nächsten Seite
- "Erstellen eines Integrationspunkts" auf Seite 103

Erstellen der Datei "simplifiedConfiguration.xml"

Gehen Sie wie folgt vor, um die Datei **simplifiedConfiguration.xml** zu erstellen:

1. Erstellen Sie die Entität **cmdb-class** wie folgt:

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

   Der CI-Typ ist **node** und der Name der RDBMS-Tabelle lautet **simpleNode**.

2. Legen Sie den Primärschlüssel der Tabelle wie folgt fest:

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

   Dieser Primärschlüssel entspricht der Entitäts-ID in der Datei **orm.xml**.

3. Legen Sie die Regel **reconciliation-by-two-nodes** wie folgt fest:

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

   Dieser Tag definiert die Beziehung zwischen den CI-Typen **Node** und **IpAddress**. Der Beziehungstyp lautet **Containment link**. Die Abstimmung erfolgt durch die zwei miteinander verbundenen CI-Typen. Die Attributzuordnung des verbundenen Knotens (in diesem Fall **IpAddress**) wird im Attribut **connected-node** definiert.

4. Fügen Sie die Bedingung **or** wie folgt zwischen den Abstimmungsattributen hinzu:

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

   Dieser Tag definiert eine ODER-Beziehung zwischen den Abstimmungsattributen, d. h. durch das erste Abstimmungsattribut, das **true** ergibt, wird die gesamte Abstimmungsregel auf **true** gesetzt.

5. Fügen Sie die folgenden Attribute hinzu:

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

   Dieser Tag legt eine Zuordnung zwischen **node.name** in UCMDB und der Spalte **host_name** in der Tabelle **simpleNode** fest.

   Gehen Sie mit dem Attribut **data_note** gleichermaßen vor:

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

Fügen Sie das Attribut **connected node** hinzu:

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

Dieser Tag legt eine Zuordnung zwischen **ip_address.name** und der Spalte **ip_address** in der Tabelle **simpleNode** fest.

6. Geben Sie die schließenden Tags in der folgenden Reihenfolge ein:

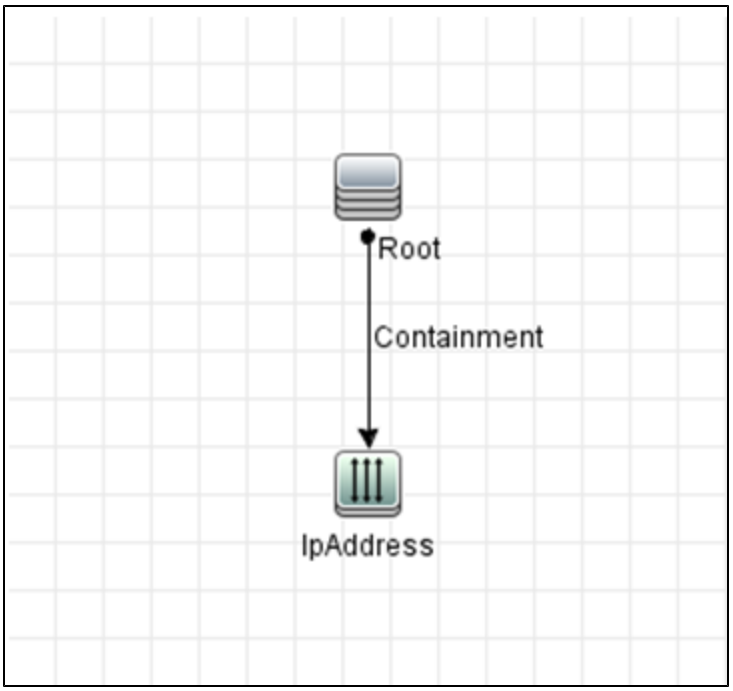
```
</or>

</reconciliation-by-two-nodes>

</cmdb-class>
```

Erstellen der TQL

Die TQL ist ein CI des Typs **node**, das über einen Containment-Link mit dem CI-Typ **ip_address** verbunden wird. Der Knoten sollte wie nachfolgend gezeigt als **root** gekennzeichnet werden.



So erstellen Sie die TQL:

1. Wechseln Sie zu **Modellieren > Modeling Studio**.
2. Klicken Sie auf die Schaltfläche **Neu** und erstellen Sie eine neue Abfrage.
3. Öffnen Sie die Registerkarte **CI-Typen** und ziehen Sie die CI-Typen **Node** und **IpAddress** auf den TQL-Bildschirm.
4. Verbinden Sie **Node** und **IpAddress** mit einem Containment-Link.

5. Klicken Sie mit der rechten Maustaste auf das Element **Node** und wählen Sie **Abfrageknoteneigenschaften** aus.
6. Ändern Sie die Einstellung für **Elementname** in **Root**
7. Öffnen Sie die Registerkarte **Elementlayout**. Wählen Sie bei **Attributbedingung** die Option **Bestimmte Attribute** aus. Wählen Sie im Fenster **Verfügbare Attribute** die Einträge **Name** und **Anmerkung** aus und verschieben Sie sie in das Fenster **Bestimmte Attribute**.
8. Klicken Sie mit der rechten Maustaste auf das Element **IpAddress** und wählen Sie **Abfrageknoteneigenschaften** aus.
9. Öffnen Sie die Registerkarte **Elementlayout**. Wählen Sie bei **Attributbedingung** die Option **Bestimmte Attribute** aus. Wählen Sie im Fenster **Verfügbare Attribute** den Eintrag **Name** aus und verschieben Sie ihn in das Fenster **Bestimmte Attribute**.
10. Speichern Sie die TQL.

Erstellen eines Integrationspunkts

Gehen Sie wie folgt vor, um den Integrationspunkt zu erstellen:

1. Wechseln Sie zu **Datenflussverwaltung > Integration Studio** und klicken Sie auf die Schaltfläche **Neuer Integrationspunkt**.
2. Fügen Sie die Details des Integrationspunkts ein und klicken Sie auf **OK**.
3. Klicken in der Registerkarte **Auffüllung** auf die Schaltfläche **Neuer Integrationsjob** und fügen Sie die zuvor erstellte TQL hinzu.
4. Speichern Sie den Integrationspunkt und klicken Sie auf die Schaltfläche **Vollständige Synchronisierung ausführen**.

Konfigurieren des Adapters – Erweiterte Methode

Die Konfigurationsdateien befinden sich im Package **db-adapter.zip** im Ordner **C:\hp\UCMDB\UCMDBServer\content\adapters**, den Sie im Schritt "Extrahieren Sie das Package in ein lokales temporäres Verzeichnis." auf Seite 95 unter "Vorbereiten des Adapter-Packages" auf Seite 95 extrahiert haben.

Diese Aufgabe umfasst folgende Schritte:

- "Konfigurieren der Datei "orm.xml"" oben
- "Konfigurieren der Datei "reconciliation_types.txt"" auf Seite 107
- "Konfigurieren der Datei "reconciliation_rules.txt" " auf Seite 107

Konfigurieren der Datei "orm.xml"

In diesem Schritt ordnen Sie die CITs und Beziehungen in der CMDB den Tabellen im RDBMS zu.

1. Öffnen Sie die Datei **orm.xml** in einem Texteditor.

   Diese Datei enthält standardmäßig eine Vorlage, die Sie zum Zuordnen der benötigten Anzahl von CITs und Beziehungen verwenden können.

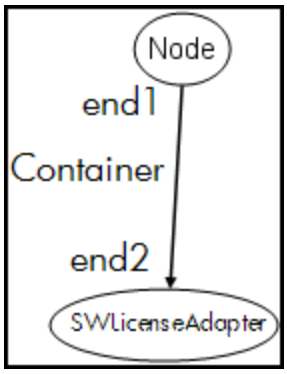
   Hinweis: Bearbeiten Sie die Datei **orm.xml** nicht mit dem Editor der Microsoft

Corporation. Verwenden Sie Notepad++, UltraEdit oder einen Texteditor eines anderen Drittanbieters.

2. Ändern Sie die Datei entsprechend den zuzuordnenden Datenentitäten. Weitere Informationen finden Sie in den folgenden Beispielen.

   Die folgenden Beziehungstypen können in der Datei **orm.xml** zugeordnet werden:

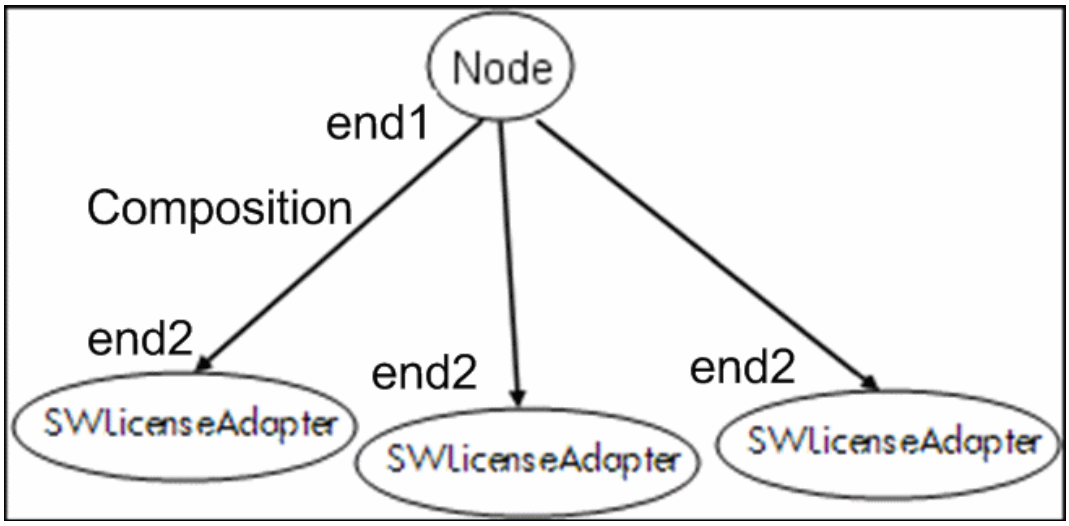
   - 1:1:



   Der Code für diesen Beziehungstyp lautet:

```
<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>
```

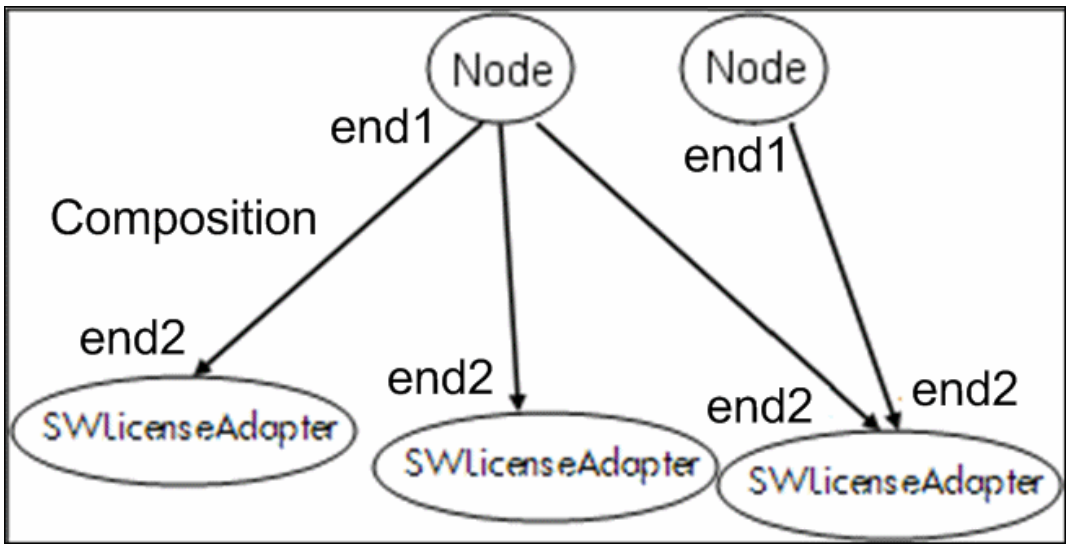
   - n:1:



   Der Code für diesen Beziehungstyp lautet:

```
<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>
```

- n:n:



Der Code für diesen Beziehungstyp lautet:

```
<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>
```

Weitere Informationen zu den Benennungskonventionen finden Sie unter "Benennungskonventionen" auf Seite 128.

Beispiel für die Entitätenzuordnung zwischen dem Datenmodell und dem RDBMS:

Hinweis: Attribute, die nicht konfiguriert werden müssen, werden bei den folgenden Beispielen ausgelassen.

- Die Klasse des CMDB-CIT:

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

- Der Name der Tabelle im RDBMS:

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

- Der Spaltenname der eindeutigen ID in der RDBMS-Tabelle:

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

- Der Name des Attributs im CMDB-CIT:

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

- Der Name des Tabellenfelds in der externen Datenquelle:

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

- Der Name des neuen CIT, den Sie unter "Erstellen eines CI-Typs" auf Seite 93 erstellt haben:

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

- Der Name der entsprechenden Tabelle im RDBMS:

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

- Die eindeutige ID im RDBMS:

- Der Attributname im CMDB-CIT und der Name des entsprechenden Attributs im RDBMS:

Beispiel für die Beziehungszuordnung zwischen dem Datenmodell und dem RDBMS:

- Die Klasse der CMDB-Beziehung:

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

- Der Name der RDBMS-Tabelle, in der die Beziehung durchgeführt wird:

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

- Die eindeutige ID im RDBMS:

```
<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>
```

- Der Beziehungstyp und der CMDB-CIT:

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

- Das Primärschlüssel- und das Fremdschlüsselfeld im RDBMS:

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

Konfigurieren der Datei "reconciliation_types.txt"

Öffnen Sie die Datei **reconciliation_types.txt** in einem Texteditor.

Weitere Informationen finden Sie unter "Die Datei "reconciliation_types.txt"" auf Seite 137.

Konfigurieren der Datei "reconciliation_rules.txt"

In diesem Schritt definieren Sie die Regeln, nach denen der Adapter die Abstimmung zwischen der CMDB und dem RDBMS vornimmt (nur wenn aus Gründen der Abwärtskompatibilität mit Version 8.x die Zuordnungs-Engine verwendet wird):

1. Öffnen Sie die Datei **META-INF\reconciliation_rules.txt** in einem Texteditor.
2. Ändern Sie die Datei entsprechend dem CIT, den Sie zuordnen. Um beispielsweise einen Node-CIT zuzuordnen, verwenden Sie die folgenden Ausdruck:

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

Hinweis:

- Wenn bei den Daten in der Datenbank zwischen Groß- und Kleinschreibung unterschieden wird, dürfen Sie das Steuerzeichen (**^**) nicht löschen.
- Vergewissern Sie sich, dass es zu jeder öffnenden eckigen Klammer eine schließende Klammer gibt.

Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 137.

Implementieren eines Plugin

Diese Aufgabe beschreibt, wie Sie einen allgemeinen DB-Adapter mit Plugins implementieren und bereitstellen.

Hinweis: Bevor Sie ein Plugin für einen Adapter schreiben, stellen Sie sicher, dass Sie die

erforderlichen Schritte unter "Vorbereiten des Adapter-Packages" auf Seite 95 abgeschlossen haben.

1. Kopieren Sie die folgenden JAR-Dateien vom Installationsverzeichnis des UCMDB-Servers in den Klassenpfad Ihrer Entwicklung:
   - Kopieren Sie die Datei **db-interfaces.jar** und die Datei **db-interfaces-javadoc.jar** aus dem Ordner **tools\adapter-dev-kit\db-adapter-framework**.
   - Kopieren Sie die Datei **federation-api.jar** und die Datei **federation-api-javadoc.jar** aus dem Ordner **\tools\adapter-dev-kit\SampleAdapters\production-lib**.

     Hinweis: Weitere Informationen zum Entwickeln eines Plugins finden Sie in den Dateien **db-interfaces-javadoc.jar** und **federation-api-javadoc.jar** sowie in der Online-Dokumentation unter:
     - **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DBAdapterFramework_JavaAPI\index.html**
     - **C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\Federation_JavaAPI\index.html**
2. Schreiben Sie eine Java-Klasse, die die Java-Schnittstelle des Plugins implementiert. Die Schnittstellen sind in der Datei **db-interfaces.jar** definiert. Die folgende Tabelle gibt die Schnittstelle an, die für jedes Plugin implementiert werden muss:

Plugin-Typ	Schnittstellenname	Methode
Synchronisieren der vollständigen Topologie	FcmdbPluginForSyncGetFullTopology	getFullTopology
Synchronisieren von Änderungen	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Synchronisieren des Layouts	FcmdbPluginForSyncGetLayout	getLayout
Abrufen unterstützter Abfragen	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Ändern der TQL-Abfragedefinition und der Ergebnisse	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Ändern der Layoutanforderung für CIs	FcmdbPluginGetCIsLayout	getCisLayout

Plugin-Typ	Schnittstellenname	Methode
Ändern der Layoutanforderung für Links	FcmdbPluginGetRelationsLayout	getRelationsLayout
Zurückgeben von IDs	FcmdbPluginPushBackIds	getPushBackIdsSQL

Die Plugin-Klasse muss einen öffentlichen Standardkonstruktor haben. Außerdem müssen alle Schnittstellen eine Methode namens "initPlugin" aufweisen. Diese Methode wird garantiert vor jeder anderen Methode aufgerufen. Sie wird zur Initialisierung des Adapters mit dem Umgebungsobjekt des enthaltenen Adapters verwendet.

Wenn **FcmdbPluginForSyncGetChangesTopology** implementiert ist, stehen zum Berichten der Änderungen zwei unterschiedliche Möglichkeiten zur Verfügung:

- **Berichten der gesamten Stammtopologie zu jeder Zeit.** Entsprechend dieser Topologie ermittelt die automatische Löschfunktion, welche CIs entfernt werden sollten. In diesem Fall sollte die automatische Löschfunktion folgendermaßen aktiviert werden:

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

- **Berichten jeder entfernten/aktualisierten CI-Instanz.** In diesem Fall sollte der automatische Löschmechanismus folgendermaßen deaktiviert werden:

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

3. Stellen Sie sicher, dass sich die SDK-JAR-Datei für die Föderation und die JAR-Dateien des allgemeinen DB-Adapters im Klassenpfad befinden, bevor Sie mit dem Kompilieren Ihres Java-Codes beginnen. Die SDK-Datei für die Föderation **federation_api.jar** befindet sich im Verzeichnis **C:\hp\UCMDB\UCMDBServer\lib**.

4. Packen Sie Ihre Klasse in eine JAR-Datei und speichern Sie diese im Adapter-Package im Ordner **adapterCode\<Name Ihres Adapters>**, bevor Sie sie bereitstellen.

   Die Plugins werden unter Verwendung der Datei **plugins.txt** konfiguriert, die sich im Ordner **\META-INF** des Adapters befindet.

   Im Folgenden finden Sie ein Beispiel der Datei des DDMi-Adapters:

```
# mandatory plugin to sync full topology
[getFullTopology]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin
# mandatory plugin to sync changes in topology
[getChangesTopology]
com.hp.ucmdb.adapters.ed.plugins.replication.EDReplicationPlugin
# mandatory plugin to sync layout
```

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

Legende:

- Kommentarzeile.

[<Adaptertyp>] – Start des Definitionsabschnitts für einen bestimmten Adaptertyp.

Unterhalb von [<Adaptertyp>] kann eine Leerzeile eingefügt werden, was bedeutet, dass keine Plugin-Klasse zugeordnet ist, oder es kann der vollqualifizierte Name Ihrer Plugin-Klasse aufgeführt werden.

5. Verpacken Sie Ihren Adapter mit der neuen JAR-Datei und der aktualisierten **plugins.xml**-Datei. Die übrigen Dateien im Package sollten identisch sein mit denen jedes anderen Adapters, der auf dem allgemeinen DB-Adapter basiert.

Bereitstellen des Adapters

1. Greifen Sie in UCMDB auf Package Manager zu. Weitere Informationen finden Sie unter "Seite "Package Manager"" im *HP Universal CMDB – Verwaltungshandbuch*.
2. Klicken Sie auf das Symbol **Packages auf Server bereitstellen (von lokalem Datenträger)** und suchen Sie Ihr Adapter-Package. Wählen Sie das Package aus und klicken Sie auf **Öffnen**. Klicken Sie dann auf **Bereitstellen**, um das Package in Package Manager anzuzeigen.
3. Wählen Sie Ihr Package in der Liste aus und klicken Sie auf das Symbol **Package-Ressourcen anzeigen**, um zu prüfen, ob der Package-Inhalt von Package Manager erkannt wird.

Bearbeiten des Adapters

Nachdem Sie den Adapter erstellt und bereitgestellt haben, können Sie ihn in UCMDB bearbeiten. Weitere Informationen finden Sie unter "Adapter Management" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Erstellen eines Integrationspunkts

In diesem Schritt prüfen Sie, ob die Föderation funktioniert. Das heißt, Sie prüfen, ob die Verbindung und die XML-Datei gültig sind. Bei dieser Prüfung wird jedoch nicht kontrolliert, ob die XML-Datei den korrekten Feldern im RDBMS zugeordnet ist.

1. Greifen Sie in UCMDB auf Integration Studio zu (**Datenflussverwaltung > Integration Studio**).
2. Erstellen Sie einen Integrationspunkt. Weitere Informationen finden Sie unter "Dialogfeld "Neuer Integrationspunkt"/"Integrationspunkt bearbeiten"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

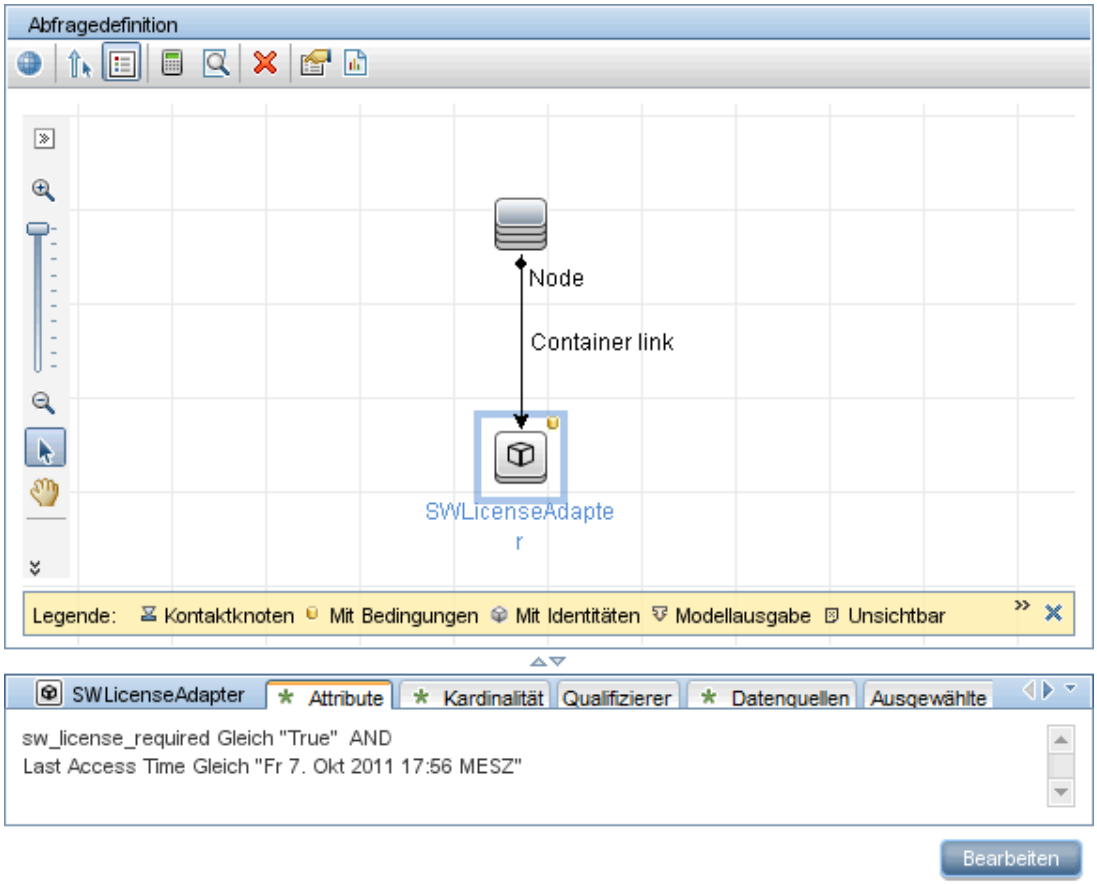
   Auf der Registerkarte **Föderation** werden alle CITs angezeigt, die unter Verwendung dieses Integrationspunkts föderiert werden können. Weitere Informationen finden Sie unter "Registerkarte "Föderation"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Erstellen einer Ansicht

In diesem Schritt erstellen Sie eine Ansicht zur Anzeige der CIT-Instanzen.

1. Greifen Sie in UCMDB auf Modeling Studio zu (**Modellieren > Modeling Studio**).
2. Erstellen Sie eine Ansicht. Weitere Informationen finden Sie unter "Erstellen einer Pattern-Ansicht" im *HP Universal CMDB – Modellierungshandbuch*.
3. Sie können Bedingungen zur TQL hinzufügen, z. B. dass die letzte Zugriffszeit mehr als 6

Monate zurückliegen soll:



Berechnen der Ergebnisse

In diesem Schritt überprüfen Sie die Ergebnisse.

1. Greifen Sie in UCMDB auf Modeling Studio zu (**Modellieren > Modeling Studio**).
2. Öffnen Sie eine Ansicht.
3. Berechnen Sie die Ergebnisse, indem Sie auf die Schaltfläche **Anzahl der Abfrageergebnisse berechnen** klicken.
4. Klicken Sie auf die Schaltfläche **Vorschau**, um die CIs in der Ansicht anzuzeigen.

Anzeigen der Ergebnisse

In diesem Schritt zeigen Sie die Ergebnisse an und debuggen Fehler in der Prozedur. Wenn in der Ansicht beispielsweise nichts angezeigt wird, prüfen Sie die Definitionen in der Datei **orm.xml**, entfernen Sie die Beziehungsattribute und laden Sie den Adapter neu.

1. Greifen Sie in UCMDB auf IT Universe Manager zu (**Modellieren > IT Universe Manager**).
2. Wählen Sie ein CI aus. Auf der Registerkarte **Eigenschaften** werden die Ergebnisse der Föderation angezeigt.

Anzeigen von Reports

In diesem Schritt zeigen Sie Topologie-Reports an. Weitere Informationen finden Sie unter "Topologie-Reports – Übersicht" im *HP Universal CMDB – Modellierungshandbuch*.

Aktivieren von Protokolldateien

Sie können die Protokolldateien zurate ziehen, um die Berechnungsabläufe und den Adapterlebenszyklus zu verstehen und Debug-Informationen anzuzeigen. Weitere Informationen finden Sie unter "Adapterprotokolldateien" auf Seite 156.

Verwenden von Eclipse für die Zuordnung zwischen CIT-Attributen und Datenbanktabellen

Achtung: Dieses Verfahren ist für Benutzer vorgesehen, die über fortgeschrittene Kenntnisse der Inhaltsentwicklung verfügen. Bei Fragen wenden Sie sich an HP Software Support.

Diese Aufgabe beschreibt, wie Sie das JPA-Plugin, das im Lieferumfang der J2EE-Version von Eclipse enthalten ist, installieren und verwenden, um Folgendes zu ermöglichen:

- Grafische Zuordnung zwischen den CMDB-Klassenattributen und den Spalten der Datenbanktabelle.
- Manuelle Bearbeitung der Zuordnungsdatei (`orm.xml`) bei gleichzeitiger Gewährleistung der Richtigkeit. Bei der Überprüfung der Richtigkeit wird unter anderem die Syntax geprüft und sichergestellt, dass die Klassenattribute und die zugeordneten Spalten der Datenbanktabelle korrekt angegeben sind.
- Bereitstellung der Zuordnungsdatei auf dem CMDB-Server und Anzeigen von Fehlern als weitere Maßnahme zur Überprüfung der Richtigkeit.
- Definition einer Beispielabfrage beim CMDB-Server und direkte Ausführung der Abfrage über Eclipse, um die Zuordnungsdatei zu testen.

Version 1.1 des Plugins ist kompatibel mit UCMDB der Version 9.01 oder höher sowie Eclipse IDE für Java EE Developers, Version 1.2.2.20100217-2310 oder höher.

Diese Aufgabe umfasst folgende Schritte:

- "Voraussetzungen" auf der nächsten Seite
- "Installation" auf der nächsten Seite
- "Vorbereiten der Arbeitsumgebung" auf der nächsten Seite
- "Erstellen eines Adapters" auf Seite 115
- "Konfigurieren des CMDB-Plugins" auf Seite 115
- "Importieren des UCMDB-Klassenmodells" auf Seite 116

1. ## Voraussetzungen

   Installieren Sie auf dem Computer, auf dem Sie Eclipse ausführen, das neueste Update für **Java Runtime Environment (JRE) 6**. Das Update steht auf der folgenden Website zum Download bereit:
   http://java.sun.com/javase/downloads/index.jsp.

2. ## Installation

   a. Laden Sie die Datei **Eclipse IDE for Java EE Developers** von der Website **http://www.eclipse.org/downloads** in einen lokalen Ordner, z. B. **C:\Program Files\eclipse**.

   b. Kopieren Sie die Datei **com.hp.plugin.import_cmdb_model_1.0.jar** von **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\bin** nach **C:\Program Files\Eclipse\plugins**.

   c. Starten Sie **C:\Program Files\Eclipse\eclipse.exe**. Wenn eine Meldung angezeigt wird, dass die Java Virtual Machine nicht gefunden wurde, starten Sie **eclipse.exe** mit folgender Befehlszeile:

   ```
   "C:\Program Files\eclipse\eclipse.exe" -vm "<JRE-Installationsordner>\bin"
   ```

3. ## Vorbereiten der Arbeitsumgebung

   In diesem Schritt richten Sie den Arbeitsbereich, die Datenbank, die Verbindungen und die Treibereigenschaften ein.

   a. Extrahieren Sie die Datei **workspaces_gdb.zip** von **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** in das Verzeichnis **C:\Documents and Settings\All Users**.

> **Hinweis:** Sie müssen den exakten Ordnerpfad verwenden. Wenn Sie die Datei verpackt lassen oder in den falschen Pfad extrahieren, funktioniert das Verfahren nicht.

b. Klicken Sie in Eclipse auf **File > Switch Workspace > Other**:

Wenn Sie mit:

- SQL Server arbeiten, wählen Sie den folgenden Ordner aus: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.
- MySQL arbeiten, wählen Sie den folgenden Ordner aus: **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
- Oracle arbeiten, wählen Sie den folgenden Ordner aus: **C:\Documents and Settings\All Users\workspace_gdb_oracle**.

c. Klicken Sie auf **OK**.

d. Zeigen Sie in Eclipse die Project Explorer-Ansicht an und wählen Sie **<Ihr_aktives_ Projekt> > JPA Content > persistence.xml > >Name_des_aktiven_Projekts< > orm.xml** aus.

e. Klicken Sie in der Data Source Explorer-Ansicht (Ausschnitt unten links) mit der rechten Maustaste auf die Datenbankverbindung und wählen Sie das Menü **Properties** aus.

f. Klicken Sie im Dialogfeld **Properties for <Verbindungsname>** auf **Common** und aktivieren Sie das Kontrollkästchen **Connect every time the workbench is started**. Wählen Sie **Driver Properties** aus und geben Sie die Verbindungseigenschaften ein. Klicken Sie auf **Test Connection** und vergewissern Sie sich, dass die Verbindung funktioniert. Klicken Sie auf **OK**.

g. Klicken Sie in der Data Source Explorer-Ansicht mit der rechten Maustaste auf die Datenbankverbindung und klicken Sie auf **Connect**. Unterhalb des Symbols für die Datenbankverbindung wird eine Struktur angezeigt, die die Datenbankschemata und -tabellen enthält.

4. Erstellen eines Adapters

Erstellen Sie einen Adapter unter Berücksichtigung der in "Schritt 1: Erstellen eines Adapters" auf Seite 30 aufgeführten Richtlinien.

5. Konfigurieren des CMDB-Plugins

a. Klicken Sie in Eclipse auf **UCMDB > Settings**, um das Dialogfeld **CMDB Settings** zu öffnen.

b. Sofern noch nicht geschehen, wählen Sie nun das neu erstellte JPA-Projekt als aktives Projekt aus.

c. Geben Sie den CMDB-Hostnamen ein, z. B. **localhost** oder **labm1.itdep1**. Es ist nicht erforderlich, die Portnummer oder das Präfix `http://` bei der Adresse anzugeben.

d. Geben Sie den Benutzernamen und das Kennwort für den Zugriff auf die CMDB-API ein, normalerweise **admin**/**admin**.

e. Stellen Sie sicher, dass der Ordner **C:\hp** auf dem CMDB-Server als Netzwerklaufwerk zugeordnet ist.

f. Wählen Sie den Basisordner des relevanten Adapters unter **C:\hp** aus. Der Basisordner ist der Ordner, der die Datei **dbAdapter.jar** und den Unterordner **META-INF** enthält. Er sollte sich im Pfad **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<Adaptername>** befinden. Achten Sie darauf, dass am Ende kein umgekehrter Schrägstrich (\) steht.

6. Importieren des UCMDB-Klassenmodells

In diesem Schritt wählen Sie die CITs aus, die als JPA-Entitäten zugeordnet werden sollen.

a. Klicken Sie auf **UCMDB > CMDB-Klassenmodell importieren**, um das Dialogfeld **CI-Typen auswählen** zu öffnen.

b. Wählen Sie die CI-Typen aus, die Sie als JPA-Entitäten zuordnen möchten. Klicken Sie auf **OK**. Die CI-Typen werden als Java-Klassen importiert. Vergewissern Sie sich, dass sie unter dem **src**-Ordner des aktiven Projekts angezeigt werden.

7. Erstellen der ORM-Datei – Zuordnen von UCMDB-Klassen zu Datenbanktabellen

In diesem Schritt ordnen Sie die Java-Klassen (die Sie im vorherigen Schritt importiert haben) den Datenbanktabellen zu.

a. Vergewissern Sie sich, dass die DB-Verbindung hergestellt ist. Klicken Sie in der Project Explorer-Ansicht mit der rechten Maustaste auf das aktive Projekt (Standardname: myProject). Wählen Sie die JPA-Ansicht aus, aktivieren Sie das Kontrollkästchen **Override default schema from connection** und wählen Sie das relevante Datenbankschema aus. Klicken Sie auf **OK**.

b. Ordnen Sie einen CIT zu: Klicken Sie in der JPA Structure-Ansicht mit der rechten Maustaste auf die Verzweigung **Entity Mappings** und wählen Sie **Add Class** aus. Das Dialogfeld **Add Persistent Class** wird geöffnet. Ändern Sie nicht Einstellung des Felds **Map as** (**Entity**).

c. Klicken Sie auf **Browse** und wählen Sie die zuzuordnende UCMDB-Klasse aus (alle UCMDB-Klassen gehören zum Package **generic_db_adapter**).

d. Klicken Sie in beiden Dialogfeldern auf **OK**. Die ausgewählte Klasse wird in der JPA Structure-Ansicht unter **Entity Mappings** angezeigt.

> **Hinweis:** Falls die Entität ohne Attributstruktur angezeigt wird, klicken Sie in der Project Explorer-Ansicht mit der rechten Maustaste auf das aktive Projekt. Klicken Sie auf **Close** und dann auf **Open**.

e. Wählen Sie in der JPA Details-Ansicht die primäre Datenbanktabelle aus, der die UCMDB-Klasse zugeordnet werden soll. Lassen Sie alle anderen Felder unverändert.

8. Zuordnen von IDs

Gemäß JPA-Standards muss jede persistente Klasse mindestens ein ID-Attribut haben. Für UCMDB-Klassen können Sie bis zu drei Attribute als IDs zuordnen. Potenzielle ID-Attribute haben die Bezeichnungen **id1**, **id2** und **id3**.

So ordnen Sie ein ID-Attribut zu:

a. Erweitern Sie in der JPA Structure-Ansicht die entsprechende Klasse unter der Verzweigung **Entity Mappings**, klicken Sie mit der rechten Maustaste auf das relevante Attribut (z. B. **id1**) und wählen Sie die Option **Add Attribute to XML and Map…** aus.

b. Das Dialogfeld **Add Persistent Attribute** wird geöffnet. Wählen Sie im Feld **Map as** die Option **Id** aus und klicken Sie auf **OK**.

c. Wählen Sie in der JPA Details-Ansicht die Spalte der Datenbanktabelle aus, der das ID-Feld zugeordnet werden soll.

9. ## Zuordnen von Attributen

In diesem Schritt ordnen Sie den Datenbankspalten Attribute zu.

a. Erweitern Sie in der JPA Structure-Ansicht die entsprechende Klasse unter der Verzweigung **Entity Mappings**, klicken Sie mit der rechten Maustaste auf das relevante Attribut (z. B. **host_hostname**) und wählen Sie die Option **Add Attribute to XML and Map…** aus.

b. Das Dialogfeld **Add Persistent Attribute** wird geöffnet. Wählen Sie im Feld **Map as** die Option **Basic** aus und klicken Sie auf **OK**.

c. Wählen Sie in der JPA Details-Ansicht die Spalte der Datenbanktabelle aus, der das Attribut-Feld zugeordnet werden soll.

10. ## Zuordnen eines gültigen Links

Führen Sie die oben im Schritt "Erstellen der ORM-Datei – Zuordnen von UCMDB-Klassen zu Datenbanktabellen" auf der vorherigen Seite beschriebenen Arbeiten durch, um eine UCMDB-Klasse zuzuordnen, die einen gültigen Link angibt. Der Name einer solchen Klasse ist wie folgt aufgebaut: **<Ende1 Name_der_Entität>_<Linkname>_<Ende2 Name_der_Entität>**. Beispiel: Ein **Contains**-Link zwischen einem Host und einem Standort wird durch eine Java-Klasse mit dem Namen **generic_db_adapter.host_contains_location** angegeben. Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 137.

a. Ordnen Sie die ID-Attribute der Link-Klasse wie unter "Zuordnen von IDs" auf der vorherigen Seite beschrieben zu. Erweitern Sie für jedes ID-Attribut in der JPA Details-Ansicht die Kontrollkästchengruppe unter **Details** und deaktivieren Sie die Kontrollkästchen **Insertable** und **Updateable**.

b. Ordnen Sie die Attribute **end1** und **end2** der Link-Klasse wie folgt zu: Führen Sie die folgenden Schritte für jedes **end1**- und **end2**-Attribut der Link-Klasse durch:

- Erweitern Sie in der JPA Structure-Ansicht die entsprechende Klasse unter **Entity Mappings**, klicken Sie mit der rechten Maustaste auf das relevante Attribut (z. B. **end1**) und wählen Sie die Option **Add Attribute to XML and Map…** aus.
- Wählen Sie im Dialogfeld **Add Persistent Attribute** im Feld **Map as** die Option **Many to One** oder **One to One** aus.
- Wählen Sie **Many to One** aus, wenn das angegebene **end1**- oder **end2**-CI mehrere Links dieses Typs haben kann. Wählen Sie andernfalls **One to One** aus. Beispiel: Für einen **host_contains_ip**-Link sollte das **Host**-Ende als **Many to One** zugeordnet werden, da ein Host mehrere IPs haben kann. Das **IP**-Ende hingegen sollte als **One to**

One zugeordnet werden, da eine IP nur einen einzigen Host haben kann.

- Wählen Sie in der JPA Details-Ansicht unter **Target entity** die Zielentität aus, z. B. **generic_db_adapter.host**.
- Aktivieren Sie im Abschnitt **Join Columns** der JPA Details-Ansicht die Option **Override Default**. Klicken Sie auf **Edit**. Wählen Sie im Dialogfeld **Edit Join Column** die Fremdschlüsselspalte der Link-Datenbanktabelle aus, die auf einen Eintrag in der Tabelle der **Ende1**/**Ende2**-Zielentität verweist. Wenn der referenzierte Spaltenname in der Tabelle der **Ende1**/**Ende2**-Zielentität seinem ID-Attribut zugeordnet ist, übernehmen Sie die Einstellung des Felds **Referenced Column Name**. Andernfalls wählen Sie den Namen der Spalte aus, auf die die Fremdschlüsselspalte verweist. Deaktivieren Sie die Kontrollkästchen **Insertable** und **Updatable** und klicken Sie auf **OK**.
- Wenn die **Ende1**/**Ende2**-Zielentität mehr als eine ID hat, klicken Sie auf die Schaltfläche **Add**, um weitere Join-Spalten hinzuzufügen, und ordnen Sie diese wie im vorherigen Schritt beschrieben zu.

11. Erstellen der ORM-Datei – Verwenden von sekundären Tabellen

JPA ermöglicht die Zuordnung einer Java-Klasse zu mehreren Datenbanktabellen. Beispiel: Die Klasse **Host** kann der Tabelle **Device** zugeordnet werden, um Persistenz der meisten ihrer Attribute zu ermöglichen. Sie kann aber auch der Tabelle **NetworkNames** zugeordnet werden, um Persistenz von **host_hostName** zu ermöglichen. In diesem Fall ist **Device** die primäre und **NetworkNames** die sekundäre Tabelle. Es kann eine beliebige Anzahl von sekundären Tabellen definiert werden. Die einzige Bedingung ist, dass es zwischen den Einträgen der primären und der sekundären Tabelle eine 1:1-Beziehung geben muss.

12. Definieren einer sekundären Tabelle

Wählen Sie die entsprechende Klasse in der JPA Structure-Ansicht aus. Greifen Sie in der **JPA Details**-Ansicht auf den Abschnitt **Secondary Tables** zu und klicken Sie auf **Add**. Wählen Sie im Dialogfeld **Add Secondary Table** die entsprechende sekundäre Tabelle aus. Lassen Sie alle anderen Felder unverändert.

Wenn die primäre und die sekundäre Tabelle nicht die gleichen Primärschlüssel haben, konfigurieren Sie die Join-Spalten im Abschnitt **Primary Key Join Columns** der **JPA Details**-Ansicht.

13. Zuordnen eines Attributs zu einer sekundären Tabelle

Gehen Sie folgendermaßen vor, um ein Klassenattribut zu einem Feld einer sekundären Tabelle zuzuordnen:

a. Ordnen Sie das Attribut wie oben unter "Zuordnen von Attributen" auf der vorherigen Seite beschrieben zu.

b. Wählen Sie im Abschnitt **Column** der JPA Details-Ansicht im Feld **Table** den Namen der sekundären Tabelle aus, um den Standardwert zu ersetzen.

14. Verwenden einer vorhandenen ORM-Datei als Basis

Führen Sie die folgenden Schritte aus, um eine vorhandene **orm.xml**-Datei als Basis für die Datei zu verwenden, die Sie entwickeln:

a. Stellen Sie sicher, dass alle CITs, die in der vorhandenen **orm.xml**-Datei zugeordnet sind, in das aktive Eclipse-Projekt importiert werden.

b. Wählen Sie alle oder einige der Entitätenzuordnungen aus der vorhandenen Datei aus und kopieren Sie sie.

c. Wählen Sie in der Eclipse JPA-Perspektive die Registerkarte **Source** der Datei **orm.xml** aus.

d. Fügen Sie unter dem Tag **<entity-mappings>** der bearbeiteten Datei **orm.xml** alle kopierten Entitätenzuordnungen unterhalb des Tags **<schema>** ein. Stellen Sie sicher, dass das Schema-Tag wie oben im Schritt "Erstellen der ORM-Datei – Zuordnen von UCMDB-Klassen zu Datenbanktabellen" auf Seite 116 beschrieben konfiguriert ist. Alle eingefügten Entitäten werden nun in der JPA Structure-Ansicht angezeigt. Ab jetzt können Zuordnungen sowohl grafisch als auch manuell über den XML-Code der Datei **orm.xml** bearbeitet werden.

e. Klicken Sie auf **Speichern**.

15. ## Importieren einer vorhandenen ORM-Datei von einem Adapter

Wenn ein Adapter bereits vorhanden ist, kann das Eclipse-Plugin zum grafischen Bearbeiten der ORM-Datei verwendet werden. Importieren Sie die Datei **orm.xml** in Eclipse, bearbeiten Sie sie mithilfe des Plugins und stellen Sie sie dann wieder auf dem UCMDB-Computer bereit. Klicken Sie zum Importieren der ORM-Datei auf die Schaltfläche in der Eclipse-Symbolleiste. Es wird ein Bestätigungsdialogfeld angezeigt. Klicken Sie auf **OK**. Die ORM-Datei wird vom UCMDB-Computer in das aktive Eclipse-Projekt kopiert und alle relevanten Klassen werden vom UCMDB-Klassenmodell importiert.

Wenn die relevanten Klassen nicht in der JPA Structure-Ansicht angezeigt werden, klicken Sie mit der rechten Maustaste in der Project Explorer-Ansicht auf das aktive Projekt, wählen Sie **Close** und dann **Open** aus.

Ab jetzt kann die ORM-Datei mithilfe von Eclipse grafisch bearbeitet und dann wieder wie unten unter "Bereitstellen der ORM-Datei in der CMDB" auf der nächsten Seite beschrieben auf dem UCMDB-Computer bereitgestellt werden.

16. ## Überprüfen der Richtigkeit der Datei "orm.xml" – Integrierte Richtigkeitsüberprüfung

Das Eclipse JPA-Plugin prüft, ob Fehler vorhanden sind, und markiert diese in der Datei **orm.xml**. Dabei wird sowohl auf Syntaxfehler (z. B. falscher Tagname, nicht geschlossenes Tag, fehlende ID) als auch auf Zuordnungsfehler (z. B. falscher Attributname oder falscher Feldname in der Datenbanktabelle) geprüft. Falls Fehler festgestellt werden, wird ihre Beschreibung in der Ansicht **Problems** angezeigt.

17. ## Erstellen eines neuen Integrationspunkts

Wenn für diesen Adapter kein Integrationspunkt in der CMDB vorhanden ist, können Sie ihn in Integration Studio erstellen. Weitere Informationen finden Sie unter "Integration Studio" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Geben Sie den Namen des Integrationspunkts in das angezeigte Dialogfeld ein. Die Datei **orm.xml** wird in den Adapterordner kopiert. Ein Integrationspunkt wird mit den gleichen importierten CI-Typen erstellt wie seine unterstützten Klassen, ausgenommen Multiknoten-

CITs, falls diese in der Datei **reconciliation_rules.txt** konfiguriert sind. Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 137.

18. **Bereitstellen der ORM-Datei in der CMDB**

Speichern Sie die Datei **orm.xml** und stellen Sie sie auf dem UCMDB-Server bereit, indem Sie auf **UCMDB > ORM bereitstellen** klicken. Die Datei **orm.xml** wird in den Adapterordner kopiert und der Adapter wird erneut geladen. Das Operationsergebnis wird im Dialogfeld **Operation Result** angezeigt. Falls während des erneuten Ladens ein Fehler auftritt, wird in dem Dialogfeld die Stapelablaufverfolgung der Java-Ausnahme angezeigt. Falls noch kein Integrationspunkt mithilfe des Adapters definiert wurde, werden bei der Bereitstellung keine Zuordnungsfehler erkannt.

19. **Ausführen einer TQL-Beispielabfrage**

a. Definieren einer Abfrage (keiner Ansicht) in Modeling Studio. Weitere Informationen finden Sie unter "Modeling Studio" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

b. Erstellen Sie einen Integrationspunkt mit dem Adapter, den Sie im Schritt "Erstellen eines neuen Integrationspunkts" erstellt haben. Weitere Informationen finden Sie unter "Dialogfeld "Neuer Integrationspunkt"/"Integrationspunkt bearbeiten"" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

c. Vergewissern Sie sich während der Erstellung des Adapters, dass die CI-Typen, die in die Abfrage einbezogen werden sollen, von diesem Integrationspunkt unterstützt werden.

d. Verwenden Sie bei der Konfiguration des CMDB-Plugins den Namen dieser Beispielabfrage im Dialogfeld **Settings**. Weitere Informationen finden Sie oben im Schritt "Konfigurieren des CMDB-Plugins" auf Seite 115.

e. Klicken Sie auf die Schaltfläche **Run TQL**, um eine Beispiel-TQL auszuführen, und prüfen Sie mithilfe der neu erstellten **orm.xml**-Datei, ob die erforderlichen Ergebnisse zurückgegeben werden

Adapterkonfigurationsdateien

Die in diesem Abschnitt behandelten Dateien befinden sich im Package **db-adapter.zip** im Ordner **C:\hp\UCMDB\UCMDBServer\content\adapters**.

In diesem Abschnitt werden die folgenden Konfigurationsdateien beschrieben:

- "Die Datei "fixed_values.txt"" auf Seite 141
- "Die Datei "persistence.xml"" auf Seite 142

Allgemeine Konfiguration

- **adapter.conf.** Die Konfigurationsdatei des Adapters. Weitere Informationen finden Sie unter "Die Datei "adapter.conf"" oben.

Einfache Konfiguration

- **simplifiedConfiguration.xml**. Konfigurationsdatei, die die Dateien **orm.xml**, **transformations.txt** und **reconciliation_rules.txt** durch weniger Funktionen ersetzt. Weitere Informationen finden Sie unter "Die Datei "simplifiedConfiguration.xml"" auf der nächsten Seite.

Erweiterte Konfiguration

- **orm.xml**. Die objektrelationale Zuordnungsdatei, in der Sie die Zuordnungen zwischen CMDB-CITs und Datenbanktabellen festlegen. Weitere Informationen finden Sie unter "Die Datei "orm.xml"" auf Seite 124.
- **reconciliation_types.txt**. Enthält die Regeln zur Konfiguration der Abstimmungstypen. Weitere Informationen finden Sie unter "Die Datei "reconciliation_types.txt"" auf Seite 137.
- **reconciliation_types.txt**. Enthält die Abstimmungsregeln. Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 137.
- **transformations.txt.** Die Transformationsdatei, in der Sie die Konverter für die Konvertierung zwischen den CMDB-Werten und den Datenbankwerten angeben. Weitere Informationen finden Sie unter "Die Datei "transformations.txt"" auf Seite 139.
- **Discriminator.properties.** Diese Datei ordnet jeden unterstützten CI-Typ einer kommagetrennten Liste der möglichen entsprechenden Werte zu. Weitere Informationen finden Sie unter "Die Datei "discriminator.properties"" auf Seite 140.
- **Replication_config.txt.**Diese Datei enthält eine kommagetrennte Liste der CI- und Beziehungstypen, deren Eigenschaftsbedingungen vom Replizierungs-Plugin unterstützt werden. Weitere Informationen finden Sie unter "Die Datei "replication_config.txt"" auf Seite 141.
- **Fixed_values.txt.**Mit dieser Datei können Sie feste Werte für bestimmte Attribute einiger CITs konfigurieren. Weitere Informationen finden Sie unter "Die Datei "fixed_values.txt"" auf Seite 141.

Hibernate-Konfiguration

- **persistence.xml.** Wird zum Überschreiben von Hibernate-Standardkonfigurationen verwendet. Weitere Informationen finden Sie unter "Die Datei "persistence.xml"" auf Seite 142.

Die Datei "adapter.conf"

Diese Datei enthält die folgenden Einstellungen:

- **use.simplified.xml.config=false.true**: Verwendet die Datei "simplifiedConfiguration.xml".

> **Hinweis:** Die Verwendung dieser Datei bedeutet, dass die Dateien `orm.xml`,

> `transformations.txt` und `reconciliation_rules.txt` durch weniger Funktionen ersetzt werden.

- **dal.ids.chunk.size=300.** Ändern Sie diesen Wert nicht.
- **dal.use.persistence.xml=false**. **true**: Der Adapter liest die Hibernate-Konfiguration aus der Datei `persistence.xml`.

  > **Hinweis:** Es wird empfohlen, die Hibernate-Konfiguration nicht zu überschreiben.

- **performance.memory.id.filtering=true**. Wenn GDBA TQLs ausführt, kann in einigen Fällen mittels SQL eine große Anzahl von IDs abgerufen und an die Datenbank zurückgesendet werden. Um diesen übermäßigen Arbeitsaufwand zu vermeiden und die Leistung zu verbessern, versucht GDBA die gesamte Ansicht/Tabelle zu lesen und filtert die Ergebnisse im Speicher.
- **id.reconciliation.cmdb.id.type=string/bytes**. Wenn Sie den allgemeinen DB-Adapter unter Verwendung der ID-Abstimmung zuordnen (Informationen hierzu finden im Schritt "Konfigurieren der Datei "reconciliation_types.txt" (für die Standard-Zuordnungs-Engine von UCMDB 9.0x)" unter "Implementieren der Zuordnungs-Engine" auf Seite 185), können Sie **cmdb_id** entweder dem Spaltentyp **string** oder dem Spaltentyp **bytes/raw** zuordnen, indem Sie die Eigenschaft **META-INF/adapter.conf** ändern.
- **performance.enable.single.sql=true**. Dies ist ein optionaler Parameter. Wenn er nicht in der Datei angegeben ist, lautet der Standardwert **true**. Die Einstellung **true** bewirkt, dass der allgemeine Datenbankadapter versucht, eine einzelne SQL-Anweisung für jede Abfrage zu generieren, die ausgeführt wird (entweder für Auffüllungs- oder für föderierte Abfragen). Mit einer einzelnen SQL-Anweisung werden die Leistung und die Speicherauslastung des allgemeinen Datenbankadapters verbessert. Die Einstellung **false** bewirkt, dass der allgemeine Datenbankadapter mehrere SQL-Anweisungen generiert, die unter Umständen mehr Zeit und mehr Speicher in Anspruch nehmen als eine einzelne Anweisung. Selbst wenn dieses Attribut auf **true** gesetzt ist, wird in folgenden Szenarios keine einzelne SQL-Anweisung generiert:
  - Die Datenbank, zu der der Adapter eine Verbindung herstellt, ist keine Oracle- oder SQL Server-Datenbank.
  - Die auszuführende TQL enthält eine andere Kardinalitätsbedingung als 0..* und 1..* (z. B. eine Kardinalitätsbedingung wie 2..* oder 0..2).
- **in.expression.size.limit=950** (Standardeinstellung). Dieser Parameter teilt den Ausdruck 'IN' der ausgeführten SQL, wenn die Größenbeschränkung der Argumentenliste erreicht ist.

Die Datei "simplifiedConfiguration.xml"

Diese Datei wird für die einfache Zuordnung von UCMDB-Klassen zu Datenbanktabellen verwendet. Um auf die Vorlage zum Bearbeiten der Datei zuzugreifen, navigieren Sie zu **Adapterverwaltung > db-adapter > Konfigurationsdateien**.

Dieser Abschnitt umfasst die folgenden Themen:

- "Vorlage für die Datei "simplifiedConfiguration.xml"" auf der nächsten Seite
- "Einschränkungen" auf Seite 124

Vorlage für die Datei "simplifiedConfiguration.xml"

Die Eigenschaft **CMDB-class-name** ist der Multiknoten-Typ (der Knoten, mit dem föderierte CITs in der TQL verbunden werden):

```
<?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="[table_
name]">
        <primary-key column-name="[column_name]" />
```

reconciliation-by-two-nodes. Die Abstimmung kann unter Verwendung von einem oder zwei Knoten erfolgen. In diesem Beispiel verwendet die Abstimmung zwei Knoten.

connected-node-CMDB-class-name. Der zweite Klassentyp, der in der Abstimmungs-TQL erforderlich ist.

CMDB-link-type. Der Beziehungstyp, der in der Abstimmungs-TQL erforderlich ist.

link-direction. Die Richtung der Beziehung in der Abstimmungs-TQL (von `node` zu `ip_address` oder von `ip_address` zu `node`):

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

Der Abstimmungsausdruck besteht aus mehreren OR-Elementen und jedes OR-Element enthält mehrere AND-Elemente.

is-ordered. Bestimmt, ob die Abstimmung der Reihe nach oder durch einen regulären `OR`-Vergleich durchgeführt wird.

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

Wenn die Abstimmungseigenschaft von der Hauptklasse (dem Multiknoten) abgerufen wird, verwenden Sie das **attribute**-Tag, andernfalls verwenden Sie das **connected-node-attribute**-Tag.

ignore-case.true: Beim Vergleich der Daten im UCMDB-Klassenmodell mit den Daten im RDBMS wird nicht zwischen Groß- und Kleinschreibung unterschieden:

```
                <attribute CMDB-attribute-name="name" column-name="
[column_name]" ignore-case="true"/>
```

Der Spaltenname ist der Name der Fremdschlüsselspalte (d. h. der Spalte mit den Werten, die auf die Primärschlüsselspalte des Multiknotens verweisen).

Wenn die Primärschlüsselspalte des Multiknotens aus mehreren Spalten besteht, sind mehrere Fremdschlüsselspalten erforderlich, eine für jede Primärschlüsselspalte.

```
        <foreign-primary-key column-name="[column_name]" CMDB-class-
primary-key-column="[column_name]"/>
```

Wenn es wenige Primärschüsselspalten gibt, duplizieren Sie diese Spalte.

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

Die Eigenschaften **from-CMDB-converter** und **to-CMDB-converter** sind Java-Klassen, die die folgenden Schnittstellen implementieren:

- com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB
- com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB

Verwenden Sie diese Konverter, wenn die Werte in der CMDB und in der Datenbank nicht identisch sind.

In diesem Beispiel wird `GenericEnumTransformer` verwendet, um den Enumerator entsprechend der in Klammern angegebenen XML-Datei zu konvertieren (**generic-enum-transformer-example.xml**):

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

Einschränkungen

- Können verwendet werden, um nur TQL-Abfragen mit einem Knoten (in der Datenbankquelle) zuzuordnen. Sie können beispielsweise eine `Knoten > Ticket`- und eine `Ticket`-TQL-Abfrage ausführen. Um die Knotenhierarchie aus der Datenbank zu lesen, müssen Sie die erweiterte **orm.xml**-Datei verwenden.
- Es werden nur 1:n-Beziehungen unterstützt. Sie können beispielsweise ein oder mehrere Tickets bei jedem Knoten verwenden. Es ist jedoch nicht möglich, Tickets zu verwenden, die zu mehreren Knoten gehören.
- Es ist nicht möglich, die gleiche Klasse mit verschiedenen Typen von CMDB-CITs zu verbinden. Wenn Sie beispielsweise definieren, dass `Ticket` mit `Knoten` verbunden werden soll, kann es nicht gleichzeitig mit `Applikation` verbunden werden.

Die Datei "orm.xml"

Diese Datei wird für die Zuordnung von CMDB-CITs zu Datenbanktabellen verwendet.

Eine Vorlage zum Erstellen einer neuen Datei befindet sich im Verzeichnis **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF**.

Um die XML-Datei für einen bereitgestellten Adapter zu bearbeiten, navigieren Sie zu **Adapterverwaltung > db-adapter > Konfigurationsdateien**.

Dieser Abschnitt umfasst die folgenden Themen:

Vorlage für die Datei "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>Generic DB adapter orm</description>
```

Ändern Sie den Package-Namen nicht.

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

entity. Der Name des CMDB-CIT. Dies ist die Multiknoten-Entität.

Stellen Sie sicher, dass **class** das Präfix **generic_db_adapter.** enthält.

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

Verwenden Sie eine sekundäre Tabelle, wenn die Entität mehreren Tabellen zugeordnet ist.

```
        <secondary-table name="" />
        <attributes>
```

Verwenden Sie für die Vererbung einer einzelnen Tabelle mit Diskriminator den folgenden Code:

```
        <inheritance strategy="SINGLE_TABLE" />
        <discriminator-value>node</discriminator-value>
        <discriminator-column name="[column_name]" />
```

Attribute mit dem Tag **id** sind die Primärschlüsselspalten. Achten Sie darauf, dass folgende Benennungskonvention für diese Primärschlüsselspalten verwendet wird: **idX** (id1, id2 usw.), wobei **X** der Spaltenindex im Primärschlüssel ist.

```
            <id name="id1">
```

Ändern Sie nur den Spaltennamen des Primärschlüssels.

```
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
```

basic. Wird zum Deklarieren der CMDB-Attribute verwendet. Stellen Sie sicher, dass Sie nur die Eigenschaften **name** und **column_name** bearbeiten.

```
            <basic name="name">
                <column updatable="false" insertable="false" name="
[column_name]" />
            </basic>
```

Ordnen Sie für die Vererbung einer einzelnen Tabelle mit Diskriminator die Erweiterungsklassen wie folgt zu:

```
    <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="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
```

Das folgende Beispiel zeigt einen CMDB-Attributnamen ohne Präfix:

```
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="
[column_name]" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="
[column_name]" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="
[column_name]" />
            </basic>
        </attributes>
    </entity>
```

Dies ist eine Beziehungsentität. Die Benennungskonvention ist **Ende1Typ_LinkTyp_Ende2Typ**. In diesem Beispiel wird **Ende1Typ** durch **node** und **LinkTyp** durch **composition** repräsentiert.

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

Die Zielentität ist die Entität, auf die diese Eigenschaft verweist. In diesem Beispiel ist **Ende1** der Entität **node** zugeordnet.

many-to-one. Viele Beziehungen können mit einem Knoten verbunden werden.

join-column. Die Spalte, die die **Ende1**-IDs enthält (d. h. die IDs der Zielentität).

referenced-column-name. Der Name der Spalte in der Zielentität (**node**), die die in der Join-Spalte verwendeten IDs enthält.

```
            <many-to-one target-entity="node" name="end1">
                <join-column updatable="false" insertable="false"
referenced-column-name="[column_name]" name="[column_name]" />
            </many-to-one>
```

one-to-one. Eine Beziehung kann mit der Zielentität **[CMDB_class_name]** verbunden werden.

```
            <one-to-one target-entity="[CMDB_class_name]"
name="end2">
                <join-column updatable="false" insertable="false"
```

```
referenced-column-name="" name="[column_name]" />
            </one-to-one>
        </attributes>
    </entity>
</entity-mappings>
```

node attribute. Dies ist ein Beispiel zur Vorgehensweise zum Hinzufügen eines Knotenattributs.

```
<entity class="generic_db_adapter.host_node">
  <discriminator-value>host_node</discriminator-value>
    <attributes/>
</entity>
<entity class="generic_db_adapter.nt">
  <discriminator-value>nt</discriminator-value>
    <attributes>
      <basic name="nt_servicepack">
        <column updatable="false" insertable="false" name="specific_type_
value"/>
      </basic>
    </attributes>
</entity>
```

Mehrere ORM-Dateien

Es werden mehrere Zuordnungsdateien unterstützt. Der Name jeder Zuordnungsdatei sollte mit **orm.xml** enden. Alle Zuordnungsdateien sollten im Ordner META-INF des Adapters gespeichert werden.

Benennungskonventionen

- In jeder Entität muss die Klasseneigenschaft mit der Namenseigenschaft mit dem Präfix `generic_db_adapter` übereinstimmen.
- Primärschlüsselspalten müssen Namen in der Form **idX** haben, wobei gilt: **X = 1**, **2**, ... entsprechend der Anzahl der Primärschlüssel in der Tabelle.
- Attributnamen müssen mit Klassenattributnamen übereinstimmen. Dies gilt auch für Groß- und Kleinschreibung.
- Der Beziehungsname hat die Form `Ende1Typ_LinkTyp_Ende2Typ`.
- CMDB-CITs, die auch reservierte Wörter in Java sind, sollten mit dem Präfix **gdba_** versehen werden. Für den CMDB-CIT **goto** beispielsweise sollte die ORM-Entität **gdba_goto** genannt werden.

Verwenden von Inline-SQL-Anweisungen anstelle von Tabellennamen

Sie können Entitäten zu eingebundenen `select`-Klauseln zuordnen anstatt zu Datenbanktabellen. Dies entspricht der Vorgehensweise, eine Ansicht in der Datenbank zu definieren und dieser Ansicht eine Entität zuzuordnen. Beispiel:

```
    <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)" />
```

In diesem Beispiel sollten die Knotenattribute den Spalten **id1**, **name** und **host_os** anstatt den Spalten **id**, **name** und **os** zugeordnet werden.

Es gelten die folgenden Einschränkungen:

- Die Inline-SQL-Anweisung ist nur verfügbar, wenn Hibernate als JPA-Provider verwendet wird.
- Die Inline-SQL-Select-Klausel muss in runde Klammern eingeschlossen werden.
- Das Element **<schema>** sollte nicht in der Datei **orm.xml** enthalten sein. Im Fall von Microsoft SQL Server 2005 bedeutet dies, dass alle Tabellennamen mit dem Präfix `dbo.` versehen und nicht global durch `<schema>dbo</schema>` definiert werden sollten.

Das Schema "orm.xml"

In der folgenden Tabelle werden die allgemeinen Elemente der Datei **orm.xml** erläutert. Das vollständige Schema finden Sie unter http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. Die Liste ist nicht vollständig. In erster Linie wird das spezifische Verhalten der standardmäßigen Java Persistence API für den allgemeinen Datenbankadapter erläutert.

Elementname und -pfad	Beschreibung	Attribute
entity-mappings	Das Stammelement für das Entitätenzuordnungsdokument. Dieses Element sollte exakt mit dem Element in der GDBA-Beispieldatei übereinstimmen.	
description (entity-mappings)	Eine frei formulierte Beschreibung des Entitätenzuordnungsdokuments. (Optional)	
package (entity-mappings)	Der Name des Java-Package, das die Zuordnungsklassen enthält. Er sollte immer den Text `generic_db_adapter` enthalten.	1. **Name:** name **Beschreibung:** Der Name des UCMDB CI-Typs, dem diese Entität zugeordnet ist. Wenn die Entität einem Link in der CMDB zugeordnet ist, sollte der Name der Entität das folgende Format

Elementname und -pfad	Beschreibung	Attribute
		aufweisen: `<Ende_1>_<Link_Name>_<Ende_2>`. Beispielweise definiert `node_composition_cpu` eine Entität, die dem Verbundlink zwischen einem Knoten und einer CPU zugeordnet wird. Wenn der Name des CI-Typs dem Namen der Java-Klasse ohne Package-Präfix entspricht, kann dieses Feld ausgelassen werden. **Erforderlich?:** Optional **Typ:** Zeichenkette 2. **Name:** class **Beschreibung:** Der vollqualifizierte Name der Java-Klasse, die für diese DB-Entität erstellt wird. Der Name des Java-Klassen-Packages sollte identisch sein mit dem im Element `package` angegebenen Namen. Es dürfen keine reservierten Java-Wörter wie **interface** oder **switch** als Klassenname verwendet werden. Fügen Sie stattdessen das Präfix `gdba_` zum Namen hinzu (d. h., aus **interface** wird dann `generic_db_adapter.gdba_interface`. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
table (entity-mappings>entity)	Dieses Element definiert die primäre Tabelle der DB-Entität. Kann nur einmal erscheinen. Erforderlich.	**Name:** name **Beschreibung:** Der Name der primären Tabelle. Wenn der Name der Tabelle nicht das zugehörige Schema enthält, wird die Tabelle nur in dem Schema des Benutzers gesucht, das zum Erstellen des Integrationspunkts verwendet wurde. Es kann sich hierbei auch um eine gültige SELECT-Anweisung handeln. Eine SELECT-Anweisung muss in

Elementname und -pfad	Beschreibung	Attribute
		Klammern eingeschlossen werden. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
secondary-table (entity-mappings > entity)	Dieses Element kann zur Definition einer sekundären Tabelle für die DB-Entität verwendet werden. Die Tabelle muss über eine 1:1-Beziehung mit der primären Tabelle verbunden werden. Sie können mehrere sekundäre Tabellen definieren. Optional.	**Name:** name **Beschreibung:** Der Name der sekundären Tabelle. Wenn der Name der Tabelle nicht das zugehörige Schema enthält, wird die Tabelle nur in dem Schema des Benutzers gesucht, das zum Erstellen des Integrationspunkts verwendet wurde. Es kann sich hierbei auch um eine gültige SELECT-Anweisung handeln. Eine SELECT-Anweisung muss in Klammern eingeschlossen werden. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
primary-key-join-column (entity-mappings > entity > secondary-table)	Wenn die sekundäre und die primäre Tabelle nicht mithilfe von gleichnamigen Feldern verbunden sind, definiert dieses Element den Namen des Primärschlüsselfelds in der sekundären Tabelle, das mit dem Primärschlüsselfeld der primären Tabelle verbunden werden muss.	**Name:** name **Beschreibung:** Der Name des Primärschlüsselfelds in der sekundären Tabelle. Falls dieses Element nicht vorhanden ist, geht das System davon aus, dass das Primärschlüsselfeld den gleichen Namen hat wie das Primärschlüsselfeld der primären Tabelle. **Erforderlich?:** Optional **Typ:** Zeichenkette
inheritance (entity-mappings > entity)	Wenn die aktuelle Entität die übergeordnete Entität für eine Familie von DB-Entitäten ist, wird sie mit diesem Element entsprechend gekennzeichnet. Optional.	**Name:** strategy **Beschreibung:** Definiert, auf welche Weise die Vererbung in Ihrer DB implementiert wird. **Erforderlich?:** Erforderlich **Typ:** Einer der folgenden Werte: • SINGLE_TABLE: Diese Entität und alle untergeordneten Entitäten befinden sich in der gleichen Tabelle. • JOINED: Die untergeordneten Entitäten befinden sich in verknüpften Tabellen.

Elementname und -pfad	Beschreibung	Attribute
		• TABLE_PER_CLASS: Jede Entität wird vollständig durch eine separate Tabelle definiert.
discriminator-column (entity-mappings > entity)	Bei einer Vererbung des Typs SINGLE_TABLE wird mithilfe dieses Elements der Name des Felds definiert, mit dem der Entitätstyp für jede Zeile bestimmt wird.	**Name:** name **Beschreibung:** Der Name der Diskriminatorspalte. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
discriminator-value (entity-mappings > entity)	Dieses Element definiert den Typ der spezifischen Entität in der Vererbungsstruktur. Der Name muss identisch sein mit dem Namen, der in der Datei **discriminator.properties** für die Wertgruppe dieses spezifischen Entitätstyps definiert wurde.	
attributes (entity-mappings > entity)	Das Stammelement für alle Attributzuordnungen für eine Entität.	
id (entity-mappings > entity attributes)	Dieses Element definiert das Schlüsselfeld für die Entität. Es muss mindestens ein ID-Feld definiert werden. Wenn mehrere ID-Elemente vorhanden sind, erstellen die Felder einen Verbundschlüssel für die Entität. Nach Möglichkeit sollten Verbundschlüssel für CI-Entitäten vermieden werden (dies gilt nicht für Links).	**Name:** name **Beschreibung:** Eine Zeichenkette des Typs **idX**, wobei X eine Zahl zwischen 1 und 9 ist. Die erste ID sollte als **id1** gekennzeichnet werden, die zweite als **id2** usw. Es handelt sich hierbei NICHT um den Namen des Schlüsselattributs in UCMDB. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
basic (entity-mappings > entity attributes)	Dieses Element definiert eine Zuordnung zwischen einem Feld in der Tabelle, das nicht Teil des Primärschlüssels der Tabelle ist, und einem UCMDB-Attribut.	**Name:** name **Beschreibung:** Der Name des UCMDB-Attributs, dem das Feld zugeordnet ist. Dieses Attribut muss im UCMDB CI-Typ vorhanden sein, dem die aktuelle Entität zugeordnet ist. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette

Elementname und -pfad	Beschreibung	Attribute
column (entity-mappings > entity > attributes >id Oder (entity-mappings > entity > attributes > basic)	Definiert den Namen der Tabellenspalte für die grundlegende Zuordnung oder definiert ein ID-Feld.	1. **Name:** name **Beschreibung:** Der Name des Felds. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** table **Beschreibung:** Der Name der Tabelle, zu der das Feld gehört. Es muss sich hierbei entweder um die primäre Tabelle oder um eine der für die Entität definierten sekundären Tabellen handeln. Wenn dieses Attribut ausgelassen wird, geht das System davon aus, dass das Feld zur primären Tabelle gehört. **Verwendung:** Optional **Typ:** Zeichenkette
one-to-one (entity-mappings > entity > attributes)	Definiert eine Spalte, deren Wert sich in einer anderen Tabelle befindet, wobei die beiden Tabellen über eine 1:1-Beziehung miteinander verbunden sind. Dieses Element wird nur für die Zuordnung von Linkentitäten unterstützt und nicht für andere CI-Typen. Dies ist die einzige Möglichkeit, um eine Zuordnung zwischen einer Tabelle und einem UCMDB-Link zu definieren.	1. **Name:** name **Beschreibung:** Gibt an, welches der beiden Enden dieses Feld repräsentiert. **Erforderlich?:** Erforderlich **Typ:** Entweder `Ende1` oder `Ende2` 2. **Name:** target-entity **Beschreibung:** Der Name der Entität, auf die sich das Ende bezieht. **Erforderlich?:** Erforderlich **Typ:** Einer der Entitätsnamen, die im Entitätenzuordnungsdokument definiert sind.
join-column (entity-mappings > entity attributes > one-to-one)	Definiert, auf welche Weise die Zielentität, die im übergeordneten 1:1-Element definiert ist, und die aktuelle Entität miteinander verknüpft werden.	1. **Name:** name **Beschreibung:** Der Name des Felds in der aktuellen Tabelle, das für die 1:1-Verknüpfung verwendet wird. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** name

Elementname und -pfad	Beschreibung	Attribute
		Beschreibung: Der Name eines Felds in der verknüpften Entität, von dem die Verknüpfung durchgeführt werden soll. Wenn dieses Attribut ausgelassen wird, geht das System davon aus, dass die verknüpfte Tabelle eine Spalte mit dem gleichen Namen hat wie das im Namensattribut definierte Feld. **Erforderlich?:** Optional **Typ:** Zeichenkette

Beispiel für das Erstellen der Datei "orm.xml"

Das hier gezeigte Beispiel verdeutlicht die Vorgehensweise beim Erstellen der Datei **orm.xml**. In diesem Beispiel werden SQL-Tabellen in einer Remote-Datenbank CI-Typen in UCMDB zugeordnet.

Erstellen Sie Tabellen mit dem folgenden Format in der Remote-Datenbank, füllen Sie die Tabelle **Hosts** mit Knoten auf, die Tabelle **IP_Addresses** mit IP-Adressen und erstellen Sie wie folgt Links zwischen den Knoten und IP-Adressen:

Tabelle 'Hosts'

host_name	host_id
Test1	1
Test2	2
Test3	3

Tabelle 'IP_Adresses'

ip_address	ip_id
10.1.1.1	1
10.2.2.2	2
10.3.3.2	3
10.4.4.4	4

Tabelle 'Host_IP_Link' (Links zwischen Knoten und IP-Adressen)

host_id	ip_id
1	1
2	2
2	3
3	4

Der Primärschlüssel für die Tabelle **Hosts** ist das Feld **host_id** und der Primärschlüssel für die Tabelle **IP_Addresses** ist das Feld **ip_id**. In der Tabelle **Host_IP_Link** sind **host_id** und **ip_id** Fremdschlüssel aus den Tabellen **Hosts** und **IP_Addresses**.

Erstellen Sie entsprechend den oben stehenden Tabellen die Datei **orm.xml**. Gehen Sie dazu wie folgt vor: Die in diesem Beispiel verwendeten Entitäten sind **node**, **ip_address** und **node_containment_ip_address**.

1. Erstellen Sie die Entität **node**, indem Sie **host_id** von der Tabelle **Hosts** wie folgt zuordnen:

```
<entity-mappings
xmlns="http://java.sun.com/xml/ns/persistence/orm"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="1.0"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm_
1_0.xsd">
    <description>test_integration</description>
    <package>generic_db_adapter</package>
    <entity class="generic_db_adapter.node">
      <table name="Hosts"/>
      <attributes>
        <id name="id1">
                <column updatable="false" insertable="false" name="host_id"/>
                <generated-value strategy="TABLE"/>
        </id>
        <basic name="name">
                <column updatable="false" insertable="false" name="host_
name"/>
        </basic>
      </attributes>
    </entity>
```

Die Entität **class** muss ein CI-Typ sein, der bereits in UCMDB vorhanden ist. Die Tabelle **name** ist die Tabelle in der Datenbank, die sowohl eine ID als auch die Hostinformationen enthält. Das ID-Attribut ist für die Identifizierung bestimmter Hosts erforderlich und wird zu einem späteren Zeitpunkt in der Zuordnung verwendet. In diesem Beispiel wird das Attribut **name** dieser Entität mit der Spalte **host_name** der Tabelle **hosts** aufgefüllt.

2. Für die nächste Entität ordnen wir IP-Adressen aus der Tabelle **Interfaces** zu.

```
<entity name="ip_address" class="generic_db_adapter.ip_address">
    <table name="IP_Addresses"/>
    <attributes>
      <id name="id1">
        <column insertable="false" updatable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
      </id>
      <basic name="name">
        <column updatable="false" insertable="false" name="ip_
address"/>
      </basic>
    </attributes>
</entity>
```

3. Als nächstes muss der Link zwischen Knoten und IP-Adresse mittels der Zuordnungstabelle erstellt werden und auf das Feld **ip_id** verweisen (bei Bedarf kann auch auf beide Felder, d. h. **host_id** und **ip_id**, verwiesen werden).

```
<entity name="node_containment_ip_address"
        class="generic_db_adapter.node_containment_ip_address">
    <table name="Host_IP_Link"/>
    <attributes>
      <id name="id1">
        <column updatable="false" insertable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
      </id>
      <many-to-one target-entity="node" name="end1">
        <join-column name="host_id"/>
      </many-to-one>
      <one-to-one target-entity="ip_address" name="end2">
```

```
        <join-column name="ip_id"/>
      </one-to-one>
    </attributes>
</entity>
```

Der Entitätsname für den Container weist das folgende Format auf: [end1 CIT]_[link CIT]_ [end2 CIT]. Da der Link-CI-Typ **containment** lautet, lautet in diesem Beispiel der Entitätsname für den Container **node_containment_ip_address** und die Entitätsklasse **generic_db_ adapter.node_containment_ip_address**. Die ID ist in diesem Codeblock erforderlich. Dieses Beispiel verwendet eine einzige ID der Schnittstelle, jedoch könnten auch beide Spalten auf **id1** und **id2** verweisen. Der Code würde in dem Fall folgendermaßen lauten:

```
<id name=”id1”>
    <column updatable=”false” insertable=”false” name=”ip_id”/>
    <generated-value strategy=”TABLE”/>
</id>
<id name=”id2”>
    <column updatable=”false” insertable=”false” name=”host_
    id”/>
    <generated-value strategy=”TABLE”/>
</id>
```

Die zwei Enden dieses Links sind n:1 und 1:1. Das bedeutet, dass jede IP-Adresse mit nur einem Knoten verknüpft wird, ein Knoten jedoch mit vielen IP-Adressen verknüpft sein kann. Die verwendeten Spalten stammen aus der Tabelle **Links** und verweisen auf die Tabellen **Hosts** und **Interfaces**.

Die Datei "reconciliation_types.txt"

Diese Datei wird zum Konfigurieren der Abstimmungstypen verwendet.

Jede Zeile in der Datei stellt einen CMDB-CIT dar, der mit einem CIT der föderierten Datenbank in der TQL-Abfrage verbunden ist.

Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)

Diese Datei wird zum Konfigurieren der Abstimmungsregeln verwendet, wenn Sie eine Abstimmung durchführen möchten und die DBMappingEngine im Adapter konfiguriert ist. Wenn Sie die DBMappingEngine nicht verwenden, kommt der allgemeine UCMDB-Abstimmungsmechanismus zum Einsatz. In diesem Fall ist es nicht erforderlich, diese Datei zu konfigurieren.

Jede Zeile in der Datei stellt eine Regel dar. Beispiel:

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

Der Wert `multinode` wird mit dem Namen des Multiknotens gefüllt (der CMDB-CIT, der mit dem CIT der föderierten Datenbank in der TQL-Abfrage verbunden ist).

Dieser Ausdruck beinhaltet die Logik, die entscheidet, ob zwei Multiknoten identisch sind (ein Multiknoten in der CMDB und der andere in der Datenbankquelle).

Der Ausdruck besteht aus mehreren `OR`- oder `AND`-Elementen.

Die Konvention bezüglich der Attributnamen im Ausdrucksteil lautet `[KlassenName].[AttributName]`. Beispiel: `AttributName` in der Klasse `ip_address` wird wie folgt geschrieben: `ip_address.name`.

Verwenden Sie für eine Übereinstimmung fester Ordnung (wenn der erste `OR`-Unterausdruck eine Antwort zurückgibt, dass die Multiknoten nicht identisch sind, wird der zweite OR-Unterausdruck nicht verglichen), den Parameter `ordered expression` anstatt `expression`.

Um die Groß-/Kleinschreibung bei einem Vergleich zu ignorieren, verwenden Sie das Steuerzeichen (**^**).

Die Parameter `end1_type`, `end2_type` und `link_type` werden nur verwendet, wenn die Abstimmungs-TQL zwei Knoten und nicht nur einen Multiknoten enthält. In diesem Fall lautet die Abstimmungs-TQL-Abfrage `end1_type > (link_type) > end2_type`.

Es ist nicht erforderlich, das relevante Layout hinzuzufügen, da dieses vom Ausdruck übernommen wird.

Arten von Abstimmungsregeln

Abstimmungsregeln haben die Form von OR- und AND-Bedingungen. Sie können diese Regeln bei mehreren verschiedenen Knoten definieren (z. B. wird ein Knoten durch den `KnotennamenUND/ODER` den `Namen der IP-Adresse` identifiziert).

Es gibt folgende Möglichkeiten, nach einer Übereinstimmung zu suchen:

- **Übereinstimmung fester Ordnung.** Der Abstimmungsausdruck wird von links nach rechts gelesen. Zwei `OR`-Unterausdrücke werden als identisch eingestuft, wenn sie identische Werte haben. Zwei `OR`-Unterausdrücke werden als nicht identisch eingestuft, wenn ihre Werte nicht identisch sind. Für alle anderen Fälle gibt es keine Entscheidung und der nächste `OR`-Unterausdruck wird auf Gleichheit getestet.

  Knotenname ODER Name der IP-Adresse. Wenn sowohl die CMDB als auch die Datenquelle die Eigenschaft `name` enthalten und diese bei beiden identisch ist, werden die Knoten als identisch eingestuft. Wenn die `name`-Eigenschaft bei beiden vorhanden, aber nicht identisch ist, werden die Knoten als nicht identisch betrachtet, ohne den Test mit der Eigenschaft `name` von `ip_address` fortzusetzen. Wenn die CMDB oder die Datenquelle keine `name`-Eigenschaft von `node` aufweisen, wird die `name`-Eigenschaft von `ip_address` geprüft.

- **Reguläre Übereinstimmung.** Wenn Gleichheit in einem der `OR`-Unterausdrücke besteht, werden die CMDB und die Datenquelle als identisch eingestuft.

  Knotenname ODER Name der IP-Adresse. Wenn keine Übereinstimmung bei der `name`-Eigenschaft von `node` besteht, wird die `name`-Eigenschaft von `ip_address` auf Gleichheit geprüft.

Für komplexe Abstimmungen, bei denen die Abstimmungsentität im Klassenmodell als mehrere CITs mit Beziehungen modelliert wird (z. B. `node`), enthält die Zuordnung eines Superset-Knotens alle relevanten Attribute von allen modellierten CITs.

Hinweis: Infolgedessen gibt es eine Einschränkung, dass alle Abstimmungsattribute in der Datenquelle in Tabellen abgelegt werden sollten, die den gleichen Primärschlüssel nutzen.

Eine andere Einschränkung gibt an, dass die Abstimmungs-TQL-Abfrage nicht mehr als zwei Knoten haben sollte. Beispiel: Die TQL `node > ticket` hat einen Knoten in der CMDB und ein Ticket in der Datenquelle.

Um die Ergebnisse abzustimmen, muss die `name`-Eigenschaft von `node` und/oder `ip_address` abgerufen werden.

Wenn `name` in der CMDB das Format `*.m.com` hat, kann ein Konverter zwischen der CMDB und der föderierten Datenbank verwendet werden, um die Werte zu konvertieren.

Die Spalte `node_id` in der Datenbanktickettabelle wird verwendet, um eine Verbindung zwischen den Entitäten herzustellen (die definierte Zuordnung kann auch in einer Knotentabelle vorgenommen werden):

DB-Knoten	
PK	node_id
	name

DB IP_Address	
PK	ip_id
	name

DB-Ticket	
PK	ticket_id
	node_id

Hinweis: Die drei Tabellen müssen Teil der föderierten RDBMS-Quelle und nicht der CMDB-Datenbank sein.

Die Datei "transformations.txt"

Diese Datei enthält alle Konverterdefinitionen.

Jede Zeile enthält eine neue Definition.

Vorlage für die Datei "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. Der Name der Entität, wie er in der Datei `orm.xml` erscheint.

attribute. Der Name des Attributs, wie er in der Datei `orm.xml` erscheint.

to_DB_class. Der vollständige, qualifizierte Name einer Klasse, die die Schnittstelle **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB** implementiert. Die Elemente in runden Klammern werden an diesen Klassenkonstruktor übergeben. Verwenden Sie diesen Konverter, um CMDB-Werte in Datenbankwerte umzuwandeln, z. B. um das Suffix **.com** an jeden Knotennamen anzuhängen.

from_DB_class. Der vollständige, qualifizierte Name einer Klasse, die die Schnittstelle **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB** implementiert. Die Elemente in runden Klammern werden an diesen Klassenkonstruktor übergeben. Verwenden Sie diesen Konverter, um Datenbankwerte in CMDB-Werte umzuwandeln, z. B. um das Suffix **.com** an jeden Knotennamen anzuhängen.

Weitere Informationen finden Sie unter "Standardkonverter" auf Seite 143.

Die Datei "discriminator.properties"

Diese Datei ordnet jeden unterstützten CI-Typ (der auch als Diskriminatorwert in der Datei "orm.xml" verwendet wird) entweder einer kommagetrennten Liste möglicher entsprechender Werte der Diskriminatorspalte oder einer Bedingung zu, um mögliche Werte der Diskriminatorspalte abzugleichen.

Verwenden Sie bei einer Bedingung die folgende Syntax: `Wie(Bedingung)`, wobei `Bedingung` eine Zeichenkette ist, die die folgenden Platzhalter enthalten kann:

- **%** (Prozentzeichen) - Ermöglicht den Abgleich mit einer Zeichenkette beliebiger Länge (einschließlich einer Zeichenkette der Länge Null)
- **_** (Unterstrich) - Ermöglicht den Abgleich mit einem einzelnen Zeichen

Beispiel: `Wie(%unix%)` stimmt überein mit unix, linux, unix-aix usw. Wie-Bedingungen können nur auf Zeichenkettenspalten angewendet werden.

Sie können auch einen einzelnen Diskriminatorwert zu einem beliebigen Wert zuordnen, der keinem anderen Diskriminator angehört, indem Sie `'all-other'` angeben.

Wenn der Adapter, den Sie erstellen, Diskriminatorfunktionen verwendet, müssen Sie alle Diskriminatorwerte in der Datei **discriminator.properties** definieren.

Beispiel für die Diskriminatorzuordnung:

Nehmen Sie an, der Adapter unterstützt die CI-Typen `node`, `nt` und `unix` und die Datenbank enthält eine einzelne Tabelle mit der Bezeichnung `t_nodes`, die wiederum eine Spalte mit der Bezeichnung **type** enthält. Wenn der Typ "10001" lautet, stellt die Zeile einen Knoten dar; wenn er "10004" lautet, einen Unix-Computer usw. Die Datei **discriminator.properties** könnte folgendermaßen aussehen:

```
node=10001, 10005 nt=10002,10003 unix=2% mainframe=all-other
```

Die Datei **orm.xml** enthält den folgenden Code:

```
<entity class="generic_db_adapter.node" >
    <table name="t_nodes" />
    ...
    <inheritance strategy="SINGLE_TABLE" />
    <discriminator-value>node</discriminator-value>
    <discriminator-column name="type" />
    ...
</entity>
<entity class="generic_db_adapter.nt" name="nt">
    <discriminator-value>nt</discriminator-value>
    <attributes>
</entity>
<entity class="generic_db_adapter.unix" name="unix">
    <discriminator-value>unix</discriminator-value>
    <attributes>
</entity>
```

Das Attribut "discriminator_column" wird wie folgt berechnet:

- Wenn **type** den Wert `10002` oder `10003` für einen bestimmten Eintrag enthält, wird der Eintrag dem CIT **nt** zugeordnet.
- Wenn **type** den Wert `10001` oder `10005` für einen bestimmten Eintrag enthält, wird der Eintrag dem CIT **node** zugeordnet.
- Wenn **type** für einen bestimmten Eintrag mit `2` beginnt, wird der Eintrag dem CIT **unix** zugeordnet.
- Jeder andere Wert in der Spalte **type** wird dem CIT **mainframe** zugeordnet.

> **Hinweis:** Der CIT **node** ist auch der übergeordnete CIT der CITs **nt** und **unix**.

Die Datei "replication_config.txt"

Diese Datei enthält eine kommagetrennte Liste der CI- und Beziehungstypen, deren Eigenschaftsbedingungen vom Replizierungs-Plugin unterstützt werden. Weitere Informationen finden Sie unter "Plugins" auf Seite 147.

Die Datei "fixed_values.txt"

Mit dieser Datei können Sie feste Werte für bestimmte Attribute einiger CITs konfigurieren. Auf diese Weise kann jedem dieser Attribute ein fester Wert zugeordnet werden, der nicht in der Datenbank gespeichert ist.

Die Datei sollte null oder mehr Einträge im folgenden Format enthalten:

```
entity[<Entitätsname>] attribute[<Attributname>] value[<Wert>]
```

Beispiel:

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

Die Datei unterstützt auch eine Liste mit Konstanten. Verwenden Sie zum Definieren einer Konstantenliste die folgende Syntax:

```
entity[<Entitätsname>] attribute[<Attributname>] value[{<Wert1>,
<Wert2>, <Wert3> ... }]
```

Die Datei "persistence.xml"

Diese Datei wird zum Überschreiben der Hibernate-Standardeinstellungen und zum Hinzufügen von Unterstützung für nicht standardmäßige Datenbanktypen verwendet (Standard-Datenbanktypen sind Oracle Server, Microsoft SQL Server und MySQL).

Wenn Sie einen neuen Datenbanktyp unterstützen müssen, stellen Sie sicher, dass Sie einen Verbindungspool-Provider (die Standardeinstellung lautet `c3p0`) und einen JDBC-Treiber für Ihre Datenbank bereitstellen (speichern Sie die `JAR`-Dateien im Adapterordner).

Um alle verfügbaren Hibernate-Werte zu sehen, die geändert werden können, prüfen Sie die Klasse **org.hibernate.cfg.Environment**. (Weitere Informationen hierzu finden Sie unter http://www.hibernate.org.)

Beispiel für die Datei "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>
```

Standardkonverter

Sie können die folgenden Konverter (Transformatoren) verwenden, um föderierte Abfragen und Replizierungsjob aus und in Datenbankdaten zu konvertieren.

Dieser Abschnitt umfasst die folgenden Themen:

Der enum-transformer-Konverter

Dieser Konverter verwendet eine XML-Datei, der als Eingabeparameter angegeben wird.

Die XML-Datei ordnet hartcodierte CMDB-Werte und Datenbankwerte (Aufzählungen) zu. Wenn einer der Werte fehlt, können Sie auswählen, ob derselbe Wert oder Null zurückgegeben oder ob eine Ausnahme ausgelöst werden soll.

Der Transformator führt einen Vergleich zwischen zwei Zeichenfolgen aus. Hierfür verwendet er eine Methode unter Beachtung oder Nichtbeachtung der Groß-/Kleinschreibung. Das Standardverhalten ist, die Groß-/Kleinschreibung zu beachten. Definieren Sie die Nichtbeachtung der Groß-/Kleinschreibung folgendermaßen: Verwenden Sie hierzu `case-sensitive="false"` im Element `enum-transformer`.

Verwenden Sie für jedes Entitätsattribut eine XML-Zuordnungsdatei.

> **Hinweis:** Dieser Konverter kann sowohl für das Feld `to_DB_class` als auch für das Feld `from_DB_class` in der Datei **transformations.txt** verwendet werden.

Eingabedatei XSD:

```
<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:attribute name="case-sensitive" use="optional">
            <xs:simpleType>
                <xs:restriction base="xs:boolean">
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="value">
    <xs:complexType>
        <xs:attribute name="cmdb-value" type="xs:string"
use="required"/>
        <xs:attribute name="external-db-value" type="xs:string"
use="required"/>
        <xs:attribute name="is-cmdb-value-null" type="xs:boolean"
use="optional"/>
        <xs:attribute name="is-db-value-null" type="xs:boolean"
use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Beispiel für die Konvertierung des Wertes 'sys' in den Wert 'System':

In diesem Beispiel wird der Wert `sys` in der CMDB in den Wert `System` in der föderierten Datenbank und der Wert `System` in der föderierten Datenbank in den Wert `sys` in der CMDB umgewandelt.

Falls der Wert nicht in der XML-Datei vorhanden ist (z. B. die Zeichenkette `demo`), gibt der Konverter den gleichen Eingabewert zurück, den er empfangen hat.

```
<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>
```

Beispiel für die Konvertierung eines externen oder CMDB-Wertes in einen Nullwert:

In diesem Beispiel wird der Wert **NNN** in der Remote-Datenbank in einen Nullwert in der CMDB-Datenbank umgewandelt.

```
<value cmdb-value="null" is-cmdb-value-null="true" external-db-
value="NNN"/>
```

In diesem Beispiel wird der Wert **OOO** in der CMDB-Datenbank in einen Nullwert in der Remote-Datenbank umgewandelt.

```
<value cmdb-value="OOO" external-db-value="null" is-db-value-
null="true"/>
```

Der SuffixTransformer-Konverter

Dieser Konverter wird verwendet, um Suffixe zum Wert der CMDB oder der föderierten Datenbankquelle hinzuzufügen oder von ihnen zu entfernen.

Es gibt zwei Implementierungen:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddSuffixTransformer**. Fügt das Suffix (gegeben als Eingabe) bei der Konvertierung vom Wert der föderierten Datenbank in den Wert der CMDB hinzu und entfernt es bei der Konvertierung vom Wert der CMDB in den Wert der föderierten Datenbank.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemoveSuffixTransformer**. Entfernt das Suffix (gegeben als Eingabe) bei der Konvertierung vom Wert der föderierten Datenbank in den Wert der CMDB und fügt es bei der Konvertierung vom Wert der CMDB in den Wert der föderierten Datenbank hinzu.

Der PrefixTransformer-Konverter

Dieser Konverter wird verwendet, um ein Präfix zum Wert der CMDB oder der föderierten Datenbank hinzuzufügen oder von ihnen zu entfernen.

Es gibt zwei Implementierungen:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddPrefixTransformer**. Fügt das Präfix (gegeben als Eingabe) bei der Konvertierung vom Wert der föderierten Datenbank in den Wert der CMDB hinzu und entfernt es bei der Konvertierung vom Wert der CMDB in den Wert der föderierten Datenbank.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemovePrefixTransformer**. Entfernt das Präfix (gegeben als Eingabe) bei der Konvertierung

vom Wert der föderierten Datenbank in den Wert der CMDB und fügt es bei der Konvertierung vom Wert der CMDB in den Wert der föderierten Datenbank hinzu.

Der BytesToStringTransformer-Konverter

Dieser Konverter wird verwendet, um Byte-Arrays in der CMDB in ihre Zeichenfolgendarstellung in der föderierten Datenbankquelle zu konvertieren.

Der Konverter ist: **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer**.

Der StringDelimitedListTransformer-Konverter

Dieser Konverter wird verwendet, um eine einzelne Zeichenfolgeliste in eine Ganzzahl-/Zeichenfolgeliste in der CMDB umzuwandeln.

Der Konverter ist: **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.StringDelimitedListTransformer.**

Plugins

Der allgemeine Datenbankadapter unterstützt die folgenden Plugins:

- Ein optionales Plugin für die Sychronisierung der vollständigen Topologie.
- Ein optionales Plugin zum Synchronisieren von Topologieänderungen. Wenn kein Plugin für die Synchronisierung von Änderungen implementiert ist, besteht die Möglichkeit, eine differenzielle Synchronisierung durchzuführen. Tatsächlich handelt es sich dabei jedoch um eine vollständige Synchronisierung.
- Ein optionales Plugin zum Synchronisieren des Layouts.
- Ein optionales Plugin zum Abrufen unterstützter Abfragen für die Synchronisierung. Falls dieses Plugin nicht definiert ist, werden alle TQL-Namen zurückgegeben.
- Ein internes, optionales Plugin zum Ändern der TQL-Definition und des TQL-Ergebnisses.
- Ein internes, optionales Plugin zum Ändern einer Layoutanforderung und eines CI-Ergebnisses.
- Ein internes, optionales Plugin zum Ändern einer Layoutanforderung und eines Beziehungsergebnisses.
- Ein internes, optionales Plugin zum Ändern der Aktion zum Zurückgeben von IDs.

Weitere Informationen zum Implementierten und Bereitstellen von Plugins finden Sie unter "Implementieren eines Plugin" auf Seite 107.

Konfigurationsbeispiele

In diesem Abschnitt finden Sie Beispiele für Konfigurationen.

Dieser Abschnitt umfasst die folgenden Themen:

- "Abstimmung unter Verwendung von zwei Knoten" auf Seite 150
- "Verwenden eines Primärschlüssels mit mehreren Spalten" auf Seite 153
- "Verwenden von Transformationen" auf Seite 155

Anwendungsfall

Anwendungsfall. Eine TQL-Abfrage ist wie folgt aufgebaut:

node > (composition) > card

Dabei gilt:

- **node** ist die CMDB-Entität
- **card** ist die Entität der föderierten Datenbankquelle
- **composition** ist die Beziehung zwischen den Entitäten

Das Beispiel wird für die `ED`-Datenbank ausgeführt. `ED nodes` werden in der `Device`-Tabelle gespeichert und `card` wird in der `hwCards`-Tabelle gespeichert. Im folgenden Beispiel wird `card` immer auf die gleiche Weise zugeordnet.

Abstimmung unter Verwendung von einem Knoten

In diesem Beispiel wird die Abstimmung für die Eigenschaft `name` ausgeführt.

Vereinfachte Definition

Die Abstimmung erfolgt auf der Basis von `node` und wird durch das Spezialtag **CMDB-class** hervorgehoben.

```
<?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>
```

Erweiterte Definition

Die Datei "orm.xml"

Achten Sie auf die Hinzufügung der Beziehungszuordnung. Weitere Informationen finden Sie im Definitionsabschnitt unter "Die Datei "orm.xml"" auf Seite 124.

Beispiel für die Datei "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>Generic DB adapter orm</description>
       <package>generic_db_adapter</package>
       <entity class="generic_db_adapter.node" >
              <table name="Device"/>
              <attributes>
                     <id name="id1">
                            <column name="Device_ID" insertable="false"
                               updatable="false"/>
                            <generated-value strategy="TABLE"/>
                     </id>
                     <basic name="name">
                            <column name="Device_Name"/>
                     </basic>
              </attributes>
       </entity>
       <entity class="generic_db_adapter.card" >
              <table name="hwCards"/>
              <attributes>
                     <id name="id1">
                            <column name="hwCards_Seq" insertable="false"
                                updatable="false"/>
                            <generated-value strategy="TABLE"/>
                     </id>
                     <basic name="card_class">
                            <column name="hwCardClass" insertable="false"
                                updatable="false"/>
                     </basic>
                     <basic name="card_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>
```

Die Datei "reconciliation_types.txt"

Weitere Informationen finden Sie unter "Die Datei "reconciliation_types.txt"" auf Seite 137.

```
node
```

Die Datei "reconciliation_rules.txt"

Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 137.

```
multinode[node] expression[node.name]
```

Die Datei "transformation.txt"

Diese Datei bleibt leer, da in diesem Beispiel keine Werte konvertiert werden müssen.

Abstimmung unter Verwendung von zwei Knoten

In diesem Beispiel wird die Abstimmung entsprechend der `name`-Eigenschaft von `node` und `ip_address` mit unterschiedlichen Variationen berechnet.

Die Abstimmungs-TQL-Abfrage lautet **node > (containment) > ip_address**.

Vereinfachte Definition

Die Abstimmung erfolgt auf Basis der `name`-Eigenschaft von `node` ODER `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>
```

Die Abstimmung erfolgt auf Basis der `name`-Eigenschaft von `node` UND `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>
```

Die Abstimmung erfolgt auf Basis der `name`-Eigenschaft von `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>
```

Erweiterte Definition

Die Datei "orm.xml"

Da der Abstimmungsausdruck in dieser Datei nicht definiert ist, sollte für alle Abstimmungsausdrücke die gleiche Version verwendet werden.

Die Datei "reconciliation_types.txt"

Weitere Informationen finden Sie unter "Die Datei "reconciliation_types.txt"" auf Seite 137.

`node`

Die Datei "reconciliation_rules.txt"

Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 137.

```
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]
```

Die Datei "transformation.txt"

Diese Datei bleibt leer, da in diesem Beispiel keine Werte konvertiert werden müssen.

Verwenden eines Primärschlüssels mit mehreren Spalten

Wenn der Primärschlüssel aus mehreren Spalten besteht, wird der folgende Code zu den XML-Definitionen hinzugefügt:

Vereinfachte Definition

Es gibt mehrere Primärschlüsseltags und für jede Spalte ist ein Tag vorhanden.

```
    <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>
```

Erweiterte Definition

Die Datei "orm.xml"

Eine neue `id`-Entität wird hinzugefügt, die den Primärschlüsselspalten zugeordnet wird. Entitäten, die diese `id`-Entität verwenden, muss ein Spezialtag hinzugefügt werden.

Wenn Sie einen Fremdschlüssel (join-column-Tag) für einen derartigen Primärschlüssel verwenden, müssen Sie jede Spalte im Fremdschlüssel einer Spalte im Primärschüssel zuordnen.

Weitere Informationen finden Sie unter "Die Datei "orm.xml"" auf Seite 124.

Beispiel für die Datei "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>
```

Verwenden von Transformationen

Im folgenden Beispiel wird der allgemeine **enum**-Transformator von den Werten `1`, `2`, `3` in die Werte `a`, `b` beziehungsweise `c` in der `name`-Spalte konvertiert.

Die Zuordnungsdatei ist die Datei `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>
```

Vereinfachte Definition

```
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address"
          CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name"
                 from-CMDB-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
                 transform.impl.GenericEnumTransformer(generic-enum-
transformer-example.
                 xml)" to-CMDB-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
                 transform.impl.GenericEnumTransformer(generic-enum-
transformer-example.
                 xml)" />
                <connected-node-attribute CMDB-attribute-name="name"
                 column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
```

Erweiterte Definition

Es gibt nur eine Änderung in der Datei **transformation.txt**.

Die Datei "transformation.txt"

Stellen Sie sicher, dass die gleichen Attribut- und Entitätsnamen verwendet werden wie in der Datei `orm.xml`.

```
entity[node] attribute[name]
to_DB_class
[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)] from_DB_
```

```
class
[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

Adapterprotokolldateien

Sie können die folgenden Protokolldateien zurate ziehen, um die Berechnungsabläufe und den Adapterlebenszyklus zu verstehen und Debug-Informationen anzuzeigen.

Dieser Abschnitt umfasst die folgenden Themen:

- "Protokollebenen" oben
- "Protokollspeicherorte" oben

Protokollebenen

Sie können für jedes Protokoll eine Protokollebene konfigurieren.

Öffnen Sie in einem Texteditor die Datei **C:\hp\UCMDB\UCMDBServer\conf\log\ fcmdb.gdba.properties**
.

Die Standardprotokollebene ist **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL loglevel=ERROR
```

- Um die Protokollebene für alle Protokolldateien zu erhöhen, ändern Sie den Eintrag **loglevel=ERROR** in **loglevel=DEBUG** oder **loglevel=INFO**.
- Um die Protokollebene für eine bestimmte Datei zu ändern, ändern Sie die Kategoriezeile **log4j** entsprechend. Um beispielsweise die Protokollebene des Protokolls **fcmdb.gdba.dal.sql.log** in **INFO** zu ändern, ändern Sie die Zeile

  ```
  log4j.category.fcmdb.gdba.dal.SQL=${loglevel},
  fcmdb.gdba.dal.SQL.appender
  ```

  in

  ```
  log4j.category.fcmdb.gdba.dal.SQL=INFO,fcmdb.gdba.dal.SQL.appender
  ```

Protokollspeicherorte

Die Protokolldateien befinden sich im Verzeichnis **C:\hp\UCMDB\UCMDBServer\runtime\log**.

- **Fcmdb.gdba.log**

  Das Protokoll zum Adapterlebenszyklus. Enthält Angaben darüber, wann der Adapter gestartet oder angehalten wurde und welche CITs von diesem Adapter unterstützt werden.

  Gibt Aufschluss über Initiierungsfehler (Laden/Entladen des Adapters).

- **fcmdb.log**

  Gibt Aufschluss über Ausnahmen.

- **cmdb.log**

  Gibt Aufschluss über Ausnahmen.

- **Fcmdb.gdba.mapping.engine.log**

  Das Protokoll zur Zuordnungs-Engine. Enthält Angaben über die von der Zuordnungs-Engine verwendete Abstimmungs-TQL-Abfrage und die Abstimmungstopologien, die während der Verbindungsphase miteinander verglichen werden.

  Ziehen Sie dieses Protokoll zurate, wenn eine TQL-Abfrage keine Ergebnisse liefert, obwohl relevante CIs in der Datenbank vorhanden sind, oder die Ergebnisse nicht den Erwartungen entsprechen (Abstimmung prüfen).

- **Fcmdb.gdba.TQL.log**

  Das TQL-Protokoll. Enthält Angaben zu den TQL-Abfragen und ihren Ergebnissen.

  Ziehen Sie dieses Protokoll zurate, wenn eine TQL-Abfrage keine Ergebnisse zurückgibt und laut Protokoll zur Zuordnungs-Engine keine Ergebnisse in der föderierten Datenquelle vorhanden sind.

- **Fcmdb.gdba.dal.log**

  Das Protokoll zum DAL-Lebenszyklus. Enthält Angaben zur CIT-Generierung und den Datenbankverbindungsdetails.

  Ziehen Sie dieses Protokoll zurate, wenn Sie keine Verbindung zur Datenbank herstellen können oder CITs oder Attribute vorhanden sind, die nicht von der Abfrage unterstützt werden.

- **Fcmdb.gdba.dal.command.log**

  Das Protokoll zu den DAL-Operationen. Enthält Angaben zu den aufgerufenen internen DAL-Operationen. (Dieses Protokoll ähnelt dem Protokoll `cmdb.dal.command.log`).

- **Fcmdb.gdba.dal.SQL.log**

  Das Protokoll zu den DAL-SQL-Abfragen. Enthält Angaben zu den aufgerufenen JPAQLs (objektorientierte SQL-Abfragen) und ihren Ergebnissen.

  Ziehen Sie dieses Protokoll zurate, wenn Sie keine Verbindung zur Datenbank herstellen können oder CITs oder Attribute vorhanden sind, die nicht von der Abfrage unterstützt werden.

- **Fcmdb.gdba.hibrnate.log**

  Das Hibernate-Protokoll. Enthält Angaben zu den ausgeführten SQL-Abfragen, den JPAQL-zu-SQL-Analysen, den Abfrageergebnissen, den Daten in Bezug auf die Hibernate-Zwischenspeicherung usw. Weitere Informationen zu Hibernate finden Sie unter "Hibernate als JPA-Provider" auf Seite 88.

Externe Referenzen

Weitere Informationen zur JavaBeans 3.0-Spezifikation finden Sie unter http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

Fehlerbehebung und Einschränkungen

In diesem Abschnitt werden die Fehlerbehebung und die Einschränkungen für den allgemeinen Datenbankadapter beschrieben.

Allgemeine Einschränkungen

- Die SQL Server NTLM-Authentifizierung wird nicht unterstützt.
- Wenn Sie ein Adapter-Package aktualisieren, verwenden Sie zum Bearbeiten der Vorlagendateien Notepad++, UltraEdit oder einen Texteditor eines anderen Drittanbieters anstelle des Editors der Microsoft Corporation, um die Verwendung von Sonderzeichen zu verhindern, die dazu führen können, dass die Bereitstellung des vorbereiteten Package fehlschlägt.

JPA-Einschränkungen

- Alle Tabellen müssen eine Primärschlüsselspalte haben.
- Die Namen von CMDB-Klassenattributen müssen die JavaBeans-Benennungskonvention einhalten (z. B. muss der Anfangsbuchstabe der Namen kleingeschrieben werden).
- Zwei CIs, die mit einer Beziehung im Klassenmodell verbunden sind, müssen eine direkte Zuweisung in der Datenbank haben (z. B. wenn `node` mit `ticket` verbunden ist, muss es einen Fremdschlüssel oder eine Verknüpfungstabelle geben, der bzw. die die beiden CIs miteinander verbindet).
- Mehrere Tabelle, die dem gleichen CIT zugeordnet sind, müssen die gleiche Primärschlüsseltabelle verwenden.

Funktionale Einschränkungen

- Sie können keine manuelle Beziehung zwischen der CMDB und föderierten CITs erstellen. Um virtuelle Beziehungen definieren zu können, muss eine spezielle Beziehungslogik definiert werden (als Basis können die Eigenschaften der föderierten Klasse verwendet werden).
- Föderierte CITs können nicht als Trigger-CITs in einer Auswirkungsregel fungieren. Sie können jedoch in eine TQL-Abfrage zu einer Auswirkungsanalyse integriert werden.
- Ein föderierter CIT kann Teil einer Enrichment-TQL sein. Er kann jedoch nicht als der Knoten verwendet werden, bei dem das Enrichment durchgeführt wird (der föderierte CIT kann nicht hinzugefügt, aktualisiert oder gelöscht werden).
- Die Verwendung eines Klassenqualifizierers in einer Bedingung wird nicht unterstützt.
- Unterdiagramme werden nicht unterstützt.
- Verbundbeziehungen werden nicht unterstützt.
- Die CMDB-`ID` für das externe CI wird aus dem Primärschlüssel gebildet und nicht aus den Schlüsselattributen.
- Eine Spalte des Typs `bytes` kann nicht als Primärschlüsselspalte in Microsoft SQL Server verwendet werden.
- Die Berechnung einer TQL-Abfrage schlägt fehl, wenn die Namen der bei einem föderierten Knoten definierten Attributbedingungen nicht in der Datei **orm.xml** zugeordnet wurden.
- Der allgemeine DB-Adapter unterstützt keine Windows-Authentifizierung für SQL-Server.

Kapitel 5

Entwickeln von Java-Adaptern

Dieses Kapitel umfasst folgende Themen:

Federation Framework – Übersicht

Hinweis:

- Der Begriff **Beziehung** hat die gleiche Bedeutung wie der Begriff **Link**.
- Der Begriff **CI** hat die gleiche Bedeutung wie der Begriff **Objekt**.
- Ein Diagramm ist eine Sammlung von Knoten und Links.

Die Federation Framework-Funktionalität verwendet eine API zum Abrufen von Informationen von föderierten Quellen. Federation Framework bietet drei Hauptfunktionen:

- **Föderation** nach dem On-the-Fly-Prinzip. Alle Abfragen werden über die ursprünglichen Daten-Repositorys ausgeführt und die Ergebnisse werden nach dem On-the-Fly-Prinzip in CMDB erstellt.
- **Auffüllung.** Füllt die CMDB mit Daten (topologische Daten und CI-Eigenschaften) von einer externen Datenquelle auf.
- **Datenpush.** Führt einen Datenpush (topologische Daten und CI-Eigenschaften) von der lokalen CMDB zu einer Remote-Datenquelle durch.

Alle Aktionstypen erfordern einen Adapter für jedes Daten-Repository, der die spezifischen Funktionen des Daten-Repository bereitstellen und die erforderlichen Daten abrufen und/oder aktualisieren kann. Jede Anforderung beim Daten-Repository erfolgt über den jeweiligen Adapter.

Dieser Abschnitt umfasst außerdem die folgenden Themen:

- "Föderation nach dem On-the-Fly-Prinzip" oben
- "Datenpush" auf der nächsten Seite
- "Auffüllung" auf Seite 162

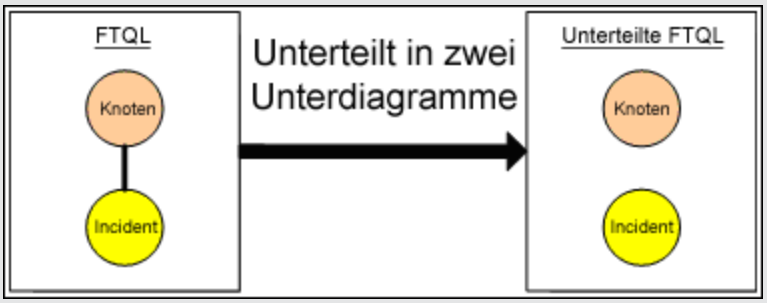
Föderation nach dem On-the-Fly-Prinzip

Föderierte TQL-Abfragen ermöglichen das Abrufen von Daten von jedem externen Daten-Repository ohne die Daten zu replizieren.

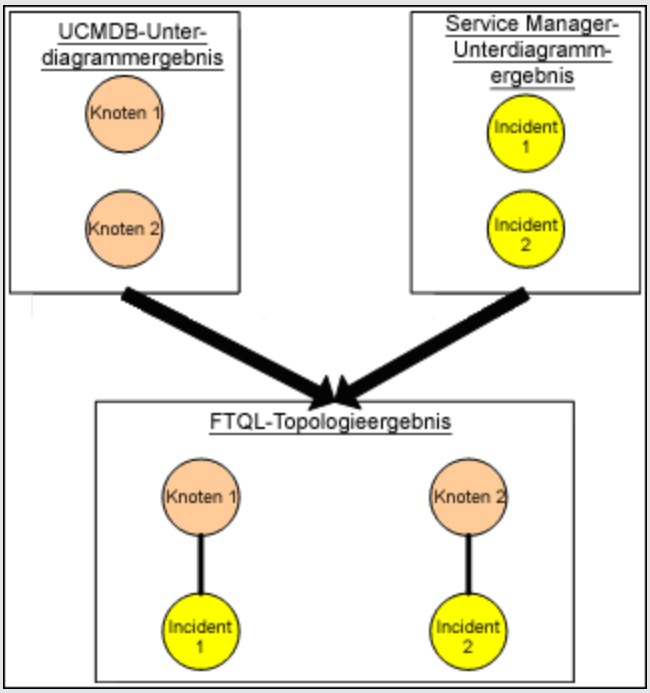
Eine föderierte TQL-Abfrage verwendet Adapter, die externe Daten-Repositorys darstellen, um entsprechende externe Beziehungen zwischen den CIs von verschiedenen externen Daten-Repositorys und den CIs von UCMDB zu erstellen.

Beispiel für einen Föderationsfluss nach dem On-the-Fly-Prinzip:

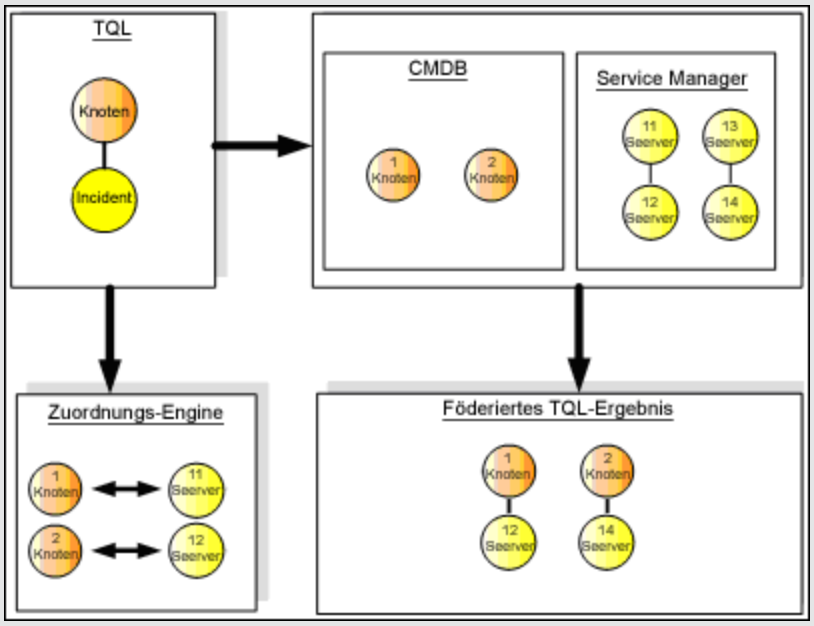
1. Federation Framework unterteilt eine föderierte TQL-Abfrage in mehrere Unterdiagramme, wobei sich alle Knoten in einem Unterdiagramm auf das gleiche Daten-Repository beziehen. Jedes Unterdiagramm ist über eine virtuelle Beziehung mit den anderen Unterdiagrammen verbunden (enthält aber selbst keine virtuellen Beziehungen).



2. Nachdem die föderierte TQL-Abfrage in Unterdiagramme unterteilt wurde, berechnet Federation Framework die Topologie jedes Unterdiagramms und verbindet zwei passende Unterdiagramme durch Erstellen von virtuellen Beziehungen zwischen den jeweiligen Knoten.



3. Nachdem die föderierte TQL-Topologie berechnet wurde, ruft Federation Framework ein Layout für das Topologieergebnis ab.



Datenpush

Sie verwenden den Datenpush-Fluss, um die Daten von Ihrer aktuellen lokalen CMDB mit einem Remote-Service oder einem Zieldaten-Repository zu synchronisieren.

Beim Datenpush werden die Daten-Repositorys in zwei Kategorien eingeteilt: Quelle (lokale CMDB) und Ziel. Die Daten werden vom Quelldaten-Repository abgerufen und im Zieldaten-

Repository aktualisiert. Der Datenpush-Prozess basiert auf Abfragenamen. Dies bedeutet, dass die Daten zwischen der Quelle (lokale CMDB) und den Zieldaten-Repositorys synchronisiert und durch einen TQL-Abfragenamen von der lokalen CMDB abgerufen werden.

Der Datenpush-Prozess besteht aus den folgenden Schritten:

1. Abrufen des Topologieergebnisses mit Signaturen vom Quelldaten-Repository.
2. Vergleichen der neuen Ergebnisse mit den vorherigen Ergebnissen.
3. Abrufen eines vollständigen Layouts (d. h. aller CI-Eigenschaften) der CIs und Beziehungen (nur für geänderte Ergebnisse).
4. Aktualisieren des Zieldaten-Repository mit dem empfangenen vollständigen Layout der CIs und Beziehungen. Wenn CIs oder Beziehungen im Quelldaten-Repository gelöscht werden und die Abfrage exklusiv ist, entfernt der Replizierungsprozess auch die CIs bzw. Beziehungen im Zieldaten-Repository.

In CMDB gibt es 2 ausgeblendete Datenquellen (**hiddenRMIDataSource** und **hiddenChangesDataSource**). Diese gehören bei den Datenpush-Flüssen grundsätzlich zur 'Quell'-Datenquelle. Um einen neuen Adapter für Datenpush-Flüsse zu implementieren, müssen Sie lediglich den 'Ziel'-Adapter implementieren.

Auffüllung

Sie verwenden den Auffüllungsfluss, um die CMDB mit Daten von externen Quellen aufzufüllen.

Der Fluss verwendet grundsätzlich eine 'Quell'-Datenquelle zum Abrufen der Daten. Die Übertragung der abgerufenen Daten an die Probe erfolgt in einem Prozess, der mit dem Fluss eines Discovery-Jobs vergleichbar ist.

Um einen neuen Adapter für Auffüllungsflüsse zu implementieren, müssen Sie nur den Quelladapter implementieren, da die Data Flow Probe als Ziel fungiert.

Der Adapter im Auffüllungsfluss wird bei der Probe ausgeführt. Das Debuggen und Protokollieren sollte bei der Probe erfolgen und nicht bei der CMDB.

Der Auffüllungsfluss basiert auf Abfragenamen, d. h., die Daten werden zwischen dem Quelldaten-Repository und der Data Flow Probe synchronisiert und durch einen Abfragenamen im Quelldaten-Repository abgerufen. In UCMDB entspricht der Abfragename beispielsweise dem Namen der TQL-Abfrage. In einem anderen Daten-Repository kann der Abfragename hingegen ein Codename sein, der Daten zurückgibt. Der Adapter ist für die korrekte Handhabung des Abfragenamens ausgelegt.

Jeder Job kann als exklusiver Job definiert werden. Dies bedeutet, dass die CIs und Beziehungen in den Job-Ergebnissen in der lokalen CMDB eindeutig sind und durch keine andere Abfrage an das Ziel übertragen werden können. Der Adapter des Quelldaten-Repository unterstützt bestimmte Abfragen und kann die Daten von diesem Daten-Repository abrufen. Der Adapter des Zieldaten-Repository ermöglicht die Aktualisierung der abgerufenen Daten bei diesem Daten-Repository.

SourceDataAdapter-Fluss

- Ruft das Topologieergebnis mit Signaturen vom Quelldaten-Repository ab.
- Vergleicht die neuen Ergebnisse mit den vorherigen Ergebnissen.

- Ruft ein vollständiges Layout (d. h. alle CI-Eigenschaften) der CIs und Beziehungen ab (nur für geänderte Ergebnisse).
- Aktualisiert das Zieldaten-Repository mit dem empfangenen vollständigen Layout der CIs und Beziehungen. Wenn CIs oder Beziehungen im Quelldaten-Repository gelöscht werden und die Abfrage exklusiv ist, entfernt der Replizierungsprozess auch die CIs bzw. Beziehungen im Zieldaten-Repository.

SourceChangesDataAdapter-Fluss

- Ruft das Topologieergebnis ab, das seit dem letzten angegebenen Datum erfasst wurde.
- Ruft ein vollständiges Layout (d. h. alle CI-Eigenschaften) der CIs und Beziehungen ab (nur für geänderte Ergebnisse).
- Aktualisiert das Zieldaten-Repository mit dem empfangenen vollständigen Layout der CIs und Beziehungen. Wenn CIs oder Beziehungen im Quelldaten-Repository gelöscht werden und die Abfrage exklusiv ist, entfernt der Replizierungsprozess auch die CIs bzw. Beziehungen im Zieldaten-Repository.

PopulateDataAdapter-Fluss

- Ruft die vollständige Topologie mit dem angeforderten Layout-Ergebnis ab.
- Verwendet den Topologie-Chunk-Mechanismus, um die Daten in Blöcken abzurufen.
- Die Probe filtert die Daten heraus, die bereits bei früheren Ausführungen übertragen wurden.
- Aktualisiert das Zieldaten-Repository mit dem empfangenen Layout der CIs und Beziehungen. Wenn CIs oder Beziehungen im Quelldaten-Repository gelöscht werden und die Abfrage exklusiv ist, entfernt der Replizierungsprozess auch die CIs bzw. Beziehungen im Zieldaten-Repository.

PopulateChangesDataAdapter-Fluss

- Ruft die Topologie mit dem angeforderten Layout-Ergebnis ab, das sich seit der letzten Ausführung geändert hat.
- Verwendet den Topologie-Chunk-Mechanismus, um die Daten in Blöcken abzurufen.
- Die Probe filtert die Daten heraus, die bereits bei früheren Ausführungen (einschließlich dieses Flusses) übertragen wurden.
- Aktualisiert das Zieldaten-Repository mit dem empfangenen Layout der CIs und Beziehungen. Wenn CIs oder Beziehungen im Quelldaten-Repository gelöscht werden und die Abfrage exklusiv ist, entfernt der Replizierungsprozess auch die CIs bzw. Beziehungen im Zieldaten-Repository.

Adapter- und Zuordnungsinteraktion mit Federation Framework

Ein Adapter ist eine Entität in UCMDB, die externe Daten darstellt (d. h. Daten, die nicht in UCMDB gespeichert sind). In föderierten Flüssen werden alle Interaktionen mit externen Datenquellen über Adapter durchgeführt. Der Federation Framework-Interaktionsfluss und die Adapterschnittstellen für die Replizierung und für föderierte TQL-Abfragen unterscheiden sich voneinander.

Dieser Abschnitt umfasst außerdem die folgenden Themen:

- "Adapterlebenszyklus" oben
- "Hilfsmethoden des Adapters" oben

Adapterlebenszyklus

Für jedes externe Daten-Repository wird eine Adapterinstanz erstellt. Der Lebenszyklus des Adapters beginnt mit der ersten Aktion, die auf den Adapter angewendet wird (z. B. `TQL berechnen` oder `Daten abrufen/aktualisieren`). Wenn die **start**-Methode aufgerufen wird, empfängt der Adapter Umgebungsinformationen, wie z. B. Angaben zur Konfiguration des Daten-Repository, zur Protokollierung usw. Der Adapterlebenszyklus endet, wenn das Daten-Repository von der Konfiguration entfernt und die **shutdown**-Methode aufgerufen wird. Dies bedeutet, dass der Adapter zustandsbehaftet ist und bei Bedarf die Verbindung zum externen Daten-Repository enthalten kann.

Hilfsmethoden des Adapters

Der Adapter verfügt über mehrere `Hilfsmethoden` zum Hinzufügen von Konfigurationen für das externe Daten-Repository. Diese Methoden sind nicht Teil des Adapterlebenszyklus. Sie erstellen bei jedem Aufruf einen neuen Adapter.

- Die erste Methode testet die Verbindung zum externen Daten-Repository für eine bestimmte Konfiguration. **testConnection** kann je nach Adaptertyp entweder beim UCMDB-Server oder bei der Data Flow Probe ausgeführt werden.
- Die zweite Methode ist nur für den Quelladapter relevant und gibt die unterstützten Abfragen für die Replizierung zurück. (Diese Methode wird nur bei der Probe ausgeführt.)
- Die dritte Methode ist nur für Föderations- und Auffüllungsflüsse relevant. Sie gibt die vom externen Daten-Repository unterstützten externen Klassen zurück. (Diese Methode wird beim UCMDB-Server ausgeführt.)

Alle diese Methoden werden verwendet, wenn Sie Integrationskonfigurationen erstellen oder anzeigen.

Federation Framework für föderierte TQL-Abfragen

Dieser Abschnitt umfasst die folgenden Themen:

- "Definitionen und Begriffe" auf der nächsten Seite
- "Zuordnungs-Engine" auf der nächsten Seite
- "Föderierter Adapter" auf der nächsten Seite

Diagramme zur Veranschaulichung der Interaktion zwischen Federation Framework, UCMDB, dem Adapter und der Zuordnungs-Engine finden Sie unter "Interaktionen zwischen Federation Framework, Server, Adapter und Zuordnungs-Engine" auf der nächsten Seite.

Definitionen und Begriffe

Abstimmungsdaten. Die Regel für die Zuordnung von CIs des angegebenen Typs, die von der CMDB und dem externen Daten-Repository empfangen werden. Es gibt drei Arten von Abstimmungsregeln:

- **ID-Abstimmung.** Diese Abstimmung kann nur verwendet werden, wenn das externe Daten-Repository die CMDB-ID der Abstimmungsobjekte enthält.
- **Eigenschaftenabstimmung.** Diese Abstimmung wird verwendet, wenn die Zuordnung nur anhand der Eigenschaften des CI-Typs **reconciliation** erfolgen kann.
- **Topologieabstimmung.** Diese Abstimmung wird verwendet, wenn die Eigenschaften zusätzlicher CITs (nicht nur des **reconciliation**-CIT) erforderlich sind, um eine Zuordnung bei Abstimmungs-CIs durchzuführen. Beispiel: Sie können eine Abstimmung des Knotentyps nach der `name`-Eigenschaft durchführen, die zum CIT `ip_address` gehört.

Abstimmungsobjekt. Das Objekt wird entsprechend den empfangenen Abstimmungsdaten vom Adapter erstellt. Dieses Objekt sollte sich auf ein externes CI beziehen. Es wird von der Zuordnungs-Engine verwendet, um eine Verbindung zwischen den externen CIs und den CMDB-CIs herzustellen.

CI-Typ "reconciliation". Der Typ von CIs, die Abstimmungsobjekte darstellen. Diese CIs müssen sowohl in der CMDB als auch in den externen Daten-Repositorys gespeichert werden.

Zuordnungs-Engine. Eine Komponente, die die Beziehungen zwischen den CIs von verschiedenen Daten-Repositorys identifiziert, zwischen denen eine virtuelle Beziehung besteht. Die Identifikation erfolgt durch Abstimmen der CMDB-Abstimmungsobjekte mit den Abstimmungsobjekten des externen CI.

Zuordnungs-Engine

Federation Framework verwendet die Zuordnungs-Engine zur Berechnung der föderierten TQL-Abfrage. Die Zuordnungs-Engine stellt eine Verbindung zwischen den CIs her, die von verschiedenen Daten-Repositorys empfangen werden und über virtuelle Beziehungen miteinander verbunden sind. Außerdem stellt die Zuordnungs-Engine die Abstimmungsdaten für die virtuelle Beziehung zur Verfügung. Ein Ende der virtuellen Beziehung muss sich auf die CMDB beziehen. Dieses Ende muss vom Typ `reconciliation` sein. Für die Berechnung der zwei Unterdiagramme ist es gleichgültig, an welchem Endknoten die virtuelle Beziehung beginnt.

Föderierter Adapter

Der föderierte Adapter überträgt zwei Arten von Daten von externen Daten-Repositorys: Externe CI-Daten und Abstimmungsobjekte, die zu externen CIs gehören.

- **Externe CI-Daten.** Die externen Daten, die nicht in der CMDB vorhanden sind. Hierbei handelt es sich um die Zieldaten des externen Daten-Repository.
- **Abstimmungsobjektdaten.** Die Zusatzdaten, die von Federation Framework verwendet werden, um eine Verbindung zwischen den CMDB-CIs und den externen Daten herzustellen. Jedes Abstimmungsobjekt sollte sich auf ein externes CI beziehen. Der Typ des Abstimmungsobjekts ist der Typ (oder Untertyp) von einem der Endpunkte der virtuellen Beziehung, von dem die Daten abgerufen werden. Abstimmungsobjekte sollten den

empfangenen Adapter an die Abstimmungsdaten anpassen. Es gibt drei Arten von Abstimmungsobjekten: `IdReconciliationObject`, `PropertyReconciliationObject` und `TopologyReconciliationObject`.

Bei den Schnittstellen auf DataAdapter-Basis (DataAdapter, PopulateDataAdapter und PopulateChangesDataAdapter) wird die Abstimmung im Rahmen der Abfragedefinition angefordert.

Interaktionen zwischen Federation Framework, Server, Adapter und Zuordnungs-Engine

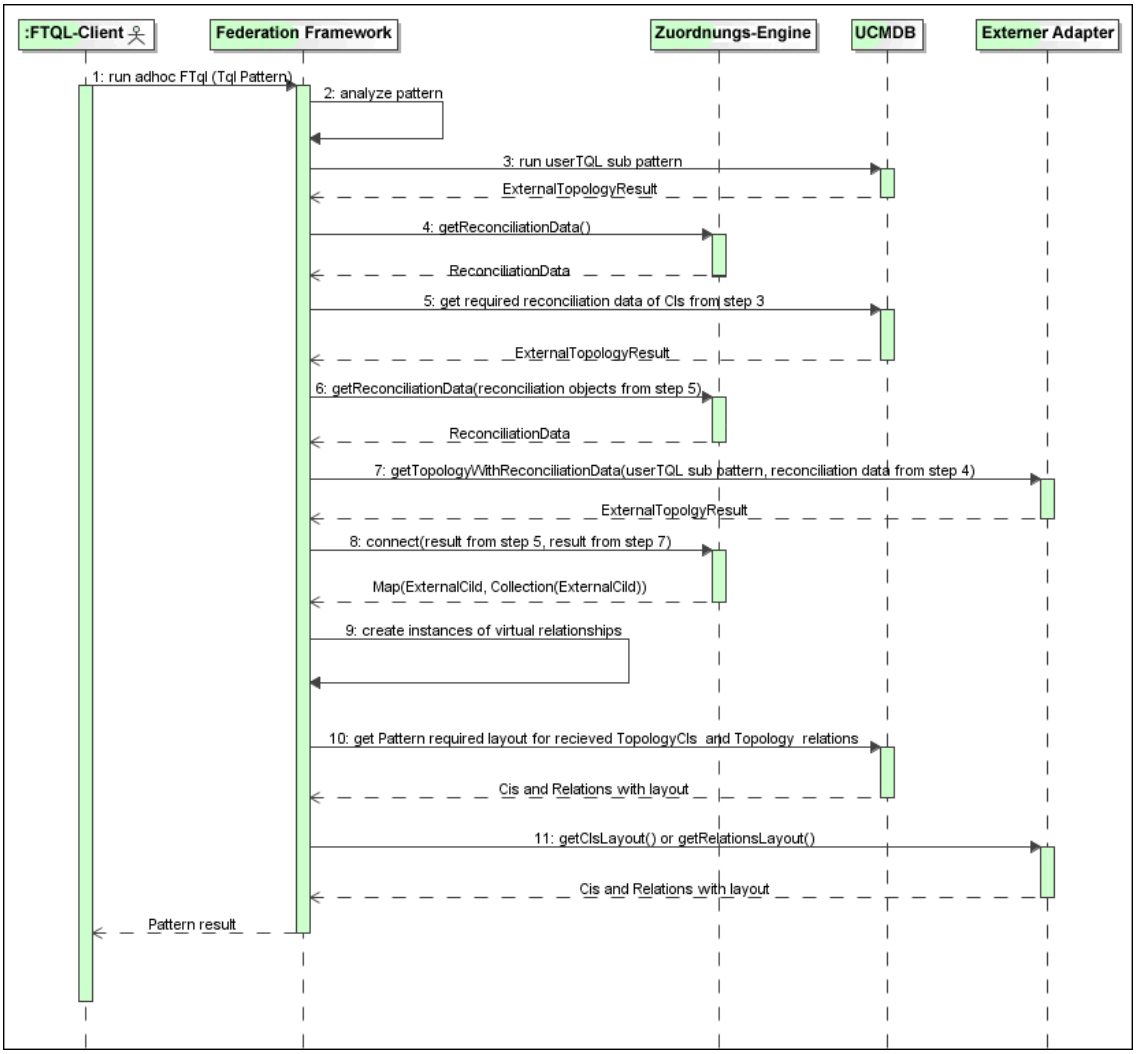
Die folgenden Diagramme veranschaulichen die Interaktionen zwischen Federation Framework, UCMDB-Server, dem Adapter und der Zuordnungs-Engine. Die föderierte TQL-Abfrage in den Beispieldiagrammen hat nur eine virtuelle Beziehung, sodass nur UCMDB und ein externes Daten-Repository in die föderierte TQL-Abfrage einbezogen werden.

Dieser Abschnitt umfasst die folgenden Themen:

Im ersten Diagram beginnt die Berechnung in UCMDB. Im zweiten Diagramm beginnt sie im externen Adapter. Jeder Schritt im Diagramm enthält Verweise auf den entsprechenden Methodenaufruf des Adapters oder die Schnittstelle der Zuordnungs-Engine.

Berechnung beginnt am Serverende

Das folgende Ablaufdiagramm veranschaulicht die Interaktion zwischen Federation Framework, UCMDB, dem Adapter und der Zuordnungs-Engine. Die föderierte TQL-Abfrage im Beispieldiagramm hat nur eine virtuelle Beziehung, sodass nur UCMDB und ein externes Daten-Repository in die föderierte TQL-Abfrage einbezogen werden.

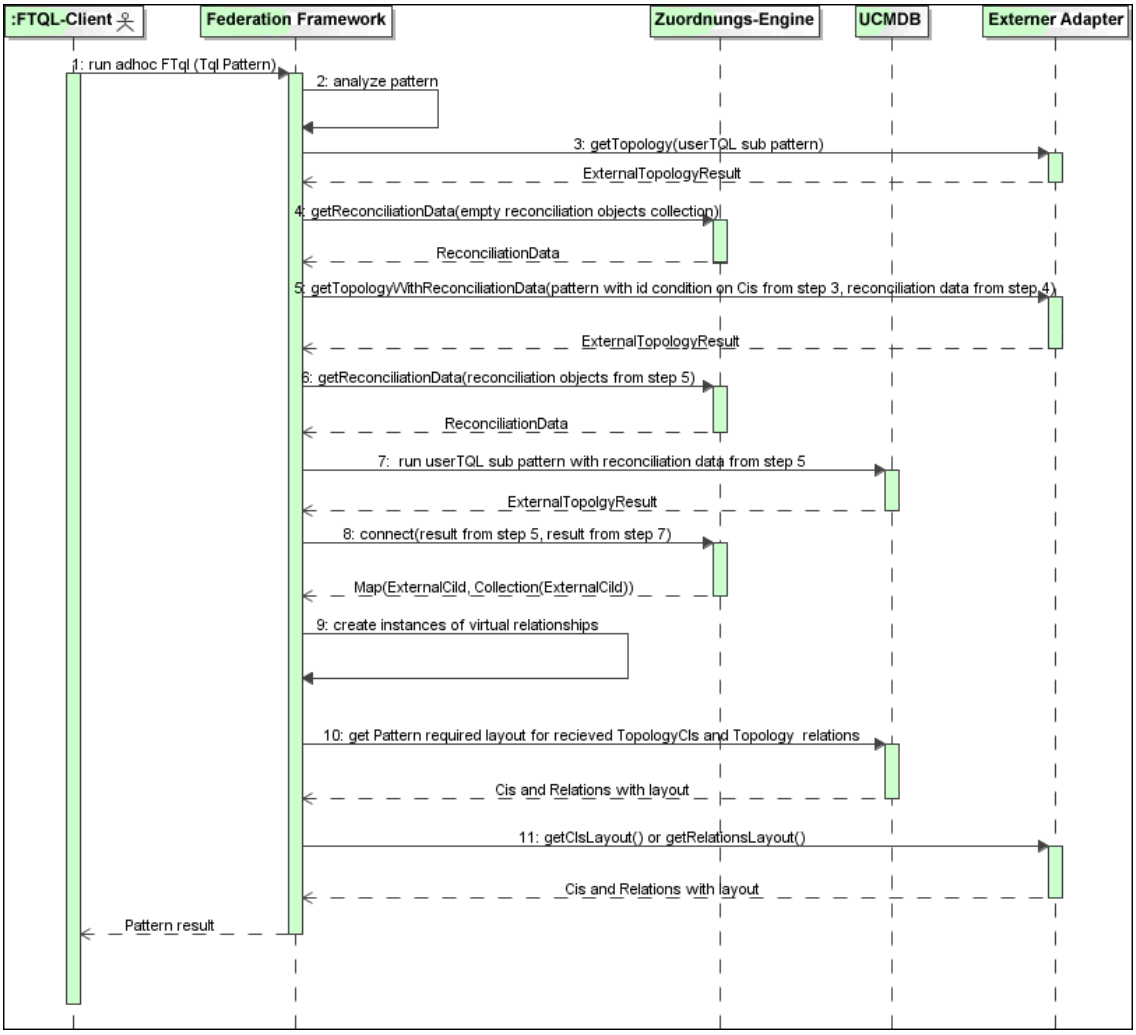


Die Zahlen in der Abbildung werden im Folgenden erläutert:

Zahl	Erklärung
1	Federation Framework empfängt einen Aufruf für die Berechnung einer föderierten TQL.
2	Federation Framework analysiert den Adapter, sucht die virtuelle Beziehung und teilt die ursprüngliche TQL in zwei Unteradapter – einen für UCMDB und einen für das externe Daten-Repository.
3	Federation Framework fordert die Topologie der Unter-TQL von UCMDB an.
4	Nach dem Empfang der Topologieergebnisse ruft Federation Framework die entsprechende Zuordnungs-Engine für die aktuelle virtuelle Beziehung auf und fordert die Abstimmungsdaten an. Der Parameter `reconciliationObject` ist zu diesem Zeitpunkt noch leer, d. h., in diesem Aufruf wird keine Bedingung zu den Abstimmungsdaten hinzugefügt. Die zurückgegebenen Abstimmungsdaten definieren, welche Daten erforderlich sind, um die Abstimmungs-CIs in UCMDB zum externen Daten-Repository zuzuordnen. Es gibt drei Arten von Abstimmungsdaten:

Zahl	Erklärung
	• **IdReconciliationData.** Die Abstimmung der CIs erfolgt auf Basis ihrer ID. • **PropertyReconciliationData.** Die Abstimmung der CIs erfolgt auf Basis der Eigenschaften eines CIs. • **TopologyReconciliationData.** Die Abstimmung der CIs erfolgt auf Basis der Topologie (z. B. wird für die Abstimmung von Knoten-CIs auch die IP-Adresse von **IP** benötigt).
5	Federation Framework fordert die Abstimmungsdaten für die CIs an den Endpunkten der virtuellen Beziehung an, die in Schritt"3" auf der vorherigen Seite von UCMDB empfangen wurden.
6	Federation Framework ruft die Zuordnungs-Engine auf, um die Abstimmungsdaten abzurufen. Im Gegensatz zu Schritt "3" auf der vorherigen Seite empfängt die Zuordnungs-Engine nun die Abstimmungsobjekte von Schritt "5" unten als Parameter. Die Zuordnungs-Engine übersetzt das empfangene Abstimmungsobjekt in die Abstimmungsdaten.
7	Federation Framework fordert die Topologie der Unter-TQL vom externen Daten-Repository an. Der externe Adapter empfängt die Abstimmungsdaten von Schritt "6" unten in Form eines Parameters.
8	Federation Framework ruft die Zuordnungs-Engine auf, um eine Verbindung zwischen den empfangenen Ergebnissen herzustellen. Der Parameter `firstResult` ist das Ergebnis der externen Topologie, das von UCMDB in Schritt "5" unten empfangen wurde. Der Parameter `secondResult` ist das Ergebnis der externen Topologie, das vom externen Adapter in Schritt "7" unten empfangen wurde. Die Zuordnungs-Engine gibt eine Karte zurück, in der die ID des externen CI vom ersten Daten-Repository (in diesem Fall UCMDB) den IDs des externen CI vom zweiten (externen) Daten-Repository zugeordnet ist.
9	Für jede Zuordnung erstellt Federation Framework eine virtuelle Beziehung.
10	Nach der Berechung der Ergebnisse der föderierten TQL-Abfrage (nur auf der Topologiestufe) ruft Federation Framework das ursprüngliche TQL-Layout für die resultierenden CIs und Beziehungen von den entsprechenden Daten-Repositorys ab.

Berechnung beginnt am Ende des externen Adapters



Die Zahlen in der Abbildung werden im Folgenden erläutert:

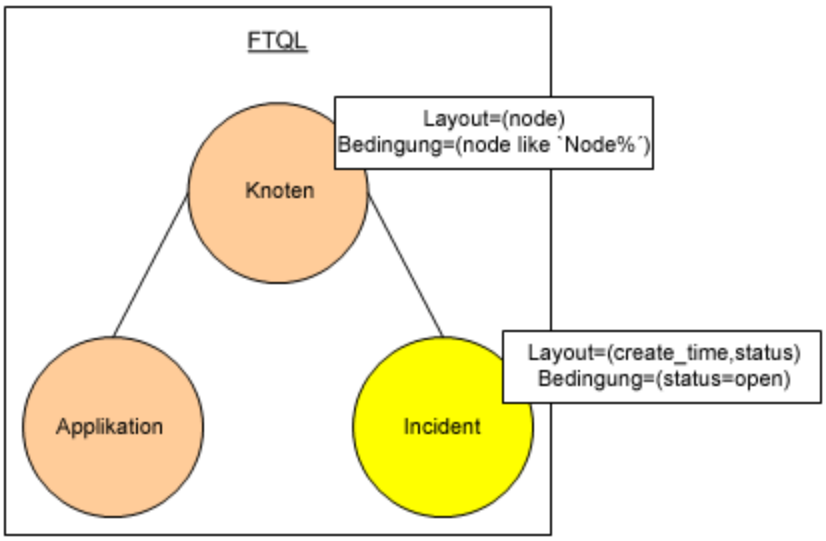
Zahl	Erklärung
1	Federation Framework empfängt einen Aufruf für die Berechnung einer föderierten TQL.
2	Federation Framework analysiert den Adapter, sucht die virtuelle Beziehung und teilt die ursprüngliche TQL in zwei Unteradapter – einen für UCMDB und einen für das externe Daten-Repository.
3	Federation Framework fordert die Topologie der Unter-TQL vom externen Adapter an. Das zurückgegebene Ergebnis `ExternalTopologyResult` sollte kein Abstimmungsobjekt enthalten, da die Abstimmungsdaten nicht Teil der Anforderung sind.
4	Nach dem Empfang der Topologieergebnisse ruft Federation Framework die entsprechende Zuordnungs-Engine mit der aktuellen virtuellen Beziehung auf und

Zahl	Erklärung
	fordert die Abstimmungsdaten an. Der Parameter `reconciliationObjects` ist zu diesem Zeitpunkt noch leer, d. h., in diesem Aufruf wird keine Bedingung zu den Abstimmungsdaten hinzugefügt. Die zurückgegebenen Abstimmungsdaten definieren, welche Daten erforderlich sind, um die Abstimmungs-CIs in UCMDB zum externen Daten-Repository zuzuordnen. Es gibt drei Arten von Abstimmungsdaten: • **IdReconciliationData.** Die Abstimmung der CIs erfolgt auf Basis ihrer ID. • **PropertyReconciliationData.** Die Abstimmung der CIs erfolgt auf Basis der Eigenschaften eines CIs. • **TopologyReconciliationData.** Die Abstimmung der CIs erfolgt auf Basis der Topologie (z. B. wird für die Abstimmung von Knoten-CIs auch die IP-Adresse von **IP** benötigt).
5	Federation Framework fordert die Abstimmungsobjekte für die CIs an, die in Schritt 3 vom externen Daten-Repository empfangen wurden. Anschließend ruft Federation Framework die Methode **getTopologyWithReconciliationData()** im externen Adapter auf. Die angeforderte Topologie ist eine Ein-Knoten-Topologie mit den in Schritt 3 empfangenen CIs als ID-Bedingung und den Abstimmungsdaten von Schritt 4.
6	Federation Framework ruft die Zuordnungs-Engine auf, um die Abstimmungsdaten abzurufen. Im Gegensatz zu Schritt 3 empfängt die Zuordnungs-Engine nun die Abstimmungsobjekte von Schritt 5 als Parameter. Die Zuordnungs-Engine übersetzt das empfangene Abstimmungsobjekt in die Abstimmungsdaten.
7	Federation Framework fordert die Topologie der Unter-TQL mit den Abstimmungsdaten von Schritt 6 von UCMDB an.
8	Federation Framework ruft die Zuordnungs-Engine auf, um eine Verbindung zwischen den empfangenen Ergebnissen herzustellen. Der Parameter `firstResult` ist das Ergebnis der externen Topologie, das vom externen Adapter in Schritt 5 empfangen wurde, und der Parameter `secondResult` ist das Ergebnis der externen Topologie, das von UCMDB in Schritt 7 empfangen wurde. Die Zuordnungs-Engine gibt eine Karte zurück, in der die ID des externen CI vom ersten Daten-Repository (in diesem Fall das externe Daten-Repository) den IDs des externen CI vom zweiten Daten-Repository (UCMDB) zugeordnet ist.
9	Für jede Zuordnung erstellt Federation Framework eine virtuelle Beziehung.
10	Nach der Berechung der Ergebnisse der föderierten TQL-Abfrage (nur auf der Topologiestufe) ruft Federation Framework das ursprüngliche TQL-Layout für die resultierenden CIs und Beziehungen von den entsprechenden Daten-Repositorys ab.

Beispiel für den Federation Framework-Fluss für föderierte TQL-Abfragen

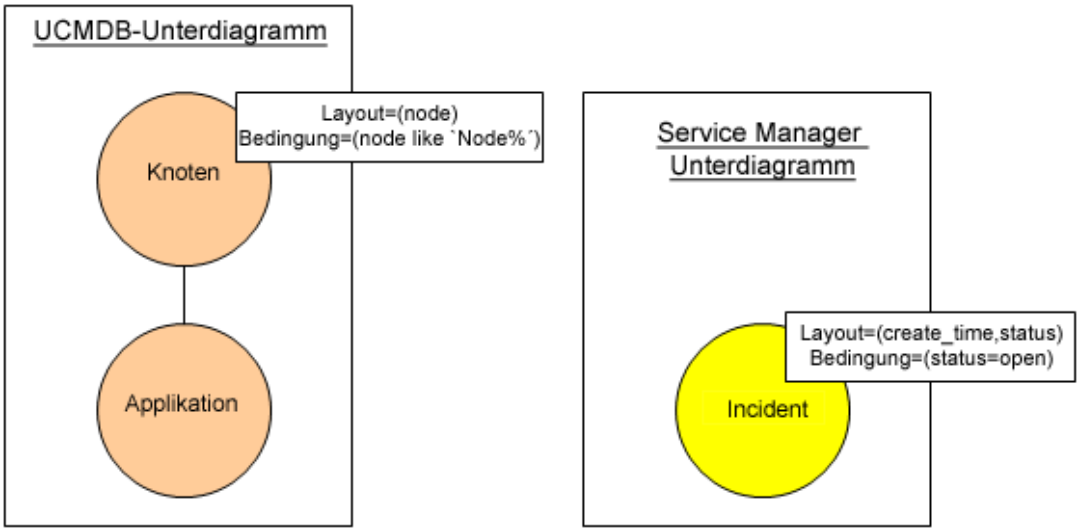
In diesem Beispiel wird erklärt, wie Sie alle offenen Vorfälle bei bestimmten Knoten anzeigen. Das ServiceCenter-Daten-Repository ist das externe Daten-Repository. Die Knoteninstanzen sind in UCMDB gespeichert und die Vorfallinstanzen in ServiceCenter. Es wird davon ausgegangen, dass

die Eigenschaften `node` und `ip_address` von Host und IP erforderlich sind, um die Vorfallinstanzen mit dem entsprechenden Knoten zu verbinden. Hierbei handelt es sich um die Abstimmungseigenschaften, die die ServiceCenter-Knoten in UCMDB identifizieren.

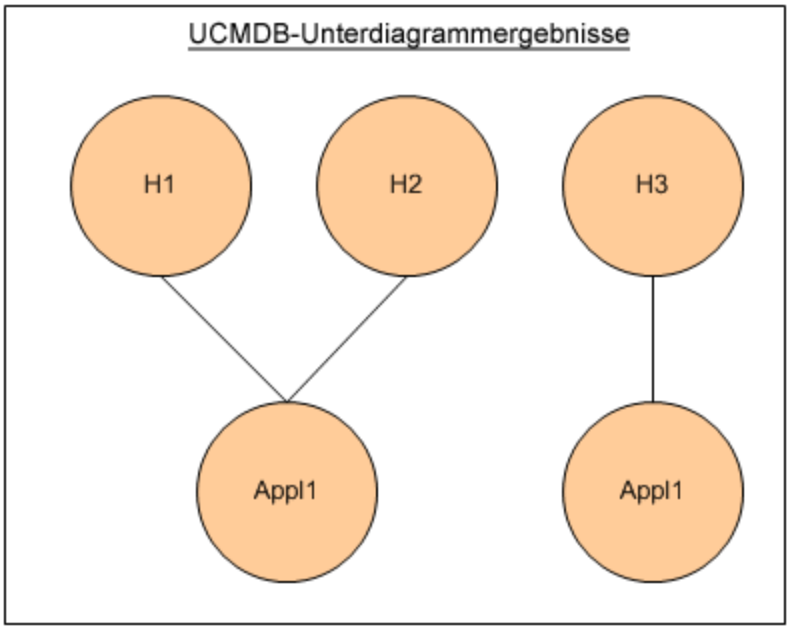


Hinweis: Für die Attributföderation wird die Methode **getTopology** des Adapters aufgerufen. Die Abstimmungsdaten werden in der Benutzer-TQL (in diesem Fall das CI-Element) angepasst.

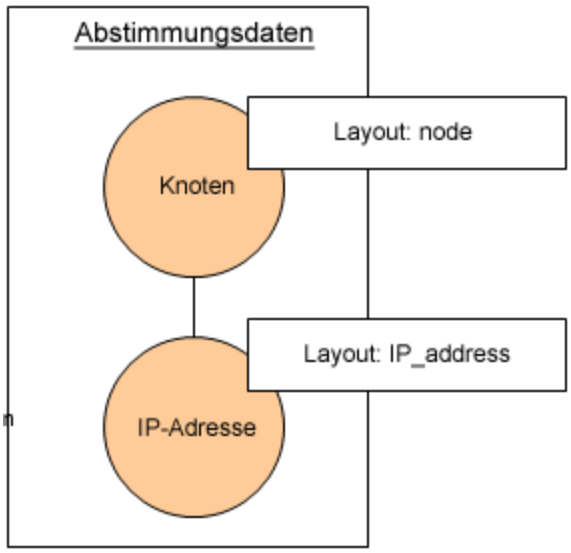
1. Nach der Analyse des Adapters erkennt Federation Framework die virtuelle Beziehung zwischen dem `Knoten` und dem `Vorfall` und unterteilt die föderierte TQL-Abfrage in zwei Unterdiagramme:



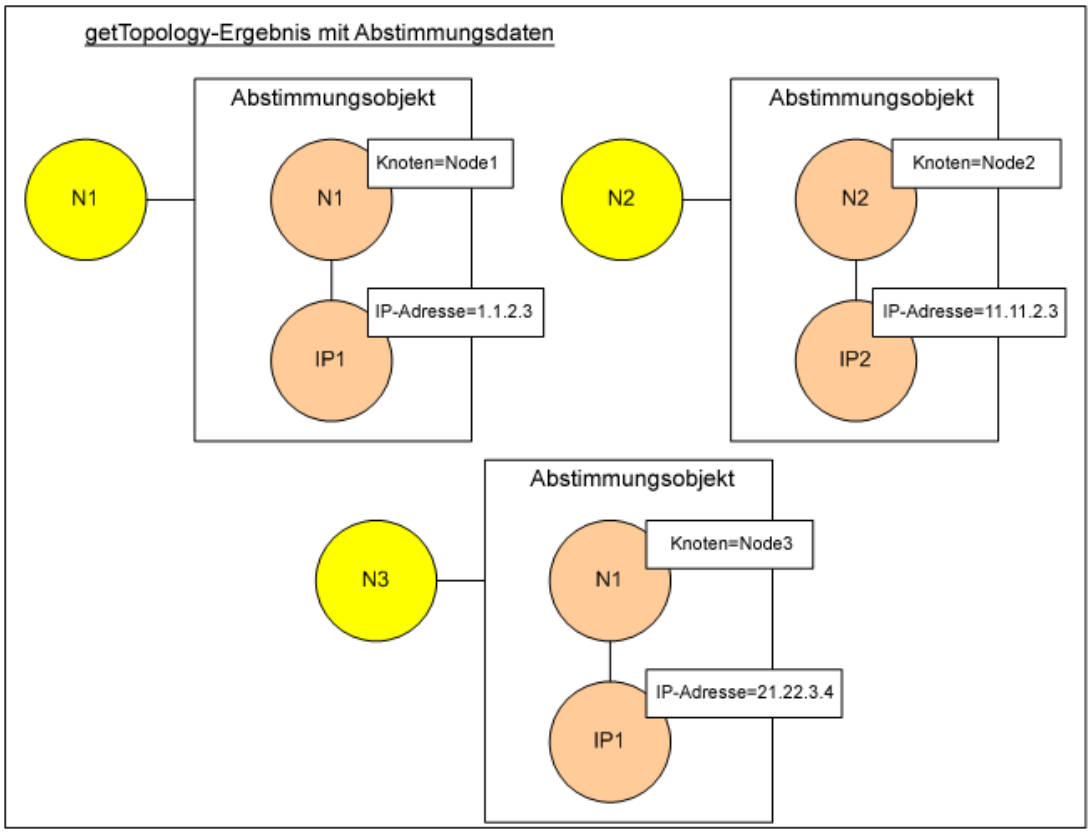
2. Federation Framework führt das UCMDB-Unterdiagramm aus, um die Topologie anzufordern, und empfängt die folgenden Ergebnisse:



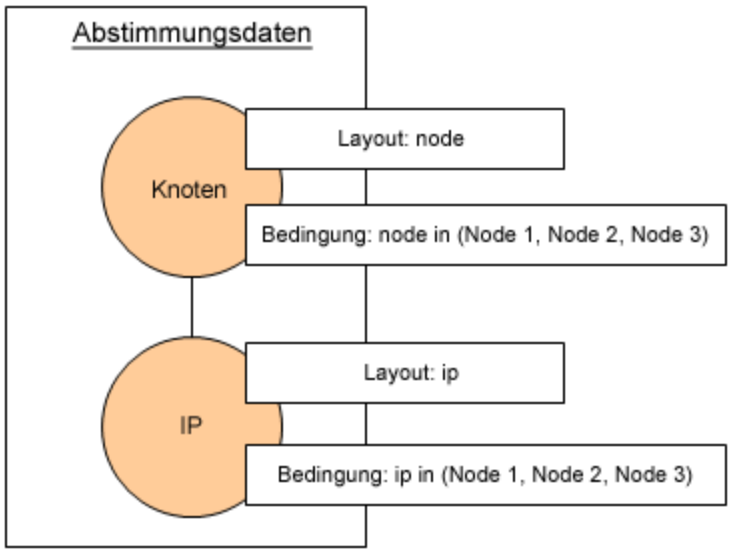
3. Federation Framework fordert von der entsprechenden Zuordnungs-Engine die Abstimmungsdaten für das erste Daten-Repository (UCMDB) an, das die Informationen zum Herstellen einer Verbindung zwischen den von den beiden Daten-Repositorys empfangenen Daten enthält. Die Abstimmungsdaten in diesem Fall lauten wie folgt:



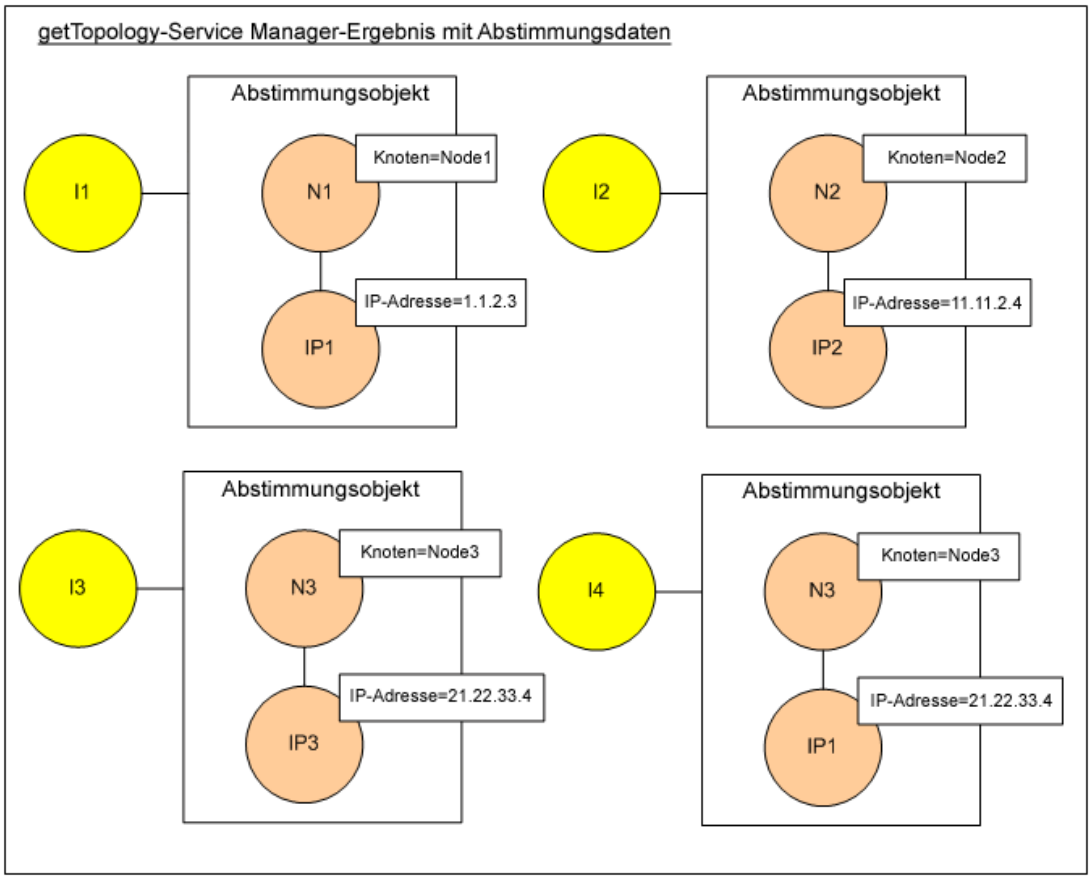
4. Federation Framework erstellt eine Ein-Knoten-Topologieabfrage mit den Knoten- und ID-Bedingungen vom vorherigen Ergebnis (`node` in H1, H2, H3) und führt diese Abfrage mit den erforderlichen Abstimmungsdaten bei UCMDB aus. Das Ergebnis enthält die für die ID-Bedingung relevanten Knoten-CIs und das entsprechende Abstimmungsobjekt für jedes CI:



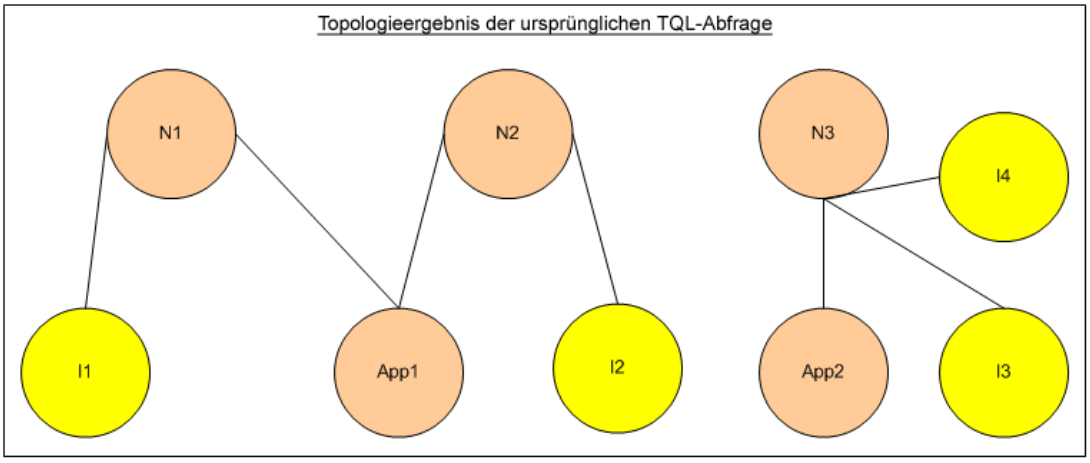
5. Die Abstimmungsdaten für ServiceCenter sollten eine Bedingung für `node` und `ip` enthalten, die von den von UCMDB empfangenen Abstimmungsobjekten abgeleitet wird:



6. Federation Framework führt das ServiceCenter-Unterdiagramm mit den Abstimmungsdaten aus, um die Topologie und die entsprechenden Abstimmungsobjekte anzufordern, und empfängt die folgenden Ergebnisse:



7. Das Ergebnis nach der Verbindung in der Zuordnungs-Engine und der Erstellung der virtuellen Beziehungen stellt sich wie folgt dar:



8. Federation Framework fordert das ursprüngliche TQL-Layout für die empfangenen Instanzen von UCMDB und ServiceCenter an.

Federation Framework-Fluss für Auffüllungen

Dieser Abschnitt umfasst die folgenden Themen:

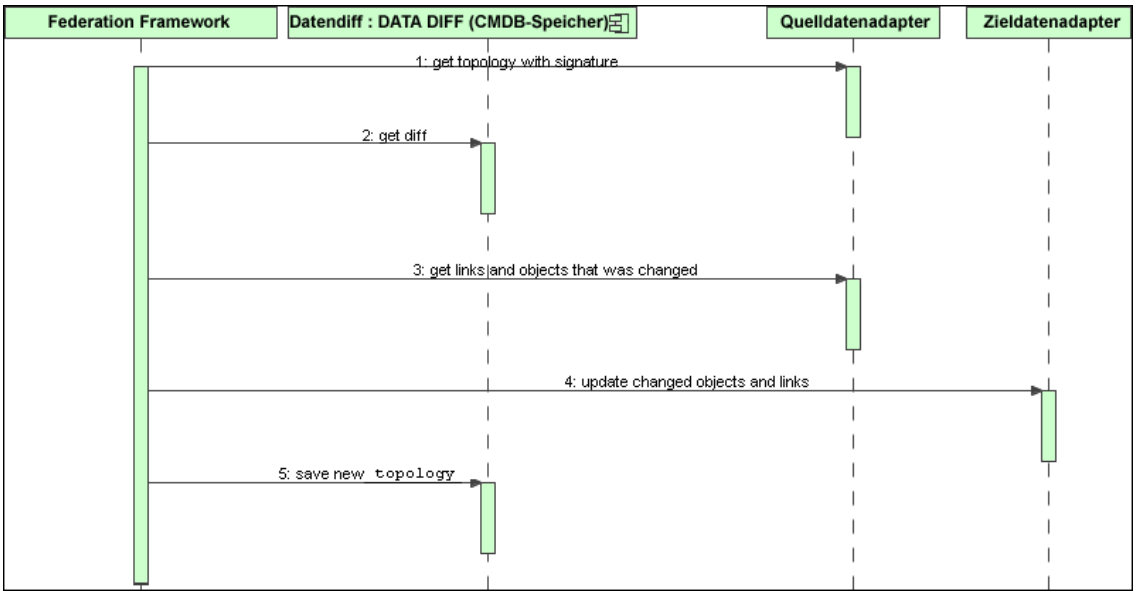
- "Definitionen und Begriffe" oben
- "Ablaufdiagramm" oben

Definitionen und Begriffe

Signatur. Bezeichnet den Status der Eigenschaften im CI. Wenn Änderungen an den Eigenschaftswerten in einem CI vorgenommen werden, muss die CI-Signatur ebenfalls geändert werden. Anhand der CI-Signatur kann festgestellt werden, ob ein CI geändert wurde, ohne dass alle CI-Eigenschaften abgerufen und verglichen werden müssen. Sowohl das CI als auch die CI-Signatur werden vom entsprechenden Adapter bereitgestellt. Der Adapter ist für die Änderung der CI-Signatur verantwortlich, nachdem die CI-Eigenschaften geändert wurden.

Ablaufdiagramm

Das folgende Ablaufdiagramm veranschaulicht die Interaktion zwischen Federation Framework und dem Quell- und Zieladapter in einem Auffüllungsfluss:



1. Federation Framework empfängt die Topologie für das Abfrageergebnis vom Quelladapter. Der Adapter erkennt die Abfrage anhand ihres Namens und führt sie beim externen Daten-Repository aus. Das Topologieergebnis enthält die ID und die Signatur für jedes CI und jede Beziehung im Ergebnis. Die ID ist die logische ID zur eindeutigen Kennzeichnung des CI im externen Daten-Repository. Die Signatur sollte geändert werden, wenn das CI oder die Beziehung geändert wird.
2. Federation Framework verwendet Signaturen, um die neu empfangenen Ergebnisse von Topologieabfragen mit den gespeicherten Ergebnissen zu vergleichen und um festzustellen, welche CIs geändert wurden.
3. Nachdem Federation Framework die geänderten CIs und Beziehungen gefunden hat, ruft es den Quelladapter mit den IDs der geänderten CIs und Beziehungen als Parameter auf, um das vollständige Layout abzurufen.
4. Federation Framework sendet die Aktualisierung an den Zieladapter. Der Zieladapter aktualisiert die externe Datenquelle mit den empfangenen Daten.
5. Nach der Aktualisierung speichert Federation Framework das Ergebnis der letzten Abfrage.

Adapterschnittstellen

Dieser Abschnitt umfasst die folgenden Themen:

Definitionen und Begriffe

Externe Beziehung. Die Beziehung zwischen zwei externen CI-Typen, die von demselben Adapter unterstützt werden.

Adapterschnittstellen für föderierte TQL-Abfragen

Verwenden Sie für jeden Adapter die entsprechende Adapterschnittstelle. Beachten Sie dabei die folgenden Hinweise.

- Eine **OneNode-Topologieschnittstelle** wird verwendet, wenn der Adapter keine externen Beziehungen unterstützt, d. h. der Adapter erhält grundsätzlich keine Anforderungen mit mehr als einem externen CI. Alle OneNode-Schnittstellen werden zur Vereinfachung des Workflows erstellt. Verwenden Sie für komplexere Abfragen die `DataAdapter`-Schnittstelle.
- Eine **DataAdapter-Schnittstelle** wird zur Definition von Adaptern verwendet, die komplexe föderierte Abfragen unterstützen. Die Abstimmungsanforderung in diesen Adaptern ist Teil des einzelnen **QueryDefinition**-Parameters. Diese Adapter können auch für Auffüllungen verwendet werden.

OneNode-Schnittstellen

Die folgenden Schnittstellen weisen unterschiedliche Arten von Abstimmungsdaten auf:

- **OneNodeTopologyIdReconciliationDataAdapter.** Verwenden Sie diese Schnittstelle, wenn der Adapter eine **Einzelknoten-TQL** unterstützt und die Abstimmung zwischen den Daten-Repositorys anhand der ID berechnet wird.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Verwenden Sie diese Schnittstelle, wenn der Adapter eine **Einzelknoten-TQL** unterstützt und die Abstimmung zwischen den Daten-Repositorys anhand der Eigenschaft eines CI erfolgt.
- **OneNodeTopologyDataAdapter.** Verwenden Sie diese Schnittstelle, wenn der Adapter eine **Einzelknoten-TQL** unterstützt und die Abstimmung zwischen den Daten-Repositorys anhand der Topologie erfolgt.

DataAdapter-Schnittstellen

- **DataAdapter.** Verwenden Sie diesen Adapter, um komplexe föderierte TQL-Abfragen zu unterstützen. Ermöglicht die größte Vielfalt.
- **PopulateDataAdapter.** Verwenden Sie diesen Adapter, um komplexe föderierte TQL-Abfragen und Auffüllungsflüsse zu unterstützen. In einem Auffüllungsfluss ruft dieser Adapter den gesamten Datensatz ab und lässt die Probe die Unterschiede seit der letzten Ausführung des Jobs herausfiltern.

- **PopulateChangesDataAdapter.** Verwenden Sie diesen Adapter, um komplexe föderierte TQL-Abfragen und Auffüllungsflüsse zu unterstützen. In einem Auffüllungsfluss unterstützt dieser Adapter das ausschließliche Abrufen der Änderungen, die seit der letzten Ausführung des Jobs vorgenommen wurden.

Hinweis: Bei der Entwicklung eines Adapters der große Datensätze zurückgeben kann, ist es wichtig, durch Implementieren der ChunkGetter-Schnittstelle die Bildung von Chunks zu ermöglichen. Weitere Informationen finden Sie im Java-Dokument des entsprechenden Adapters.

Zusätzliche Schnittstellen

- **SortResultDataAdapter.** Verwenden Sie diese Schnittstelle, wenn Sie die resultierenden CIs im externen Daten-Repository sortieren können.
- **FunctionalLayoutDataAdapter.** Verwenden Sie diese Schnittstelle, wenn Sie das funktionale Layout im externen Daten-Repository berechnen können.

Adapterschnittstellen für die Synchronisierung

- **SourceDataAdapter.** Verwenden Sie diese Schnittstelle für Quelladapter in Auffüllungsflüssen.
- **TargetDataAdapter.** Verwenden Sie diese Schnittstelle für Zieladapter in Datenpush-Flüssen.

Debuggen von Adapterressourcen

Im Rahmen dieser Aufgabe wird beschrieben, wie Sie die JMX-Konsole verwenden, um Adapterstatusressourcen (d. h. Ressourcen, die mit den Methoden zur Ressourcenbearbeitung in der DataAdapterEnvironment-Schnittstelle erstellt wurden, die in der UCMDB-Datenbank oder der Probendatenbank gespeichert sind) zu Debugging- und Entwicklungszwecken zu erstellen, anzuzeigen und zu löschen.

1. Starten Sie den Webbrowser, und geben Sie die Serveradresse wie folgt ein:
   - Für den UCMDB-Server: `http://localhost:8080/jmx-console`
   - Für die Probe: `http://localhost:1977`

   Eventuell müssen Sie sich mit einem Benutzernamen und einem Kennwort anmelden (die Standardeinstellung lautet **sysadmin/sysadmin**).
2. Führen Sie eine der folgenden Aktionen aus, um die Seite **JMX MBEAN View** zu öffnen:
   - Auf dem UCMDB-Server: Klicken Sie auf **UCMDB:service=FCMDB Adapter State Resource Services.**
   - Auf der Probe: Klicken Sie auf **type=AdapterStateResources**.
3. Geben Sie Werte in die Operationen ein, die Sie verwenden möchten, und klicken Sie auf **Invoke**.

Hinzufügen eines Adapters für eine neue externe Datenquelle

Im Folgenden wird beschrieben, wie Sie einen Adapter zur Unterstützung einer neuen externen Datenquelle definieren.

Diese Aufgabe umfasst folgende Schritte:

1. ## Voraussetzungen

   Modellunterstützte Adapterklassen für CIs und Beziehungen im UCMDB-Datenmodell. Als Adapterentwickler sollten Sie:

   - über Kenntnisse der Hierarchie von UCMDB-CITs verfügen, um das Zusammenspiel zwischen externen CITs und UCMDB-CITs zu verstehen;
   - die externen CITs im UCMDB-Klassenmodell modellieren;
   - die Definitionen für neue CI-Typen und ihre Beziehungen hinzufügen;
   - gültige Beziehungen im UCMDB-Klassenmodell für die gültigen Beziehungen zwischen den internen Adapterklassen definieren. (Die CITs können an beliebiger Stelle in der Struktur des UCMDB-Klassenmodells platziert werden.)

   Die Modellierung sollte ungeachtet des Föderationstyps auf die gleiche Weise erfolgen (nach dem On-the-Fly- oder dem Replizierungsprinzip). Weitere Informationen zum Hinzufügen neuer CIT-Definitionen zum UCMDB-Klassenmodell finden Sie unter "Verwenden der CI-Auswahl" im HP Universal CMDB – Modellierungshandbuch.

   Damit der Adapter föderierte Attribute bei CITs unterstützt, fügen Sie diesen CIT zu den unterstützten Klassen mit den unterstützten Attributen und der Abstimmungsregel für diesen CIT hinzu.

2. ## Definieren gültiger Beziehungen für virtuelle Beziehungen

   Hinweis: Dieser Abschnitt ist nur für die Föderation relevant.

Um föderierte CITs abzurufen, die mit lokalen CMDB-CITs verbunden sind, muss eine gültige Linkdefinition zwischen den beiden CITs in der CMDB vorhanden sein.

a. Erstellen Sie eine XML-Datei mit gültigen Links (sofern diese noch nicht vorhanden sind).

b. Fügen Sie die XML-Datei mit den Links zum Adapter-Package im Ordner **\validlinks** hinzu. Weitere Informationen finden Sie unter "Package Manager" im *HP Universal CMDB – Verwaltungshandbuch*.

Beispiel für die Definition einer gültigen Beziehung:

Im folgenden Beispiel ist der Beziehungstyp `containment` zwischen den Instanzen vom Typ `node` und den Instanzen vom Typ `myclass1` eine gültige Beziehungsdefinition.

```
<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. Definieren einer Adapterkonfiguration

a. Öffnen Sie **Adapterverwaltung**.

b. Klicken Sie auf die Schaltfläche **Neue Ressource erstellen**  .

c. Wählen Sie im Dialogfeld **Neuer Adapter** die Optionen **Integration** und **Java-Adapter** aus.

d. Klicken Sie mit der rechten Maus auf den soeben erstellten Adapter und wählen Sie im Kontextmenü die Option **Adapter-Quelle bearbeiten** aus.

e. Bearbeiten Sie die folgenden XML-Tags:

```
<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="newAdapterIdName"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
description="Adapter Description" schemaVersion="9.0"
displayName="New Adapter Display Name">

<deletable>true</deletable>

<discoveredClasses>

<discoveredClass>link</discoveredClass>

<discoveredClass>object</discoveredClass>

</discoveredClasses>

<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.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.BaseDestinationDat-
a">

<!-- 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-->

<Topologie>

<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>
```

Weitere Informationen zu den XML-Tags finden Sie unter "Tags und Eigenschaften für die XML-Konfiguration " auf Seite 187.

4. ## Definieren der unterstützten Klassen

Definieren Sie die unterstützten Klassen entweder mithilfe des Adaptercodes, indem Sie die Methode *getSupportedClasses()* implementieren, oder mithilfe der Pattern-XML-Datei.

```
<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	Der Name des CI-Typs.
is-derived	Gibt an, ob diese Definition alle erbenden untergeordneten Elemente enthält.
is-reconciliation-supported	Gibt an, ob diese Klasse für die Abstimmung verwendet wird.
is-id-reconciliation-supported	Gibt an, ob diese Klasse für die Abstimmung per ID verwendet wird.
federation-not-supported	Gibt an, ob dieser CIT nicht für die Föderation zugelassen werden soll (beispielsweise kann durch das Blockieren bestimmter CITs ein CIT ausschließlich für die Föderation definiert werden).
<supported-conditions>	Gibt die bei jedem Attribut unterstützten Bedingungen an.

5. ## Implementieren des Adapters

Wählen Sie anhand der definierten Funktionen die korrekte Klasse für die Adapterimplementierung aus. Diese Klasse implementiert die entsprechenden Schnittstellen gemäß den definierten Funktionen.

Die Unterstützung der Adapterabstimmung kann gemäß **global_id** definiert werden, wobei **global_id** als Teil der Abstimmungsattribute in den vom Adapter unterstützten Klassen definiert sein muss. Wird die Unterstützung der Adapterabstimmung gemäß **global_id** definiert, muss **global_id** als Teil der Abstimmungsobjekteigenschaften von **getTopologyWithReconciliationData()** zurückgegeben werden. UCMDB verwendet **global_id** anstelle der Identifikationsregel für die Abstimmung der Föderationsergebnisse für ein CIT.

6. ## Definieren der Abstimmungsregeln oder Implementieren der Zuordnungs-Engine

Wenn Ihr Adapter föderierte TQL-Abfragen unterstützt, haben Sie drei Möglichkeiten, um eine Zuordnungs-Engine zu definieren:

- Sie können die Standard-Zuordnungs-Engine von CMDB 9.0x verwenden, die für die Zuordnung die internen Abstimmungsregeln von CMDB nutzt. Wenn Sie diese Engine verwenden möchten, lassen Sie das XML-Tag **<default-mapping-engine>** leer.

  Weitere Informationen finden Sie unter "Die Datei "reconciliation_types.txt"" auf Seite 137.

- Sie können die Zuordnungs-Engine von CMDB 8.0x verwenden. Hierzu benötigen Sie das folgende XML-Tag: **<default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>**

  Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 137.

- Sie können Ihre eigene Zuordnungs-Engine schreiben, indem Sie die Schnittstelle der Zuordnungs-Engine implementieren und die JAR-Dateien mit dem Rest des Adaptercodes

anordnen. Verwenden Sie hierzu das folgende XML-Tag: **<default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>**

7. ## Hinzufügen der für die Implementierung erforderlichen JAR-Dateien zum Klassenpfad

   Um Ihre Klassen zu implementieren, müssen Sie die Datei **federation_api.jar** zum Klassenpfad Ihres Code-Editors hinzufügen.

8. ## Bereitstellen des Adapters

   Stellen Sie das Adapter-Package bereit. Allgemeine Informationen zum Bereitstellen eines Package finden Sie unter "Package Manager" im *HP Universal CMDB – Verwaltungshandbuch*.

   Das Package sollte die folgenden Entitäten enthalten:

   - Definition neuer CITs (optional):
   - Wird nur verwendet, wenn der Adapter neue CI-Typen unterstützt, die noch nicht in UCMDB vorhanden sind.
   - Die neuen Definitionen werden im Ordner `class` des Package gespeichert.
   - Definition neuer Datentypen (optional):
   - Wird nur verwendet, wenn die neuen CITs neue Datentypen erfordern.
   - Die neuen Definitionen werden im Ordner `typedef` des Package gespeichert.
   - Definition neuer gültiger Beziehungen (optional):
   - Wird nur verwendet, wenn der Adapter die föderierte TQL unterstützt.
   - Die neuen Definitionen werden im Ordner `validlinks` des Package gespeichert.
   - Die XML-Datei mit der Pattern-Konfiguration sollte im Ordner `discoveryPatterns` des Package gespeichert werden.
   - **Deskriptor.** Definiert die Package-Definitionen.
   - Speichern Sie Ihre kompilierten Klassen (normalerweise eine JAR-Datei) im Ordner **adapterCode\<Adapter_ID>** des Package.

     Hinweis: Der Name des Ordners `Adapter_ID` ist identisch mit dem Wert in der Adapterkonfiguration.

   - Wenn Sie Ihre eigene Konfigurationsdatei erstellen, sollten Sie diese im Ordner **adapterCode\<Adapter_ID>** des Package speichern.

9. ## Aktualisieren des Adapters

   Änderungen an den nicht binären Dateien des Adapters können im Modul **Adapterverwaltung** vorgenommen werden. Wenn Änderungen an den Konfigurationsdateien im Modul **Adapterverwaltung** vorgenommen werden, wird der Adapter mit den neuen Konfigurationen neu geladen.

   Aktualisierungen können ebenfalls durch Bearbeiten der Dateien im Package (sowohl binäre als auch nicht binäre Dateien) und das anschließende erneute Bereitstellen des Package

mithilfe von Package Manager durchgeführt werden. Weitere Informationen hierzu finden Sie unter "Bereitstellen eines Package" im *HP Universal CMDB – Verwaltungshandbuch*.

Implementieren der Zuordnungs-Engine

Die Konfiguration der Zuordnungs-Engine hängt von der verwendeten Engine ab.

Diese Aufgabe umfasst folgende Schritte:

- "Konfigurieren der Datei "reconciliation_types.txt" (für die Standard-Zuordnungs-Engine von UCMDB 9.0x)" oben
- "Konfigurieren der Datei "reconciliation_rules.txt" (für die Zuordnungs-Engine von UCMDB 8.0x) " oben

1. **Konfigurieren der Datei "reconciliation_types.txt" (für die Standard-Zuordnungs-Engine von UCMDB 9.0x)**

   Mit dieser Datei wird definiert, welche CI-Typen für die Abstimmung im Adapter verwendet werden.

   Schreiben Sie jeden für die Abstimmung verwendeten CI-Typ wie im folgenden Beispiel gezeigt in eine einzelne Zeile:

   ```
   node business_application
   ```

   Speichern Sie die Datei im Adapter-Package im Ordner **adapterCode\<Adapter-ID>\META-INF**. Um die ID-Abstimmung zu unterstützen (d. h. die Abstimmung, die auf der ID-Zuordnung zwischen der CMDB-ID in der CMDB und einem Wert in der Remote-Datenbank basiert), sollten Sie ein spezielles CMDB-Attribut mit der Bezeichnung **cmdb_id** zu einer Datenbankspalte vom Typ "String" (char, varchar) oder "Byte[]" (raw/bytes) zuordnen.

2. **Konfigurieren der Datei "reconciliation_rules.txt" (für die Zuordnungs-Engine von UCMDB 8.0x)**

   Diese Datei wird zum Konfigurieren der Abstimmungsregeln verwendet. Jede Zeile in der Datei stellt eine Regel dar. Beispiel:

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

   Der Parameter **reconcilition_type** wird mit dem Typ des CI gefüllt, bei dem die Abstimmung durchgeführt wird (der UCMDB-Klassenname, der mit der föderierten Klasse in der TQL verbunden ist).

   Der Parameter **expression** ist die Logik, die entscheidet, ob zwei Abstimmungsobjekte identisch sind (ein Abstimmungsobjekt von der UCMDB-Seite und das andere von der Seite des föderierten Adapters).

   Der **expression**-Ausdruck besteht aus mehreren OR- und AND-Elementen.

   Die Konvention bezüglich der Attributnamen im Ausdrucksteil lautet **[KlassenName].[AttributName]**. Beispiel: Das Attribut **ip_address** in der Klasse **ip** wird wie folgt geschrieben: **ip.ip_address**.

   Sie können Übereinstimmungen fester Ordnung definieren. Dabei wird zunächst der erste OR-Unterausdruck geprüft. Wenn zwei Abstimmungsobjekte einen Wert bei den Attributen des

Unterausdrucks aufweisen und **false** zurückgegeben wird (d. h., die Abstimmungsobjekte sind nicht identisch), dann wird der zweite OR-Unterausdruck nicht verglichen.

Verwenden Sie für eine Übereinstimmung fester Ordnung den Parameter **ordered expression** anstatt **expression**.

Das Zirkumflex-Zeichen (**^**) gibt an, dass bei den Vergleichen nicht zwischen Groß- und Kleinschreibung unterschieden werden soll.

Die anderen Parameter (**end1_type**, **end2_type** und **link_type**) werden nur verwendet, wenn die Abstimmungsdaten zwei Knoten enthalten und nicht nur den Knoten des Abstimmungstyps (topologische Abstimmungsdaten). In diesem Fall lauten die Abstimmungsdaten **end1_type -(link_type)> end2_type**.

Das relevante Layout muss nicht hinzugefügt werden, da es vom Ausdruck abgerufen wird.

Um die Abstimmung anhand der UCMDB-ID durchzuführen, verwenden Sie **cmdb_id** als Attributnamen im Ausdruck.

Speichern Sie die Datei im Adapter-Package im Ordner **adapterCode\<Adapter_ID>\META-INF**.

Beispiele:

- Abstimmungsregeln können nur für CIs des Typs **node** hinzugefügt werden. Der Grund hierfür ist, dass nur CIs des Typs **node** gültige Beziehungen mit externen CITs haben. Beispiel: Ein Knoten-CI in der CMDB wird über das Attribut `node.name` oder `ip_address.name` einem Knoten-CI in ServiceCenter zugeordnet.
- Die Abstimmungsregel ist in diesem Fall eine Topologieregel und der Ausdrucksparameter lautet **ordered expression**. Die Regel führt im Rahmen des Vergleichs die folgenden Prüfungen bei den CIs durch:
  - Wenn das Attribut `node.name` identisch ist, ordnet die Regel die Knoten zu.
  - Wenn das Attribut `node.name` nicht identisch ist, ordnet die Regel die Knoten nicht zu.
  - Wenn das Attribut `node.name` bei einem der verglichenen CIs null ist, prüft die Regel das Attribut `ip_address.name`. Wenn das Attribut `ip_address.name` identisch ist, ordnet die Regel die Knoten zu.

Erstellen eines Beispieladapters

Im Folgenden wird beschrieben, wie Sie einen Beispieladapter erstellen. Diese Aufgabe umfasst folgende Schritte:

- "Auswählen der Adapterlogik" oben
- "Laden des Projekts" auf der nächsten Seite

1. Auswählen der Adapterlogik

Wenn Sie einen Adapter implementieren, müssen Sie auswählen, wie die Bedingungslogik in der Implementierung gehandhabt werden soll (Eigenschaftsbedingungen, ID-Bedingungen, Abstimmungsbedingungen und Linkbedingungen).

a. Rufen Sie die gesamten Daten im Adapterspeicher ab und lassen Sie die erforderlichen CI-Instanzen automatisch auswählen oder filtern.

b. Konvertieren Sie alle Bedingungen in die Sprache der Datenquelle und lassen Sie die Daten automatisch filtern und auswählen. Beispiel:

   - Konvertieren Sie die Bedingung in eine SQL-Abfrage.
   - Konvertieren Sie die Bedingung in ein Java API-Filterobjekt.

c. Filtern Sie einige der Daten beim Remote-Service und lassen Sie den Adapter die übrigen Daten auswählen und filtern.

Im MyAdapter-Beispiel wird die Logik aus Option *a* verwendet.

2. **Laden des Projekts**

Kopieren Sie die Dateien aus dem Ordner **C:\hp\UCMDB\UCMDBServer\tools\adapter-dev-kit\SampleAdapters** und folgen Sie den Anweisungen in den Readme-Dateien.

> **Hinweis:** Wenn Sie einen Adapter mit großen Datensätzen verwenden, müssen Sie zur Verbesserung der Leistung für die Föderation eventuell mit Zwischenspeicherung und Indizierung arbeiten.

Die Online-Dokumentation für Java-Dokumente ist verfügbar unter:

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

Tags und Eigenschaften für die XML-Konfiguration

id="newAdapterIdName"		Definiert den tatsächlichen Namen des Adapters. Wird für Protokolle und die Suche in Ordnern verwendet.
displayName="New Adapter Display Name"		Definiert den Anzeigenamen des Adapters, wie er auf der Benutzeroberfläche erscheint.
<className>…</className>		Definiert die Schnittstelle des Adapters, die die Java-Klasse implementiert.
<category >My Category</category>		Definiert die Kategorie des Adapters.
<parameters>		Definiert die Eigenschaften für die Konfiguration, die auf der Benutzeroberfläche zur Verfügung stehen, wenn ein neuer Integrationspunkt eingerichtet wird.
	name	Der Name der Eigenschaft (wird hauptsächlich vom Code verwendet)
	description	Der Anzeigehinweis zur Eigenschaft

	type	Zeichenkette oder Ganzzahl (verwenden Sie gültige Werte mit Zeichenketten für boolesche Werte).
	display-name	Der Name der Eigenschaft auf der Benutzeroberfläche.
	mandatory	Gibt an, ob diese Konfigurationseigenschaft für den Benutzer obligatorisch ist.
	order-index	Die Platzierungsreihenfolge der Eigenschaft (klein = oben)
	valid-values	Eine semikolongetrennte Liste der möglichen gültigen Werte (z. B. valid-values="Oracle;SQLServer;MySQL" oder valid-values="True;False").
<adapterInfo>		Enthält die Definition der statischen Einstellungen und Funktionen des Adapters.
	<support-federated-query>	Definiert diesen Adapter als föderationsfähig.
	<one-node-topology>	Die Fähigkeit, föderierte Abfragen mit einem föderierten Abfrageknoten durchzuführen.
	<pattern-topology>	Die Fähigkeit, komplexe Abfragen zu föderieren.
	<support-replication-data>	Definiert die Fähigkeit, Datenpush- und Auffüllungsflüsse auszuführen.
	<source>	Dieser Adapter kann für Auffüllungsflüsse verwendet werden.
	<push-back-ids>	Die globale ID des CI wird zur Spalte **global_id** der Tabelle zurückgegeben (muss in der Datei **orm.xml** definiert sein). Das Verhalten kann durch Implementieren des Plugins **FcmdbPluginPushBackIds** überschrieben werden.
	<changes-source>	Dieser Adapter kann für Auffüllungsänderungsflüsse verwendet werden.
	<target>	Dieser Adapter kann für Datenpush-Flüsse verwendet werden.
	<default-mapping-engine>	Ermöglicht die Definition einer Zuordnungs-Engine für den Adapter (standardmäßig verwendet der Adapter die Standard-Zuordnungs-Engine). Um eine andere Zuordnungs-Engine zu verwenden, geben Sie den Namen der implementierenden Klasse der Zuordnungs-Engine ein (für die Zuordnungs-

		Engine von UCMDB 8.0x lautet dieser wie folgt: **com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine**)
	<removedAttributes>	Erzwingt die Entfernung bestimmter Attribute vom Ergebnis.
	<full-population-days-interval>	Gibt an, wann ein vollständiger Auffüllungsjob anstelle eines differenziellen Jobs durchgeführt werden soll (alle 'x' Tage). Verwendet den Alterungsmechanismus zusammen mit dem Änderungsfluss.
	<adapter-settings>	Die Liste der Einstellungen des Adapters.
	<list.attributes.for.set>	Bestimmt, welche Attribute den vorherigen Wert (sofern vorhanden) überschreiben.

Kapitel 6

Entwickeln von Push-Adaptern

Dieses Kapitel umfasst folgende Themen:

Entwickeln von Push-Adaptern – Übersicht

Der generische Push-Adapter stellt eine Plattform zur Verfügung, die eine schnelle Entwicklung von Integrationen ermöglicht, um Daten der UCMDB-Version 9.0x per Push an externe Daten-Repositorys (Datenbanken und Applikationen von Drittanbietern) zu übertragen. Die Entwicklung einer benutzerdefinierten Integration auf Basis eines generischen Push-Adapters erfordert Folgendes:

- Eine XML-Zuordnungsdatei zwischen den CI-Linktypen von UCMDB und den externen Datenelementen.
- Ein Jython-Skript, um die Datenelemente per Push in das externe Daten-Repository zu übertragen.

Differenzielle Synchronisierung

Damit der Push-Adapter die differenzielle Synchronisierung unterstützt, muss die Funktion **DiscoveryMain** ein Objekt zurückgeben, das die **DataPushResults**-Schnittstelle implementiert, welche die Zuordnungen zwischen den IDs, die das Jython-Skript von der XML empfängt, und den IDs, die das Jython-Skript auf dem Remote-Computer erstellt, enthält. Letztere sind IDs des Typs **ExternalId**.

Der Befehl **ExternalIdUtil.restoreExternal**, der die ID des CI in der CMDB als Parameter empfängt, stellt die externe ID von der ID des CI in der CMDB wieder her. Dieser Befehl kann beispielsweise während der Durchführung der differenziellen Synchronisierung verwendet werden. Wenn sich ein Ende nicht im Bundle befindet (d. h. bereits synchronisiert wurde), wird ein Link empfangen.

Wenn die **DiscoveryMain**-Methode im Jython-Skript, auf dem der Push-Adapter basiert, eine leere **ObjectStateHolderVector**-Instanz zurückgibt, wird die differenzielle Synchronisierung nicht vom Adapter unterstützt. Dies bedeutet, dass selbst wenn ein Job mit differenzieller Synchronisierung ausgeführt wird, tatsächlich eine vollständige Synchronisierung erfolgt. Es ist daher nicht möglich, Daten auf dem Remote-System zu aktualisieren oder zu entfernen, da während der Synchronisierungen alle Daten zur CMDB hinzugefügt werden.

Vorbereiten der Zuordnungsdateien

Hinweis: Sie können alle CIs und Beziehungen ohne Zuordnung direkt aus der CMDB abrufen, indem Sie auf die Erstellung der Datei **mappings.xml** verzichten. In diesem Fall werden alle CIs und Beziehungen mit allen ihren Attributen zurückgegeben.

Es gibt zwei verschiedene Möglichkeiten zum Vorbereiten von Zuordnungsdateien:

- Sie können eine einzelne, globale Zuordnungsdatei vorbereiten.

  Alle Zuordnungen werden in einer einzelnen Datei namens **mappings.xml** gespeichert.

- Sie können für jede Push-Abfrage eine separate Datei vorbereiten.

  Jede Zuordnungsdatei hat den Namen **<Abfragename>.xml**.

Weitere Informationen finden Sie unter "Schema der Zuordnungsdatei" auf Seite 199.

Diese Aufgabe umfasst folgende Schritte:

- "Erstellen der Zuordnungsdatei" oben
- "Zuordnen von CIs" oben
- "Zuordnen von Links " auf der nächsten Seite

1. ## Erstellen der Zuordnungsdatei

   Die Zuordnungsdatei ist wie folgt aufgebaut:

```
<?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. ## Zuordnen von CIs

   Es gibt zwei Möglichkeiten zur Zuordnung von CMDB-CI-Typen:

- Zuordnen eines CI-Typ, sodass sämtliche CIs des Typs und alle geerbten Typen auf die gleiche Weise zugeordnet werden:

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

- Zuordnen eines CI-Typs, sodass nur CIs des Typs verarbeitet werden. CIs von geerbten Typen werden nicht verarbeitet, sofern ihr Typ nicht ebenfalls zugeordnet wird (auf eine der zwei folgenden Arten):

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

Ein indirekt zugeordneter CI-Typ (eines seiner übergeordneten Elemente wurde unter Verwendung von **source_ci_type_tree** zugeordnet) kann auch seine übergeordnete Karte überschreiben, indem er sie in sein eigenes **source_ci_type_tree**- oder **source_ci_type**-Element aufnimmt.

Es wird empfohlen, nach Möglichkeit **source_ci_type_tree** zu verwenden. Andernfalls werden die resultierenden CIs eines CI-Typs, die nicht in den Zuordnungsdateien erscheinen, nicht an das Jython-Skript übertragen.

3. ## Zuordnen von Links

Es gibt zwei Möglichkeiten zum Zuordnen von Links:

- Zuordnen eines Links, der auch alle von ihm erbenden Link-Typen zuordnet:

```
<source_link_type_tree name="dependency" target_link_
type="dependency" mode="update_else_insert" source_ci_type_
end1="webservice" source_ci_type_end2="sap_gateway">
    <target_ci_type_end1 name="webservice" >
    <target_ci_type_end2 name="sap_gateway" >
        <target_attribute name="name" datatype="STRING">
```

```
            <map type="direct" source_attribute="name" >
        </target_attribute>
</source_link_type_tree>
```

- Zuordnen eines Links, der nur den jeweiligen Link-Typ zuordnet und nicht die von ihm erbenden Link-Typen:

```
<link source_link_type="dependency" target_link_type="dependency"
mode="update_else_insert" source_ci_type_end1="webservice"
source_ci_type_end2="sap_gateway">
        <target_ci_type_end1 name="webservice" >
        <target_ci_type_end2 name="sap_gateway" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
    </target_attribute>
</link>
```

Schreiben von Jython-Skripts

Das Zuordnungsskript ist ein reguläres Jython-Skript und sollte den Regeln für Jython-Skripts folgen. Weitere Informationen finden Sie unter "Entwickeln von Jython-Adaptern" auf Seite 41.

Das Skript sollte die Funktion **DiscoveryMain** enthalten, die bei erfolgreicher Ausführung entweder eine leere **OSHVResult**-Instanz oder eine **DataPushResults**-Instanz zurückgeben kann.

Um Fehler zu melden, sollte das Skript eine Ausnahme auslösen, z. B.:

```
raise Exception('Failed to insert to remote UCMDB using
TopologyUpdateService. See log of the remote UCMDB')
```

In der DiscoveryMain-Funktion können die Datenelemente, die per Push an die externe Applikation übertragen oder von ihr gelöscht werden sollen, wie folgt abgerufen werden:

```
# get add/update/delete result objects (in XML format) from the
Framework
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
```

The client object to the external application can be obtained as follows:

```
oracleClient = Framework.createClient()
```

Das Client-Objekt für die externe Applikation kann wie folgt abgerufen werden:

```
oracleClient = Framework.createClient()
```

Dieses Client-Objekt verwendet automatisch die Anmeldeinformationen-ID, den Hostnamen und die Portnummer, die vom Adapter über das Framework übertragen wurden.

Wenn Sie die Verbindungsparameter verwenden müssen, die Sie für den Adapter definiert haben (weitere Informationen hierzu finden Sie in Schritt "Bearbeiten Sie die Datei discoveryPatterns\push_adapter.xml." unter "Erstellen eines Adapter-Package " auf Seite 198), verwenden Sie den folgenden Code:

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

Beispiel:

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

Dieser Abschnitt beinhaltet auch folgende Themen:

- "Arbeiten mit den Ergebnissen der Zuordnung" oben
- "Behandeln der Testverbindung im Skript" auf Seite 196

Arbeiten mit den Ergebnissen der Zuordnung

Der generische Push-Adapter erstellt XML-Zeichenketten, die die Daten beschreiben, die hinzugefügt, aktualisiert oder vom Zielsystem gelöscht werden sollen. Das Jython-Skript muss diese XML analysieren und führt dann beim Ziel den entsprechenden Hinzufügungs-, Aktualisierung- oder Löschvorgang durch.

In der XML von Hinzufügungsvorgängen, die das Jython-Skript empfängt, entspricht das `mamId`-Attribut für die Objekte und Links immer der UCMDB-ID des ursprünglichen Objekts oder Links, bevor dessen Typ, Attribut oder andere Informationen in das Schema des Remote-Systems geändert wurden.

In der XML von Aktualisierungs- oder Löschvorgängen enthält das `mamId`-Attribut jedes Objekts oder Links die Zeichenfolgendarstellung der `ExternalId`-ID, die von der vorherigen Synchronisierung vom Jython-Skript zurückgegeben wurde.

In der XML enthält das `id`-Attribut eines CI das `cmdbId` als eine externe ID oder die `ExternalId` dieses CI, wenn das CI beim Senden an das Skript eine `ExternalId` aufwies. Die `end1Id`- und `end2Id`-Felder des Links enthalten für jedes Ende der Links die `cmdbId` als externe ID oder die `ExternalId` des Ende dieses Links, wenn das CI am Ende des Links beim Senden an das Skript eine `ExternalId` aufwies.

Beim Verarbeiten der CIs im Jython-Skript ist der Rückgabewert des Skripts eine Zuordnung zwischen der CMDB-ID des CI und der angegebenen ID (die für jedes CI im Skript angegebene ID). Wenn ein CI zum ersten Mal übertragen wird, handelt es sich bei der ID, die für dieses CI in der XML vorhanden ist, um die CMDB-ID. Wenn ein CI nicht zum ersten Mal übertragen wird, ist die ID des CI dieselbe, die dem CI im Skript bei der ersten Übertragung zugewiesen wurde.

Die ID wird vom CI-XML-Skript wie folgt abgerufen:

1. Rufen Sie vom CI-Element in der XML die ID aus dem ID-Attribut ab. Beispiel: `id = objectElement.getAttributeValue('id').`
2. Stellen Sie nach dem Abrufen der ID auf der XML die ID aus dem Attribut (Zeichenfolge) wieder her. Beispiel: `objectId = CmdbObjectID.Factory.restoreObjectID(id).`
3. Überprüfen Sie, ob es sich bei der im vorherigen Schritt abgerufenen `objectId` um die CMDB-ID handelt. Überprüfen Sie hierzu, ob die `objectId` die neue ID aufweist, die das Skript ihr zuweist. Ist dies der Fall, handelt es sich bei der zurückgegebenen ID nicht um die CMDB-ID. Beispiel:
   `newId = objectId.getPropertyValue(<der Name des vom Skript zugewiesenen ID-Attributs>).`

Wenn `newId` null ist, handelt es sich bei der in der XML zurückgegebenen ID um eine CMDB-ID.

4. Wenn es sich bei der ID um eine CMDB-ID handelt (also `newId` null ist), führen Sie folgenden Schritte aus (wenn es sich bei der ID um keine CMDB-ID handelt, fahren Sie mit Schritt 5 fort):

   a. Erstellen Sie eine Eigenschaften für dieses CI, die die neue ID enthält. Beispiel:
   ```
   propArray =[TypesFactory.createProperty('<der Name des vom Skript zugewiesenen ID-Attributs>', '<neue ID>')].
   ```

   b. Erstellen Sie eine `externaId` für dieses CI. Beispiel:
   ```
   cmdbId = extI.getPropertyValue('internal_id')
   className = extI.getType()
   externalId = ExternalIdFactory.createExternalCiId(className, propArray)
   ```

   c. Ordnen Sie die CMDB-ID der neu erstellten `externalId` zu (und geben Sie diese Zuordnung im nächsten Schritt zurück an den Adapter). Beispiel: `objectMappings.put (cmdbId, externalId)`

   d. Wenn alle CIs und Links zugeordnet sind:
   ```
   updateResult = DataPushResultsFactory.createDataPushResults (objectMappings, linkMappings);
   return updateResult
   ```

5. Wenn es sich bei der ID um die neue ID handelt (also `newId` nicht null ist), ist die `externalId` die `newId`.

Beispiel für das XML-Ergebnis

```
<root>
  <data>
    <objects>
      <Object mode="update_else_insert" name="UCMDB_UNIX"
operation="add" mamId="0c82f591bc3a584121b0b85efd90b174"
id="HiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A">
        <field name="NAME" key="false" datatype="char"
length="255">UNIX5</field>
        <field name="DATA_NOTE" key="false" datatype="char"
length="255"></field>
      </Object>
    </objects>
    <links>
        <link targetRelationshipClass="TALK" targetParent="unix"
targetChild="unix" operation="add" mode="update_else_insert"
mamId="265e985c6ec51a8543f461b30fa58f81"
id="end1id%5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_
```

```
id%3DSTRING%3D41372a1cbcaba27b214b84a2ec9eb535%0A%5D%0Aend2id%
5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A%5D%0AHiddenRmi
DataSource%0Atalk%0A1%0Ainternal_
id%3DSTRING%3D265e985c6ec51a8543f461b30fa58f81%0A">

        <field
name="DiscoveryID1">41372a1cbcaba27b214b84a2ec9eb535</field>

        <field
name="DiscoveryID2">0c82f591bc3a584121b0b85efd90b174</field>

        <field name="end1Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D41372a1cbcaba27b214b84a2ec9eb535%0A</field>

        <field name="end2Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A</field>

        <field name="NAME" key="false" datatype="char"
length="255">TALK4</field>

        <field name="DATA_NOTE" key="false" datatype="char"
length="255"></field>

        </link>

    </links>

  </data>

</root>
```

Hinweis: Bei `datatype="BYTE"` ist der Wert des zurückgegebenen Ergebnisses eine folgendermaßen generierte **Zeichenfolge**: `new String([the byte array attribute])`. Das `byte[] object` kann folgendermaßen rekonstruiert werden: `<die empfangene Zeichenfolge>.getBytes()`. Möglicherweise weisen der Server und die Probe abweichende Standardgebietsschemata auf. In diesem Fall erfolgt die Rekonstruktion entsprechend dem Standardgebietsschema des Servers.

Behandeln der Testverbindung im Skript

Ein Jython-Skript kann aufgerufen werden, um die Verbindung mit einer externen Applikation zu testen. In diesem Fall hat das Zielattribut `testConnection` die Einstellung `true`. Dieses Attribut kann wie folgt vom Framework abgerufen werden:

```
testConnection = Framework.getTriggerCIData('testConnection')
```

Bei der Ausführung im Testverbindungsmodus sollte ein Skript eine Ausnahme auslösen, wenn keine Verbindung zur externen Applikation hergestellt werden kann. Wenn die Verbindung erfolgreich aufgebaut wurde, sollte die **DiscoveryMain**-Funktion eine leere **OSHVResult**-Instanz zurückgeben.

Unterstützung der differenziellen Synchronisierung

Wichtig: Wenn Sie die differenzielle Synchronisierung bei einem vorhandenen Adapter implementieren, der in der Version 9.00 oder 9.01 erstellt wurde, müssen Sie die Datei **push-adapter.zip** der Version 9.02 oder höher verwenden, um Ihr Adapter-Package neu zu erstellen. Weitere Informationen finden Sie unter "Erstellen eines Adapter-Package " auf der nächsten Seite.

In dieser Aufgabe richten Sie den Push-Adapter für die Durchführung der differenziellen Synchronisierung ein. Weitere Informationen finden Sie unter "Differenzielle Synchronisierung " auf Seite 190.

Das Jython-Skript gibt das Objekt **DataPushResults** zurück, das zwei Java-Zuordnungen enthält - eine für Objekt-ID-Zuordnungen (Schlüssel und Werte sind Objekte vom Typ **ExternalCiId**) und eine für Link-IDs (Schlüssel und Werte sind Objekte vom Typ **ExternalCiId**).

- Fügen Sie die folgenden **from**-Anweisungen zu Ihrem Jython-Skript hinzu:

```
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
```

- Verwenden Sie die Werk-Klasse **DataPushResultsFactory**, um das **DataPushResults**-Objekt von der **DiscoveryMain**-Funktion abzurufen.

```
# Create the UpdateResult object

updateResult = DataPushResultsFactory.createDataPushResults
(objectMappings, linkMappings);
```

- Verwenden Sie die folgenden Befehle, um Java-Zuordnungen für das **DataPushResults**-Objekt zu erstellen:

```
# Prepare the maps to store the mappings if IDs

objectMappings = HashMap()

linkMappings = HashMap()
```

- Verwenden Sie die Klasse **ExternalIdFactory**, um die folgenden ExternalId-IDs zu erstellen:
  - ExternalId für Objekte oder Links, die von einer CMDB stammen (z. B. alle CIs in einem Hinzufügungsvorgang stammen von der CMDB):

```
externaCIlId = ExternalIdFactory.createExternalCmdbCiId(ciType,
ciIDAsString)
```

```
externalRelationId =
ExternalIdFactory.createExternalCmdbRelationId(linkType,
end1ExternalCIId, end2ExternalCIId, linkIDAsString)
```

- ExternalId für Objekte oder Links, die nicht von einer CMDB stammen (normalerweise enthält jeder Aktualisierungs- und Löschvorgang derartige Objekte):

```
myIDField = TypesFactory.createProperty("systemID", "1")

myExternalId = ExternalIdFactory.createExternalCiId(type,
myIDField)
```

> **Hinweis:** Wenn das Jython-Skript vorhandene Informationen aktualisiert hat und demzufolge die ID des Objekts (oder Links) geändert wurde, müssen Sie eine Zuordnung zwischen der vorherigen externen ID und der neuen ID zurückgeben.

- Verwenden Sie die Methoden **restoreCmdbCiIDString** oder **restoreCmdbRelationIDString** von der Klasse **ExternalIdFactory**, um die UCMDB ID-Zeichenkette von einer externen ID eines aus UCMBD stammenden Objekts oder Links abzurufen.
- Verwenden Sie die Methoden **restoreExternalCiId** und **restoreExternalRelationId** von der Klasse **ExternalIdUtil**, um das Objekt **ExternalId** vom **mamId**-Attributwert der XML der Aktualisierungs- oder Löschvorgänge abzurufen.

> **Hinweis: ExternalId**-Objekte sind tatsächlich Eigenschaften-Arrays. Dies bedeutet, dass Sie in einem **ExternalId**-Objekt sämtliche Informationen speichern können, die Sie zur Identifizierung der Daten auf dem Remote-System benötigen.

Erstellen eines Adapter-Package

1. Extrahieren Sie den Inhalt von **C:\hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip** in einen temporären Ordner. Im Adapter-Package enthält die Datei **sql_queries**, die sich unter **adapterCode > PushAdapter > sqlTablesCreation** befindet, die Abfragen, die erforderlich sind, um in einem neuen Schema in Oracle Tabellen zum Testen des Adapters zu erstellen. Die Tabellen entsprechen der Datei **adapterCode\<Adapter-ID>\mappings\mappings.xml**.

   > **Hinweis:** Die Datei **sql_queries** wird für den Adapter nicht benötigt. Sie dient lediglich als Beispiel.

2. Bearbeiten Sie die Datei **discoveryPatterns\push_adapter.xml**.

   a. Modifizieren Sie das <pattern>-Tag mit einer neuen ID und einem neuen Anzeige-Label. Ersetzen Sie

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

   durch den folgenden Code:

```
<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. Aktualisieren Sie die Parameterliste, sodass sie die erforderlichen Verbindungsattribute widerspiegelt. Entfernen Sie das Attribut **probeName** nicht.

3. Benennen Sie den Ordner **adapterCode\PushAdapter** in die im vorherigen Schritt verwendete Adapter-ID um (z. B. **adapterCode\MyPushAdapter**).

4. Die Datei **discoveryScript** enthält das Skript **script pushScript.py**, das die CIs und Links in eine externe Oracle-Datenbank einfügt. Ersetzen Sie das Skript **discoveryScripts\pushScript.py** durch das Skript, das Sie geschrieben haben (weitere Informationen hierzu finden Sie unter "Schreiben von Jython-Skripts " auf Seite 193). Wenn Sie das Skript umbenennen, sollte die Eigenschaft `jythonScript.name` in **adapterCode\<Adapter_ID>\push.properties** entsprechend aktualisiert werden.

5. Die Datei **adapterCode\<Adapter-ID>\mappings\mappings.xml**, die sich unter **adapterCode\<Adapter-ID>\mappings** befindet, ist eine Beispielzuordnungsdatei mit folgenden Zuordnungen:

   - CI-Typ **Node** mit allen davon geerbten CI-Typen
   - CI-Typ **UNIX** ohne die davon geerbten CI-Typen
   - Abhängigkeitsverknüpfung mit allen davon geerbten Link-Typen
   - Link-Typ **Talk** ohne die davon geerbten Link-Typen

   Dieses Zuordnungsbeispiel entspricht dem Beispiel der Tabellen, die in ORACLE in der Datei **sql_queries** erstellt wurden (siehe Schritt 1).

   Ersetzen Sie die Datei **adapterCode\<Adapter_ID>\mappings\mappings.xml** durch die Zuordnungsdateien, die Sie vorbereitet haben (weitere Informationen hierzu finden Sie unter "Vorbereiten der Zuordnungsdateien" auf Seite 191).

   Wenn Sie für jede TQL-Methode eine Zuordnungsdatei verwenden möchten, ordnen Sie jeder XML-Datei den Namen der entsprechenden TQL, gefolgt von dem Suffix **.xml** zu. In diesem Fall wird standardmäßig die Datei **mappings.xml** verwendet, wenn für den aktuellen TQL-Namen keine spezifische Zuordnungsdatei gefunden wird. Der Name der Standardzuordnungsdatei kann durch Ändern der Eigenschaft `mappingFile.default` in **adapterCode\<Adapter_ID>\push.properties** geändert werden.

Schema der Zuordnungsdatei

Elementname und -pfad	Beschreibung	Attribute
Integration	Definiert die Zuordnungsinhalte der Datei. Abgesehen von der Anfangszeile und	

Elementname und -pfad	Beschreibung	Attribute
	eventuellen Kommentaren muss dies der äußerste Block in der Datei sein.	
info (integration)	Definiert die Informationen über die zu integrierenden Daten-Repositorys.	
source (integration > info)	Definiert die Informationen über das Quelldaten-Repository.	1. **Name:** type **Beschreibung:** Name des Quelldaten-Repository. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** versions **Beschreibung:** Version(en) der Quelldaten-Repositorys **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 3. **Name:** vendor **Beschreibung:** Anbieter des Quelldaten-Repository **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
target (integration > info)	Definiert die Informationen über das Zieldaten-Repository.	1. **Name:** type **Beschreibung:** Name des Quelldaten-Repository. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** versions **Beschreibung:** Version(en) des Quelldaten-Repository. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 3. **Name:** vendor **Beschreibung:** Anbieter des Quelldaten-Repository **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
targetcis (integration)	Container-Element für alle CIT-Zuordnungen.	
source_ci_type_tree	Definiert einen Quell-	1. **Name:** name

Elementname und -pfad	Beschreibung	Attribute
(integration > targetcis)	CIT und alle davon geerbten CI-Typen.	**Beschreibung:** Name des Quell-CIT. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** mode **Beschreibung:** Der für den aktuellen CI-Typ erforderliche Aktualisierungstyp. **Erforderlich?:** Erforderlich **Typ:** Eine der folgenden Zeichenketten: a. **insert:** Verwenden Sie diese Zeichenkette nur, wenn das CI noch nicht vorhanden ist. b. **update:** Verwenden Sie diese Zeichenkette nur, wenn das CI bekanntermaßen vorhanden ist. c. **update_else_insert:** Wenn das CI vorhanden ist, wird es aktualisiert. Andernfalls wird ein neues CI erstellt. d. **ignore:** Verwenden Sie diese Zeichenkette, wenn mit dem CI-Typ nichts gemacht werden soll.
source_ci_type (integration > targetcis)	Definiert einen Quell-CIT ohne die davon geerbten CI-Typen.	1. **Name:** name **Beschreibung:** Name des Quell-CIT. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** mode **Beschreibung:** Der für den aktuellen CI-Typ erforderliche Aktualisierungstyp. **Erforderlich?:** Erforderlich **Typ:** Eine der folgenden Zeichenketten: a. **insert:** Verwenden Sie diese Zeichenkette nur, wenn das CI noch nicht vorhanden ist. b. **update:** Verwenden Sie diese Zeichenkette nur, wenn das CI bekanntermaßen vorhanden ist. c. **update_else_insert:** Wenn das CI vorhanden ist, wird es aktualisiert. Andernfalls wird ein neues CI erstellt. d. **ignore:** Verwenden Sie diese Zeichenkette, wenn mit dem CI-Typ nichts gemacht werden soll.
target_ci_type (integration > targetcis >	Definiert einen Ziel-CIT.	1. **Name:** name **Beschreibung:** Name des Ziel-CI-Typs. **Erforderlich?:** Erforderlich

Elementname und -pfad	Beschreibung	Attribute
source_ci_type Oder integration > targetcis > source_ci_type_ tree)		**Typ:** Zeichenkette 2. **Name:** schema **Beschreibung:** Der Name des Schemas, das zum Speichern dieses CI-Typs am Ziel verwendet wird. **Erforderlich?:** Nicht erforderlich **Typ:** Zeichenkette 3. **Name:** namespace **Beschreibung:** Gibt den Namespace dieses CI-Typs beim Ziel an. **Erforderlich?:** Nicht erforderlich **Typ:** Zeichenkette
targetprimarykey (integration > targetcis > source_ci_type) Oder (integration > targetcis > source_ci_type_tree Oder (integration > targetrelations > link) Oder (integration > targetrelations > source_link_type_ tree)	Identifiziert die Primärschlüsselattribute des Ziel-CITs.	
pkey (integration > targetcis> source_ci_ type > targetprimarykey Oder integration > targetcis > source_ci_type_tree > targetprimarykey Oder (integration >	Identifiziert ein Primärschlüsselattribut. Nur erforderlich, wenn **mode** = **update** oder **insert_else_update**.	

Elementname und -pfad	Beschreibung	Attribute
targetrelations > link > targetprimarykey) Oder integration > targetrelations > source_link_type_tree > targetprimarykey)		
target_attribute (integration > targetcis> source_ci_type Oder integration > targetcis > source_ci_type_tree Oder integration > targetrelations > link Oder integration > targetrelations > source_link_type_tree)	Definiert das Attribut des Ziel-CIT.	1. **Name:** name **Beschreibung:** Name des Attributs des Ziel-CIT. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** datatype **Beschreibung:** Datentyp des Attributs des Ziel-CIT. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 3. **Name:** length **Beschreibung:** Für Daten vom Typ "Zeichenkette/Zeichen" die Ganzzahlgröße des Zielattributs. **Erforderlich?:**Nicht erforderlich **Typ.** Ganzzahl 4. **Name.** option **Beschreibung.** Die auf den Wert anzuwendende Konvertierungsfunktion. **Verwendung.** False **Typ.** Eine der folgenden Zeichenketten: a. **uppercase** – In Großbuchstaben konvertieren b. **lowercase** – In Kleinbuchstaben konvertieren Wenn dieses Attribut leer ist, findet keine Konvertierung statt.
map (integration > targetcis > source_ci_type > target_attribute Oder	Gibt an, wie der Attributwert des Quell-CIT abgerufen werden soll.	1. **Name.** type **Beschreibung.** Der Typ der Zuordnung zwischen den Quell- und Zielwerten. **Verwendung.** Erforderlich **Typ.** Eine der folgenden Zeichenketten: a. **direct** – Gibt eine 1:1-Zuordnung vom Wert des Quellattributs zum Wert des

Elementname und -pfad	Beschreibung	Attribute
integration > targetcis > source_ci_type_tree > target_attribute) Oder (integration > targetrelations > link > target_attribute Oder integration > targetrelations > source_link_type_tree > target_attribute)		Zielattributs an. b. **compoundstring** – Unterelemente werden in einer einzelnen Zeichenkette zusammengefasst und der Wert des Zielattributs wird gesetzt. c. **childattr** – Unterelemente sind ein oder mehrere Attribute des untergeordneten CIT. Untergeordnete CITs sind definiert als CITs mit der Beziehung **composition** oder **containment**. d. **constant** – Statische Zeichenkette 2. **Name.** value **Beschreibung.** Konstante Zeichenkette für type=**constant** **Verwendung.** Nur erforderlich, wenn **type=constant** **Typ.** Zeichenkette 3. **Name.** attr **Beschreibung.** Name des Quellattributs für **type=direct** **Verwendung.** Nur erforderlich, wenn **type=direct** **Typ.** Zeichenkette
aggregation (integration > targetcis > source_ci_type > target_attribute > map Oder integration > targetcis > source_ci_type_tree > target_attribute > map Oder (integration > targetrelations > link > target_attribute > map Oder	Gibt an, wie die Attributwerte eines dem Quell-CI untergeordneten CI für die Zuordnung zum Ziel-CI-Attribut zu einem einzelnen Wert zusammengefasst werden sollen. Optional.	**Name:** type **Beschreibung.** Der Typ der Aggregationsfunktion. **Erforderlich?:** Erforderlich **Typ.** Eine der folgenden Zeichenketten: • **csv** – Verkettet alle eingeschlossenen Werte in einer kommagetrennten Liste (numerisch oder Zeichenkette/Zeichen). • **count** – Gibt eine numerische Anzahl alle eingeschlossenen Werte zurück. • **sum** – Gibt eine numerische Anzahl alle eingeschlossenen Werte zurück. • **average** – Gibt einen numerischen Durchschnitt aller eingeschlossenen Werte zurück. • **min** – Gibt den niedrigsten eingeschlossenen Wert (numerisch/Zeichen) zurück.

Elementname und -pfad	Beschreibung	Attribute
integration > targetrelations > source_link_type_tree > target_attribute > map) Nur gültig für Zuordnungen des Typs **childattr**		• **max** – Gibt den höchsten eingeschlossenen Wert (numerisch/Zeichen) zurück.
source_child_ci_type (integration > targetcis> source_ci_ type > target_attribute > map Oder integration > targetcis > source_ci_type_tree > target_attribute > map Oder (integration > targetrelations > link > target_attribute > map Oder integration > targetrelations > source_link_type_tree > target_attribute > map) Nur gültig für Zuordnungen des Typs **childattr**.	Gibt an, von welchem verbundenen CI das untergeordnete Attribut genommen wird.	1. **Name.** name **Beschreibung.** Der Typ des untergeordneten CI **Verwendung.** Erforderlich **Typ.** Zeichenkette 2. **Name.** source_attribute **Beschreibung.** Das Attribut des untergeordneten CI, das zugeordnet wird. **Verwendung.** Nur erforderlich, wenn der Aggregationstyp **childAttr** (in demselben Pfad) nicht **=count** ist. **Typ.** Zeichenkette
validation (integration > targetcis > source_ci_type > target_attribute > map Oder	Erlaubt die Ausschlussfilterung der dem Quell-CI untergeordneten CIs auf Basis von Attributwerten. Wird mit dem Unterelement aggregation verwendet,	1. **Name.** minlength **Beschreibung.** Schließt Zeichenketten aus, die kürzer als der angegebene Wert sind. **Erforderlich?:** Nicht erforderlich **Typ.** Ganzzahl 2. **Name.** maxlength **Beschreibung.** Schließt Zeichenketten

Elementname und -pfad	Beschreibung	Attribute
integration > targetcis > source_ci_type_tree > target_attribute > map Oder (integration > targetrelations > link > target_attribute > map Oder integration > targetrelations > source_link_type_tree > target_attribute > map) Nur gültig für Zuordnungen des Typs **childatt**	um Granularität darüber zu erreichen, welche untergeordneten Attribute dem Attributwert des Ziel-CIT zugeordnet sind. Optional.	aus, die länger als der angegebene Wert sind. **Erforderlich?:** Nicht erforderlich **Typ.** Ganzzahl 3. **Name.** minvalue **Beschreibung.** Schließt Zahlen aus, die kleiner als der angegebene Wert sind. **Erforderlich?:** Nicht erforderlich **Typ.** Numerisch 4. **Name.** maxvalue **Beschreibung.** Schließt Zahlen aus, die größer als der angegebene Wert sind. **Erforderlich?:** Nicht erforderlich **Typ.** Numerisch
targetrelations (integration)	Container-Element für alle Beziehungszuordnunge-n. Optional.	
source_link_type_tree (integration > targetrelations)	Ordnet einen Quellbeziehungstyp ohne die davon geerbten Typen zu einer Zielbeziehung zu. Nur obligatorisch, wenn **targetrelation** vorhanden ist.	1. **Name:** name **Beschreibung.** Name der Quellbeziehung. **Verwendung?:** Erforderlich **Typ.** Zeichenkette 2. **Name:** target_link_type **Beschreibung.** Name der Zielbeziehung **Erforderlich?:** Erforderlich **Typ.** Zeichenkette 3. **Name:** nameSpace **Beschreibung:** Der Namespace für den Link, der beim Ziel erstellt wird. **Erforderlich?:** Nicht erforderlich **Typ:** Zeichenkette 4. **Name:** mode **Beschreibung:** Der für den aktuellen Link erforderliche Aktualisierungstyp. **Erforderlich?:** Erforderlich

Elementname und -pfad	Beschreibung	Attribute
		Typ: Eine der folgenden Zeichenketten: ▪ **insert** – Verwenden Sie diese Zeichenkette nur, wenn das CI noch nicht vorhanden ist. ▪ **update** – Verwenden Sie diese Zeichenkette nur, wenn das CI bekanntermaßen vorhanden ist. ▪ **update_else_insert** – Wenn das CI vorhanden ist, wird es aktualisiert. Andernfalls wird ein neues CI erstellt. ▪ **ignore** – Verwenden Sie diese Zeichenkette, wenn mit dem CI-Typ nichts gemacht werden soll. 5. **Name:** source_ci_type_end1 **Beschreibung:** CI-Typ von Ende 1 der Quellbeziehung. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 6. **Name:** source_ci_type_end2 **Beschreibung:** CI-Typ von Ende 2 der Quellbeziehung. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
link (integration > targetrelations)	Ordnet eine Quellbeziehung zu einer Zielbeziehung zu. Nur obligatorisch, wenn **targetrelation** vorhanden ist.	1. **Name:** source_link_type **Beschreibung:** Name der Quellbeziehung. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** target_link_type **Beschreibung:** Name der Zielbeziehung. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 3. **Name:** nameSpace **Beschreibung:** Der Namespace für den Link, der beim Ziel erstellt wird. **Erforderlich?:** Nicht erforderlich **Typ:** Zeichenkette 4. **Name:** mode **Beschreibung:** Der für den aktuellen Link erforderliche Aktualisierungstyp. **Erforderlich?:** Erforderlich

Elementname und -pfad	Beschreibung	Attribute
		Typ: Eine der folgenden Zeichenketten: ▪ **insert** – Verwenden Sie diese Zeichenkette nur, wenn das CI noch nicht vorhanden ist. ▪ **update** – Verwenden Sie diese Zeichenkette nur, wenn das CI bekanntermaßen vorhanden ist. ▪ **update_else_insert** – Wenn das CI vorhanden ist, wird es aktualisiert. Andernfalls wird ein neues CI erstellt. ▪ **ignore** – Verwenden Sie diese Zeichenkette, wenn mit dem CI-Typ nichts gemacht werden soll. 5. **Name:** source_ci_type_end1 **Beschreibung:** CI-Typ von Ende 1 der Quellbeziehung **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 6. **Name:** source_ci_type_end2 **Beschreibung:** CI-Typ von Ende 2 der Quellbeziehung **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
target_ci_type_end1 (integration > targetrelations > link Oder integration > targetrelations > source_link_type_ tree)	CI-Typ von Ende 1 der Zielbeziehung.	1. **Name:** name **Beschreibung:**Name des CI-Typs von Ende 1 der Zielbeziehung. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** superclass **Beschreibung:** Name der Superklasse des CI-Typs von Ende 1. **Erforderlich?:** Nicht erforderlich **Typ:** Zeichenkette
target_ci_type_end2 (integration > targetrelations > link Oder integration > targetrelations > source_link_type_	CI-Typ von Ende 2 der Zielbeziehung.	1. **Name:** name **Beschreibung:** Name des CI-Typs von Ende 2 der Zielbeziehung. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** superclass **Beschreibung:** Name der Superklasse des CI-Typs von Ende 2.

Elementname und -pfad	Beschreibung	Attribute
tree)		**Erforderlich?:** Nicht erforderlich **Typ:** Zeichenkette

Schema der Zuordnungsergebnisse

Elementname und -pfad	Beschreibung	Attribute
root	Der Stamm des Ergebnisdokuments.	
data (root)	Der Stamm der Daten selbst.	
objects (root > data)	Das Stammelement für die zu aktualisierenden Objekte.	
Object (root > data > objects)	Beschreibt den Aktualisierungsvorgang für ein einzelnes Objekt und alle seine Attribute.	1. **Name:** name **Beschreibung:** Der Name des CI-Typs. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** mode **Beschreibung:** Der für den aktuellen CI-Typ erforderliche Aktualisierungstyp. **Erforderlich?:** Erforderlich **Typ:** Eine der folgenden Zeichenketten: a. **insert** – Verwenden Sie diese Zeichenkette nur, wenn das CI noch nicht vorhanden ist. b. **update** – Verwenden Sie diese Zeichenkette nur, wenn das CI bekanntermaßen vorhanden ist. c. **update_else_insert** – Wenn das CI vorhanden ist, wird es aktualisiert. Andernfalls wird ein neues CI erstellt. d. **ignore** – Verwenden Sie diese Zeichenkette, wenn mit dem CI-Typ nichts gemacht werden soll. 3. **Name:** operation **Beschreibung:** Die mit diesem CI durchzuführende Operation. **Verwendung:** Erforderlich **Typ:** Eine der folgenden Zeichenketten:

Elementname und -pfad	Beschreibung	Attribute
		a. **add** – Das CI soll hinzugefügt werden. b. **update** – Das CI soll aktualisiert werden. c. **delete** – Das CI soll gelöscht werden. Wenn kein Wert angegeben ist, wird der Standardwert **add** verwendet. 4. **Name:** mamId **Beschreibung:** Die ID des Objekts bei der Quell-CMDB. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette
field (root > data > objects> Object Oder root > data > links > link)	Beschreibt den Wert eines einzelnen Felds für ein Objekt. Der Text des Felds ist der neue Wert im Feld. Wenn das Feld einen Link enthält, ist der Wert die ID eines Link-Endes. Die ID jedes Endes erscheint als Objekt (unter <objects>).	1. **Name:** name **Beschreibung:** Der Name des Felds. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** key **Beschreibung:** Gibt an, ob dieses Feld ein Schlüssel für das Objekt ist. **Erforderlich?:** Erforderlich **Typ:** Boolesch 3. **Name:** datatype **Beschreibung:** Der Typ des Felds. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 4. **Name:** length **Beschreibung:** Für Daten vom Typ "Zeichenkette/Zeichen" ist dies die Ganzzahlgröße des Zielattributs. **Erforderlich?:** Nicht erforderlich **Typ:** Ganzzahl
links (root > data)	Das Stammelement für die zu aktualisierenden Links.	1. **Name:** targetRelationshipClass **Beschreibung:** Der Name der Beziehung (des Links) im Zielsystem. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 2. **Name:** targetParent **Beschreibung:** Der Typ des ersten Endes des Links (übergeordnetes Element). **Erforderlich?:** Erforderlich **Typ:** Zeichenkette 3. **Name:** targetChild

Elementname und -pfad	Beschreibung	Attribute
		Beschreibung: Der Typ des zweiten Endes des Links (untergeordnetes Element). **Verwendung:** Erforderlich **Typ:** Zeichenkette 4. **Name:** mode **Beschreibung:** Der für den aktuellen CI-Typ erforderliche Aktualisierungstyp. **Erforderlich?:** Erforderlich **Typ:** Eine der folgenden Zeichenketten: a. **insert** – Verwenden Sie diese Zeichenkette nur, wenn das CI noch nicht vorhanden ist. b. **update** – Verwenden Sie diese Zeichenkette nur, wenn das CI bekanntermaßen vorhanden ist. c. **update_else_insert** – Wenn das CI vorhanden ist, wird es aktualisiert. Andernfalls wird ein neues CI erstellt. d. **ignore** – Verwenden Sie diese Zeichenkette, wenn mit dem CI-Typ nichts gemacht werden soll. 5. **Name:** operation **Beschreibung:** Die mit diesem CI durchzuführende Operation. **Erforderlich?:** Erforderlich **Typ:** Eine der folgenden Zeichenketten: a. **add** – Das CI soll hinzugefügt werden. b. **update** – Das CI soll aktualisiert werden. c. **delete** – Das CI soll gelöscht werden. Wenn kein Wert angegeben ist, wird der Standardwert **add** verwendet. 6. **Name:** mamId **Beschreibung:** Die ID des Objekts bei der Quell-CMDB. **Erforderlich?:** Erforderlich **Typ:** Zeichenkette

Kapitel 7

Entwickeln von erweiterten generischen Push-Adaptern

Dieses Kapitel umfasst folgende Themen:

Entwickeln von erweiterten Push-Adaptern – Übersicht

Ein erweiterter Push-Adapter ist eine Datenstruktur, die das TQL-Abfrageergebnis darstellt. Jeder Adapter, der über den erweiterten Push-Adapter erstellt wird, verarbeitet diese Datenstruktur und überträgt sie an das erforderliche Ziel.

Die Datenstruktur wird als **ResultTreeNode (RTN)** bezeichnet. Der RTN wird entsprechend der Zuordnungsdatei des Adapters und der Ergebnisse der TQL-Abfrage erstellt. Die für den erweiterten Push-Adapter verwendeten Abfragen müssen stammbasiert sein, d. h. die Abfrage muss einen Abfrageknoten mit dem Elementnamen **root** oder mindestens ein Beziehungselement mit dem Präfix **root** enthalten. Diese CI bzw. diese Beziehung fungiert als Stammelement der Abfrage. Weitere Informationen finden Sie unter "Datenpush" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

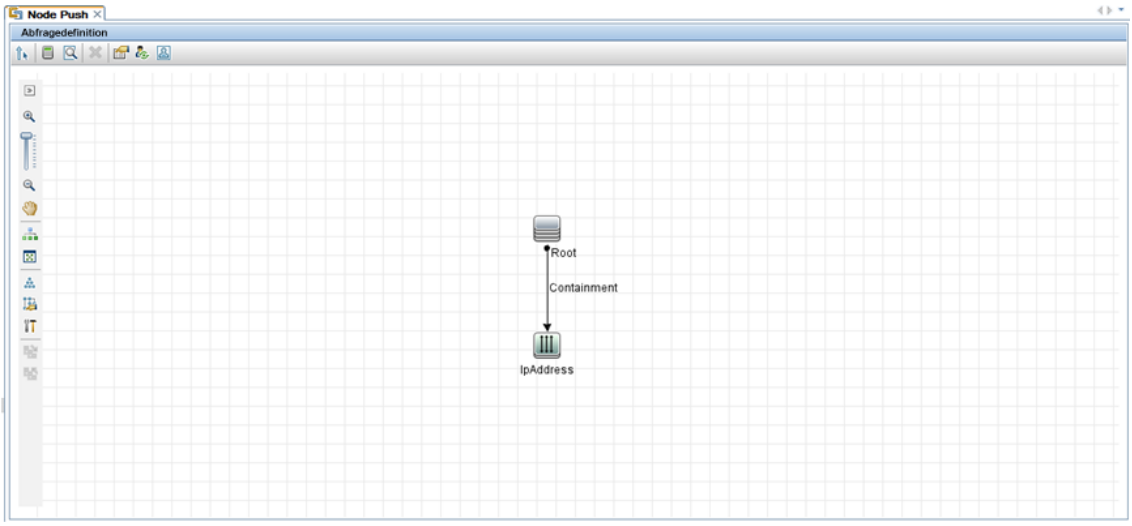
Die Entwicklung erweiterter Push-Adapter beinhaltet zwei grundlegende Schritte:

1. Implementieren der PushConnector-Schnittstelle – Diese Schnittstelle empfängt die hinzuzufügenden, zu aktualisierenden oder zu löschenden Daten in Form einer RTN-Liste und führt den Datenpush zum Ziel durch.
2. Erstellen der Zuordnungsdatei – Die Zuordnungsdatei bestimmt die Erstellung der RTN-Struktur, indem sie CIs und Attribute des TQL-Ergebnisses zuordnet.

Zuordnungsdatei

Das folgende Beispiel verdeutlicht die Vorgehensweise beim Erstellen der Zuordnungsdatei.

In diesem Beispiel simulieren wir den Push eines Knotens und einer IP-Adresse. Wir erstellen eine TQL-Abfrage mit der Bezeichnung **Node Push**. Dazu gehen wir wie folgt vor:



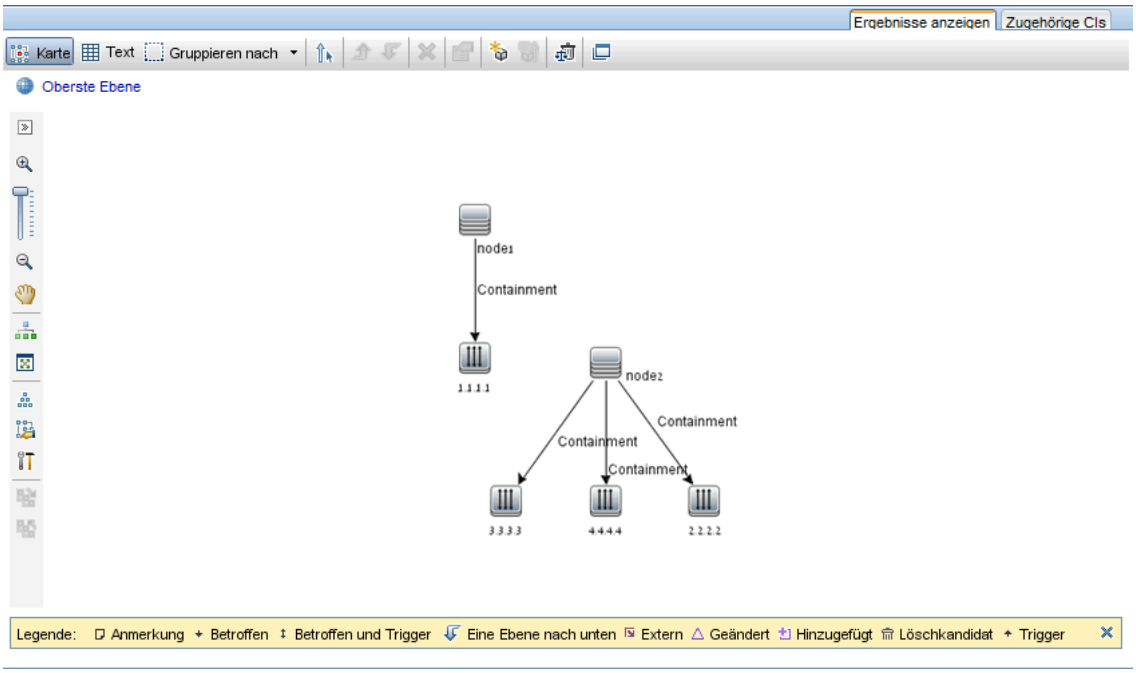
In der Zuordnungsdatei erstellen wird zwei Ziel-CI-Typen: **Computer** und **IP**. **Computer** weist eine Variable und zwei Attribute auf. **IP** besitzt ein Attribut.

Die XML-Zuordnungsdatei lautet wie folgt:

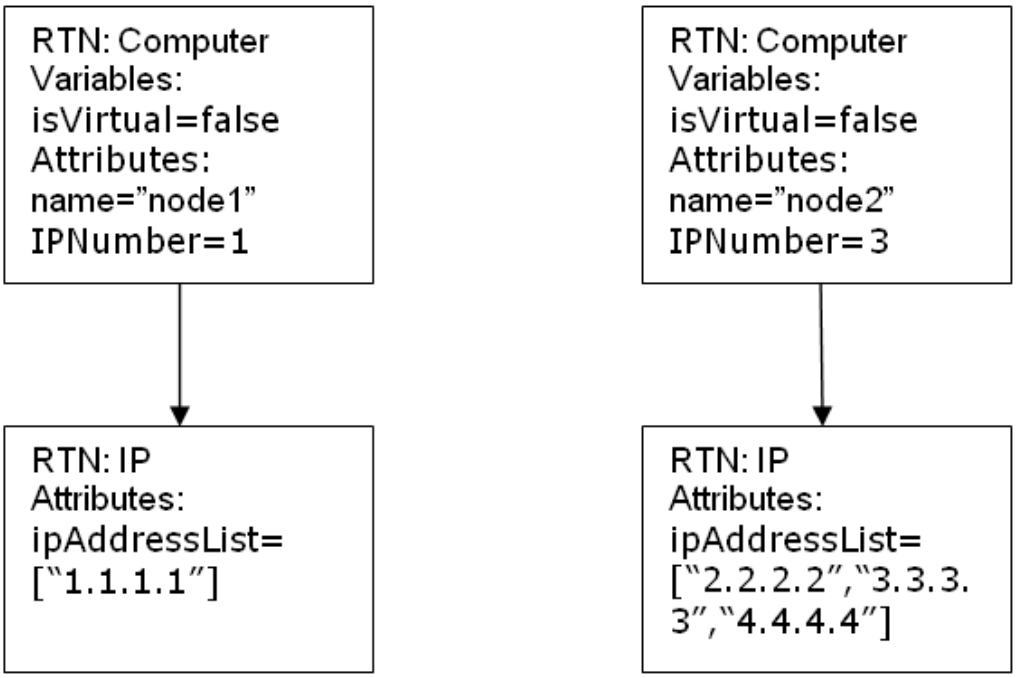
```
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://schemas.hp.com/ucmdb/1/pushAdap
ter">
        <info>
            <source name="UCMDB" versions="10.0" vendor="HP" />
            <target name="PushProduct" versions="9.3" vendor="HP" />
        </info>
        <import>
            <!—imports the Groovy script file. -->
            <scriptFile path="mappings.scripts.PushFunctions" />
        </import>
<targetcis>
<source_instance_type query-name="Node Push" root-element-name="Root">
 <target_ci_type name="Computer" is-
    valid="(Root['root_iscandidatefordeletion']==null) ? true :
    !Root['root_iscandidatefordeletion']">
  <variable name="isVirtual" datatype="BOOLEAN"
    value="PushFunctions.isVirtual(Root['root_class']) " />

  <target_mapping name="name" datatype="String" value="Root[`name’] " />
  <target_mapping name="ipNumber" datatype="INTEGER"
    value="Root.IpAddress.size()" />
  <target_mapping name="description" datatype="STRING" value=" PushFunctions
    .getDescription(isVirtual)" />
  <target_ci_type name="IP">
    <target_mapping name="ipAddressList" datatype="STRING_LIST"
    value="Root.IpAddress*.getAt(`name’) " />
</target_ci_type>
</target_ci_type>
</targetcis>
</integration>
```

Die Abfrageergebnisse werden wie folgt angezeigt:



Die RTN-Liste, die gemäß dieser Zuordnungsdatei erstellt wird, sieht folgendermaßen aus:



Jede Stamminstanz wird mithilfe der Zuordnungsdatei separat zugeordnet. Folglich empfängt der PushConnector in diesem Beispiel eine Liste mit zwei RTN-Stämmen.

Hinweis: Der vorherige Push-Adapter war in der Lage, eine allgemeine Zuordnung für einen CI-Typen zu erstellen. Die Zuordnung des neuen Push-Adapters erfolgt per TQL-Abfrage.

Während der Ausführung eines Push-Jobs, der eine Abfrage namens **x** verwendet, sucht der Adapter nach der entsprechenden Zuordnungsdatei (die das Attribut **query-name=x** aufweist).

Sie können die Werte in der Zuordnungsdatei mit der Groovy-Skriptsprache berechnen. Weitere Informationen finden Sie unter "Groovy Traveler" oben.

Groovy Traveler

Wir können wie folgt auf die TQL-Abfrageergebnisse zugreifen:

- **Root[attr]** gibt das Attribut **attr** des Stammelements zurück.
- **Root.Query_Element_Name** gibt eine Liste der CI-Instanzen zurück, die in **Query_Element_Name** in der TQL benannt und mit der aktuellen Stamm-CI verknüpft sind.
- **Root.Query_Element_Name[2][attr]** gibt das Attribut **attr** des dritten **Query_Element_Name** zurück, das mit dem aktuellen Stamm-CI verknüpft ist.
- **Root.Query_Element_Name*.getAt(attr)** gibt eine Liste der Attribute **attr** der CI-Instanzen zurück, die in **Query_Element_Name** in der TQL benannt und mit der aktuellen Stamm-CI verknüpft sind.

Es gibt weitere Attribute, auf die Groovy Traveler zugreifen kann:

- **cmdb_id** – Gibt die UCMDB-ID des CI oder der Beziehung als Zeichenkette zurück.
- **external_cmdb_id** – Gibt die externe ID des CI oder der Beziehung als Zeichenkette zurück.
- **Element_type** – Gibt den Elementtyp des CI oder der Beziehung als Zeichenkette zurück.

Tag "import":

```
<import>

<scriptFile path="mappings.scripts.PushFunctions"/>

</import>
```

Dies bedeutet, dass wir einen Import für alle Groovy-Skripts in der Zuordnungsdatei deklarieren. In diesem Beispiel ist **PushFunctions** eine Groovy-Skriptdatei, die einige statische Funktionen enthält. Auf diese können wir während der Zuordnung zugreifen (d. h. value=" PushFunctions.foo()")

source_instance_type

Die Zuordnung erfolgt per TQL. Der Wert für **query-name** ist die zugehörige TQL der aktuellen Zuordnung. Das Sternchen (*) gibt an, dass diese Zuordnungsdatei allen TQL-Abfragen zugeordnet ist, die mit dem Präfix **Node Push** beginnen.

```
<source_instance_type query-name="Node Push*" root-element-
name="Root">
```

Der Tag **source_instance_type** kennzeichnet das Stammelement, das wir zuordnen.

root-element-name muss exakt mit dem Namen des Stamms in der TQL übereinstimmen..

target_ci_type

Dieser Tag wird für die RTN-Erstellung verwendet.

Das Attribut **name** repräsentiert den Namen von **target_ci_type**: name=Computer

Das Attribut **is-valid** ist ein boolescher Wert, der während der Zuordnung berechnet wird. Er bestimmt, ob das aktuelle **target_ci**-Element gültig ist. Ungültige **target_ci_type**-Elemente werden nicht zum RTN hinzugefügt. In diesem Beispiel möchten wir keine **target_ci_type**-Instanz erstellen, für die das Attribut **root_iscandidatefordeletion** in UCMDB auf **true** gesetzt ist.

Das **target_ci_type**-Element kann Variablen aufweisen, die während der Zuordnung berechnet werden:

```
<variable name="vSerialNo" datatype="STRING" value="Root['serial_
number']"/>
```

Die Variable **vSerialNo** erhält den Wert vom Element **serial_number** des aktuellen Stamms.

Das RTN-Attribut wird von dem Tag **target_mapping** erstellt. Das Ergebnis der Ausführung des Groovy-Skripts im Feld **value** wird dem RTN-Attribut zugewiesen.

```
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
```

SerialNo weist den Wert der Variablen **vSerialNo** zu.

Es ist möglich, ein **target_ci_type**-Element als untergeordnetes Element eines anderen **target_ci_type**-Elements zu definieren. Dies geschieht wie folgt:

```
<target_ci_type name="Portfolio">

<variable name="vSerialNo" datatype="STRING" value="Root['global_id']
"/>

<target_mapping name="CMDBId" datatype="STRING" value="globalId"/>

<target_ci_type name=Asset">

<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>

</target_ci_type>

</target_ci_type>
```

Der RTN **Portfolio** weist einen untergeordneten RTN mit der Bezeichnung **Asset** auf.

for-each-source-ci

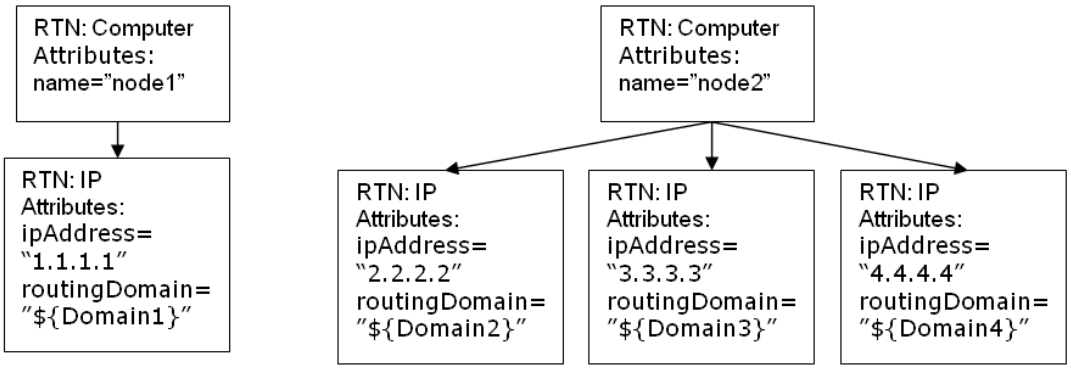
Dieser Tag listet die spezifischen CIs der Stamminstanz auf. Er weist die folgenden Felder auf:

- source-cis=" " – Liste der CIs, für die ein Ziel-CI erstellt wird. Diese Liste wird von Groovy Traveler im Feld **Root.IpAddress** definiert.
- count-index=" " – Variable, die den Index des CI in der aktuellen Iteration der Schleife enthält.
- var-name=" " – Name des CI in der aktuellen Iteration der Schleife.

Verändern wir nun unsere Zuordnungsbeispieldatei:

```
<target_ci_type name="Computer">
  <target_mapping name="name" datatype="String" value="Root[`name'] " />
   <for-each-source-ci count-index="i" var-name="currIP" source-cis="Root.IpAddress" >
     <target_ci_type name="IP">
         <target_mapping name="ipAddress" datatype="STRING"
     value="Root.IpAddress[i][`name']"/>
         <target_mapping name="routingDomain" datatype="STRING"
     value="currIP['routing_domain']"/>
   </target_ci_type>
 </for-each-source-ci>
 </target_ci_type>
```

Die RTN-Liste, die gemäß dieser Zuordnungsdatei erstellt wird, sieht folgendermaßen aus:



dynamic_mapping

Dieser Tag bewirkt, dass während der Erstellung der RTN-Struktur eine Datenzuordnung vom Zieldatenspeicher erstellt werden kann.

Beispiel: Nehmen wir an, das Ziel ist eine Datenbank mit einer Tabelle namens **Computer**. Diese besteht aus einer Spalte **id** und einer Spalte **name**, die mit **Node.name** in UCMDB zusammenhängt. Beide Spalten sind eindeutig. Die Datenbank weist außerdem eine Tabelle namens **IP** auf. Diese besitzt einen referenzierten Schlüssel, der auf **parentID** in der Tabelle **Computer** verweist. Der Tag **dynamic_mapping** kann eine Zuordnung erstellen, die den Namen und die ID als <name,id> speichert. Auf Basis dieser Zuordnung kann der Adapter IDs mit Computern abgleichen und den korrekten Wert an das Attribut **parentID** in der Tabelle **IP** übertragen. Mithilfe dieser Zuordnung können Sie dem Attribut **parentID** während der Erstellung der RTN-Struktur einen Wert zuweisen.

Die Zuordnung wird durch **map_property** bestimmt. Der Tag **dynamic_mapping** wird einmal für jeden Chunk ausgeführt.

```
<dynamic_mapping name="IdByName " keys-unique="true">
```

Das Attribut **name** stellt den Namen der Zuordnung dar. Das Attribut **keys-unique** gibt an, ob die Schlüssel eindeutig sind (jeder Schlüssel wird einem Wert oder einem Satz von Werten zugeordnet).

Der Name der Zuordnung in diesem Beispiel lautet **IdByName**. Die Schlüssel sind eindeutig. Führen Sie den folgenden Befehl aus, um im Skript auf die Zuordnung zuzugreifen:

```
DynamicMapHolder.getMap('IdByName')
```

Dieser Befehl gibt einen Verweis auf die Zuordnung zurück.

Der Tag **map_property** erstellt die Eigenschaft, auf der die Zuordnung basiert.

Beispiel:

```
<map_property property-name="SQLQuery" datatype="STRING"
property-value="SELECT name, id FROM Computer"/>
```

In diesem Beispiel hat die Eigenschaft den Namen **SQLQuery** und ihr Wert ist eine SQL-Anweisung, die die Zuordnung erstellt. Die Implementierung der Methoden **retrieveUniqueMapping** und **retrieveNonUniqueMapping** für die PushConnector-Schnittstelle bestimmt den tatsächlichen Inhalt der zurückgegebenen Zuordnung.

Globale Variablen

Die folgenden globalen Variablen sind für das Groovy-Skript in der Zuordnungsdatei zugänglich:

- **Topology** – Typ: Topologie. Eine Instanz der Topologie des aktuellen Chunks.
- **QueryDefinition** - Typ: QueryDefinition. Eine Instanz der Abfragedefinition der aktuellen TQL.
- **OutputCI** – Typ: ResultTreeNode. Der RTN des Stammelements in der aktuellen Strukturzuordnung.
- **ClassModel** – Typ: ClassModel. Eine Instanz des Klassenmodells.
- **CustomerInformation** – Typ: CustomerInformation. Informationen über den Kunden, der den Job ausführt.
- **Logger** – Typ: DataAdapterLogger. Diese Protokollierung ist im Adapter verfügbar, um Protokolle in das UCMDB-Protokollierungs-Framework zu schreiben.

Schreiben von Groovy-Skripts

In diesem Abschnitt erstellen wir die Datei **PushFunctions.groovy**. Diese Datei enthält statische Funktionen, die während der Zuordnung der Stamminstanz verwendet werden.

```
package mappings.scripts

public class PushFunctions {

	public static boolean isVirtual(def nodeRole){
		return isListContainsOne(def list, "MY_VM", "MY_SIMULATOR");
	}



public static String getDescription(boolean isVirtual){
		if(isVirtual){
			return "This is a VM";
		}
		else{
			return "This is physical machine";
		}
	}
```

```
private static boolean isListContainsOne(def list, ...stringList){
                //returns true if the list contains one of the values.
        }
}
```

Implementieren der PushConnector-Schnittstelle

Die Implementierung sollte die folgenden grundlegenden Schritte unterstützen:

```
public class PushExampleAdapter implements PushConnector
```

```
public class PushExampleAdapter implements PushConnector
```

```
{

public UpdateResult pushTreeNodes(ResultTreeNodeActionData
resultTreeNodes, QueryDefinition queryDefinition) throws
DataAccessException{

// 1. build an UpdateResult instance - the UpdateResult is used to
return mappings between the sent ids to the actual ids that entered
the data store.
// Also has an update status which allows to pass errors to the
statistics in the UI.
// 2. handle the data:
// a. handle data to add. Can be retrieved by:
resultTreeNodeActionData.getDataToAdd();
//  b. handle data to update.
//  c. handle data to delete.
// 3. Return the Update result.
        }
```

```
public void start(PushDataAdapterEnvironment env) throws
DataAccessException{
        // this method is called when the integration point created, or when
the adapter is reloaded
        //(i.e after changing one of the mapping files
        // and pressing 'save').

}
```

```
public void testConnection(PushDataAdapterEnvironment env) throws
DataAccessException {
                // this method is called when pressing the 'test connection' button
in the
                //creation of the integration point.
                // For example if we push data to RDBMS this method can create a
connection
```

```
                //to the database and will run a dummy SQL statement.
                // If it fails it writes an error message to the log and throws an
exception.
        }



Map<Object, Object> retrieveUniqueMapping(MappingQuery mappingQuery){
//This method will create the map according to the given mappingQuery.
It will be called in the
// mapping stage of the adapter execution, before the 'UpdateResult
pushTreeNodes' method.
// This method is called when the 'keys-unique' attribute of the
'dynamic_mapping' tag is true.
}

Map<Object, Set<Object>> retrieveNonUniqueMapping(MappingQuery
mappingQuery){
 // This method is called when the 'keys-unique' attribute of the
'dynamic_mapping' tag is false.
//  In this case a key can be mapped to several values.
}
}
```

Erstellen eines Adapter-Package

Das Adapter-Package sollte die folgenden Ordner enthalten:

- **adapterCode.** Erstellen Sie unter diesem Ordner einen Ordner mit der Bezeichnung **PushExampleAdapter**, in dem die JAR-Datei gespeichert wird, die wir von der Datei **PushExampleAdapter.java** erstellt haben. Außerdem wird der Ordner einen Ordner mit der Bezeichnung **mappings** enthalten, in dem Sie die zuvor erstellte Zuordnungsdatei **computerIPMapping.xml** speichern können. Darüber hinaus sollte der Ordner einen weiteren Ordner namens **scripts** enthalten, in dem die Datei **PushFunctions.groovy** abgelegt wird.
- **discoveryConfigFiles.** Für Konfigurationsdateien, wie z. B. die verwendeten Fehlercodes bei der Meldung eines Fehlers mittels **UpdateResult**. In diesem Beispiel ist der Ordner.
- **discoveryPatterns.** Für die Datei **push_example_adapter.xml**.
- **tql.** Für die TQL-Abfrage, die für das Beispiel erstellt wurde. Dieser Ordner ist optional. Die TQL wird jedoch automatisch erstellt, wenn das Package bereitgestellt wird.

Schema der Zuordnungsdatei

Elementname und -pfad	Beschreibung	Attribute
Integration	Definiert die Zuordnungsinhalte der Datei. Abgesehen von der Anfangszeile und eventuellen Kommentaren muss dies der äußerste Block in der Datei	

Elementname und -pfad	Beschreibung	Attribute
	sein.	
info (integration)	Definiert die Informationen über die zu integrierenden Daten-Repositorys.	
source (info)	Das Quellprodukt.	**Name:** name **Beschreibung:** Der Name des Produkts. **Verwendung:** Erforderlich **Typ:** Zeichenkette
		Name: vendor **Beschreibung:** Der Anbieter des Produkts. **Verwendung:** Erforderlich **Typ:** Zeichenkette
		Name: versions **Beschreibung:** Die Version des Produkts. **Verwendung:** Erforderlich **Typ:** Dezimalziffer
target (info)	Das Zielprodukt.	**Name:** name **Beschreibung:** Der Name des Produkts. **Verwendung:** Erforderlich **Typ:** Zeichenkette
		Name: vendor **Beschreibung:** Der Anbieter des Produkts. **Verwendung:** Erforderlich **Typ:** Zeichenkette

Elementname und -pfad	Beschreibung	Attribute
		Name: versions **Beschreibung:** Die Version des Produkts. **Verwendung:** Erforderlich **Typ:** Dezimalziffer
Import (integration)	Ein Container-Element für importierte Skriptdateien.	
scriptFile (integration>import)	Definiert die zu importierende Groovy-Skriptdatei.	Name: path Description: Der Pfad der Skriptdatei. Verwendung: Erforderlich. Typ: string
Targetcis (integration)	Ein Container-Element für Ziel-CI-Typen.	
Source_instance_type (integration>targetcis)	Definiert den Typ der Quell-CI-Instanz. Muss das in der TQL definierte Stammelement sein.	**Name:** query-name. **Beschreibung:** Der Name der relevanten TQL. **Verwendung:** Erforderlich **Typ:** Zeichenkette
		Name: name **Beschreibung:** Muss das in der TQL definierte Stammelement sein. **Verwendung:** Erforderlich **Typ:** Zeichenkette
dynamic_mapping (integration>targetcis> source_instance_type)	Definiert eine Zuordnung, die einmal pro Chunk erstellt wird.	**Name:** name **Beschreibung:** Der Name der Zuordnung. **Verwendung:** Erforderlich **Typ:** Zeichenkette

Elementname und -pfad	Beschreibung	Attribute
		Name: keys-unique **Beschreibung:** Gibt an, ob die Schlüssel eindeutig sind. **Verwendung:** Erforderlich **Typ:** Boolescher Wert
target_ci_type (integration>targetcis> source_instance_type) - ODER- (integration>targetcis> source_instance_type> for-each-source-ci)	Definiert den Ziel-CI-Typ, der zum RTN hinzugefügt werden soll.	**Name:** name **Beschreibung:** Der Name des Ziel-CI-Typs. **Verwendung:** Erforderlich **Typ:** Zeichenkette
		Name: is-valid **Beschreibung:** Prüft, ob das aktuelle Ziel-CI gemäß dem angegebenen Skript gültig ist. **Verwendung:** Nicht erforderlich **Typ:** Zeichenkette (Groovy-Skript)
Variable (target_ci_type)	Definiert eine Variable für den Ziel-CI-Typ.	**Name:** name **Beschreibung:** Der Name der Variablen. **Verwendung:** Erforderlich **Typ:** Zeichenkette
		Name: datatype **Beschreibung:** Der Datentyp der Variablen. **Verwendung:** Erforderlich

Elementname und -pfad	Beschreibung	Attribute
		Typ: type-enum (Mögliche Typen sind: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_ LIST, STRING_LIST)
		Name: value **Beschreibung:** Der Wert, der der Variablen zugewiesen werden soll. **Verwendung:** Erforderlich **Typ:** Groovy-Skript
target_mapping (target_ci_ type)	Definiert ein Attribut für den Ziel-CI-Typ.	**Name:** name **Beschreibung:** Der Name des Attributs. **Verwendung:** Erforderlich **Typ:** Zeichenkette
		Name: datatype **Beschreibung:** Der Datentyp der Variablen. **Verwendung:** Erforderlich **Typ:** type-enum (Mögliche Typen sind: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_ LIST, STRING_LIST)
		Name: value **Beschreibung:** Der Wert, der der

Elementname und -pfad	Beschreibung	Attribute
		Variablen zugewiesen werden soll. **Verwendung:** Erforderlich **Typ:** Groovy-Skript
		Name: ignore-on-null **Beschreibung:** Wenn der Wert der Zielzuordnung null beträgt und dieses Attribut auf TRUE gesetzt ist, wird das Attribut ignoriert. **Verwendung:** Nicht erforderlich **Typ:** Boolescher Wert (Groovy-Skript)
before-mapping (target_ci_type)	Ein Groovy-Skript, das vor der Zuordnung des Ziel-CI-Typs ausgeführt wird.	
after-mapping (target_ci_type)	Ein Groovy-Skript, das nach der Zuordnung des Ziel-CI-Typs ausgeführt wird.	
for-each-source-ci (target_ci_type)	Definiert eine Iteration spezifischer CIs der Stamminstanz.	**Name:** count-index **Beschreibung:** Index des aktuell wiederholten CI. **Verwendung:** Erforderlich **Typ:** Zeichenkette
		Name: var-name **Beschreibung:** Die Variable, die auf das aktuell wiederholte CI verweist. **Verwendung:** Nicht erforderlich **Typ:** Zeichenkette

Elementname und -pfad	Beschreibung	Attribute
		Name: source-cis **Beschreibung:** Der CI-Name in der Abfrage, die wiederholt werden soll. **Verwendung:** Erforderlich **Typ:** Zeichenkette (Groovy-Skript)

Verwenden von APIs

Kapitel 8

Einführung zu APIs

Dieses Kapitel umfasst folgende Themen:

APIs – Übersicht

Die folgenden APIs sind in HP Universal CMDB enthalten:

- **UCMDB-Java-API.** Erklärt, wie benutzerdefinierte Werkzeuge oder Werkzeuge von Drittanbietern die Java-API verwenden können, um Daten und Berechnungen zu extrahieren und Daten in UCMDB (Universal Configuration Management Database) zu schreiben. Weitere Informationen finden Sie unter "HP Universal CMDB-API" auf Seite 229.
- **UCMDB-Webservice-API.** Ermöglicht das Schreiben von CI-Definitionen und topologischen Beziehungen in UCMDB (Universal Configuration Management Database) sowie das Abfragen der Informationen mittels TQL- und Ad-hoc-Abfragen. Weitere Informationen finden Sie unter "HP Universal CMDB-Webservice-API" auf Seite 237.
- **API des Webservice "Datenflussverwaltung".** Ermöglicht das Verwalten von Proben, Jobs, Triggern und Anmeldeinformationen für die Datenflussverwaltung. Weitere Informationen finden Sie unter "API zur Datenflussverwaltung" auf Seite 295.

Hinweis: Um den vollen Nutzen aus der API-Dokumentation zu ziehen, wird empfohlen, auf die Online-Dokumentation zuzugreifen. Die PDF-Version enthält keine Links zu der im HTML-Format generierten API-Dokumentation.

Kapitel 9

HP Universal CMDB-API

Dieses Kapitel umfasst folgende Themen:

Konventionen

In diesem Kapitel werden die folgenden Konventionen verwendet:

- **UCMDB** ist die Abkürzung für Universal Configuration Management Database und bezeichnet die Datenbank für die Konfigurationsverwaltung. HP Universal CMDB bezieht sich auf die Applikation.
- UCMDB-Elemente und Methodenargumente werden so geschrieben, wie sie in den Schnittstellen angegeben sind.

Die vollständige Dokumentation zu den verfügbaren APIs finden Sie in der HP UCMDB API Reference.

Die Dateien befinden sich im folgenden Ordner:

\\<UCMDB-Stammverzeichnis>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Verwenden der HP Universal CMDB-API

Hinweis: Verwenden Sie dieses Kapitel zusammen mit der API-Java-Dokumentation, das in der Online-Dokumentationsbibliothek zur Verfügung steht.

Die HP Universal CMDB-API wird für die Integration von Anwendungen mit Universal CMDB (CMDB) verwendet. Die API stellt Methoden für folgende Aufgaben zur Verfügung:

- Hinzufügen, Entfernen und Aktualisieren von CIs und Beziehungen in der CMDB
- Abrufen von Informationen zum Klassenmodell
- Abrufen von Informationen aus der UCMDB-Historie
- Ausführen von Was-wäre-wenn-Szenarien
- Abrufen von Informationen zu Konfigurationselementen und Beziehungen

Methoden zum Abrufen von Informationen zu Konfigurationselementen und Beziehungen verwenden im Allgemeinen TQL-Abfragen (TQL = Topology Query Language, Topologieabfragesprache). Weitere Informationen finden Sie unter "Topology Query Language" im *HP Universal CMDB – Modellierungshandbuch*.

Benutzer der HP Universal CMDB-API sollten mit Folgendem vertraut sein:

- Der Programmiersprache Java
- HP Universal CMDB

Dieser Abschnitt umfasst die folgenden Themen:

- "Verwendung der API" oben
- "Berechtigungen" oben

Verwendung der API

Die API kann für eine Reihe von Geschäftsanforderungen eingesetzt werden. Beispielsweise kann ein Drittanbietersystem das Klassenmodell nach Informationen zu den verfügbaren Konfigurationselementen (CIs) abfragen. Weitere Anwendungsfälle finden Sie unter "Anwendungsfälle" auf Seite 234.

Berechtigungen

Der Administrator stellt die Anmeldeinformationen zum Herstellen der Verbindung zur API zur Verfügung. Der API-Client benötigt den Benutzernamen und das Kennwort eines in der CMDB definierten Integrationsbenutzers. Bei diesen Benutzern handelt es sich nicht um menschliche Benutzer der CMDB, sondern vielmehr um Anwendungen, die mit der CMDB verbunden sind.

Achtung: Der API-Client kann auch mit normalen Benutzern arbeiten, wenn diese über eine API-Authentifizierungsberechtigung verfügen. Diese Option wird jedoch nicht empfohlen.

Weitere Informationen finden Sie unter "Erstellen eines Integrationsbenutzers" auf Seite 232.

Allgemeine Struktur einer Applikation

Es gibt nur eine statische Factory, die UcmdbServiceFactory. Diese Factory ist der Einstiegspunkt für eine Applikation. Die UcmdbServiceFactory macht die getServiceProvider-Methoden verfügbar. Diese Methoden geben eine Instanz der **UcmdbServiceProvider**-Schnittstelle zurück.

Der Client erstellt andere Objekte unter Verwendung von Schnittstellenmethoden. Zum Erstellen einer neuen Abfragedefinition führt der Client beispielsweise folgende Schritte durch:

1. Er ruft den Abfrageservice vom Hauptserviceobjekt der CMDB ab.
2. Er ruft ein Abfrage-Factory-Objekt vom Serviceobjekt ab.
3. Er ruft eine neue Abfragedefinition von der Factory ab.

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

Folgende Services sind vom **UcmdbService** verfügbar:

Servicemethoden	Verwendung
getClassModelService	Informationen über CI-Typen und Beziehungen.
getConfigurationService	Verwaltung der Infrastruktureinstellungen für die Serverkonfiguration
getDDMConfigurationService	Konfigurieren der Datenflussverwaltung
getDDMManagementService	Analysieren und Anzeigen des Fortschritts, der Ergebnisse und der Fehler der Datenflussverwaltung
getHistoryService	Informationen zur Historie der überwachten CIs (Änderungen, Löschungen usw.)
getImpactAnalysisService	Ausführen des Auswirkungsanalyseszenarios (auch bekannt als **Korrelation**).
getQueryManagementService	Verwalten des Zugriffs auf Anfragen - Speichern, Löschen, Auflisten vorhandener Elemente. Ermöglicht außerdem die Prüfung von Abfragen und die Discovery von Abfrageabhängigkeiten.
getResourceBundleManagementService	Ressourcen-Tagging (Bundling-Services). Ermöglicht die explizite Erstellung neuer Tags und die Entfernung von Tags von allen getaggten Ressourcen.
getStateService	Bereitstellen von Services für die Verwaltung von Status (Auflisten, Hinzufügen, Entfernen usw.)
getSoftwareSignatureService	Definieren der vom Discovery- und Dependency-

Servicemethoden	Verwendung
	Management-System zu ermittelnden Softwareelemente.
getSnapshotService	Bereitstellen von Services für die Verwaltung von Baselines (Abrufen, Speichern, Vergleichen usw.)
getTopologyQueryService	Abrufen von Informationen über IT Universe.
getTopologyUpdateService	Ändern von Informationen in IT Universe.
getViewService	Anzeigen des Ausführungsservice (Ausführungsdefinition, Ausführung gespeichert) und des Managementservice (Speichern, Löschen, Auflisten vorhandener Elemente). Ermöglicht außerdem die Prüfung von Ansichten und die Discovery von Abhängigkeiten.
getViewArchiveService	Anzeigen der Ergebnisse der Archivierungsservices. Ermöglicht das Speichern des aktuellen Ansichtsergebnisses und das Abrufen zuvor gespeicherter Ergebnisse.
SystemHealthService	Bereitstellen von System Health-Services (Indikatoren zur grundlegenden Systemleistung, Kapazitäts- und Verfügbarkeitskennzahlen)

Die Client kommuniziert mit dem Server über HTTP.

Speichern der API-JAR-Datei im Klassenpfad

Um diesen API-Satz verwenden zu können, benötigen Sie die Datei **ucmdb-api.jar**. Sie können die Datei herunterladen, indem Sie `http://<localhost>:8080` in einen Webbrowser eingeben (wobei `localhost` dem Computer entspricht, auf dem UCMDB installiert ist) und auf den Link **API Client Download** klicken.

Speichern Sie die JAR-Datei im Klassenpfad, bevor Sie Ihre Applikation kompilieren oder ausführen.

Hinweis: Um die JAR-Dateien der UCMDB-Java-API verwenden zu können, muss die JRE-Version 6 oder höher installiert sein.

Erstellen eines Integrationsbenutzers

Für Integrationen zwischen anderen Produkten und UCMDB können Sie einen dedizierten Benutzer erstellen. Mit diesem Benutzer ist es möglich, ein Produkt, das das UCMDB-Client-SDK verwendet, beim Server-SDK zu authentifizieren und die APIs auszuführen. Applikationen, die mit diesem API-Satz geschrieben wurden, müssen sich mit den Anmeldeinformationen des Integrationsbenutzers anmelden.

Achtung: Ferner ist es möglich, eine Verbindung zu einem regulären UCMDB-Benutzer (z. B. admin) herzustellen, jedoch wird diese Option nicht empfohlen. Um eine Verbindung zu einem UCMDB-Benutzer herzustellen, müssen Sie dem Benutzer eine API-Authentifizierungsberechtigung erteilen.

So erstellen Sie einen Integrationsbenutzer:

1. Starten Sie den Webbrowser und geben Sie die folgende Serveradresse ein:

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

   Eventuell müssen Sie sich mit einem Benutzernamen und einem Kennwort anmelden (die Standardeinstellung lautet **sysadmin/sysadmin**).

2. Klicken Sie unter `UCMDB` auf **service=UCMDB Authorization Services**.

3. Suchen Sie den Vorgang **createUser**. Diese Methode verwendet die folgenden Parameter:

   - **customerId**. Die Kunden-ID.
   - **username**. Der Name des Integrationsbenutzers.
   - **userDisplayName**. Der Anzeigename des Integrationsbenutzers.
   - **userLoginName**. Der Anmeldename des Integrationsbenutzers.
   - **password**. Das Kennwort des Integrationsbenutzers.

4. Klicken Sie auf **Invoke**.

5. Suchen Sie in einer Einzelmandantenumgebung die Methode **setRolesForUser** und geben Sie die folgenden Parameter ein:

   - **userName**. Der Name des Integrationsbenutzers.
   - **roles.** SuperAdmin.

   Klicken Sie auf **Invoke**.

6. Suchen Sie in einer Mehrmandantenumgebung die Methode **grantRolesToUserForAllTenants** und geben Sie die folgenden Parameter ein, um die Rolle in Verbindung mit allen Mandanten zuzuweisen:

   - **userName**. Der Name des Integrationsbenutzers.
   - **roles.** SuperAdmin.

   Klicken Sie auf **Invoke**.

   Um die Rolle in Verbindung mit bestimmten Mandanten zuzuweisen, rufen Sie die Methode **grantRolesToUserForTenants** auf und verwenden Sie für die Parameter **userName** und **roles** die gleichen Werte. Geben Sie für den Parameter **tenantNames** die erforderlichen Mandanten ein.

7. Sie können nun entweder weitere Benutzer erstellen oder die JMX-Konsole schließen.

8. Melden Sie sich bei UCMDB als Administrator an.

9. Klicken Sie auf die Registerkarte **Verwaltung** und führen Sie **Package Manager** aus.

10. Klicken Sie auf das Symbol **Benutzerdefiniertes Package erstellen**.
11. Geben Sie einen Namen für das neue Package ein und klicken Sie auf **Weiter**.
12. Öffnen Sie die Registerkarte **Ressourcenauswahl** und klicken Sie unter **Verwaltung** auf **Benutzer**.
13. Wählen Sie einen oder mehrere der Benutzer aus, die Sie mithilfe der JMX-Konsole erstellt haben.
14. Klicken Sie auf **Weiter** und anschließend auf **Fertig stellen**. Ihr neues Package wird in Package Manager in der Liste unter **Package-Name** angezeigt.
15. Stellen Sie das Package für die Benutzer bereit, die die API-Applikationen ausführen werden.

    Weitere Informationen hierzu finden Sie unter "Bereitstellen eines Package" im *HP Universal CMDB – Verwaltungshandbuch*.

Hinweis:

Der Integrationsbenutzer gilt nur für einen Kunden. Um einen stärkeren Integrationsbenutzer für die kundenübergreifende Verwendung zu erstellen, verwenden Sie einen **systemUser**, bei dem das Flag **isSuperIntegrationUser** auf **true** festgelegt wird. Verwenden Sie die Methoden **systemUser** (**removeUser**, **resetPassword**, **UserAuthenticate** usw.).

Es gibt zwei vordefinierte Systembenutzer. Es wird empfohlen, deren Kennwörter nach der Installation mit der Methode **resetPassword** zu ändern.

- **sysadmin**/**sysadmin**
- **UISysadmin**/**UISysadmin** (Dieser Benutzer ist auch der Superintegrationsbenutzer **SuperIntegrationUser**).

  Wenn Sie das Kennwort **UISysadmin** mit der Methode **resetPassword** ändern, müssen Sie die folgende Methode ausführen: Lokalisieren Sie in der JMX-Konsole den Service **UCMDB-UI:name=UCMDB Integration**. Führen Sie **setCMDBSuperIntegrationUser** mit dem Benutzernamen und dem neuen Kennwort des Integrationsbenutzers aus.

Referenz zur HP Universal CMDB-API

Die Dateien befinden sich im folgenden Ordner:

\\<UCMDB-Stammverzeichnis>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Anwendungsfälle

Die folgenden Anwendungsfälle setzen zwei Systeme voraus:

- HP Universal CMDB Server
- Ein Drittanbietersystem, das ein Repository der Konfigurationselemente enthält.

Dieser Abschnitt umfasst die folgenden Themen:

- "Auffüllen der CMDB" oben
- "Abfragen der CMDB" oben
- "Abfragen des Klassenmodells" oben
- "Analysieren von Änderungsauswirkungen " oben

Auffüllen der CMDB

Anwendungsfälle:

- Ein Asset-Management-Werkzeug eines Drittanbieters aktualisiert die CMDB mit den Informationen, die nur ihm zur Verfügung stehen.
- Mehrere Drittanbietersysteme füllen die CMDB auf, um eine zentrale CMDB zu erstellen, die Änderungen verfolgen und Auswirkungsanalysen durchführen kann.
- Ein Drittanbietersystem erstellt Konfigurationselemente und Beziehungen gemäß der Geschäftslogik des Drittanbieters, um die Abfragefunktionen der CMDB zu nutzen.

Abfragen der CMDB

Anwendungsfälle:

- Ein Drittanbietersystem ruft die Konfigurationselemente und Beziehungen ab, die das SAP-System darstellen, indem es die Ergebnisse der SAP-TQL abruft.
- Ein Drittanbietersystem ruft die Liste der Oracle-Server ab, die innerhalb der letzten 5 Stunden hinzugefügt oder geändert wurden.
- Ein Drittanbietersystem ruft die Liste der Server ab, deren Hostname die Unterzeichenfolge `lab` enthält.
- Ein Drittanbietersystem sucht die zu einem bestimmten CI gehörenden Elemente, indem es dessen Nachbarn abruft.

Abfragen des Klassenmodells

Anwendungsfälle:

- Ein Drittanbietersystem ermöglicht den Benutzern, den Datensatz anzugeben, der von der CMDB abgerufen werden soll. Über dem Klassenmodell kann eine Benutzeroberfläche angeordnet werden, um den Benutzern die möglichen Eigenschaften anzuzeigen und sie zur Eingabe der erforderlichen Daten aufzufordern. Der Benutzer kann dann die abzurufenden Informationen auswählen.
- Ein Drittanbietersystem durchsucht das Klassenmodell, wenn der Benutzer nicht auf die UCMDB-Benutzeroberfläche zugreifen kann.

Analysieren von Änderungsauswirkungen

Anwendungsfall:

Ein Drittanbietersystem gibt eine Liste der Geschäftsservices aus, die von einer Änderung bei einem bestimmten Host betroffen sein könnten.

Beispiele

Siehe folgende Codebeispiele:

- Create a Connection
- Create and Execute an Ad-Hoc Query
- Create and Execute a View
- Add and Delete Data
- Execute an Impact Analysis
- Query the Class Model
- Query a History Sample

Diese Dateien befinden sich im folgenden Verzeichnis:

\\<UCMDB-Stammverzeichnis>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples

Kapitel 10

HP Universal CMDB-Webservice-API

Dieses Kapitel umfasst folgende Themen:

Konventionen

In diesem Kapitel werden die folgenden Konventionen verwendet:

- **UCMDB** ist die Abkürzung für Universal Configuration Management Database und bezeichnet die Datenbank für die Konfigurationsverwaltung. HP Universal CMDB bezieht sich auf die Applikation.
- UCMDB-Elemente und Methodenargumente werden genauso geschrieben wie im Schema angegeben. Ein Element oder Argument einer Methode wird kleingeschrieben. Beispiel: `relation` ist ein Element des Typs `Relation`, das an eine Methode übergeben wird.

Die vollständige Dokumentation zu den Anforderungs- und Antwortstrukturen finden Sie unter HP UCMDB Web Service API Reference. Die Dateien befinden sich im folgenden Ordner:

<UCMDB-Stammverzeichnis>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

HP Universal CMDB-Webservice-API – Übersicht

Hinweis: Verwenden Sie dieses Kapitel zusammen mit der UCMDB-Schemadokumentation, die in der Online-Dokumentationsbibliothek zur Verfügung steht.

Die HP Universal CMDB-Webservice-API wird für die Integration von Applikationen mit HP Universal CMDB (UCMDB) verwendet. Die API stellt Methoden für folgende Aufgaben zur Verfügung:

- Hinzufügen, Entfernen und Aktualisieren von CIs und Beziehungen in der CMDB
- Abrufen von Informationen zum Klassenmodell
- Abrufen von Auswirkungsanalysen
- Abrufen von Informationen zu Konfigurationselementen und Beziehungen
- Verwalten von Anmeldeinformationen: Anzeigen, Hinzufügen, Aktualisieren und Entfernen
- Verwalten von Jobs: Anzeigen des Status, Aktivieren und Deaktivieren
- Verwalten von Probe-Bereichen: Anzeigen, Hinzufügen und Aktualisieren
- Verwalten von Triggern: Hinzufügen oder Entfernen eines Trigger-CI sowie Hinzufügen, Entfernen oder Deaktivieren einer Trigger-TQL
- Anzeigen von allgemeinen Daten zu Domänen und Proben

Methoden zum Abrufen von Informationen zu Konfigurationselementen und Beziehungen verwenden im Allgemeinen TQL-Abfragen (TQL = Topology Query Language, Topologieabfragesprache). Weitere Informationen finden Sie unter "Topology Query Language" im *HP Universal CMDB – Modellierungshandbuch*.

Benutzer der HP Universal CMDB-Webservice-API sollten mit Folgendem vertraut sein:

- Der SOAP-Spezifikation
- Einer objektorientierten Programmiersprache wie C++, C# oder Java
- HP Universal CMDB
- Datenflussverwaltung

Dieser Abschnitt umfasst die folgenden Themen:

- "Verwendung der API" oben
- "Berechtigungen" auf der nächsten Seite

Verwendung der API

Die API kann für eine Reihe von Geschäftsanforderungen eingesetzt werden. Beispiel:

- Ein Drittanbietersystem kann das Klassenmodell nach Informationen zu den verfügbaren Konfigurationselementen (CIs) abfragen.
- Ein Asset-Management-Werkzeug eines Drittanbieters kann die CMDB mit Informationen aktualisieren, die nur ihm zur Verfügung stehen, und so seine Daten mit den von den HP-

Applikationen erfassten Daten zusammenführen.

- Mehrere Drittanbietersysteme können die CMDB auffüllen, um eine zentrale CMDB zu erstellen, die Änderungen verfolgen und Auswirkungsanalysen durchführen kann.
- Ein Drittanbietersystem kann Entitäten und Beziehungen gemäß seiner Geschäftslogik erstellen und die Daten anschließend in die CMDB schreiben, um die zugehörigen Abfragefunktionen zu nutzen.
- Andere Systeme, wie beispielsweise Systeme zur Versionskontrolle (CCM-Systeme), können die Methoden zur Auswirkungsanalyse für Änderungsanalysen verwenden.

Berechtigungen

Um auf die WDSL-Datei für den Webservice zuzugreifen, öffnen Sie **http://localhost:8080/axis2/services/UcmdbService?wsdl**. Sie müssen sich als Serveradministrator anmelden, um die WSDL-Datei anzuzeigen.

Der Benutzer muss die Berechtigung für die allgemeine Aktion **Legacy-API ausführen** besitzen, um sich anmelden zu können.

In der folgenden Tabelle sind die weiteren erforderlichen Berechtigungen für jeden Webservice-API-Befehl aufgeführt:

Webservice-API-Befehl	Erforderliche Berechtigungen
addCIsAndRelations deleteCIsAndRelations updateCIsAndRelations	Allgemeine Aktion: **Daten aktualisieren**
executeTopologyQueryByName(AdHoc) executeTopologyQueryByNameWithParameters(AdHoc) executeTopologyQueryWithParameters(AdHoc)	Allgemeine Aktion: **Abfrage über Definition ausführen** Für jede Abfrage: Berechtigung **Anzeigen**
getTopologyQueryExistingResultByName getTopologyQueryResultCountByName releaseChunks pullTopologyMapChunks getCINeighbours getFilteredCIsByType getCIsById getCIsByType getRelationsById	Allgemeine Aktion: **CIs anzeigen** Für jede Abfrage: Berechtigung **Anzeigen**
getQueryNameOfView	Allgemeine Aktion: **CIs anzeigen** Für jede Ansicht: Berechtigung **Anzeigen**
getChangedCIs	Allgemeine Aktion: **Historie anzeigen**, **CIs anzeigen**

Webservice-API-Befehl	Erforderliche Berechtigungen
calculateImpact getImpactPath getImpactRulesByGroupName getImpactRulesByNamePrefix	Allgemeine Aktion: **Auswirkungsanalyse ausführen**
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Keine

HP Universal CMDB – Referenz zur HP UCMDB-Webservice-API

Die vollständige Dokumentation zu den Anforderungs- und Antwortstrukturen finden Sie unter HP UCMDB Web Service API Reference. Die Dateien befinden sich im folgenden Ordner:

<UCMDB-Stammverzeichnis>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

Aufrufen des Webservice

Zum Aufrufen serverseitiger Methoden verwenden Sie standardmäßige SOAP-Programmierverfahren in HP Universal CMDB-Webservice. Wenn die Anweisung nicht analysiert werden kann oder ein Problem beim Aufrufen der Methode auftritt, lösen die API-Methoden eine `SoapFault`-Ausnahme aus. Daraufhin füllt UCMDB eines oder mehrere der für Fehlermeldungen, Fehlercodes und Ausnahmemeldungen vorgesehenen Felder mit den entsprechenden Daten. Wenn kein Fehler festgestellt wird, werden die Ergebnisse des Aufrufs zurückgegeben.

SOAP-Programmierer können über die folgende Adresse auf die WSDL zugreifen:

http://<Server>[:port]/axis2/services/UcmdbService?wsdl

Die Portspezifikation ist nur für nicht standardmäßige Installationen erforderlich. Bitte erfragen Sie die korrekte Portnummer bei Ihrem Systemadministrator.

Der URL zum Aufrufen des Service lautet wie folgt:

http://<Server>[:port]/axis2/services/UcmdbService

Beispiele zum Herstellen einer Verbindung zur CMDB finden Sie unter "Anwendungsfälle" auf Seite 268.

Abfragen der CMDB

Die CMDB wird mithilfe der unter "UCMDB-Abfragemethoden" auf Seite 251 beschriebenen APIs abgefragt.

Die Abfragen und die zurückgegebenen CMDB-Elemente enthalten grundsätzlich reale UMDB -IDs.

Beispiele für die Verwendung der Abfragemethoden finden Sie unter "Abfragebeispiel" auf Seite 271.

Dieser Abschnitt umfasst die folgenden Themen:

Just In Time-Antwortberechnung

Für alle Abfragemethoden berechnet der UMDB-Server die von der Abfragemethode angeforderten Werte, wenn die Anforderung eingeht, und gibt die Ergebnisse auf Basis der letzten Daten zurück. Das Ergebnis wird immer zum Zeitpunkt des Eingangs der Anforderung berechnet. Dies gilt auch dann, wenn die TQL-Abfrage aktiv ist und ein zuvor berechnetes Ergebnis existiert. Die Ergebnisse einer Abfrage, die an die Client-Applikation zurückgegeben werden, können daher von den Ergebnissen der gleichen Abfrage abweichen, die auf der Benutzeroberfläche angezeigt wird.

Tipp: Wenn Ihre Applikation die Ergebnisse einer bestimmten Abfrage mehrmals verwendet und zwischen den einzelnen Verwendungen der Ergebnisdaten keine wesentlichen Datenänderungen zu erwarten sind, können Sie die Daten in der Client-Applikation speichern anstatt eine neue Abfrage auszuführen und auf diese Weise die Systemressourcen schonen.

Verarbeiten großer Antworten

Die Antwort auf eine Abfrage umfasst immer die Strukturen für die von der Abfragemethode angeforderten Daten, selbst wenn tatsächlich keine Daten übertragen werden. Für viele Methoden, bei denen die Daten eine Sammlung oder Karte sind, umfasst die Antwort auch die `ChunkInfo`-Struktur, die aus den Elementen `chunksKey` und `numberOfChunks` besteht. Das Feld `numberOfChunks` gibt die Anzahl der Chunks an, die abzurufende Daten enthalten.

Die maximale Datenübertragungsgröße wird vom Systemadministrator festgelegt. Wenn die von der Abfrage zurückgegebenen Daten die maximale Größe überschreiten, enthalten die Datenstrukturen in der ersten Antwort keine aussagekräftigen Informationen und das Feld `numberOfChunks` weist den Wert `2` oder einen höheren Wert auf. Wenn die Daten das Maximum nicht überschreiten, weist das Feld `numberOfChunks` den Wert `0` (Null) auf und die Daten werden in der ersten Antwort übertragen. Prüfen Sie deshalb beim Verarbeiten einer Antwort zunächst den

Wert des Felds `numberOfChunks`. Wenn dieser größer ist als `1`, verwerfen Sie die Daten in der Übertragung und fordern Sie Daten-Chunks an. Andernfalls verwenden Sie die Daten in der Antwort.

Weitere Informationen zum Verarbeiten von Daten-Chunks finden Sie im Abschnitt "pullTopologyMapChunks" auf Seite 259 und im Abschnitt "releaseChunks" auf Seite 261.

Festlegen der zurückzugebenden Eigenschaften

CIs und Beziehungen haben im Allgemeinen viele Eigenschaften. Einige Methoden, die Sammlungen oder Diagramme dieser Elemente zurückgeben, akzeptieren Eingabeparameter, die angeben, welche Eigenschaftswerte mit jedem Element zurückgegeben werden sollen, das mit der Abfrage übereinstimmt. Die CMDB gibt keine leeren Eigenschaften zurück. Die Antwort auf eine Abfrage kann daher weniger Eigenschaften enthalten als in der Abfrage angefordert wurden.

Dieser Abschnitt beschreibt die zum Festlegen der zurückzugebenden Eigenschaften verwendeten Arten von Eigenschaftensätzen.

Es gibt zwei Möglichkeiten, um auf Eigenschaften zu verweisen:

- Anhand ihrer Namen
- Anhand der Namen von vordefinierten Eigenschaftsregeln. Vordefinierte Eigenschaftsregeln werden von der CMDB verwendet, um eine Liste der realen Eigenschaftsnamen zu erstellen.

Wenn eine Applikation anhand der Namen auf Eigenschaften verweist, übergibt sie ein Element des Typs `PropertiesList`.

> **Tipp:** Verwenden Sie nach Möglichkeit das Element `PropertiesList` anstelle eines regelbasierten Satzes, um die Namen der gewünschten Eigenschaften anzugeben. Bei Verwendung vordefinierter Eigenschaftsregeln werden fast immer mehr Eigenschaften zurückgegeben als benötigt werden, was auf Kosten der Leistung geht.

Es gibt zwei Arten von vordefinierten Eigenschaften: Qualifizier-Eigenschaften und einfache Eigenschaften.

- **Qualifizierer-Eigenschaften.** Verwenden Sie diese Eigenschaften, wenn die Client-Applikation ein `QualifierProperties`-Element übergeben soll (eine Liste von Qualifizierern, die auf Eigenschaften angewendet werden können). Die CMDB konvertiert die von der Client-Applikation übergebene Qualifizierer-Liste in die Liste der Eigenschaften, auf die mindestens ein Qualifizierer zutrifft. Die Werte dieser Eigenschaften werden mit dem Element `CI` oder `Relation` zurückgegeben.
- **Einfache Eigenschaften.** Um einfache regelbasierte Eigenschaften zu verwenden, übergibt die Client-Applikation ein Element des Typs `SimplePredefinedProperty` oder `SimpleTypedPredefinedProperty`. Diese Elemente enthalten den Namen der Regel, nach der die CMDB die Liste der zurückzugebenden Eigenschaften generiert. Folgende Regeln können in einem Element des Typs `SimplePredefinedProperty` oder `SimpleTypedPredefinedProperty` angegeben werden: `CONCRETE` (KONKRET), `DERIVED` (ABGELEITET) und `NAMING` (NAME).

Konkrete Eigenschaften

Konkrete Eigenschaften sind der Satz von Eigenschaften, die für den angegebenen CIT definiert wurden. Die von abgeleiteten Klassen hinzugefügten Eigenschaften werden für die Instanzen

dieser abgeleiteten Klassen nicht zurückgegeben.

Eine Sammlung von Instanzen, die von einer Methode zurückgegeben werden, kann Instanzen eines im Methodenaufruf angegebenen CIT und Instanzen der von diesem CIT erbenden CITs enthalten. Die abgeleiteten CITs erben die Eigenschaften des angegebenen CIT. Außerdem erweitern die abgeleiteten CITs den übergeordneten CIT, indem sie weitere Eigenschaften hinzufügen.

Beispiel für konkrete Eigenschaften:

Der CIT `T1` weist die Eigenschaften `P1` und `P2` auf. Der CIT `T11` erbt von `T1` und erweitert `T1` um die Eigenschaften `P21` und `P22`.

Die Sammlung der CIs des Typs `T1` enthält die Instanzen von `T1` und `T11`. Die konkreten Eigenschaften aller Instanzen in dieser Sammlung sind `P1` und `P2`.

Abgeleitete Eigenschaften

Abgeleitete Eigenschaften sind der Satz von Eigenschaften, die für den angegebenen CIT definiert wurden, sowie für jeden abgeleiteten CIT die vom abgeleiteten CIT hinzugefügten Eigenschaften.

Beispiel für abgeleitete Eigenschaften:

Wenn wir auf das Beispiel für die konkreten Eigenschaften zurückgreifen, sind `P1` und `P2` die abgeleiteten Eigenschaften der Instanzen von `T1`. Die abgeleiteten Eigenschaften der Instanzen von `T11` sind `P1`, `P2`, `P21` und `P22`.

Namenseigenschaften

Die Namenseigenschaften sind `display_label` und `data_name`.

Andere Elemente für die Spezifikation von Eigenschaften

- **PredefinedProperties**

  `PredefinedProperties` können ein Element des Typs `QualifierProperties` und ein Element des Typs `SimplePredefinedProperty` für jede der anderen möglichen Regeln enthalten. In einem `PredefinedProperties`-Satz sind nicht zwangsläufig alle Listentypen enthalten.

- **PredefinedTypedProperties**

  Das Element `PredefinedTypedProperties` wird verwendet, um auf jeden CIT einen anderen Eigenschaftensatz anzuwenden. `PredefinedTypedProperties` können ein Element des Typs `QualifierProperties` und ein Element des Typs `SimpleTypedPredefinedProperty` für jede der anderen anwendbaren Regeln enthalten. Da das Element `PredefinedTypedProperties` auf jeden CIT einzeln angewendet wird, sind abgeleitete Eigenschaften nicht relevant. In einem `PredefinedProperties`-Satz sind nicht zwangsläufig alle anwendbaren Listentypen enthalten.

- **CustomProperties**

  `CustomProperties` können jede beliebige Kombination aus dem grundlegenden `PropertiesList`-Element und den regelbasierten Eigenschaftenlisten enthalten. Der Eigenschaftenfilter ist die Gesamtmenge aller von allen Listen zurückgegebenen Eigenschaften.

- **CustomTypedProperties**

  `CustomTypedProperties` können jede beliebige Kombination aus dem grundlegenden `PropertiesList`-Element und den anwendbaren regelbasierten Eigenschaftenlisten enthalten. Der Eigenschaftenfilter ist die Gesamtmenge aller von allen Listen zurückgegebenen Eigenschaften.

- **TypedProperties**

  Das Element `TypedProperties` wird verwendet, um für jeden CIT einen anderen Eigenschaftensatz zu übergeben. `TypedProperties` sind eine Sammlung von Paaren, die sich aus den Typnamen und den Eigenschaftensätzen aller Typen zusammensetzen. Jeder Eigenschaftensatz wird nur auf den entsprechenden Typ angewendet.

Aktualisieren von UCMDB

Sie aktualisieren die CMDB mit den APIs für die Aktualisierung. Weitere Informationen zu den API-Methoden finden Sie unter "UCMDB-Aktualisierungsmethoden" auf Seite 262. Beispiele für die Verwendung der Aktualisierungsmethoden finden Sie unter "Aktualisierungsbeispiel" auf Seite 282.

Diese Aufgabe umfasst folgende Schritte:

- "Aktualisieren von UCMDB" unten
- "Verwendung von ID-Typen mit Aktualisierungsmethoden" oben

Parameter für die UCMDB-Aktualisierung

Dieses Thema beschreibt die Parameter, die nur von den Aktualisierungsmethoden des Service verwendet werden. Weitere Informationen finden Sie in der schema documentation.

- **CIsAndRelationsUpdates**

  Der Typ `CIsAndRelationsUpdates` besteht aus den Elementen `CIsForUpdate`, `relationsForUpdate`, `referencedRelations` und `referencedCIs`. Eine `CIsAndRelationsUpdates`-Instanz muss nicht zwangsläufig alle drei Elemente enthalten.

  `CIsForUpdate` ist eine `CIs`-Sammlung. `relationsForUpdate` ist eine `Relations`-Sammlung. Die `CI`- und `relation`-Elemente in den Sammlungen weisen ein `props`-Element auf. Beim Erstellen eines CI oder einer Beziehung müssen Eigenschaften, die das Attribut `required` oder das Attribut `key` in der CIT-Definition enthalten, mit Werten aufgefüllt werden. Die Elemente in diesen Sammlungen werden von der Methode aktualisiert oder erstellt.

  `referencedCIs` und `referencedRelations` sind Sammlungen von CIs, die bereits in der CMDB definiert wurden. Die Elemente in der Sammlung werden mit einer temporären ID in Kombination mit allen Schlüsseleigenschaften identifiziert. Diese Elemente werden verwendet, um die Identitäten der CIs und Beziehungen für die Aktualisierung aufzulösen. Sie werden grundsätzlich nicht durch die Methode erstellt oder aktualisiert.

  Jedes `CI`- und `relation`-Element in diesen Sammlungen weist eine Eigenschaftensammlung auf. Neue Elemente werden mit den Eigenschaftswerten in diesen Sammlungen erstellt.

Verwendung von ID-Typen mit Aktualisierungsmethoden

Im Folgenden werden ID-CITs, CIs und Beziehungen beschrieben. Wenn es sich bei der ID nicht um eine reale CMDB -ID handelt, sind die Attribute **type** und **key** erforderlich.

- **Löschen oder Aktualisieren von Konfigurationselementen**

  Beim Aufrufen einer Methode zum Löschen oder Aktualisieren eines Elements kann vom Client eine temporäre oder leere ID verwendet werden. In diesem Fall müssen der CI-Typ und die "Schlüsselattribute" zur Identifizierung des CI festgelegt werden.

- **Löschen oder Aktualisieren von Beziehungen**

  Wenn Sie Beziehungen löschen oder aktualisieren, kann die Beziehungs-ID leer, temporär oder real sein.

  Wenn die ID eines CI temporär ist, muss das CI in der Sammlung `referencedCIs` übergeben werden. Außerdem müssen seine Schlüsselattribute angegeben werden. Weitere Informationen finden Sie unter referencedCIs in "CIsAndRelationsUpdates" auf der vorherigen Seite.

- **Einfügen neuer Konfigurationselemente indie CMDB**

  Zum Einfügen eines neuen CI können Sie entweder eine leere oder eine temporäre ID verwenden. Wenn die ID leer ist, kann der Server jedoch nicht die reale CMDB-ID in der Struktur `createIDsMap` zurückgeben, da das Element `clientIDFEHLT`. Weitere Informationen finden Sie unter "addCIsAndRelations" auf Seite 262 und "UCMDB-Abfragemethoden" auf Seite 251.

- **Einfügen neuer Beziehungen indie CMDB**

  Die Beziehungs-ID kann entweder temporär oder leer sein. Wenn die Beziehung jedoch neu ist und die Konfigurationselemente an einem Ende der Beziehung bereits in der CMDB definiert wurden, dann müssen die bereits vorhandenen CIs durch eine reale CMDB-ID identifiziert oder in einer Sammlung des Typs `referencedCIs` angegeben werden.

Abfragen des UCMDB-Klassenmodells

Die Klassenmodellmethoden geben Informationen über CITs und Beziehungen zurück. Das Klassenmodell wird mit CIT Manager konfiguriert. Weitere Informationen finden Sie unter "CIT Manager" im *HP Universal CMDB – Modellierungshandbuch*.

Beispiele für die Verwendung der Klassenmodellmethoden finden Sie unter "Beispiel für das Klassenmodell" auf Seite 287.

Dieser Abschnitt enthält Informationen über die folgenden Methoden, die Informationen über CITs und Beziehungen zurückgeben:

- "getClassAncestors" oben
- "getAllClassesHierarchy" auf der nächsten Seite
- "getCmdbClassDefinition" auf der nächsten Seite

getClassAncestors

Die `getClassAncestors`-Methode ruft den Pfad zwischen dem angegebenen CIT und seinem Stamm, einschließlich des Stamms, ab.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf der nächsten Seite.
className	Der Typname. Weitere Informationen finden Sie unter "Typname" auf Seite 249.

Ausgabe

Parameter	Kommentar
classHierarchy	Eine Sammlung von Paaren bestehend aus Klassenname und übergeordnetem Klassennamen.
comments	Nur für den internen Gebrauch.

getAllClassesHierarchy

Die `getAllClassesHierarchy`-Methode ruft die gesamte Klassenmodellstruktur ab.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf der nächsten Seite.

Ausgabe

Parameter	Kommentar
classesHierarchy	Eine Sammlung von Paaren bestehend aus Klassenname und übergeordnetem Klassennamen.
comments	Nur für den internen Gebrauch.

getCmdbClassDefinition

Die `getCmdbClassDefinition`-Methode ruft Informationen über die angegebene Klasse ab.

Wenn Sie die `getCmdbClassDefinition`-Methode zum Abrufen der Schlüsselattribute verwenden, müssen Sie auch die übergeordneten Klassen bis zur Basisklasse abrufen. `getCmdbClassDefinition` identifiziert als Schlüsselattribute nur solche Attribute, die in der durch das Element `className` angegebenen Klassendefinition mit dem Qualifizierer ID_ ATTRIBUTE gekennzeichnet sind. Geerbte Schlüsselattribute werden nicht als Schlüsselattribute der angegebenen Klasse erkannt. Daher ist die vollständige Liste der Schlüsselattribute für die angegebene Klasse die Gesamtmenge aller Schlüssel der Klasse sowie ihrer übergeordneten Klassen bis zum Stamm.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" oben.
className	Der Typname. Weitere Informationen finden Sie unter "Allgemeine UCMDB-Parameter " oben.

Ausgabe

Parameter	Kommentar
cmdbClass	Die Klassendefinition, bestehend aus den Elementen `name`, `classType`, `displayLabel`, `description` und `parentName` sowie Qualifizierern und Attributen.
comments	Nur für den internen Gebrauch.

Abfrage für Auswirkungsanalysen

Die `ID` in den Methoden zur Auswirkungsanalyse verweist auf die Antwortdaten des Service. Sie ist für die aktuelle Antwort eindeutig und wird nach 10-minütigem Nichtgebrauch aus dem Zwischenspeicher des Servers gelöscht.

Beispiele für die Verwendung der Methoden zur Auswirkungsanalyse finden Sie unter "Beispiel für die Auswirkungsanalyse" auf Seite 288.

Allgemeine UCMDB-Parameter

In diesem Abschnitt werden die gebräuchlichsten Parameter der Servicemethoden beschrieben. Weitere Informationen finden Sie in der Schemadokumentation.

Dieser Abschnitt umfasst die folgenden Themen:

CmdbContext

Alle Serviceaufrufe der UCMDB-Webservice-API erfordern ein `CmdbContext`-Argument. `CmdbContext` ist eine `callerApplication`-Zeichenkette, die die Applikation identifiziert, die den Service aufruft. `CmdbContext` wird für die Protokollierung und Fehlersuche verwendet.

ID

Jedes `CI` und jede Beziehung hat ein `ID`-Feld. Es besteht aus einer ID-Zeichenkette, die zwischen Groß- und Kleinschreibung unterscheidet, und einem optionalen `temp`-Kennzeichen, das angibt, ob es sich bei der ID um eine temporäre ID handelt.

Schlüsselattribute

In einigen Kontexten können zur Identifizierung eines CI oder einer Beziehung Schlüsselattribute anstelle einer CMDB-ID verwendet werden. Schlüsselattribute sind in der Klassendefinition mit dem Qualifizierer `ID_ATTRIBUTE` gekennzeichnet.

Auf der Benutzeroberfläche werden die Schlüsselattribute in der Liste der CIT-Attribute mit einem Schlüsselsymbol angezeigt. Weitere Informationen finden Sie unter "Dialogfeld "Attribut hinzufügen/bearbeiten"" im HP Universal CMDB – Modellierungshandbuch. Weitere Informationen darüber, wie die Schlüsselattribute von der API-Client-Applikation aus identifiziert werden können, finden Sie unter "getCmdbClassDefinition" auf Seite 246.

ID-Typen

Ein `ID`-Element kann eine reale oder temporäre ID enthalten.

Eine reale ID ist eine von der CMDB zugewiesene Zeichenkette, die eine Entität in der Datenbank identifiziert. Eine temporäre ID kann jede beliebige Zeichenkette sein, die in der aktuellen Anforderung eindeutig ist.

Eine temporäre ID kann vom Client zugewiesen werden und entspricht häufig der ID des CI so wie sie vom Client gespeichert wurde. Sie stellt nicht zwangsläufig eine bereits in der CMDB erstellte Entität dar. Wenn der Client eine temporäre ID übergibt und die CMDB ein vorhandenes Datenkonfigurationselement anhand der Schlüsseleigenschaften des CI identifizieren kann, wird das CI als für den Kontext geeignet verwendet, obwohl es mit einer realen ID identifiziert wurde.

CIProperties

Ein `CIProperties`-Element besteht aus mehreren Sammlungen, wobei jede Sammlung eine Folge von Name-Wert-Elementen enthält, die die Eigenschaften des vom Namen der Sammlung angegebenen Typs spezifizieren. Keine der Sammlungen ist erforderlich. Daher kann das Element `CIProperties` jede beliebige Kombination von Sammlungen enthalten.

`CIProperties` werden von den Elementen `CI` und `Relation` verwendet. Weitere Informationen finden Sie unter "Konfigurationselement (CI)" auf der nächsten Seite und "Relation" auf der nächsten Seite.

Es gibt folgende Eigenschaftensammlungen:

- `dateProps` – Sammlung von `DateProp`-Elementen
- `doubleProps` – Sammlung von `DoubleProp`-Elementen
- `floatProps` – Sammlung von `FloatProp`-Elementen
- `intListProps` – Sammlung von `intListProp`-Elementen
- `intProps` – Sammlung von `IntProp`-Elementen
- `strProps` – Sammlung von `StrProp`-Elementen
- `strListProps` – Sammlung von `StrListProp`-Elementen

- `longProps` – Sammlung von `LongProp`-Elementen
- `bytesProps` – Sammlung von `BytesProp`-Elementen
- `xmlProps` – Sammlung von `XmlProp`-Elementen

Typname

Der Typname ist der Klassenname eines CI- oder Beziehungstyps. Der Typname wird im Code verwendet, um auf die Klasse zu verweisen. Er sollte nicht mit dem Anzeigenamen verwechselt werden, der auf der Benutzeroberfläche angezeigt wird, wo die Klasse erwähnt wird, aber im Code keine Bedeutung hat.

Konfigurationselement (CI)

Ein `CI`-Element besteht aus einer `ID`, einem `Typ` und einer `props`-Sammlung.

Bei Verwendung der "UCMDB-Aktualisierungsmethoden" zur Aktualisierung eines `CI` kann das `ID`-Element eine reale CMDB-ID oder eine vom Client zugeordnete temporäre ID enthalten. Wenn eine temporäre ID verwendet wird, muss das `temp`-Kennzeichen auf **true** gesetzt werden. Beim Löschen eines Konfigurationselements kann die `ID` leer sein. Die "UCMDB-Abfragemethoden" verwenden reale `ID`s als Eingabeparameter und geben in den Abfrageergebnissen reale `ID`s zurück.

Der `Typ` kann eine beliebiger, in CIT Manager definierter Typname sein. Weitere Informationen finden Sie unter "CIT Manager" im HP Universal CMDB – Modellierungshandbuch.

Das `props`-Element ist eine CIProperties-Sammlung. Weitere Informationen finden Sie unter "Allgemeine UCMDB-Parameter " auf Seite 247.

Relation

Ein `Relation`-Element ist eine Entität, die zwei Konfigurationselemente miteinander verknüpft. Es besteht aus einer `ID`, einem `Typ`, den IDs der beiden verknüpften Konfigurationselemente `(end1ID` und `end2ID)` sowie einer `props` -Sammlung.

Bei Verwendung der "UCMDB-Aktualisierungsmethoden" zur Aktualisierung eines `Relation`-Elements kann der Wert der zugehörigen `ID` eine reale CMDB-ID oder eine temporäre ID sein. Beim Löschen eines Konfigurationselements kann die `ID` leer sein. Die "UCMDB-Abfragemethoden" verwenden reale `ID`s als Eingabeparameter und geben in den Abfrageergebnissen reale `ID`s zurück.

Der Beziehungstyp ist das `Type Name`-Element der UCMDB-Klasse, von der die Beziehung instanziiert wird. Der Typ kann ein beliebiger, in der CMDB definierter Beziehungstyp sein. Weitere Informationen zu Klassen oder Typen finden Sie unter "Abfragen des UCMDB-Klassenmodells " auf Seite 245.

Weitere Informationen finden Sie unter "CIT Manager" im *HP Universal CMDB – Modellierungshandbuch*.

Die IDs der beiden Endpunkte der Beziehung dürfen keine leeren IDs sein, da sie zum Erstellen der ID der aktuellen Beziehung verwendet werden. Sie können jedoch temporäre IDs aufweisen, die ihnen vom Client zugewiesen wurden.

Das `props`-Element ist eine CIProperties-Sammlung. Weitere Informationen finden Sie unter "CIProperties" auf der vorherigen Seite.

UCMDB-Ausgabeparameter

In diesem Abschnitt werden die gebräuchlichsten Ausgabeparameter der Servicemethoden beschrieben. Weitere Informationen finden Sie in der schema documentation..

Dieser Abschnitt umfasst die folgenden Themen:

CIs

`CIs` sind eine Sammlung von CI-Elementen.

ShallowRelation

Ein `ShallowRelation`-Element ist eine Entität, die zwei Konfigurationselemente miteinander verknüpft. Es besteht aus einer `ID`, einem `Typ` und den IDs der beiden verknüpften Konfigurationselemente `(end1ID` und `end2ID)`. Der Beziehungstyp ist das `Typ Name`-Element der CMDB-Klasse, von der die Beziehung instanziiert wird. Der Typ kann ein beliebiger, in der CMDB definierter Beziehungstyp sein.

Topology

`Topology` ist ein aus `CI`-Elementen und Beziehungen bestehendes Diagramm. Ein `Topology`-Element besteht aus einer `CIs`-Sammlung und einer `Relations`-Sammlung, die ein oder mehrere `Relation`-Elemente enthält.

CINode

`CINode` besteht aus einer `CIs`-Sammlung mit einem `Label`. Das `Label` in `CINode` entspricht dem Label, das im Knoten der in der Abfrage verwendeten TQL definiert ist.

RelationNode

`RelationNode` ist ein Satz von `Relations`-Sammlungen mit einem `Label`. Das `Label` in `RelationNode` entspricht dem Label, das im Knoten der in der Abfrage verwendeten TQL definiert ist.

TopologyMap

`TopologyMap` ist die Ausgabe einer Abfrageberechnung, die mit einer TQL-Abfrage übereinstimmt. Die `Label` in `TopologyMap` entsprechen den Knoten-Labeln, die in der in der Abfrage verwendeten TQL definiert sind.

Die Daten von `TopologyMap` werden in folgender Form zurückgegeben:

- `CINodes`. Hierbei handelt es sich um ein oder mehrere Elemente des Typs `CINode` (siehe "CINode" auf der vorherigen Seite).
- `relationNodes`. Hierbei handelt es sich um ein oder mehrere Elemente des Typs `RelationNode` (siehe "RelationNode" auf der vorherigen Seite).

Die `Label` in diesen beiden Strukturen sortieren die Listen der Konfigurationselemente und Beziehungen.

ChunkInfo

Wenn eine Abfrage eine große Datenmenge zurückgibt, speichert der Server die Daten in einzelnen Segmenten, sogenannten Chunks. Die Informationen, die der Client zum Abrufen von Daten-Chunks verwendet, befinden sich in der `ChunkInfo`-Struktur, die von der Abfrage zurückgegeben wird. `ChunkInfo` besteht aus dem Element `numberOfChunks`, das abgerufen werden muss, und dem Element `chunksKey`. Das Element `chunksKey` ist eine eindeutige ID der Daten auf dem Server für diesen speziellen Abfrageaufruf.

Weitere Informationen finden Sie unter "Verarbeiten großer Antworten" auf Seite 241.

UCMDB-Abfragemethoden

Dieser Abschnitt enthält Informationen zu den folgenden Methoden:

executeTopologyQueryByNameWithParameters

Die `executeTopologyQueryByNameWithParameters`-Methode ruft ein `topologyMap`-Element ab, das mit der angegebenen parametrisierten Abfrage übereinstimmt.

Die Werte für die Abfrageparameter werden im Argument `parameterizedNodes` übergeben. In der angegebenen TQL müssen für jedes `CINode`-Element und jedes `relationNode`-Element eindeutige Label definiert sein. Andernfalls schlägt der Methodenaufruf fehl.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
queryName	Der Name der parametrisierten TQL in der CMDB, für die die Karte abgerufen werden soll.
parameterizedNodes	Die Bedingungen, die jeder Knoten erfüllen muss, um in die Abfrageergebnisse einbezogen zu werden.
queryTypedProperties	Eine Sammlung von Eigenschaftensätzen, um Elemente eines bestimmten CI-Typs abzurufen.

Ausgabe

Parameter	Kommentar
topologyMap	Weitere Informationen finden Sie unter "TopologyMap" auf Seite 250.
chunkInfo	Weitere Informationen finden Sie unter: "ChunkInfo" auf der vorherigen Seite und "Verarbeiten großer Antworten" auf Seite 241.

executeTopologyQueryWithParameters

Die `executeTopologyQueryWithParameters`-Methode ruft ein `topologyMap`-Element ab, das mit der parametrisierten Abfrage übereinstimmt.

Die Abfrage wird im Argument `queryXML` übergeben. Die Werte für die Abfrageparameter werden im Argument `parameterizedNodes` übergeben. In der TQL müssen für jedes `CINode`-Element und jedes `relationNode`-Element eindeutige Label definiert sein.

Die `executeTopologyQueryWithParameters`-Methode wird eher für die Übergabe von Ad-hoc-Abfragen verwendet als für den Zugriff auf eine in der CMDB definierte Abfrage. Sie können diese Methode zum Definieren einer Abfrage verwenden, wenn Sie keinen Zugriff auf die UCMDB-Benutzeroberfläche haben oder wenn Sie die Abfrage nicht in der Datenbank speichern möchten.

Führen Sie folgende Schritte aus, um eine exportierte TQL als Eingabe für diese Methode zu verwenden:

1. Starten Sie den Webbrowser und geben Sie die folgende Adresse ein: **http://localhost:8080/jmx-console**.

   Eventuell müssen Sie sich mit einem Benutzernamen und einem Kennwort anmelden. Die Standardeinstellung lautet **sysadmin**/**sysadmin**.

2. Klicken Sie auf **UCMDB:service=TQL Services**.
3. Suchen Sie die Operation **exportTql**.
   - Geben Sie für den Parameter **customerId** den Wert **1** (Standardwert) ein.
   - Geben Sie im Feld **patternName** einen gültigen TQL-Namen ein.
4. Klicken Sie auf **Invoke**.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
queryXML	Eine XML-Zeichenkette, die eine TQL ohne Ressourcen-Tags darstellt.
parameterizedNodes	Die Bedingungen, die jeder Knoten erfüllen muss, um in die Abfrageergebnisse einbezogen zu werden.

Ausgabe

Parameter	Kommentar
topologyMap	Weitere Informationen finden Sie unter "TopologyMap" auf Seite 250.
chunkInfo	Weitere Informationen finden Sie unter "ChunkInfo" auf Seite 251 und "Verarbeiten großer Antworten" auf Seite 241.

getChangedCIs

Die `getChangedCIs`-Methode gibt die Änderungsdaten für alle CIs zurück, die sich auf die angegebenen CIs beziehen.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
ids	Die Liste der IDs der Stamm-CIs, deren zugehörige CIs auf Änderungen geprüft werden. In dieser Sammlung sind nur reale CMDB-IDs gültig. .
fromDate	Der Beginn des Zeitraums, der auf geänderte CIs geprüft werden soll.
toDate	Das Ende des Zeitraums, der auf geänderte CIs geprüft werden soll.

Ausgabe

Parameter	Kommentar
changeDataInfo	Null oder mehr Sammlungen mit `ChangedDataInfo`-Elementen.

getCINeighbours

Die `getCINeighbours`-Methode gibt die unmittelbaren Nachbarn des angegebenen CI zurück.

Wenn sich die Abfrage beispielsweise auf die Nachbarn von CI `A` bezieht und CI `A` CI `B` enthält, welches CI `C` verwendet, wird zwar CI `B`, aber nicht CI `C` zurückgegeben. Das heißt, es werden nur die Nachbarn des angegebenen Typs zurückgegeben.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
ID	Die ID des CI, dessen Nachbarn abgerufen werden sollen. Es muss sich hierbei um eine reale CMDB-ID handeln.
neighbourType	Der CIT-Name der abzurufenden Nachbarn. Es werden die Nachbarn des angegebenen Typs und der von dem Typ abgeleiteten Typen zurückgegeben. Weitere Informationen finden Sie unter "Typname" auf Seite 249.
CIProperties	Die zu jedem Konfigurationselement zurückzugebenden Daten (das sogenannte Abfragelayout in der Benutzeroberfläche). Weitere Informationen finden Sie unter "TypedProperties" auf Seite 244.
relationProperties	Die zu jeder Beziehung zurückzugebenden Daten (das sogenannte Abfragelayout in der Benutzeroberfläche). Weitere Informationen finden Sie unter "TypedProperties" auf Seite 244

Ausgabe

Parameter	Kommentar
Topologie	Weitere Informationen finden Sie unter "Topology" auf Seite 250.
comments	Nur für den internen Gebrauch.

getCIsByID

Die `getCIsByID`-Methode ruft Konfigurationselemente nach ihren CMDB-IDs ab.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
CIsTypedProperties	Eine Sammlung von typisierten Eigenschaften. Weitere Informationen finden Sie unter "Andere Elemente für die Spezifikation von Eigenschaften" auf Seite 243.
IDs	In dieser Sammlung sind nur reale CMDB-IDs gültig. .

Ausgabe

Parameter	Kommentar
CIs	Sammlung von CI-Elementen.
chunkInfo	Weitere Informationen finden Sie unter: "ChunkInfo" auf Seite 251 und "Verarbeiten großer Antworten" auf Seite 241.

getClsByType

Die `getCIsByType`-Methode gibt die Sammlung der Konfigurationselemente des angegebenen Typs sowie aller von dem angegebenen Typ erbenden Typen zurück.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
type	Der Klassenname. Weitere Informationen finden Sie unter "Typname" auf Seite 249.
properties	Die zu jedem Konfigurationselement zurückzugebenden Daten. Weitere Informationen finden Sie unter "CustomProperties" auf Seite 243.

Ausgabe

Parameter	Kommentar
CIs	Sammlung von CI-Elementen.
chunkInfo	Weitere Informationen finden Sie unter: "ChunkInfo" auf Seite 251 und "Verarbeiten großer Antworten" auf Seite 241.

getFilteredClsByType

Die `getFilteredCIsByType`-Methode ruft die CIs des angegebenen Typs ab, die die von der Methode verwendeten Bedingungen erfüllen. Eine Bedingung besteht aus folgenden Elementen:

- Einem Namensfeld mit dem Namen einer Eigenschaft
- Einem Operatorfeld mit einem Vergleichsoperator
- Einem optionalen Feld mit einem Wert oder einer Liste von Werten

Zusammen bilden Sie einen booleschen Ausdruck:
`<`*`Konfigurationselement`*`>.Eigenschaft.value [Operator] <`*`Bedingung`*`>.value`

Beispiel: Für den Bedingungsnamen `root_actualdeletionperiod`, den Bedingungswert `40` und den Operator `Gleich` lautet die boolesche Anweisung wie folgt:

`<Konfigurationselement>.`*`root_actualdeletionperiod`*`.value = = 40`

Vorausgesetzt, dass keine weiteren Bedingungen erfüllt werden müssen, gibt die Abfrage alle Konfigurationselemente zurück, deren `root_actualdeletionperiod` den Wert `40` aufweist.

Wenn das `conditionsLogicalOperator`-Argument `AND` lautet, gibt die Abfrage die Konfigurationselemente zurück, die alle Bedingungen in der `Bedingung`ssammlung erfüllen. Wenn das `conditionsLogicalOperator`-Argument `OR` lautet, gibt die Abfrage die Konfigurationselemente zurück, die mindestens eine der Bedingungen in der `Bedingung`ssammlung erfüllen.

Die Vergleichsoperatoren sind in der folgenden Tabelle aufgeführt:

Operator	Bedingungstyp / Anmerkungen
Geändert bei	Datum Dies ist eine Bereichsprüfung. Der Bedingungswert wird in Stunden angegeben. Wenn der Wert der Datumseigenschaft plus/minus dem Bedingungswert innerhalb des Zeitbereichs liegt, in dem die Methode aufgerufen wird, trifft die Bedingung zu. Beispiel: Wenn der Bedingungswert `24` beträgt, trifft die Bedingung zu, wenn der Wert der Datumseigenschaft zwischen gestern um diese Zeit und morgen um diese Zeit liegt. **Hinweis:** Der Name `Geändert bei` wird aus Gründen der Abwärtskompatibilität beibehalten. In früheren Versionen wurde der Operator nur bei den Eigenschaften zum Erstellen und Ändern von Zeitangaben verwendet.
Gleich	Zeichenkette und numerisch
Gleich (ohne Groß-/Kleinschr.)	Zeichenkette
Größer als	Numerisch
Größer als oder gleich	Numerisch
In	Zeichenkette, numerisch und Liste Der Bedingungswert ist eine Liste. Die Bedingung trifft zu, wenn der Eigenschaftswert einem der Werte in der Liste entspricht.
In Liste	Liste Der Bedingungswert und der Eigenschaftswert sind Listen. Die Bedingung trifft zu, wenn alle Werte in der Bedingungsliste auch in der Eigenschaftsliste des Elements vorhanden sind. Die Bedingung trifft auch zu, wenn es mehr Eigenschaftswerte gibt als in der Bedingung angegeben sind.
Ist Null	Zeichenkette, numerisch und Liste Die Eigenschaft des Elements hat keinen Wert. Wenn der Operator `Ist Null` verwendet wird, wird der Wert der Bedingung ignoriert und kann in einigen Fällen gleich null sein.
Kleiner als	Numerisch
Kleiner als oder gleich	Numerisch
Wie	Zeichenkette Der Bedingungswert ist eine Unterzeichenfolge des Eigenschaftswertes. Er muss mit Prozentzeichen in Klammern gesetzt werden (`%`). Beispiel: `%Bi%` stimmt mit

Operator	Bedingungstyp / Anmerkungen
	`Bismark` und `Bay of Biscay`, aber nicht mit `biscuit` überein.
Wie (ohne Groß-/Kleinschr.)	Zeichenkette Der Operator `Wie (ohne Groß-/Kleinschr.)` wird genauso verwendet wie der Operator `Wie`. Die Groß-/Kleinschreibung wird bei der Übereinstimmung jedoch nicht berücksichtigt. Daher stimmt `%Bi%` auch mit `biscuit` überein.
Ungleich	Zeichenkette und numerisch
Nicht geändert bei	Datum Dies ist eine Bereichsprüfung. Der Bedingungswert wird in Stunden angegeben. Wenn der Wert der Datumseigenschaft plus/minus dem Bedingungswert innerhalb des Zeitbereichs liegt, in dem die Methode aufgerufen wird, trifft die Bedingung nicht zu. Liegt er außerhalb des Bereichs, trifft die Bedingung zu. Beispiel: Wenn der Bedingungswert `24` beträgt, trifft die Bedingung zu, wenn der Wert der Datumseigenschaft vor gestern um diese Zeit oder später als morgen um diese Zeit liegt. **Hinweis:** Der Name `Nicht geändert bei` wird aus Gründen der Abwärtskompatibilität beibehalten. In früheren Versionen wurde der Operator nur bei den Eigenschaften zum Erstellen und Ändern von Zeitangaben verwendet.

Beispiel für das Festlegen einer Bedingung:

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.Equal);
```

Beispiel für das Abfragen von geerbten Eigenschaften:

Das Ziel-CI hat die Bezeichnung `sample` und verfügt über die Attribute `name` und `size`. `sampleII` erweitert das CI um die Attribute `level` und `grade`. In diesem Beispiel wird eine Abfrage für die vom CI `sample` geerbten Eigenschaften des CI `sampleII` eingerichtet, wobei die Eigenschaften mit ihren Namen angegeben werden.

```
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)
```

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
type	Der Klassenname. Weitere Informationen finden Sie unter "Typname" auf Seite 249. Dies kann ein beliebiger Typ der unter Verwendung von CIT Manager definierten Typen sein. Weitere Informationen finden Sie unter "CIT Manager" im *HP Universal CMDB – Modellierungshandbuch*.
properties	Die zu jedem CI zurückzugebenden Daten (das sogenannte Abfragelayout in der Benutzeroberfläche). Weitere Informationen finden Sie unter "CustomProperties" auf Seite 243.
conditions	Eine Sammlung von Name-Wert-Paaren und ihren Operatoren. Beispiel: `Host_Hostname wie QA`.
conditionsLogicalOperator	• **AND**. Alle Bedingungen müssen erfüllt werden. • **OR**. Mindestens eine Bedingung muss erfüllt werden.

Ausgabe

Parameter	Kommentar
CIs	Sammlung von CI-Elementen.
chunkInfo	Weitere Informationen finden Sie unter "ChunkInfo" auf Seite 251 und "Verarbeiten großer Antworten" auf Seite 241.

getQueryNameOfView

Die `getQueryNameOfView`-Methode ruft den Namen der TQL ab, auf der die angegebene Ansicht basiert.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
viewName	Der Name einer Ansicht, d. h. einer Teilmenge des Klassenmodells in der CMDB

Ausgabe

Parameter	Kommentar
queryName	Der Name der TQL in der CMDB, auf der die Ansicht basiert.

getTopologyQueryExistingResultByName

Die `getTopologyQueryExistingResultByName`-Methode ruft das neueste Ergebnis der Ausführung der angegebenen TQL ab. Durch den Aufruf wird die TQL nicht ausgeführt. Wenn keine Ergebnisse von einer früheren Ausführung vorhanden sind, wird nichts zurückgegeben.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
queryName	Der Name einer TQL.
queryTypedProperties	Eine Sammlung von Eigenschaftssätzen, die für die Elemente eines bestimmten CI-Typs abgerufen werden sollen.

Ausgabe

Parameter	Kommentar
queryName	Der Name der TQL in der CMDB, auf der die Ansicht basiert.

getTopologyQueryResultCountByName

Die `getTopologyQueryResultCountByName`-Methode ruft die Anzahl der Instanzen jedes Knotens ab, der mit der angegebenen Abfrage übereinstimmt.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
queryName	Der Name einer TQL.
countInvisible	True: Die Ausgabe berücksichtigt auch CIs, die in der Abfrage als unsichtbar definiert wurden.

Ausgabe

Parameter	Kommentar
queryName	Der Name der TQL in der CMDB, auf der die Ansicht basiert.

pullTopologyMapChunks

Die `pullTopologyMapChunks`-Methode ruft einen der Chunks ab, die die Antwort auf eine Methode enthalten.

Jeder Chunk enthält ein `topologyMap`-Element, das Teil der Antwort ist. Der erste Chunk hat die Nummer `1`, sodass der Abrufschleifenzähler von 1 bis <*Antwortobjekt*>`.getChunkInfo().getNumberOfChunks()` durchläuft.

Weitere Informationen finden Sie unter "ChunkInfo" auf Seite 251 und "Abfragen der CMDB" auf Seite 241.

Die Client-Applikation muss in der Lage sein, die Teilkarten zu verarbeiten. Weitere Informationen finden Sie im folgenden Beispiel zur Verarbeitung einer CI-Sammlung sowie im Beispiel zum Zusammenführen von Chunks zu einer Karte unter "Abfragebeispiel" auf Seite 271.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
ChunkRequest	Die Anzahl der abzurufenden Chunks und der von der Abfragemethode zurückgegebene Wert des Parameters `ChunkInfo`.

Ausgabe

Parameter	Kommentar
topologyMap	Weitere Informationen finden Sie unter "TopologyMap" auf Seite 250.
comments	Nur für den internen Gebrauch.

Beispiel für das Verarbeiten von Chunks:

```
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
            }
```

```
        }
}

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

Die `releaseChunks`-Methode löscht die Chunks aus dem Speicher, die die Daten von der Abfrage enthalten.

Tipp: Der Server löscht die Daten nach zehn Minuten. Sie können diese Methode aufrufen, um die Daten unmittelbar nach dem Lesen zu löschen und so die Serverressourcen zu schonen.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
chunksKey	Die ID der Daten auf dem Server, wo der Chunk erstellt wurde. Der Schlüssel ist ein Element des Typs `ChunkInfo`.

UCMDB-Aktualisierungsmethoden

Dieser Abschnitt enthält Informationen zu den folgenden Methoden:

addCIsAndRelations

Mit der `addCIsAndRelations`-Methode können CIs und Beziehungen hinzugefügt oder aktualisiert werden.

Wenn die CIs oder Beziehungen nicht in der CMDB vorhanden sind, werden sie hinzugefügt und ihre Eigenschaften entsprechend den Inhalten des Arguments `CIsAndRelationsUpdates` festgelegt.

Wenn die CIs oder Beziehungen in der CMDB vorhanden sind und `updateExisting` den Wert **true** hat, werden sie mit den neuen Daten aktualisiert.

Wenn `updateExisting` den Wert **false** hat, kann `CIsAndRelationsUpdates` nicht auf vorhandene Konfigurationselemente oder Beziehungen verweisen. Jeder Versuch, auf vorhandene Elemente zu verweisen, wenn `updateExisting` den Wert **false** hat, führt zu einer Ausnahme.

Wenn `updateExisting` den Wert **true** hat, werden Hinzufügungs- oder Aktualisierungsvorgänge ungeachtet des Wertes von `ignoreValidation` ohne Prüfung der CIs durchgeführt.

Wenn `updateExisting` den Wert **false** und `ignoreValidation` den Wert **true** hat, wird der Hinzufügungsvorgang ohne Prüfung der CIs durchgeführt.

Wenn `updateExisting` den Wert **false** und `ignoreValidation` den Wert **false** hat, werden die CIs vor dem Hinzufügungsvorgangs geprüft.

Beziehungen werden grundsätzlich nicht geprüft.

`CreatedIDsMap` ist eine Karte oder ein Dictionary des Typs `ClientIDToCmdbID`, das die temporären Client-IDs mit den zugehörigen realen CMDB-IDs verbindet.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
updateExisting	Setzen Sie den Parameter auf *true*, um die bereits in der CMDB vorhandenen Elemente zu aktualisieren. Setzen Sie den Parameter auf *false*, um eine Ausnahme auszulösen, wenn ein Element bereits

Parameter	Kommentar
	vorhanden ist.
CIsAndRelationsUpdates	Die zu aktualisierenden oder zu erstellenden Elemente. Weitere Informationen finden Sie unter "CIsAndRelationsUpdates" auf Seite 244.
ignoreValidation	True: Vor der Aktualisierung der CMDB wird keine Prüfung durchgeführt.

Ausgabe

Parameter	Kommentar
CreatedIDsMap	Die Karte der Client-IDs und CMDB-IDs. Weitere Informationen finden Sie in der vorgenannten Beschreibung.
comments	Nur für den internen Gebrauch.

addCustomer

Die `addCustomer`-Methode fügt einen Kunden hinzu.

Eingabe

Parameter	Kommentar
CustomerID	Die numerische ID des Kunden.

deleteCIsAndRelations

Die `deleteCIsAndRelations`-Methode entfernt die angegebenen Konfigurationselemente und Beziehungen aus der CMDB.

Wenn ein CI gelöscht wird, das ein Endpunkt eines oder mehrerer `Beziehung`selemente ist, werden die `Beziehung`selemente ebenfalls gelöscht.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
CIsAndRelationsUpdates	Die zu löschenden Elemente. Weitere Informationen finden Sie unter "CIsAndRelationsUpdates" auf Seite 244

removeCustomer

Die `removeCustomer`-Methode löscht einen Kundendatensatz.

Eingabe

Parameter	Kommentar
CustomerID	Die numerische ID des Kunden.

updateCIsAndRelations

Die `updateCIsAndRelations`-Methode aktualisiert die angegebenen CIs und Beziehungen.

Für die Aktualisierung werden die Eigenschaftswerte vom Argument `CIsAndRelationsUpdates` verwendet. Für CIs oder Beziehungen, die nicht in der CMDB vorhanden sind, wird eine Ausnahme ausgelöst.

`CreatedIDsMap` ist eine Karte oder ein Dictionary des Typs `ClientIDToCmdbID`, das die temporären Client-IDs mit den zugehörigen realen CMDB-IDs verbindet.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
CIsAndRelationsUpdates	Die zu aktualisierenden Elemente. Weitere Informationen finden Sie unter "CIsAndRelationsUpdates" auf Seite 244.
ignoreValidation	True: Vor der Aktualisierung der CMDB wird keine Prüfung durchgeführt.

Ausgabe

Parameter	Kommentar
CreatedIDsMap	Die Karte der Client-IDs und CMDB-IDs. Weitere Informationen finden Sie unter "addCIsAndRelations" auf Seite 262.

UCMDB-Methoden zur Auswirkungsanalyse

Dieser Abschnitt enthält Informationen zu den folgenden Methoden:

- "calculateImpact" oben
- "getImpactPath" auf der nächsten Seite
- "getImpactRulesByNamePrefix" auf Seite 266

calculateImpact

Die `calculateImpact`-Methode berechnet, welche CIs nach den in der CMDB definierten Regeln von einem bestimmten CI betroffen sind.

Sie zeigt die Auswirkung einer Ereignisauslösung der Regel. Der Ausgabeparameter `identifier` der `calculateImpact`-Methode wird als Eingabe für "getImpactPath" oben verwendet.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
impactCategory	Der Typ des Ereignisses, das die simulierte Regel auslösen würde.
IDs	Eine Sammlung von ID-Elementen.
impactRulesNames	Eine Sammlung von Elementen des Typs `ImpactRuleName`.
severity	Der Schweregrad des auslösenden Ereignisses.

Ausgabe

Parameter	Kommentar
impactTopology	Weitere Informationen finden Sie unter "Topology" auf Seite 250.
identifier	Der Schlüssel zur Serverantwort.

getImpactPath

Die `getImpactPath`-Methode ruft das Topologiediagramm des Pfads zwischen dem betroffenen CI und dem CI, das es betrifft, ab.

Der Ausgabeparameter `identifier` der "calculateImpact" auf der vorherigen Seite-Methode wird als `identifier`-Eingabeargument der `getImpactPath`-Methode verwendet.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
identifier	Der Schlüssel zu der von der `calculateImpact`-Methode zurückgegebenen Serverantwort.
relation	Der Wert des `Relation`-Parameters basiert auf einem der "ShallowRelation"-Werte, die von der `calculateImpact`-Methode im Element `impactTopology` zurückgegeben werden.

Ausgabe

Parameter	Kommentar
impactPathTopology	Eine Sammlung von `CIs` und eine Sammlung von `ImpactRelations`.
comments	Nur für den internen Gebrauch.

Ein `ImpactRelations`-Element besteht aus einer `ID`, einem `Typ`, der `ID von Ende 1`, der `ID von Ende 2`, einer `Regel` und einer `Aktion`.

getImpactRulesByNamePrefix

Die `getImpactRulesByNamePrefix`-Methode ruft Regeln unter Verwendung eines Präfixfilters ab.

Diese Methode ist für Auswirkungsregeln vorgesehen, deren Name ein Präfix enthält, das den Kontext angibt, auf den sie angewendet werden, z. B. `SAP_MeineRegel`, `ORA_MeineRegel` usw. Die Methode filtert alle Namen von Auswirkungsregeln heraus, die mit dem vom Argument `ruleNamePrefixFilter` angegebenen Präfix beginnen.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
ruleNamePrefixFilter	Eine Zeichenkette, die die Anfangsbuchstaben der abzugleichenden Regelnamen enthält.

Ausgabe

Parameter	Kommentar
impactRules	Der Ausgabeparameter `impactRules` besteht aus null oder mehr `impactRule`-Elementen. Ein `impactRule`-Element, das die Auswirkung einer Änderung angibt, besteht aus den Unterelementen `ruleName`, `description`, `queryName` und `isActive`.

Webservice-API des Status "Tatsächlich"

Die Webservice-API des Status "Tatsächlich" wird von Service Manager hauptsächlich verwendet, um Informationen zum Status "Tatsächlich" für eine bestimmte CMDB-ID oder globale ID und eine bestimmte Kunden-ID abzurufen. Die API findet eine passende Abfrage im Ordner **Integration/SM Query**, führt die TQL mit der CMDB-ID und der globalen ID als Bedingung aus und gibt die Ausgabe der Abfrage zurück.

Webservice-URL: http://[machine_name]:8080/axis2/services/ucmdbSMService

Webservice-Schema: http://[machine_name]:8080/axis2/services/ucmdbSMService?xsd=xsd0

Flow

Wenn die API-Methode aufgerufen wird, sucht diese im Ordner **Integration/SM Query** nach einer geeigneten Abfrage. Sie versucht, den Typ der angeforderten **CMDBID/GlobalID** zuerst einer der in diesem Ordner enthaltenen Abfragen zuzuordnen. Hierzu sucht sie nach einem **QueryElement** namens **Root**. Wird keines gefunden, versucht sie, einen beliebigen **QueryNode** desselben Typs wie die angeforderte **CMDBID/GlobalID** zu verwenden. Sobald eine geeignete Abfrage und ein QueryNode gefunden sind, legt sie CMDBID/GlobalID auf dem QueryNode als eine Bedingung fest

und führt die Abfrage aus. Das Ergebnis wird anschließend an die aufrufende Anwendung (Caller) der API zurückgegeben.

Bearbeiten des Ergebnisses mit Transformationen

Mitunter empfiehlt es sich, auf die XML zusätzliche Transformationen anzuwenden (beispielsweise um die Größen aller Datenträger zu berechnen und diese Summe dem CI als ein zusätzliches Attribut hinzuzufügen). Fügen Sie zum Hinzufügen zusätzlicher Transformatoren zu den TQL-Ergebnissen eine Ressource namens **[tql_name].xslt** wie folgt der Adapterkonfiguration hinzu: **Adapterverwaltung > ServiceDeskAdapter7-1 > Konfigurationsdateien > [tql_name].xslt.**

Protokolle für die Webservice-API des Status "Tatsächlich"

Die Protokollkonfiguration für UCMDB befindet sich im folgenden Verzeichnis: **UCMDBServer/Conf/log** in den verschiedenen ***.properties**-Dateien.

So zeigen Sie die Protokolle des SM-Flows des Status "Tatsächlich" an:

1. Öffnen Sie die Datei **cmdb_soaapi.properties** und ändern Sie die Protokollebene wie folgt auf **DEBUG**: **loglevel=DEBUG**.
2. Öffnen Sie die Datei **fcmdb.properties** und ändern Sie die Protokollebene wie folgt auf **DEBUG**: **loglevel=DEBUG**.
3. Warten Sie eine Minute, bis der Server die Änderungen abgerufen hat.
4. Führen Sie den **Status "Tatsächlich"** in SM aus.
5. Rufen Sie die folgenden Protokolldateien unter **UCMDBServer/Runtime/log** auf:
   - cmdb.soaapi.log
   - fcmdb.log

Aktivieren des Status "Tatsächlich" von replizierten CIs nach Ändern des Stammkontextes

Wenn Sie den Stammkontext für den Zugriff auf UCMDB geändert haben, müssen Sie die folgenden Konfigurationsänderungen vornehmen, um den Status "Tatsächlich" für replizierte CIs zu aktivieren:

1. Öffnen Sie die Datei **web.xml** unter **UCMDBServer\deploy\axis2\WEB-INF**.
2. Fügen Sie den folgenden **servlet init**-Parameter zu AxisServlet hinzu (fügen Sie diese 4 Zeilen nach Zeile 28 ein):

```
<init-param>

<param-name>axis2.find.context</param-name>

<param-value>false</param-value>

</init-param>
```

Diese Einstellung verhindert, dass Axis2 versucht, den Kontextstamm zu berechnen, sondern explizit in der Datei **axis2.xml** nach ihm sucht.

3. Öffnen Sie die Datei **axis2.xml** unter **UCMDBServer\deploy\axis2\WEB-INF\conf**.
4. Entfernen Sie in Zeile 58 die Kommentare vom Parameter **contextRoot** und bearbeiten Sie den Parameter wie folgt:

   ```
   <parameter name="contextRoot" locked="false">test/axis2</parameter>
   ```

   (Dabei steht **test** für den neuen Stammkontext in der Datei **cmdb.xml**).

   Hinweis: Vor **test/axis2** steht kein Schrägstrich.

Anwendungsfälle

Die folgenden Anwendungsfälle setzen zwei Systeme voraus:

- HP Universal CMDB Server
- Ein Drittanbietersystem, das ein Repository der Konfigurationselemente enthält.

Dieser Abschnitt umfasst die folgenden Themen:

Auffüllen der CMDB

Anwendungsfälle:

- Ein Asset-Management-Werkzeug eines Drittanbieters aktualisiert die CMDB mit den Informationen, die nur ihm zur Verfügung stehen.
- Mehrere Drittanbietersysteme füllen die CMDB auf, um eine zentrale CMDB zu erstellen, die Änderungen verfolgen und Auswirkungsanalysen durchführen kann.
- Ein Drittanbietersystem erstellt Konfigurationselemente und Beziehungen gemäß der Geschäftslogik des Drittanbieters, um die Abfragefunktionen der CMDB zu nutzen.

Abfragen der CMDB

Anwendungsfälle:

- Ein Drittanbietersystem ruft die Konfigurationselemente und Beziehungen ab, die das SAP-System darstellen, indem es die Ergebnisse der SAP-TQL abruft.
- Ein Drittanbietersystem ruft die Liste der Oracle-Server ab, die innerhalb der letzten 5 Stunden hinzugefügt oder geändert wurden.
- Ein Drittanbietersystem ruft die Liste der Server ab, deren Hostname die Unterzeichenfolge *lab* enthält.

- Ein Drittanbietersystem sucht die zu einem bestimmten CI gehörenden Elemente, indem es dessen Nachbarn abruft.

Abfragen des Klassenmodells

Anwendungsfälle:

- Ein Drittanbietersystem ermöglicht den Benutzern, den Datensatz anzugeben, der von der CMDB abgerufen werden soll. Über dem Klassenmodell kann eine Benutzeroberfläche angeordnet werden, um den Benutzern die möglichen Eigenschaften anzuzeigen und sie zur Eingabe der erforderlichen Daten aufzufordern. Der Benutzer kann dann die abzurufenden Informationen auswählen.
- Ein Drittanbietersystem durchsucht das Klassenmodell, wenn der Benutzer nicht auf die UCMDB-Benutzeroberfläche zugreifen kann.

Analysieren von Änderungsauswirkungen

Anwendungsfall:

Ein Drittanbietersystem gibt eine Liste der Geschäftsservices aus, die von einer Änderung bei einem bestimmten Host betroffen sein könnten.

Beispiele

Dieser Abschnitt umfasst die folgenden Themen:

Die Beispielbasisklasse

```
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;
     }
    }
```

Abfragebeispiel

```
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 relations1;
    }

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

Aktualisierungsbeispiel

```
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.UcmdbFault;
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;
import java.util.ArrayList;
import java.util.List;
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();
      List<CI> listCI = new ArrayList<CI>();
      CI ci = new CI();
      ID id = new ID();
      id.setString("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.getStrProps().add(strProp);
      props.setStrProps(strProps);
      ci.setProps(props);
      listCI.add(ci);
      cis.setCIs(listCI);
```

```
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
        try {
          AddCIsAndRelationsResponse response  = getStub().addCIsAndRelations
(request);
          for(int i = 0  ;  i < response.getCreatedIDsMaps().size() ; i++) {
             ClientIDToCmdbID idsMap = response.getCreatedIDsMaps().get(i);
             //do something
          }
        } catch (RemoteException e) {
          //handle exception
        } catch (UcmdbFault e) {
          //handle exception
        }
    }
    public void getUpdateCIsAndRelationsDemo() {
        UpdateCIsAndRelations request = new UpdateCIsAndRelations();
        request.setCmdbContext(getContext());
        CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        List<CI> listCI = new ArrayList<CI>();
        CI ci = new CI();
        ID id = new ID();
        id.setString("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.getStrProps().add(hostKeyProp);
```

```
    StrProp hostOSProp = new StrProp();
    hostOSProp.setName("host_os");
    String hostOSValue = "winXP";
    hostOSProp.setValue(hostOSValue);
    strProps.getStrProps().add(hostOSProp);
    StrProp hostDNSProp = new StrProp();
    hostDNSProp.setName("host_dnsname");
    String hostDNSValue = "dnsname";
    hostDNSProp.setValue(hostDNSValue);
    strProps.getStrProps().add(hostDNSProp);
    props.setStrProps(strProps);
    ci.setProps(props);
    listCI.add(ci);
    cis.setCIs(listCI);
    updates.setCIsForUpdate(cis);
    request.setCIsAndRelationsUpdates(updates);
    try {
      getStub().updateCIsAndRelations(request);
    } catch (RemoteException e) {
      //handle exception
    } catch (UcmdbFault e) {
      //handle exception
    }
}
public void getDeleteCIsAndRelationsDemo() {
    DeleteCIsAndRelations request = new DeleteCIsAndRelations();
    request.setCmdbContext(getContext());
    CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
    CIs cis = new CIs();
    List<CI> listCI = new ArrayList<CI>();
    CI ci = new CI();
    ID id = new ID();
    id.setString("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.getStrProps().add(strProp1);
    props.setStrProps(strProps);
    ci.setProps(props);
    listCI.add(ci);
    cis.setCIs(listCI);
    updates.setCIsForUpdate(cis);
    request.setCIsAndRelationsUpdates(updates);
    try {
      getStub().deleteCIsAndRelations(request);
    } catch (RemoteException e) {
      //handle exception
    } catch (UcmdbFault e) {
      //handle exception
    }
}
public static void main(String[] args) {
  try {
     UpdateDemo demo = new UpdateDemo();
     demo.initDemo();
     demo.getAddCIsAndRelationsDemo();
    } catch(Exception e){
     System.out.println(e.getMessage());
     e.printStackTrace();
    }
```

```
    }
}
```

Beispiel für das Klassenmodell

```
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
        }
    }
}
```

Beispiel für die Auswirkungsanalyse

```
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
        }
}
}
```

Beispiel für das Hinzufügen von Anmeldeinformationen

```
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.AUTHENTICATE,auth);

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

```
        }
    } // End class
```

Kapitel 11

API zur Datenflussverwaltung

Dieses Kapitel umfasst folgende Themen:

API zur Datenflussverwaltung – Übersicht

In diesem Kapitel wird beschrieben, wie Sie mit benutzerdefinierten Werkzeugen oder Werkzeugen von Drittanbietern den HP Webservice "Datenflussverwaltung" für die Datenflussverwaltung nutzen können.

Die vollständige Dokumentation zu den verfügbaren Operationen finden Sie in der *Schemareferenz zu HP Discovery and Dependency Mapping*. Die Dateien befinden sich im folgenden Ordner:

<UCMDB-Stammverzeichnis>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

Konventionen

In diesem Kapitel werden die folgenden Konventionen verwendet:

- Die Schreibweise `Element` gibt an, dass es sich bei einem Element um eine Entität in der Datenbank oder um ein im Schema definiertes Element handelt, einschließlich der Strukturen, die an die Methoden übergeben oder von diesen zurückgegeben werden. Klartext gibt an, dass das Element in einem allgemeinen Kontext erörtert wird.
- Elemente und Methodenargumente, die sich auf die Datenflussverwaltung beziehen, werden genauso geschrieben wie im Schema angegeben. In der Regel bedeutet dies, dass ein Klassenname oder ein allgemeiner Verweis auf eine Instanz der Klasse großgeschrieben wird. Ein Element oder Argument einer Methode wird kleingeschrieben. Beispiel: `credential` ist ein Element des Typs `Credential`, das an eine Methode übergeben wird.

Webservice "Datenflussverwaltung"

Der HP Webservice "Datenflussverwaltung" ist eine API, die für die Integration von Applikationen mit HP Universal CMDB verwendet wird. Die API stellt Methoden für folgende Aufgaben zur Verfügung:

- **Verwalten von Anmeldeinformationen**. Anzeigen, Hinzufügen, Aktualisieren und Entfernen.
- **Verwalten von Jobs**. Anzeigen des Status, Aktivieren und Deaktivieren.
- **Verwalten von Probe-Bereichen**. Anzeigen, Hinzufügen und Aktualisieren.
- **Verwalten von Triggern**. Hinzufügen oder Entfernen eines Trigger-CI sowie Hinzufügen, Entfernen oder Deaktivieren einer Trigger-TQL.
- **Anzeigen von allgemeinen Daten**. Daten zu Domänen und Proben.

Benutzer des HP Webservice "Datenflussverwaltung" sollten mit Folgendem vertraut sein:

- Der SOAP-Spezifikation
- Einer objektorientierten Programmiersprache wie C++, C# oder Java
- HP Universal CMDB
- Datenflussverwaltung

Berechtigungen

Der Administrator stellt die Anmeldeinformationen zum Herstellen der Verbindung zum Webservice zur Verfügung. Die Berechtigungsebenen lauten "Anzeigen", "Aktualisieren" und "Ausführen". Weitere Informationen über die für die einzelnen Operationen erforderlichen Berechtigungen finden Sie in der Anforderungsdokumentation der jeweiligen Operation in der *Schemareferenz zu HP Discovery and Dependency Mapping*.

Aufrufen des Webservice

Der Webservice von HP Discovery and Dependency Mapping ermöglicht das Aufrufen von serverseitigen Methoden mittels standardmäßiger SOAP-Programmierverfahren. Wenn die Anweisung nicht analysiert werden kann oder ein Problem beim Aufrufen der Methode auftritt, lösen die API-Methoden eine `SoapFault`-Ausnahme aus. Daraufhin füllt der Service eines oder mehrere der für Fehlermeldungen, Fehlercodes und Ausnahmemeldungen vorgesehenen Felder mit den entsprechenden Daten. Wenn kein Fehler festgestellt wird, werden die Ergebnisse des Aufrufs zurückgegeben.

So rufen Sie den Service auf:

- `Protocol: http oder https (je nach Serverkonfiguration)`
- `URL: <UCMDB-Server>:8080/axis2/services/DiscoveryService`
- `Standardkennwort: "admin"`
- `Standardbenutzername: "admin"`

SOAP-Programmierer können über die folgende Adresse auf die WSDL zugreifen:

- `axis2/services/DiscoveryService?wsdl`

Methoden der Datenflussverwaltung

Dieser Abschnitt enthält eine Liste der Webservice-Operationen zusammen mit einer kurzen Zusammenfassung ihrer Verwendungsbereiche. Die vollständige Dokumentation zu den

Anforderungen und Antworten für jede Operation finden Sie in der *Schemareferenz zu HP Discovery and Dependency Mapping*.

Dieser Abschnitt umfasst die folgenden Themen:

Datenstrukturen

Dies sind einige der in der API des Webservice "Datenflussverwaltung" verwendeten Datenstrukturen.

CIProperties

`CIProperties` sind eine Sammlung von Sammlungen. Jede Sammlung enthält Eigenschaften eines anderen Datentyps. Es kann beispielsweise eine `dateProps`-Sammlung, eine `strListProps`-Sammlung, eine `xmlProps`-Sammlung usw. geben.

Jede Typsammlung enthält individuelle Eigenschaften des jeweiligen Typs. Die Namen dieser Eigenschaftenelemente sind identisch mit dem Namen des Containers. Sie werden allerdings in der Singularform verwendet. Beispiel: `dateProps` enthält `dateProp`-Elemente. Jede Eigenschaft ist ein Name-Wert-Paar.

Weitere Informationen finden Sie unter CIProperties in der *Schemareferenz zu HP Discovery and Dependency Mapping*.

IPList

Eine Liste der `IP`-Elemente, von denen jedes eine IPv4-Adresse enthält.

Weitere Informationen finden Sie unter IPList in der *Schemareferenz zu HP Discovery and Dependency Mapping*.

IPRange

`IPRange` besteht aus zwei Elementen, dem Element `Start` und dem Element `End`. Jedes Element enthält ein `Address`-Element. Dabei handelt es sich um eine IPv4-Adresse.

Weitere Informationen finden Sie unter IPLRange in der *Schemareferenz zu HP Discovery and Dependency Mapping*.

Scope

Zwei `IPRanges`. `Exclude` ist eine Sammlung von `IPRanges`, die von dem Job ausgeschlossen werden sollen. `Include` ist eine Sammlung von `IPRanges`, die in den Job aufgenommen werden sollen.

Weitere Informationen finden Sie unter Scope in der *Schemareferenz zu HP Discovery and Dependency Mapping*.

Verwalten von Discovery-Job-Methoden

activateJob

Aktiviert den angegebenen Job.

Siehe "Codebeispiel" auf Seite 307.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Name des Jobs.

deactivateJob

Deaktiviert den angegebenen Job.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Name des Jobs.

dispatchAdHocJob

Verteilt ad hoc einen Job auf der Probe. Der Job muss aktiv sein und das angegebene Trigger-CI enthalten.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Name des Jobs.
CIID	Die ID des Trigger-CI.
ProbeName	Der Name der Probe.
Zeitüberschreitung	In Millisekunden

getDiscoveryJobsNames

Gibt die Liste der Jobnamen zurück.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.

Ausgabe

Parameter	Kommentar
strList	Die Liste der Jobnamen.

isJobActive

Prüft, ob der Job aktiv ist.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Name des zu prüfenden Jobs.

Ausgabe

Parameter	Kommentar
JobState	**True**, wenn der Job aktiv ist.

Verwalten von Trigger-Methoden

addTriggerCI

Fügt ein neues Trigger-CI zum angegebenen Job hinzu.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Name des Jobs.
CIID	Die ID des Trigger-CI.

addTriggerTQL

Fügt eine neue Trigger-TQL zum angegebenen Job hinzu.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Name des Jobs.
TqlName	Der Name der hinzuzufügenden TQL.

disableTriggerTQL

Verhindert, dass die TQL den Job auslöst, entfernt sie aber nicht permanent aus der Liste der Abfragen, die den Job auslösen.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Name des Jobs.

removeTriggerCI

Entfernt das angegebene CI aus der Liste der CIs, die den Job auslösen.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Jobname.
CIID	Die ID des Trigger-CI.

removeTriggerTQL

Entfernt die angegebene TQL aus der Liste der Abfragen, die den Job auslösen.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Die Sammlung der zu prüfenden Jobnamen.
CIID	Die ID der zu entfernenden TQL.

setTriggerTQLProbesLimit

Schränkt die Proben, in denen die TQL im Job aktiv ist, auf die angegebene Liste ein.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
JobName	Der Name des Jobs.
tqlName	Der TQL-Name.
probesLimit	Die Liste der Proben, für die die TQL aktiv ist.

Methoden in Bezug auf Domänen- und Probe-Daten

getDomainType

Gibt den Domänentyp zurück.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Der Name der Domäne.

Ausgabe

Parameter	Kommentar
domainType	Der Domänentyp.

getDomainsNames

Gibt die Namen der aktuellen Domänen zurück.

Siehe "Codebeispiel" auf Seite 307.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.

Ausgabe

Parameter	Kommentar
domainNames	Die Liste der Domänennamen.

getProbeIPs

Gibt die IP-Adressen der angegebenen Probe zurück.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die zu prüfende Domäne.
probeName	Der Name der Probe, die in dieser Domäne verwendet wird.

Ausgabe

Parameter	Kommentar
probeIPs	Die "IPList" der Adressen in der Probe.

getProbesNames

Gibt die Namen der Proben in der angegebenen Domäne zurück.

Siehe "Codebeispiel" auf Seite 307.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die zu prüfende Domäne.

Ausgabe

Parameter	Kommentar
probesName	Die Liste der Proben in der Domäne.

getProbeScope

Gibt die Definition des Gültigkeitsbereichs der angegebenen Probe zurück.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die zu prüfende Domäne.
probeName	Der Name der Probe.

Ausgabe

Parameter	Kommentar
probeScope	Der "Scope" der Probe.

isProbeConnected

Prüft, ob die angegebene Probe verbunden ist.

Siehe "Codebeispiel" auf Seite 307.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.

Parameter	Kommentar
domainName	Die zu prüfende Domäne.
probeName	Die zu prüfende Probe

Ausgabe

Parameter	Kommentar
isConnected	**True**, wenn die Probe verbunden ist.

updateProbeScope

Legt den Gültigkeitsbereich der angegebenen Probe fest und überschreibt den vorhandenen Gültigkeitsbereich.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die Domäne.
probeName	Die zu aktualisierende Probe.
newScope	Der für die Probe festzulegende "Scope".

Methoden in Bezug auf Anmeldeinformationen

addCredentialsEntry

Fügt einen Eintrag für Anmeldeinformationen zum angegebenen Protokoll für die angegebene Domäne hinzu.

Siehe "Codebeispiel" auf Seite 307.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die zu aktualisierende Domäne.
protocolName	Der Name des Protokolls.
credentialsEntryParameters	Die "CIProperties"-Sammlung der neuen Anmeldeinformationen.

Ausgabe

Parameter	Kommentar
credentialsEntryID	Die CI-ID des Eintrags der neuen Anmeldeinformationen.

getCredentialsEntriesIDs

Gibt die IDs der für das angegebene Protokoll definierten Anmeldeinformationen zurück.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die Domäne, für die die Anmeldeinformationen abgerufen werden.
protocolName	Der Name eines Protokolls, das in dieser Domäne verwendet wird.

Ausgabe

Parameter	Kommentar
credentialsEntryIDs	Die Liste der IDs der Anmeldeinformationen für das Protokoll in der Domäne.

getCredentialsEntry

Gibt die für das angegebene Protokoll definierten Anmeldeinformationen zurück. Verschlüsselte Attribute werden leer zurückgegeben.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die Domäne, für die die Anmeldeinformationen abgerufen werden.
protocolName	Ein in dieser Domäne verwendetes Protokoll.
credentialsEntryID	Die abzurufende ID der Anmeldeinformationen.

Ausgabe

Parameter	Kommentar
credentialsEntryParameters	Die "CIProperties"-Sammlung der Anmeldeinformationen.

removeCredentialsEntry

Entfernt die angegebenen Anmeldeinformation aus dem Protokoll.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die Domäne.
protocolName	Ein in der Domäne verwendetes Protokoll.
credentialsEntryID	Die ID der zu entfernenden Anmeldeinformationen.

updateCredentialsEntry

Legt neue Werte für die Eigenschaften des angegebenen Eintrags für Anmeldeinformationen fest.

Die vorhandenen Eigenschaften werden gelöscht und diese Eigenschaften werden festgelegt. Alle Eigenschaften, deren Werte in diesem Aufruf nicht festgelegt sind, bleiben undefiniert.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
domainName	Die Domäne, in der Anmeldeinformationen aktualisiert werden sollen.
protocolName	Ein in der Domäne verwendetes Protokoll.
credentialsEntryID	Die ID der zu aktualisierenden Anmeldeinformationen.
credentialsEntryParameters	Die "CIProperties"-Sammlung, die als Eigenschaften für die Anmeldeinformationen festgelegt werden sollen.

Datenaktualisierungsmethoden

rediscoverCIs

Sucht die Trigger, die die angegebenen CI-Objekte ermittelt haben, und führt diese erneut aus.

rediscoverCIs wird asynchron ausgeführt. Rufen Sie **checkDiscoveryProgress** auf, um festzustellen, ob die erneute Discovery abgeschlossen ist.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
CmdbIDs	Sammlung der IDs der erneut zu findenden Objekte.

Ausgabe

Parameter	Kommentar
isSucceed	**True**, wenn die erneute Discovery der CIs erfolgreich war.

checkDiscoveryProgress

Gibt den Fortschritt des letzten **rediscoverCIs**-Aufrufs bei den angegebenen IDs zurück. Die Antwort ist ein Wert zwischen `0` und `1`. Wenn die Antwort `1` lautet, ist der **rediscoverCIs**-Aufruf abgeschlossen.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
CmdbIDs	Sammlung der IDs der Objekte in dem zu verfolgenden Re-Discovery-Aufruf.

Ausgabe

Parameter	Kommentar
progress	Ein abgeschlossener Job hat einen Fortschritt von 1. Nicht abgeschlossene Jobs haben einen Bruchteil kleiner als 1.

rediscoverViewCIs

Sucht die Trigger, die die Daten zum Auffüllen der angegebenen Ansicht erstellt haben, und führt diese erneut aus.

rediscoverViewCIs wird asynchron ausgeführt. Rufen Sie **checkViewDiscoveryProgress** auf, um festzustellen, ob die erneute Discovery abgeschlossen ist.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
viewName	Die zu prüfenden Ansichten.

Ausgabe

Parameter	Kommentar
isSucceed	**True**, wenn die erneute Discovery der CIs erfolgreich war.

checkViewDiscoveryProgress

Gibt den Fortschritt des letzten **rediscoverViewCIs**-Aufrufs bei der angegebenen Ansicht zurück. Die Antwort ist ein Wert zwischen 0 und 1. Wenn die Antwort 1 lautet, ist der **rediscoverCIs**-Aufruf abgeschlossen.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 247.
viewName	Die Sammlung der zu überprüfenden Ansichten.

Ausgabe

Parameter	Kommentar
progress	Ein abgeschlossener Job hat einen Fortschritt von 1. Nicht abgeschlossene Jobs haben einen Bruchteil kleiner als 1.

Codebeispiel

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.*;
import com.hp.ucmdb.generated.types.*;
public class test {
    static final String HOST_NAME = "<my_hostname>";
    static final int PORT = 8080;
    private static final String PROTOCOL = "http";
    private static final String FILE =
"/axis2/services/DiscoveryService";

    private static final String PASSWORD = "<my_password>";
    private static final String USERNAME = "<my_username>";

    private static CmdbContext cmdbContext = new CmdbContext("ws
tests");

    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest(
            "Range IPs by ICMP", cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protcol
        addNTCMDCredentialsEntry();
    }
```

```
    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(
                new GetDomainsNamesRequest(cmdbContext)).
                getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create
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.AUTHENTICATE,auth);
```

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


Java™
COMPATIBLE