HP Universal CMDB

Softwareversion: 10.20

Entwicklerreferenzhandbuch

Datum der Dokumentveröffentlichung: Januar 2015
Datum des Software-Release: Januar 2015

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 US-Regierung per Standardlizenz lizenziert.

Copyright-Hinweis

Marken

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.

Dieses Produkt enthält eine Oberfläche der Komprimierungsbibliothek 'zlib' für allgemeine Zwecke, Copyright © 1995-2002 Jean-loup Gailly und Mark Adler.

Aktualisierte Dokumentation

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

- Software-Versionsnummer zur Angabe der Version der Software
- 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: **https://softwaresupport.hp.com**

Für die Anmeldung an dieser Website benötigen Sie einen HP Passport. Hier können Sie sich für eine HP Passport-ID registrieren: **https://hpp12.passport.hp.com/hppcf/createuser.do**

Oder klicken Sie auf den Link für die Registrierung oben auf der Seite des HP Software Support.

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: **https://softwaresupport.hp.com**

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

Der Online-Software-Support bietet Kunden mit Hilfe interaktiver technischer Support-Werkzeuge für die Unternehmensverwaltung die Möglichkeiten, ihre Probleme auf schnelle und effiziente Weise 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-Supportkontakten
- 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:

https://hpp12.passport.hp.com/hppcf/createuser.do

Weitere Informationen zu Zugriffsebenen finden Sie unter:

https://softwaresupport.hp.com/web/softwaresupport/access-levels

HP Software Solutions Now greift auf die Website von HPSW Solution and Integration Portal zu. Auf dieser Website finden Sie HP-Produktlösungen für Ihre Unternehmensanforderungen, einschließlich einer Liste aller Integrationsmöglichkeiten zwischen HP-Produkten sowie eine Aufstellung der ITIL-Prozesse. Der URL für diese Website lautet **http://h20230.www2.hp.com/sc/solutions/index.jsp**

Info zur PDF-Version der Online-Hilfe

Bei diesem Dokument handelt es sich um eine PDF-Version der Online-Hilfe. Diese PDF-Datei wird Ihnen bereitgestellt, um Ihnen das Drucken mehrerer Themen der Hilfe oder das Lesen der Online-Hilfe im PDF-Format zu ermöglichen. Da diese Inhalte ursprünglich als Online-Hilfe für die Anzeige in einem Webbrowser erstellt wurden, sind einige Themen möglicherweise nicht ordnungsgemäß formatiert. Einige interaktive Themen sind eventuell nicht in dieser PDF-Version vorhanden. Diese fehlenden Themen können Sie problemlos direkt aus der Online-Hilfe drucken.

Inhalt

Teil I: 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:

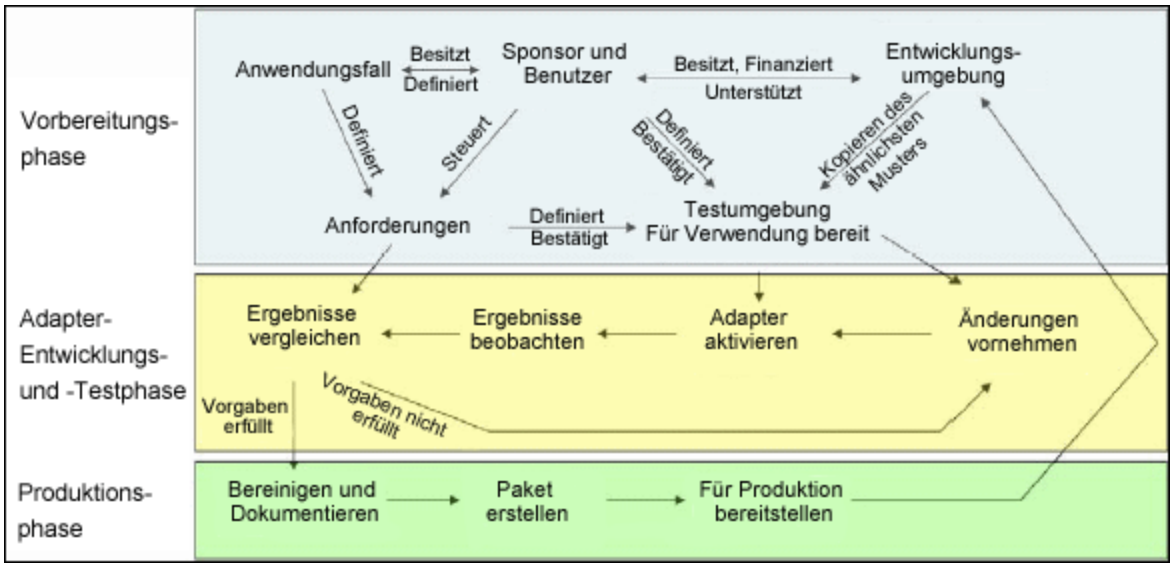
- Es wird davon ausgegangen, dass Sie über praktische Kenntnisse der HP Universal CMDB verfügen und mit den Elementen des Systems grundlegend vertraut sind. Die folgenden Ausführungen sind nicht als detaillierter Leitfaden vorgesehen, sondern sollen Sie lediglich bei Ihrem Lernprozess unterstützen.
- 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

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.

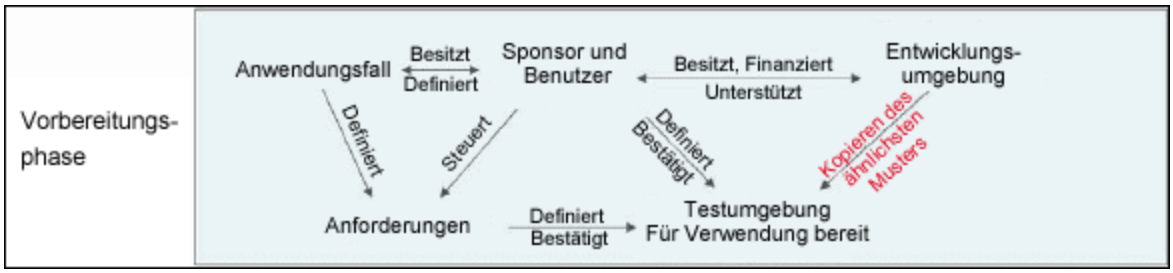


Die einzelnen Phasen der Adapterentwicklung bauen aufeinander auf.

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 finden Sie unter Package Manager im *HP Universal CMDB – Verwaltungshandbuch*.

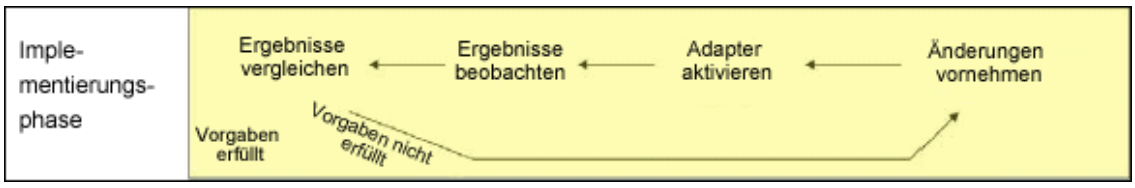
Die folgenden Abschnitte befassen sich mit den einzelnen Phasen der Adaptererstellung unter besonderer Berücksichtigung der wichtigsten Schritte und Best Practices:

Untersuchungs- und Vorbereitungsphase



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. Legen Sie die Methode fest, die der Adapter für die Datenerfassung verwenden 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 folgende 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 12.

Dieser Abschnitt umfasst die folgenden Themen:

- "Ändern eines vorhandenen Adapters" unten
- "Schreiben eines neuen Adapters" unten
- "Modelluntersuchung" unten
- "Technologieuntersuchung" auf der nächsten Seite
- "Richtlinien für die Auswahl von Datenzugriffsmöglichkeiten" auf der nächsten Seite
- "Übersicht" auf Seite 20

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. Sie können die gegenwärtig unterstützten HP Universal CMDB-Verbindungsprotokolle unter **Datenflussverwaltung > Data Flow Probe einrichten > Ausschnitt "Domänen und Proben"** anzeigen. Weitere Informationen zu den einzelnen Protokollen finden Sie im Abschnitt über die unterstützten Protokolle im *HP UCMDB Discovery and Integrations Content Guide*

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?

  Wenn Sie diese Fragen mit "Ja" beantworten, dann ist **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?

  Wenn Sie diese Fragen mit "Ja" beantworten, dann ist **Föderation** erforderlich.
- Soll die Integration Daten per Push an Remote-Datenquellen übertragen?

  Wenn Sie diese Fragen mit "Ja" beantworten, dann ist **Datenpush** erforderlich.
- Überschreitet die Länge einer CI-ID 60 Zeichen?

  Wenn Sie diese Frage mit "Ja" beantworten, **verringern** Sie die ID-Länge für alle relevanten CIs um sicherzustellen, dass die maximale Länge von 60 Zeichen nicht überschritten wird.

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 in dem Handbuch *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

Für die Erstellung von Integrationsadaptern stehen fünf 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 37.
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, die mit einer einzelnen Datenquelle oder einem einzelnen Datenziel verbunden werden.

   Weitere Informationen finden Sie unter "Entwickeln von Java-Adaptern" auf Seite 194.
3. Allgemeiner DB-Adapter:
   - Ein auf dem Java-Adapter basierender abstrakter Adapter, der das 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, die mit einer einzelnen Datenquelle verbunden werden.

   Weitere Informationen finden Sie unter "Entwickeln von allgemeinen Datenbankadaptern" auf Seite 121.
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, die mit einem einzelnen Datenziel verbunden werden.
- Wird für Datenpushes verwendet.

Weitere Informationen finden Sie unter "Entwickeln von Push-Adaptern" auf Seite 228.

5. Erweiterter generischer Push-Adapter:
   - Alle oben genannten Funktionen des generischen Push-Adapters
   - Ein auf dem Stammelement basierender Adapter
   - Ordnet eine UCMDB-Datenstruktur zu einer Ziel-Datenstruktur zu

   Weitere Informationen finden Sie unter "Umsetzen des Datenpush unter Verwendung des allgemeinen Adapters" auf Seite 276.

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

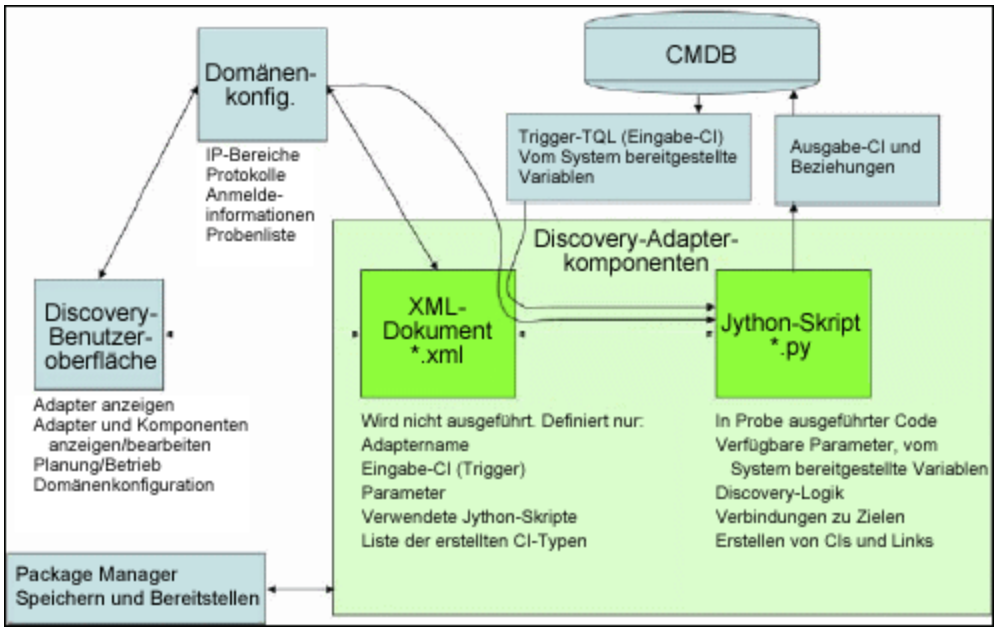
Fluss/Adapter	Jython-Adapter	Java-Adapter	Allgemeiner DB-Adapter	Generischer Push-Adapter	Erweiterter generischer Push-Adapter
Auffüllung	✓	✓	✓	×	×
Föderation	×	✓	✓	×	×
Datenpush	×	✓	×	✓	✓

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 **Universal Discovery**, **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

Eine vollständige Discovery könnte in einem einzigen 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 28.

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 34.

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 35.

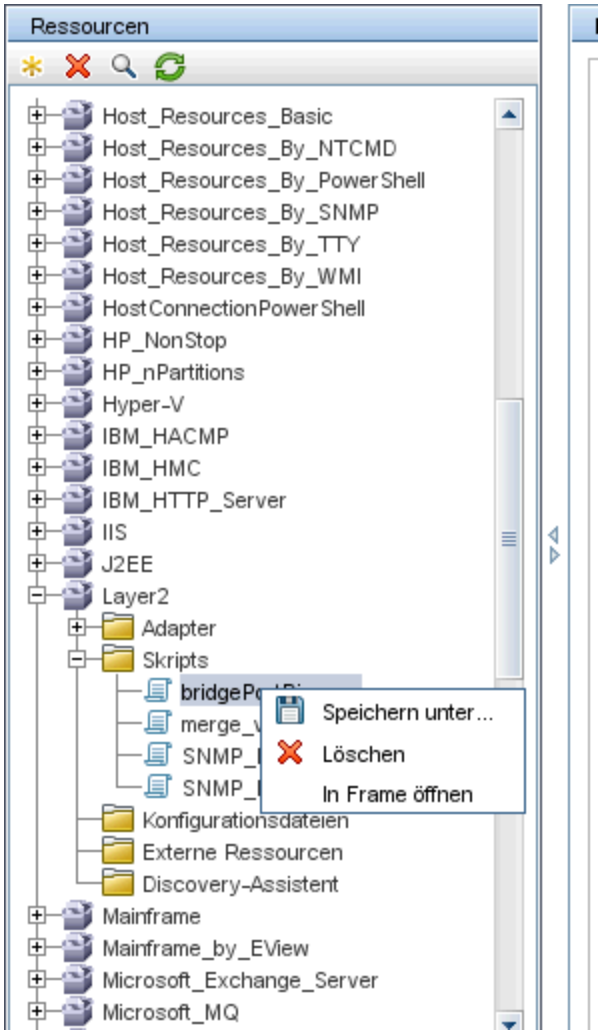
Um neue Adapter zu schreiben, müssen Sie alle oben aufgeführten Komponenten erstellen. Jede Komponente wird dann automatisch an die Komponente im vorherigen Schritt gebunden. Nachdem Sie zum Beispiel einen Job erstellt und den entsprechenden Adapter ausgewählt haben, wird die Adapterdatei an den Job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

Adaptercode

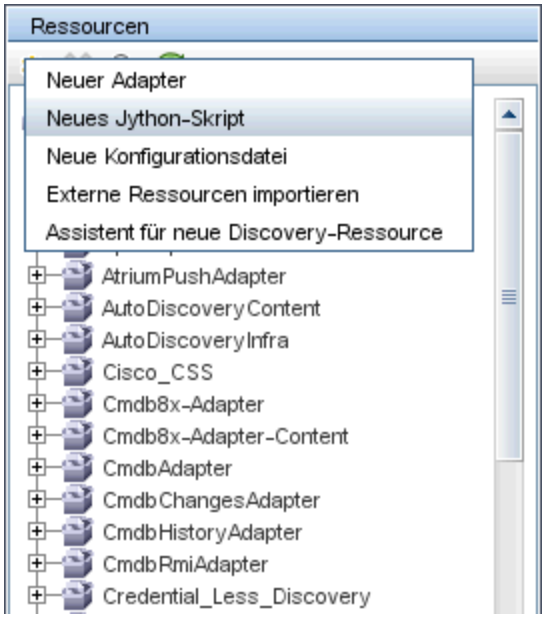
Die eigentliche Implementierung, d. h. das Herstellen einer Verbindung zum Remote-System, das Abfragen seiner Daten und deren Zuordnung als CMDB-Daten, wird mittels Jython-Code durchgeführt. Der Code enthält beispielsweise die Logik zum Herstellen einer Verbindung zu einer Datenbank und zum Extrahieren 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.

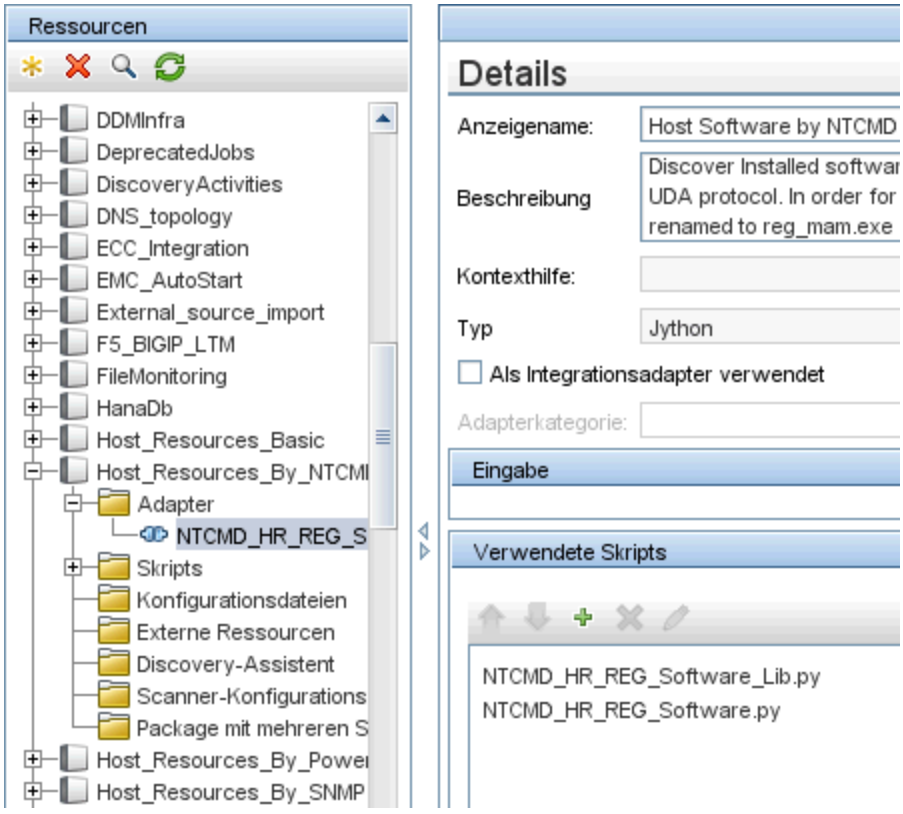
Ein Adapter kann eine Liste von Skripts enthalten, die bei der Discovery-Ausführung verwendet werden sollen. Bei der Erstellung eines neuen Adapters erstellen Sie in der Regel ein neues Skript, das Sie dann dem Adapter zuweisen. Ein neues Skript enthält Basisvorlagen. Sie können jedoch auch ein anderes 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 35. 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 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 "Konzepte der Datenflussverwaltung" 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 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.

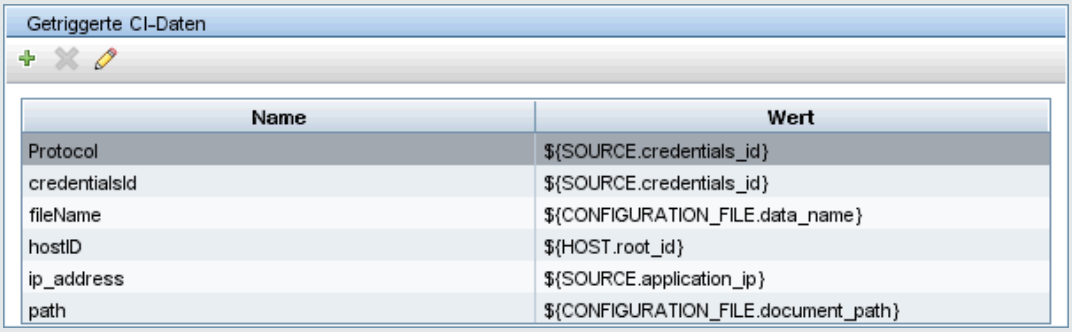
- Wenn der Wert der Zieldaten eine Liste ist, können Sie von der Liste die Anzahl der Elemente definieren, die an die Probe gesendet werden sollen. Fügen Sie zum Definieren einen Doppelpunkt hinter dem Standardwert gefolgt von der Anzahl der Elemente hinzu. Ist kein Standardwert für die Zieldaten vorhanden, geben Sie zwei Doppelpunkte ein.

  Wenn beispielsweise die folgenden Zieldaten eingegeben werden: `name=portId, value= ${PHYSICALPORT.root_id:NA:1}` oder `name=portId, value= ${PHYSICALPORT.root_id::1}`, wird nur der erste Port aus der Port-Liste an die Probe gesendet.

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:

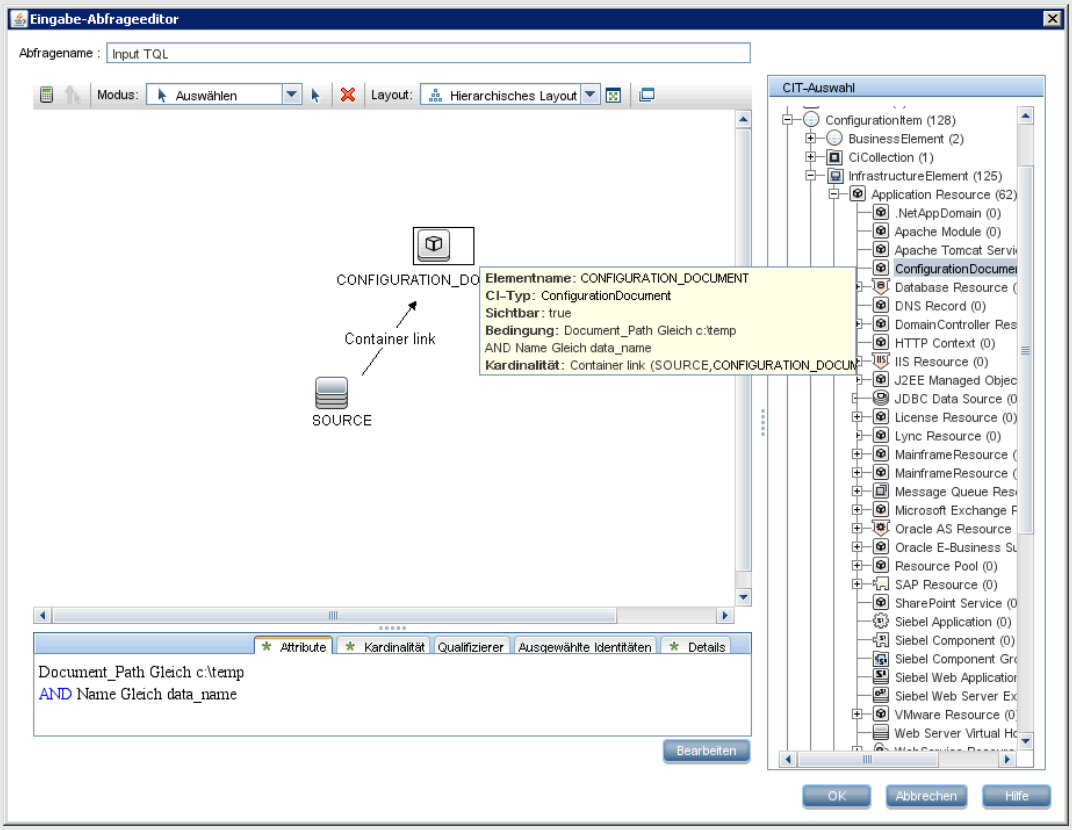

Getriggerte CI-Daten

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:

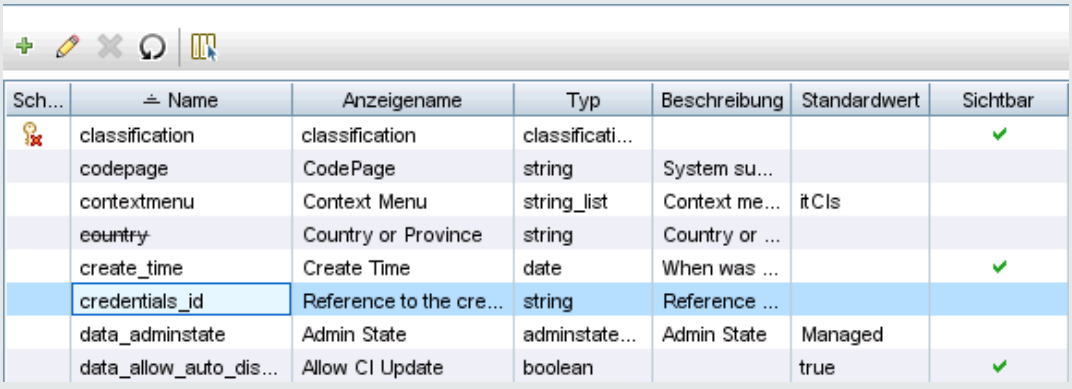
```
Framework.getTriggerCIData ('ip_address')
```

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 auf die Miniaturansicht, um das Bild in voller Größe anzuzeigen.

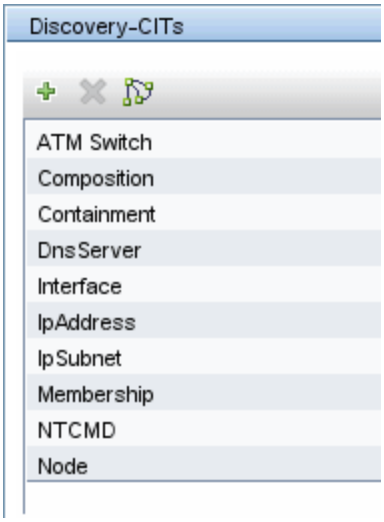
Die Variablen `Protocol`, `credentialsId` und `ip_address` verwenden die Attribute `root_class`, `credentials_id` und `application_ip`:



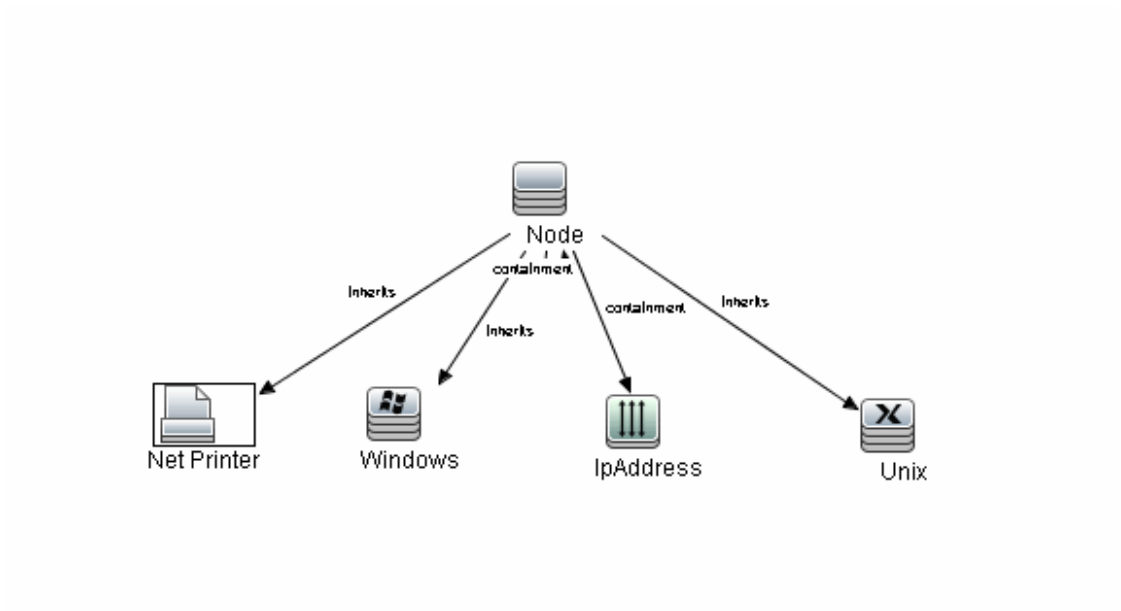
Sch...	Name	Anzeigename	Typ	Beschreibung	Standardwert	Sichtbar
	classification	classification	classificati...			✔
	codepage	CodePage	string	System su...		
	contextmenu	Context Menu	string_list	Context me...	itCIs	
	~~country~~	Country or Province	string	Country or ...		
	create_time	Create Time	date	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 42.

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.

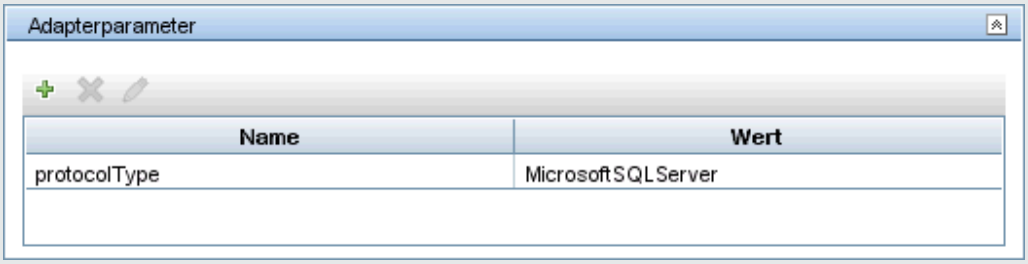
Beispiel für das Überschreiben eines Adapterparameters:

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**.

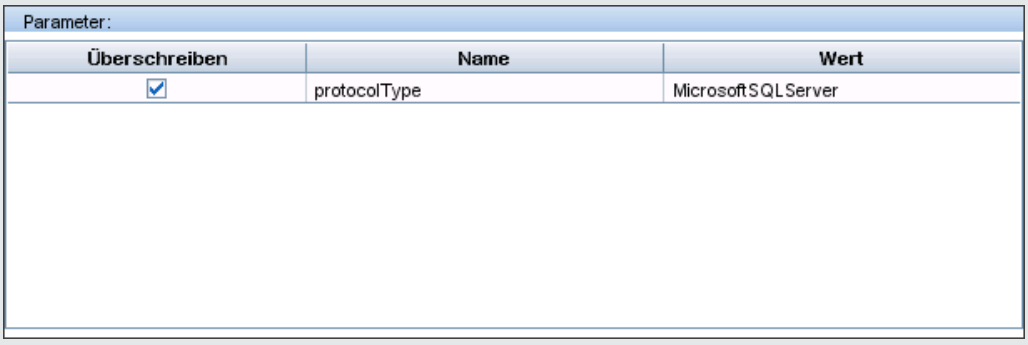
b. Wählen Sie im Ausschnitt **Ressourcen** die Optionen **Database_Basic > Adapter > SQL_NET_Dis_Connection** aus.

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 jedes Trigger-CI ausführen soll.

- **Für den IP-Adress-CI-Typ:** Nehmen Sie die Probe, die für diese IP-Adresse definiert ist.
- **Für den CI-Typ "Aktive Software":** Verwenden Sie die Attribute **application_ip** und **application_ip_domain** und wählen Sie die Probe aus, die für die IP-Adresse in der entsprechenden Domäne definiert ist.
- **Für andere CI-Typen:** Nehmen Sie die IP-Adresse 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.

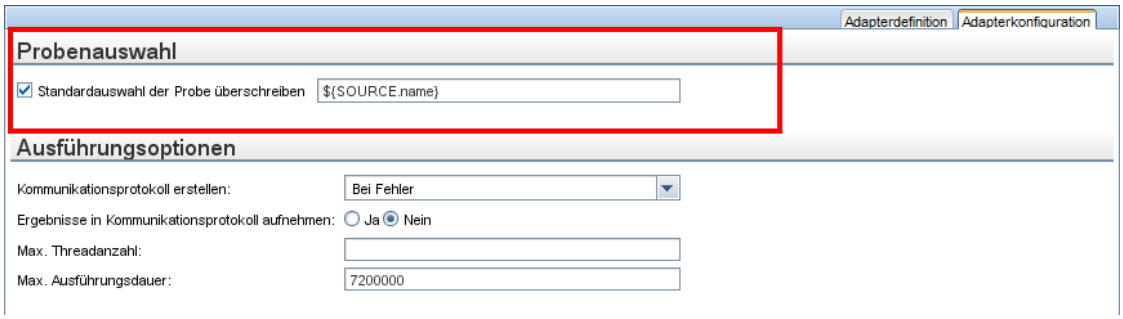
So geben Sie manuell an, welche Probe mit dem Adapter verwendet werden soll:

- Wählen Sie den Adapter aus und klicken Sie auf die Registerkarte **Adapterkonfiguration**.
- Wählen Sie unter **Optionen für Trigger-Verteilung** die Einstellung **Standardauswahl der Probe überschreiben** aus.
- Geben Sie die Probe in einem der folgenden Formate in das Feld ein:

Probenname	Der Name der Probe

IP-Adresse	Die IP-Adresse der Probe. Sie kann entweder im IPv4- oder im IPv6-Format definiert werden.
IP,Domäne	IPv4-Format: **16.59.63.86,Standarddomäne** IPv6-Format: **2001:0:9d46:953c:34a9:1e6b:f2ff:fffe,BenutzerdefinierteDomäne**
Domänenname	Die Domäne, von der die Probe ausgewählt werden soll.

Beispiel:



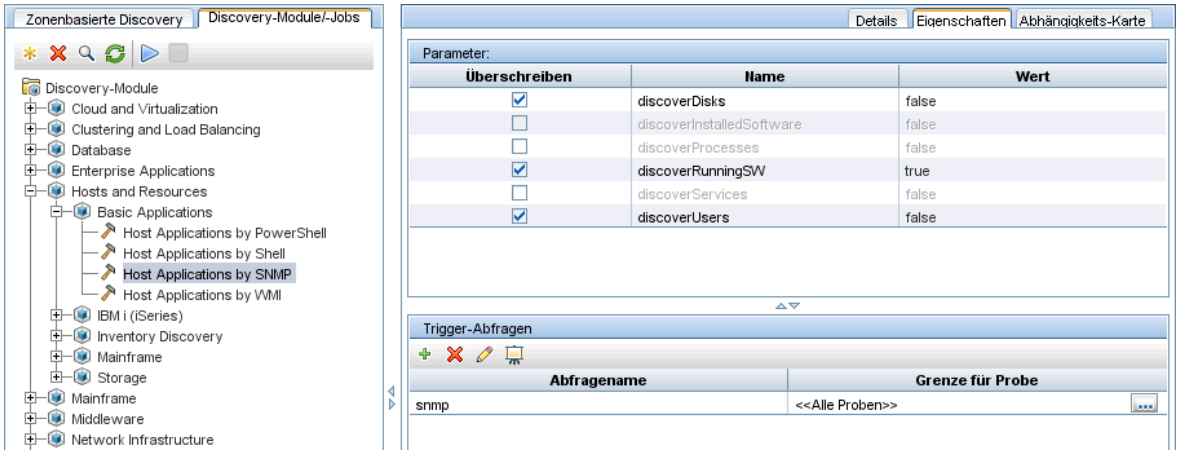
5. Konfigurieren eines Klassenpfads für einen Remoteprozess – optional

   Weitere Informationen finden Sie unter "Konfigurieren der Remoteprozess-Ausführung" auf Seite 36.

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 in der Strukturansicht unter **Discovery-Module** angezeigt. Diese Entität wird vom Benutzer aktiviert. Siehe hierzu die untenstehende Abbildung.



Auswählen einer Trigger-TQL

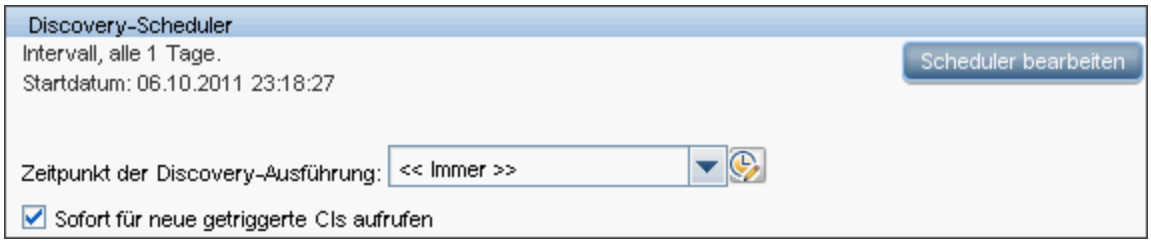
Jedem Job sind Trigger-TQLs zugeordnet. Diese Trigger-TQLs veröffentlichen Ergebnisse, die als Eingabe-Trigger-CIs für den Adapter dieses Jobs verwendet werden.

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 Discovery Scheduler Dialog Box 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 32.

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 37.

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 **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:

<UCMDB-Installationsverzeichnis>\UCMDBServer\deploymdb-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 eine Verbindung zu Computern mittels SNMP 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 Universal Discovery geschrieben werden, sollten mit der Jython-Version 2.5.3 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 erforderlich, die als Java-Dienstprogrammarchive, Verbindungsarchive (z. B. JDBC-Treiber-JAR-Dateien) oder ausführbare Dateien (**nmap.exe** wird beispielsweise für die Durchführung einer Discovery ohne Anmeldeinformationen benötigt) verwendet werden.

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 dem 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:

```
#   Importe
from appilog.common.system.types import ObjectStateHolder
from appilog.common.system.types.vectors import ObjectStateHolderVector
# Funktionsdefinition
def foo:
    # do something
#   Hauptfunktion
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:

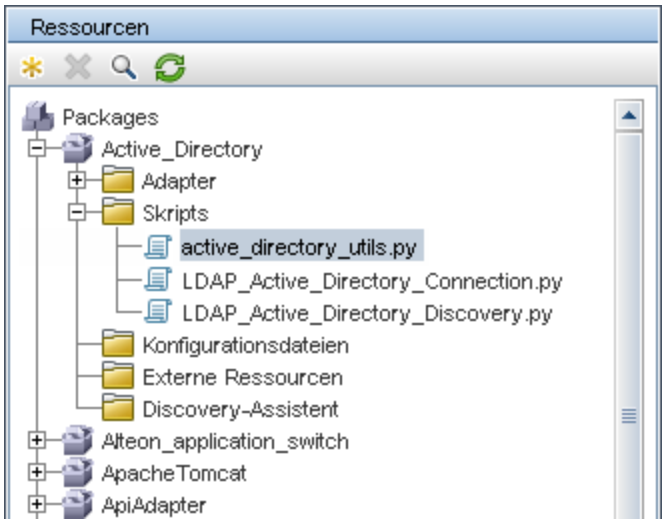
```
def DiscoveryMain(Framework):
```

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.createClient(Framework.getTriggerCIData
(BaseClient.CREDENTIALS_ID))
    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" auf Seite 62.

Senden großer Datenmengen

Das Senden großer Datenmengen (in der Regel mehr als 20 KB) ist in der UCMDB schwierig zu verarbeiten. Datenmengen dieser Größenordnung sollten vor dem Senden an die UCMDB in kleinere Chunks aufgeteilt werden. Damit alle Chunks korrekt in die UCMDB eingefügt werden, muss jeder Chunk die erforderlichen Identifikationsinformationen für die CIs im Chunk enthalten. Dies ist ein übliches Szenario bei der Entwicklung von Jython-Integrationen. Um die Ergebnisse in Chunks zu senden, wird die Methode sendObjects verwendet. Wenn das Jython-Skript eine große Anzahl von Ergebnissen sendet (der Standardwert ist 20.000, er kann jedoch in der Datei "DataFlowProbe.properties" mittels des Schlüssels **appilog.agent.local.maxTaskResultSize** konfiguriert werden), sollte es die Ergebnisse entsprechend ihrer Topologie in Chunks aufteilen. Die Bildung von Chunks sollte unter Berücksichtigung von Identifikationsregeln erfolgen, damit die Ergebnisse korrekt in die UCMDB eingegeben werden. Wenn das Jython-Skript die Ergebnisse nicht in Chunks aufteilt, versucht die Probe dies zu tun, was allerdings bei einem großen Ergebnissatz die Leistung beeinträchtigen kann.

Hinweis: Chunks sollten für Jython-Integrationsadapter verwendet werden und nicht für reguläre Discovery-Jobs. Der Grund hierfür ist, dass Discovery-Jobs in der Regel Informationen in Bezug auf einen bestimmten Trigger ermitteln und keine großen Datenmengen senden. Bei Jython-Integrationen werden bei einem einzelnen Trigger der Integration große Datenmengen ermittelt.

Es ist auch möglich, eine kleine Anzahl von Ergebnissen in Chunks aufzuteilen. In einem solchen Fall besteht eine Beziehung zwischen den CIs in den verschiedenen Chunks und dem Entwickler des Jython-Skripts stehen zwei Optionen zur Verfügung:

- Erneutes Senden des vollständigen CI sowie aller zugehörigen Identifikationsinformationen in jedem Chunk, der einen Link auf das CI enthält.
- Verwenden der UCMDB-ID des CI. Zu diesem Zweck muss das Jython-Skript warten, bis jeder Chunk im UCMDB-Server verarbeitet wurde, um die UCMDB-IDs zu erhalten. Um diesen Modus (die sogenannte synchrone Ergebnisübertragung) zu aktivieren, müssen Sie das Tag **SendJythonResultsSynchronously** zum Adapter hinzufügen. Durch dieses Tag wird sichergestellt, dass die UCMDB-IDs der CIs im Chunk bereits von der Probe empfangen wurden, wenn der Sendevorgang des Chunk abgeschlossen ist. Der Adapterentwickler kann die UCMDB-IDs für die Generierung des nächsten Chunk verwenden. Um die UCMDB-IDs verwenden zu können, benötigen Sie die Framework-API **getIdMapping**.

Beispiel für die Verwendung von "getIdMapping"

Im ersten Chunk senden Sie Knoten. Im zweiten Chunk senden Sie Prozesse. Der Root-Container des Prozesses ist ein Knoten. Anstatt die gesamte **objectStateHolder**-Instanz des Knotens im Attribut **process root_container** zu senden, können Sie mithilfe der API **getIdMapping** die UCMDB-ID des Knotens abrufen und im Attribut **process root_container** nur die Knoten-ID verwenden, um den Chunk zu verkleinern.

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 37.

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:

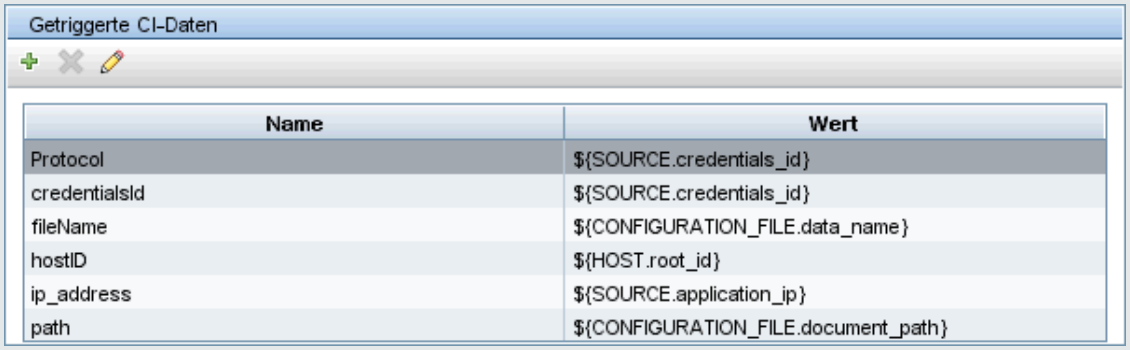
- "Framework.getTriggerCIData(String attributeName)" unten
- "Framework.createClient(credentialsId, props)" unten
- "Framework.getParameter (parameterName-Zeichenkette)" auf Seite 47
- "Framework.reportError(Zeichenfolgennachricht) und Framework.reportWarning (Zeichenfolgennachricht)" auf Seite 47

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 UCMDB Discovery and Integrations 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.

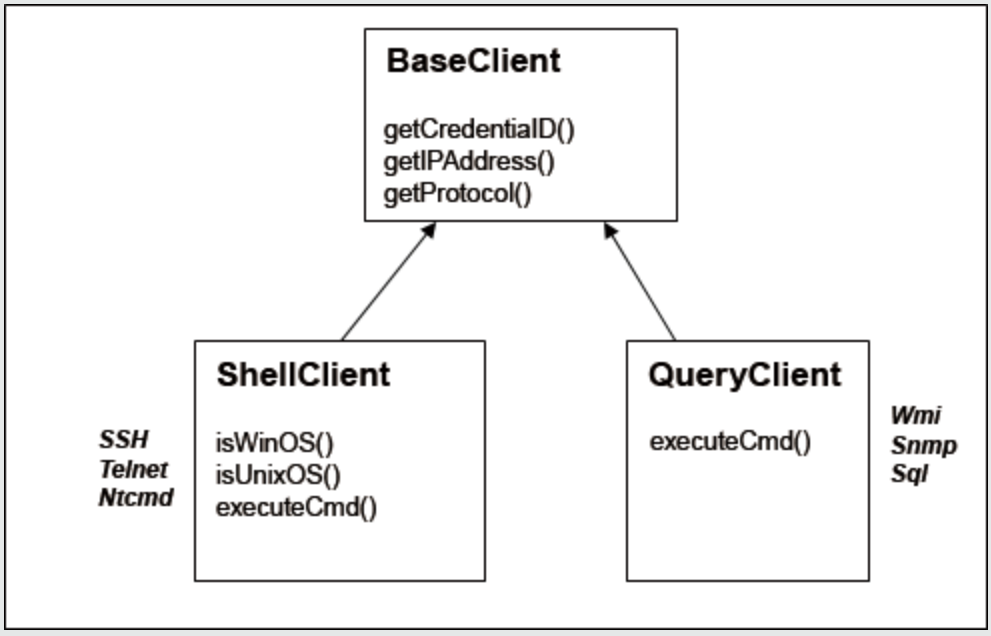
Beispiel für das Erstellen eines WMI-Client und das Ausführen einer WMI-Abfrage:

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:

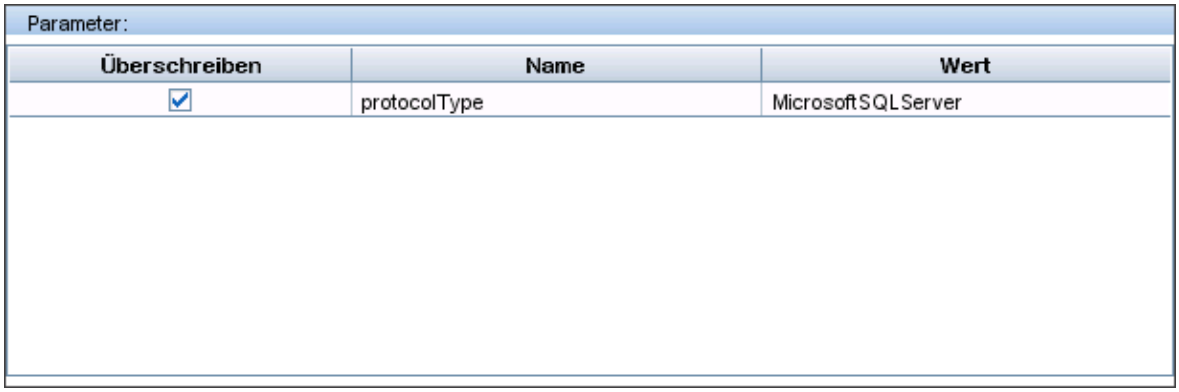


Weitere Informationen zu den Clients und ihren unterstützten APIs finden Sie unter BaseClient, ShellClient und QueryClient im DFM Framework. Die Dateien befinden sich im folgenden Ordner:

<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.createClient(Framework.getTriggerCIData
(BaseClient.CREDENTIALS_ID))

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 64.

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 erforderlich sind, ist die Anmeldeinformationen-ID. Das Verbindungsskript erhält Zugriff auf die möglichen Sätze mit Anmeldeinformationen und probiert diese unter Verwendung der Methode `Framework.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).

Hinweis: Der Zugriff auf sensible Daten (Kennwörter, private Schlüssel usw.) ist für die folgenden Protokolltypen blockiert:

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

Die Nutzung dieser Protokolltypen sollte mittels dedizierter Clients erfolgen.

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 23.

Beispiel

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

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

```
TRUE = 1
FALSE = 0

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

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

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

          connected = 0
          # Go over all the protocol credentials
          for protocol in protocols:
              client = None
              try:
                  try:
                      logger.debug('try to get snmp agent for: %s:%s' % (ip_
          address, ip_domain))
                      if (isClient == TRUE):
                          properties = Properties()
                          properties.setProperty
          (CollectorsConstants.DESTINATION_DATA_IP_ADDRESS, ip_address)
                          properties.setProperty
          (CollectorsConstants.DESTINATION_DATA_IP_DOMAIN, ip_domain)
                          client = Framework.createClient(protocol, properties)
                      else:
                          properties = Properties()
                          properties.setProperty
          (CollectorsConstants.DESTINATION_DATA_IP_ADDRESS, ip_address)
                          client = Framework.createClient(protocol, properties)
                      logger.debug('Running test connection queries')
                      testConnection(client)
                      Framework.saveState(protocol)
                      logger.debug('got snmp agent for: %s:%s' % (ip_address,
          ip_domain))
                      isMultiOid = client.supportMultiOid()
```

```
                    logger.debug('snmp server isMultiOid state=%s'
          %isMultiOid)

                   client.close()
                   client = None

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

        return (_vector, error)
```

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 an die Benutzeroberfläche ausgeben.

Hinweis: Es ist sehr wichtig, eine Java-basierte Ausnahmeklasse wie im folgenden Beispiel gezeigt zu importieren, weil in Python eine Basisausnahmeklasse gleichen Namens vorhanden ist.

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

Fehlerbehebung bei der Migration von Jython-Version 2.1 auf 2.5.3

Universal Discovery verwendet jetzt die Jython-Version 2.5.3. Alle vordefinierten Skripts wurden ordnungsgemäß migriert. Wenn Sie vor diesem Upgrade Ihre eigenen Jython-Skripts für Discovery

entwickelt haben, können die folgenden Probleme auftreten, die wie beschrieben behoben werden müssen.

Hinweis: Diese Änderungen sollten nur von erfahrenen Jython-Entwicklern vorgenommen werden.

Zeichenkettenformatierung

- **Fehlermeldung:** `TypeError: int argument required`
- **Mögliche Ursache:** Verwendung der Zeichenkettenformatierung für eine dezimale Ganzzahl von einer Zeichenkettenvariablen, die ganzzahlige Daten enthält.
- **Problematischer Jython 2.1-Code:**

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

- **Korrekter Jython 2.5.3-Code:**

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

oder

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

Überprüfen des Zeichenkettentyps

Der folgende Code funktioniert möglicherweise nicht einwandfrei, wenn die Eingabe Unicode-Zeichenketten enthält:

- **Problematischer Jython 2.1-Code:** `isinstance(unicodeStringVariable,'')`
- **Korrekter Jython 2.5.3-Code:** `isinstance(unicodeStringVariable,basestring)`

  Der Vergleich sollte mit `basestring` durchgeführt werden, um zu testen, ob ein Objekt eine Instanz von `str` oder Unicode ist.

Nicht-ASCII-Zeichen in Datei

- **Fehlermeldung:**
  `SyntaxError: Non-ASCII character in file 'x', , but no encoding declared; see http://www.python.org/peps/pep-0263.html for details`
- **Korrekter Jython 2.5.3-Code:** (Fügen Sie diesen Code zur ersten Zeile in der Datei hinzu)

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

Importieren von Sub-Packages

- **Fehlermeldung:**
  `AttributeError: 'module' object has no attribute 'sub_package_name'`
- **Mögliche Ursache:** Ein Sub-Package wurde importiert, ohne dass sein Name explizit in der Importanweisung angegeben wurde.
- **Problematischer Jython 2.1-Code:**

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

Das Sub-Package wurde nicht explizit importiert.

- **Korrekter Jython 2.5.3-Code:**

  ```
  import a.b
  ```

  oder

  ```
  from a import b
  ```

Iterator-Änderungen

Ab Jython 2.2 wird die Methode **__iter__** verwendet, um eine Schleife über einer Sammlung im Bereich eines **for-in**-Blocks auszuführen. Der Iterator sollte die Methode **next** implementieren, die ein entsprechendes Element zurückgibt oder den Fehler **StopIteration** auslösen, falls das Ende der Sammlung erreicht wurde. Wenn die Methode **__iter__** nicht implementiert ist, wird stattdessen die Methode **GetItem** verwendet.

Auslösen von Ausnahmen

- **Die Jython 2.1-Methode zum Auslösen von Ausnahmen ist veraltet:**
  ```
  raise Exception, 'Failed getting contents of file'
  ```
- **Die empfohlene Jython 2.5.3-Methode zum Auslösen von Ausnahmen lautet wie folgt:**
  ```
  raise Exception('Failed getting contents of file')
  ```

Lokalisierungsunterstützung in Jython-Adaptern

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

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 57. 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 Seite 52.

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 331.

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 in dem Schritt über das Abrufen eines Schlüsselworts

aus dem Ressource-Bundle unter "Definieren eines neuen Jobs für die Ausführung mit lokalisierten Daten" auf Seite 54.

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**.

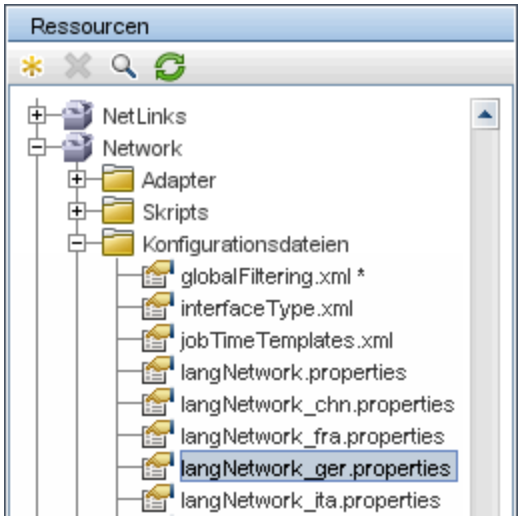
Der reguläre Anwendungsfall sieht folgendermaßen aus:

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 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

Name	Beschreibung
	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" auf der vorherigen Seite.

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

Das osLanguage-Feld

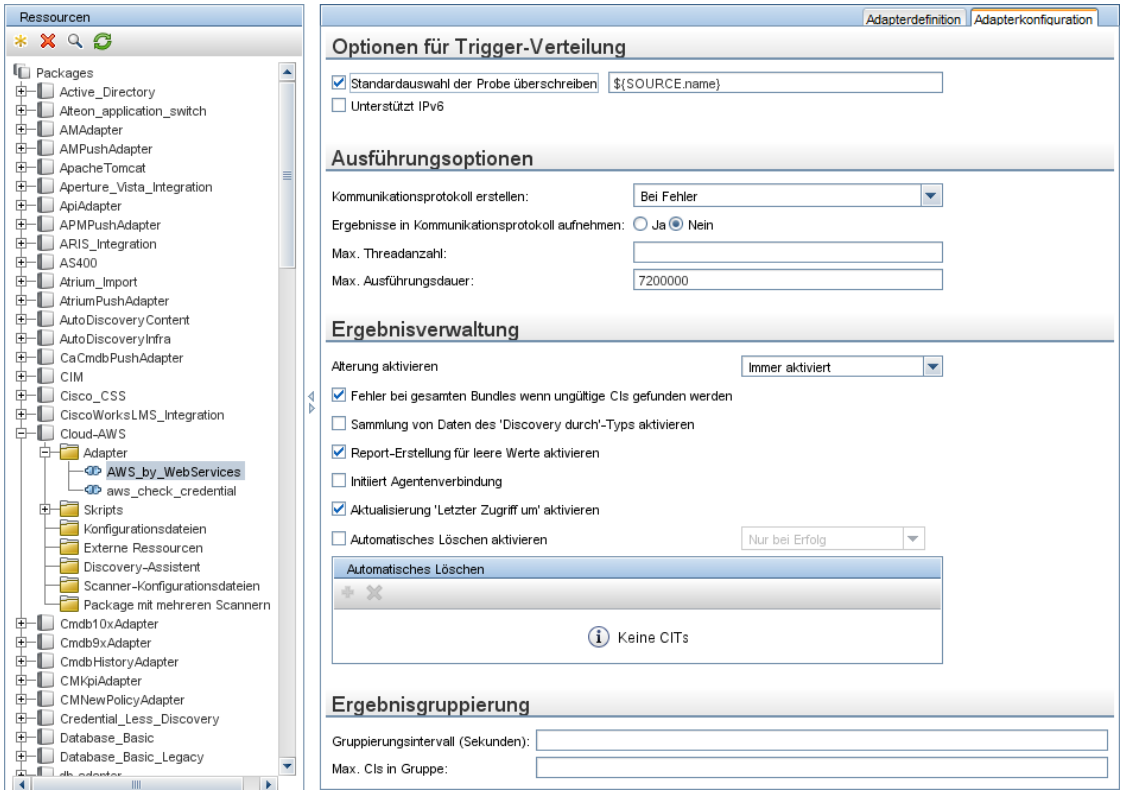
Dieses Feld enthält ein Objekt, das die Sprache repräsentiert.

Aufzeichnen von DFM-Code

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.

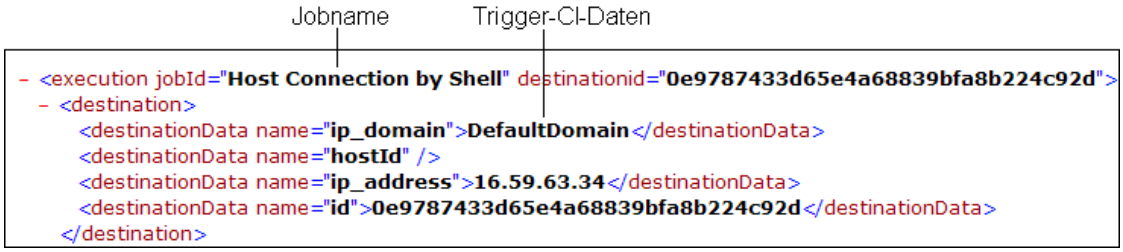
So zeichnen Sie DFM-Code auf:

1. Klicken Sie auf **Datenflussverwaltung > Universal Discovery**. 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**.

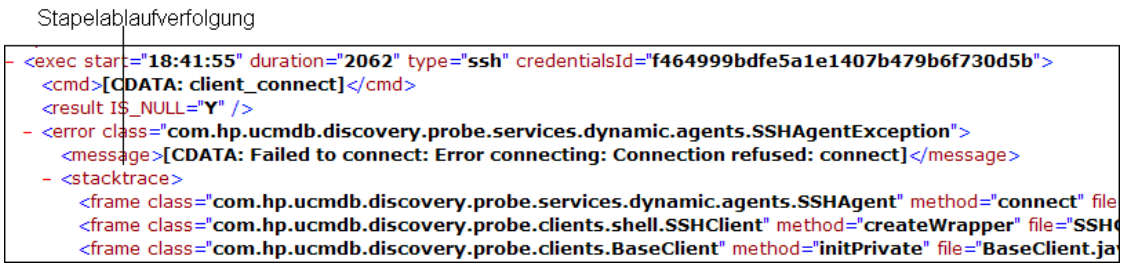


3. Ändern Sie die Einstellung des Felds **Kommunikationsprotokoll erstellen** auf **Immer**. Weitere Informationen zum Einstellen der Protokollierungsoptionen finden Sie unter Ausschnitt "Ausführungsoptionen" im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

   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

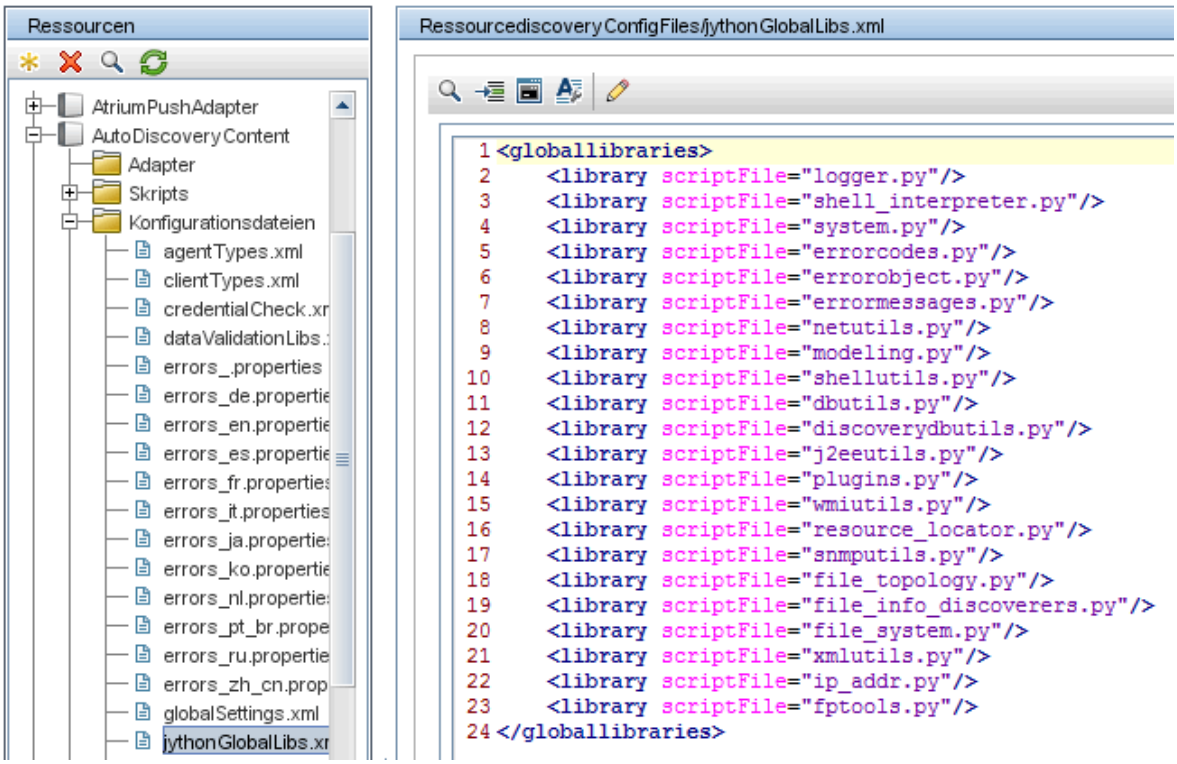
In Adaptern werden häufig mehrere Dienstprogrammskripts verwendet. Diese Skripts sind Teil des `AutoDiscovery`-Package, das in folgendem Ordner gespeichert ist: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts**. In diesem Ordner befinden sich auch die anderen Skripts, die zur Probe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

- "netutils.py" auf der nächsten Seite
- "shellutils.py" auf der nächsten Seite

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 67.

```
#########################################################################
################         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 = Framework.createClient(Framework.getTriggerCIData
(BaseClient.CREDENTIALS_ID), 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. Diese Fehler werden im Fenster **Universal Discovery** angezeigt, 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" unten.

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 "Discovery-Fortschritt und Ergebnisse" 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

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: Zuordnen von Benutzer-Provider-Abhängigkeiten

Dieses Kapitel umfasst folgende Themen:

Übersicht über das Abhängigkeiten-Discovery

Die Abhängigkeitszuordnung bietet eine flexible Methode zur Ermittlung der Beziehungen zwischen bereitstellbaren Komponenten oder aktiver Software. Diese Methode ermöglicht es, mit benutzerdefinierten Regeln für die Abhängigkeitszuordnung zu arbeiten (unter Verwendung einfacher Programmiersyntax), die der Universal Discovery-Prozess verwendet, um automatisch Abhängigkeiten zu ermitteln.

Ein Service kann entweder ein Geschäfts- oder ein IT-Service sein. Ein Geschäftsservice ist ein Service, den ein Geschäftsunternehmen einem anderen bereitstellt (wird auch als Firmenkundengeschäft oder B2B bezeichnet) bzw. den eine Organisation innerhalb einer Geschäftsabteilung einer anderen bereitstellt (zum Beispiel Zahlungsverarbeitung). Ein IT-Service ist ein Geschäftsservice, den eine IT-Organisation bereitstellt, um Geschäftsservices oder eigene Operationen aus der IT zu unterstützen.

Eine bereitstellbare Komponente ist eine Softwarekomponente, die innerhalb einer aktiven Software bereitgestellt wird, zum Beispiel ein Applikations- oder Webserver. Beispiele für bereitstellbare Komponenten sind JEE EAR-Komponenten oder ein Schema innerhalb einer Oracle-Datenbank. Zum Zweck der Abhängigkeiten-Discovery wird aktive Software als eine bereitstellbare Komponente betrachtet.

Eine durch den **Provider** bereitstellbare Komponente liefert einen Service und deklariert wie andere bereitstellbaren Komponenten diesen Service benutzen können. Eine durch den **Benutzer** bereitstellbare Komponente "benutzt" einen Service, der von einer durch den Provider bereitstellbaren Komponente zur Verfügung gestellt wurde. Die Abhängigkeit zwischen diesen bereitstellbaren Komponenten ist eine **Benutzer-Provider**-Abhängigkeit.

Hinweis:

- Damit Sie in der Lage sind, Benutzer-Provider-Abhängigkeitsadapter zu erstellen und das Framework für Abhängigkeitszuordnung zu verwenden, müssen Sie die Option **Suche aktivieren** auswählen, wenn Sie während der Installation das UCMDB-Schema einrichten.
- Adapter für Benutzer-Provider-Abhängigkeiten können nur für Data Flow Probes im gemeinsamen Modus ausgeführt werden.

Weitere Informationen finden Sie unter:

Provider und Benutzer

Sie stellen über Verbindungszeichenketten eine Verbindung zu Providern her. Wenn zum Beispiel eine Oracle-Datenbank ein Provider ist, benötigen Sie unter Umständen Folgendes, um eine Verbindung zu den Services der Datenbank herzustellen:

- die IP-Adresse des Computers
- die SID
- den TCP-Port

Diese drei Informationen wären die Bestandteile der Verbindungszeichenketten, die von einem Benutzer benötigt werden, damit er eine Verbindung zu dem Service herstellen kann, den dieser Provider anbietet. Eine Oracle-Verbindungszeichenkette könnte beispielsweise folgende Informationen enthalten:

- IP-Adressen: 1.1.1.1, 2.2.2.2
- Port: 1521
- SID: abcd

Einem Benutzer ist mindestens eine Verbindungszeichenkette für einen Provider bekannt und diese Verbindungszeichenkette ist an einem bekannten Speicherort, zum Beispiel in einem Konfigurationsdokument, in einer Datenbanktabelle oder in der Windows-Registrierung, hinterlegt. Indem diese Speicherorte durchsucht werden, werden Abhängigkeiten zwischen Benutzern und Providern ermittelt.

Wenn die Verbindungszeichenketten eines Providers in einem bestimmten Konfigurationsdokument gefunden werden, werden der Provider und der Container des Konfigurationsdokuments über eine Benutzer-Provider-Beziehung miteinander verbunden.

Der Prozess der Ermittlung von Benutzer-Provider-Abhängigkeiten ist dann einfach: Verbindungszeichenketten vom Provider werden in den Konfigurationsdokumenten des Benutzers gesucht und die Suchergebnisse weisen alle Konfigurationsdokumente aus, die sich im Besitz der Benutzer des angegebenen Providers befinden.

Weitere Informationen finden Sie unter "Definieren von Abhängigkeiten" auf Seite 78.

Abhängigkeitssignaturen

Es kann für jedes Konfigurationsdokument und jeden Provider-Typ ein anderer Suchbegriff verwendet werden. Diese Suchbegriffe werden in einer Abhängigkeitssignaturdatei definiert.

Eine Abhängigkeitssignatur ist eine Regel, die definiert, ob eine Benutzer-Provider-Abhängigkeit zwischen einem bestimmten Provider und einem bestimmten Benutzer existiert. Dazu werden die Verbindungszeichenkette des Providers und der Konfigurationsdokumente des Benutzers verwendet.

Eine Abhängigkeitssignatur besteht aus Suchausdrücken. Diese Suchausdrücke sind von der Definition der Verbindungszeichenketten abhängig, nicht von den tatsächlichen Werten, die für einen bestimmten

Provider gelten. Darüber hinaus hängen die Suchausdrücke von dem Namen, Speicherort und Format der Konfigurationsdokumente des Benutzers ab, nicht jedoch vom tatsächlichen Inhalt der Dateien. Wenn die Verbindungszeichenketten des Providers bekannt sind, werden sie in die Suchausdrücke eingegeben. Daraus resultieren dann konkrete Suchausdrücke. Die konkreten Suchausdrücke werden dann unter Verwendung der Konfigurationsdokumente des Benutzers evaluiert. Die Suchausdrücke geben nur dann "True" zurück wenn in den Konfigurationsdokumenten des Benutzers auf eine bestimmte Weise eine Provider-Verbindungszeichenkette hinterlegt ist.

Weitere Informationen finden Sie unter "Dateien der Abhängigkeitssignaturen" unten.

Dependency Mapping-Flow

Dieser Abschnitt bietet einen kurzen Überblick über den grundlegenden Flow, der während der Abhängigkeitszuordnung ausgeführt wird:

1. Die bereitstellbaren Komponenten und deren Verbindungszeichenketten werden ermittelt.
2. Alle Typen der durch den Provider bereitstellbaren Komponenten lösen einen bestimmten Dependency Mapping-Job aus. Jeder Adapter des Jobs ist in der Lage die relevante Verbindungszeichenkette für den bestimmten Typ der bereitstellbaren Komponente zu extrahieren. Der Adapter sucht nach anderen bereitstellbaren Komponenten, die diesen Service benutzen.
3. Zwischen jeder gefundenen bereitstellbaren Komponente und ihrem Trigger (Provider) wird eine Benutzer-Provider-Beziehung erstellt.

Dateien der Abhängigkeitssignaturen

Dieser Abschnitt umfasst Folgendes:

Aufbau der Abhängigkeitssignaturdatei

Eine Abhängigkeitssignaturdatei definiert eine oder mehrere Abhängigkeiten zwischen bereitstellbaren Komponenten.

Der Benutzerteil der Abhängigkeit ist als das `<Deployable>`-Element definiert. Jedes `<Deployable>`-Element kann eines oder mehrere `<Dependency>`-Elemente enthalten. Jedes `<Dependency>`-Element enthält wiederum die Bedingungen unter denen eine Abhängigkeit zwischen einem Benutzer (einem `<Deployable>`-Element) und einem Provider besteht.

Das `<Dependency>`-Element kann als boolesche Funktion betrachtet werden, die als Eingabe das Provider-CI und die Konfigurationsdokumente des Benutzers verwendet. Es wird nur dann "True" zurückgegeben, wenn eine Abhängigkeit zwischen diesen beiden CIs vorhanden ist, und zwar vom Benutzer zum Provider.

Im `<Dependency>`-Element ist der Provider in der Datei nur über seinen CI-Typ angegeben. Jeder Abhängigkeit kann für einen einzelnen CI-Typ eines Provider verwendet werden. Das folgende Beispiel

definiert eine Abhängigkeitsfunktion zwischen einem beliebigen Benutzer, in diesem Fall eine aktive Software, und einem Provider, in diesem Fall ebenfalls eine aktive Software:

```
<Deployable name="ApolloOnNod">
   <Descriptor cit="running_software">
   </Descriptor>
   <Dependency name="history_db" providerCiType="running_software"
scope="default">
      …
```

Variablen und Konzepte

Eine Variable kann während der Ausführung des Programms unterschiedliche Werte enthalten. Im Falle der Zuordnung von Abhängigkeiten kann eine Variable unterschiedliche Werte für verschiedene Abhängigkeitssuchen enthalten. Diese Variablen werden bei der Definition von Suchausdrücken verwendet, um zu bestimmen, ob eine Verbindungszeichenkette in einem Konfigurationsdokument vorhanden ist. Da die Verbindungszeichenketten sich je nach Provider unterscheiden, ermöglichen Variablen die Definition generischer Suchausdrücke, unabhängig von den spezifischen Werten der Verbindungszeichenketten.

Variablen

Hinweis: Der Begriff "Variable" bezeichnet sowohl Variablen als auch Konzeptvariablen.

Die Syntax für die Verwendung einer Variablen lautet `${VARIABLE_NAME}`. Jeder Variablenwert muss entweder ein Zeichenkettenwert oder eine Zeichenkettenliste sein. Ein Teil der Verbindungszeichenkette kann zum Beispiel aus der IP-Adresse (oder den IP-Adressen) des Providers bestehen. Um im Konfigurationsdokument nach einer IP-Adresse zu suchen, können Sie eine Variable namens IP_ADDRESS definieren und diese folgendermaßen innerhalb des Ausdrucks verwenden: `${IP_ADDRESS}`.

Variablen müssen zunächst definiert werden. Dabei müssen sie sich entweder auf die gesamte Abhängigkeitssignaturdatei (globaler Gültigkeitsbereich) oder auf den Gültigkeitsbereich einer Abhängigkeit (lokaler Gültigkeitsbereich) beziehen. Die Definition der Variablen ermöglicht es Ihnen, diese Variable zu verwenden.

Variablen, die ohne vorherige Definition verwendet werden, generieren einen Fehler, wenn sie in der Signaturdatei bereitgestellt werden.

Globaler Gültigkeitsbereich

Eine Variable für einen globalen Gültigkeitsbereich kann von jeder Abhängigkeit innerhalb der Datei verwendet werden, in der sie definiert wurde. So definieren Sie eine Variable für einen globalen Gültigkeitsbereich:

```
<DependencySignatures xmlns="http://www.hp.com/ucmdb/1-0-0/Dependencies">
   <VariableDeclarations>
      <Variable name="IPADDRESS"/>
```

```
      <Variable name="PROTOCOL"/>
</DependencySignatures>
```

Die oben dargestellte Syntax definiert zwei Variablen für einen globalen Gültigkeitsbereich: IPADDRESS und PROTOCOL. Wie alle Variablen können sie beliebige Zeichenkettenwerte oder -listen enthalten.

Werte globaler Variablen können nur über die Zieldaten eines Triggers eingerichtet werden.

Lokaler Gültigkeitsbereich

Eine Variable für einen lokalen Gültigkeitsbereich kann nur für die Abhängigkeit verwendet werden, für die sie definiert wurde. Für jede andere Abhängigkeit ist die Variable nicht sichtbar, unabhängig davon, ob es sich um dieselbe bereitstellbare Komponente handelt oder nicht.

Sie können mehrere Variablen für einen lokalen Gültigkeitsbereich mit demselben Namen in unterschiedlichen Abhängigkeiten definieren. Jede dieser Variablen ist eigenständig und und ihre Werte können nur innerhalbe des Gültigkeitsbereichs verwendet werden, für den sie definiert wurde.

Hinweis: Eine Variable für einen lokalen Gültigkeitsbereich kann nicht denselben Namen haben wie die für einen globalen Gültigkeitsbereich. Ebenso ist es nicht möglich, zwei Variablen für einen lokalen Gültigkeitsbereich zu definieren, die im Kontext einer Abhängigkeit über denselben Namen verfügen.

Verwenden Sie die folgende Syntax, um eine Variable für einen lokalen Gültigkeitsbereich zu definieren:

```
<Deployable name="StrongXmlApplication">
   <Descriptor cit="cluster_software"/>
   <Dependency name="app_cluster" providerCiType="running_software"
scope="default">
      <VariableDeclarations>
         <Variable name="IPADDRESS"/>
         <Variable name="PROTOCOL"/>
      </VariableDeclarations>
      …
   </Dependency>
</Deployable>
```

Variablen für einen lokalen Gültigkeitsbereich können Werten nur unter Verwendung von Einfügeanweisungen zugeordnet werden. Abhängig vom Typ der Datei, aus der der Wert extrahiert wird, gibt es drei verschiedene, spezifische Einfügeanweisungen. Eine Einfügeanweisung kann an zwei Stellen im Konfigurationsdokument angezeigt werden.

- In einem dedizierten `<Variables>`-Abschnitt. Die Einfügeanweisungen im `<Variables>`-Abschnitt werden nur evaluiert, wenn der gesamte Suchausdruck der Datei als "True" evaluiert wurde.
- In einem `<Variables>`-Abschnitt als Bestandteil einer Suchbedingung. Mit dieser Option wird die Variable nur dann eingefügt, wenn die Suchbedingung als "True" evaluiert wurde. Diese Option wird nicht unterstützt, wenn alternative Ausdrücke verwendet werden. Weitere Informationen finden Sie unter "Verwenden von Standardwerten für Variablen" auf Seite 81.

Weitere Informationen zum Zuweisen von Variablen finden Sie unter:

- "Konfigurationsdokumente für Eigenschaften" auf Seite 89
- "XML-Konfigurationsdokumente" auf Seite 92
- "Textkonfigurationsdokumente" auf Seite 95

Konzepte

Suchausdrücke werden auf jedes relevante Konfigurationsdokument angewendet. Einige Werte in den Verbindungszeichenketten sind miteinander gekoppelt und sollten nur gemeinsam in konkreten Suchausdrücken verwendet werden. Jeder Satz gekoppelter Verbindungszeichenketten muss über einen eindeutigen Namen verfügen und wird als "Konzept" bezeichnet.

> Auf eine durch den Provider bereitstellbare Komponente kann zum Beispiel über `1.1.1.1:8080` oder über `2.2.2.2:85` zugegriffen werden. Jeder Benutzer des Providers muss somit in einer seiner Konfigurationsdateien den Eintrag `1.1.1.1 :8080` oder `2.2.2.2 :85` führen. Allerdings ist es unwahrscheinlich, dass der Eintrag `1.1.1.1:85` im Konfigurationsdokument eines beliebigen Benutzers dieses Providers enthalten ist, da dieser Provider nicht den Port mit der IP-Adresse `1.1.1.1:85` abhört. Selbst wenn der Eintrag `1.1.1.1 :85` in einem Konfigurationsdokument eines Benutzers gefunden wird, stellt er keine Abhängigkeit mit diesem Provider dar, sondern eine Abhängigkeit mit einer anderen bereitstellbaren Komponente, die auf dem gleichen Knoten ausgeführt wird wie der Provider, und den Port mit der IP-Adresse `1.1.1.1 :85` abhört.
>
> Die Suche muss korrekte Übereinstimmungen für IP-Adresse und Port ermitteln, da die Suche andernfalls falschpositive Abhängigkeiten zurückgeben würde.
>
> Darüber hinaus muss die Suche auch korrekte Übereinstimmungen für den Namen aller IP-Adressen und Ports ermitteln, da jede IP-Adresse über mehrere Namen verfügen kann (zum Beispiel über einen maßgeblichen DNS-Namen und mehrere Aliasnamen).
>
> Aus diesem Grund werden wahrscheinlich die folgenden Übereinstimmungen oder Paare in den Konfigurationsdokumenten des Benutzers dieses Providers ermittelt: `XYZ:8080`, `FOO:8080` oder `ABC:85` (wobei `XYZ`, `FOO` und `ABC` andere Namen für die Provideradresse sind), da diese Übereinstimmungen für die Abhängigkeiten mit diesem Provider stehen. Die Übereinstimmung `ABC:8080` steht jedoch nicht für eine Abhängigkeit mit dem Provider und deshalb sollte eine solche Übereinstimmung nicht gesucht werden.

Zu diesem Zweck werden Konzepte verwendet. Es empfiehlt sich daher, jedes Verbindungszeichenkettenattribut, das in Kombination mit einem anderen Attribut verwendet werden soll, in demselben Konzept zu definieren. Jedes Verbindungszeichenkettenattribut sollte in diesem Konzept eine Variable sein.

Konzepte können nur für einen globalen Gültigkeitsbereich definiert werden. Für jede Konzeptinstanz die einen Satz eng gekoppelter Verbindungszeichenketten darstellt, müssen die Werte über den Adapter festgelegt werden, ähnlich wie im Fall von Variablen.

Jedes Konzept besteht aus genau einer Schlüsselvariablen und zusätzlichen Variablen. Jede dieser Variablen (Schlüsselvariable und zusätzliche Variablen) werden verwendet, um die gekoppelten Verbindungszeichenketten zu speichern. Die Schlüsselvariable wird verwendet, um zwischen verschiedenen Instanzen desselben Konzepts zu unterscheiden. Dies bedeutet, in einem Konzept kann es nicht zwei Konzeptinstanzen geben, die in ihrer Schlüsselvariablen einen identischen Wert haben. Weitere Informationen finden Sie unter "Angeben der Werte der Konzeptvariablen" auf Seite 108.

Hinweis: Eine Schlüsselvariable kann in derselben Weise verwendet werden wie alle anderen Konzeptvariablen. Ihre Bedeutung liegt in der Art und Weise, wie Konzeptvariablen während der Ausführung eines Triggers instanziiert werden.

Um Konzepte und die zugehörigen Variablen zu definieren, verwenden Sie die folgende Syntax:

```
<Concept name="ConceptName">
   <Properties>
      <KeyProperty name="KeyVariableName"/>
      <Property name="VariableName1"/>
      <Property name="VariableName2"/>
   </Properties>
</Concept>
```

Um die Variable eines Konzepts in einem Suchausdruck zu verwenden, verwenden Sie die folgende Syntax:

```
${ConceptName.VariableName}.
```

Für jedes Konzept, das in einem Suchausdruck verwendet wird, muss ein logischer Operator vorhanden sein, der alle für das Konzept relevanten Variablen, jedoch keine Variablen anderer Konzepte enthält.

Im Folgenden finden Sie Beispiele für gültige Suchausdrücke, die Konzepte enthalten:

```
${C.A} = X AND ${X.B} = Y
(${C1.A} = X AND ${C1.B} = Y) OR ${C2.C} = Z
```

Dies ist ein Beispiel für einen ungültigen Suchausdruck:

```
${C1.A} = X AND ${C2.B} = Y AND ${C2.X} = Z
```

Standardwerte

Globale Variablen und Konzeptvariablen können optional einen Standardwert aufweisen. Wenn eine Variable aus dem Adapter mit einem Wert eingefügt wird, der mit dem Standardwert identisch ist, dann können alternative Suchausdrücke definiert werden. Weitere Informationen zu alternativen Suchausdrücken finden Sie unter "Erstellen eines Suchausdrucks" auf Seite 78.

Verwenden Sie die folgende Syntax, um einen Standardwert zu definieren:

```
<Variable name=”PORT” defaultValue=”8080” />
```

Variablen des Typs "IP-Adresse"

Eine IP-Adresse ist ein wichtiger Verbindungszeichenkettentyp. Die Adresse des Providers ist aber möglicherweise nicht immer in den Konfigurationsdateien angegeben.

Dies ist beispielsweise häufig der Fall, wenn Benutzer und Provider sich auf einer Hostmaschine befinden. In diesem Fall können anstelle der Adresse des Providers Zeichenketten wie "localhost" oder "127.0.0 .1" angezeigt werden. Möglicherweise wird auch gar keine Adresse angezeigt.

Indem eine Variable als Variable des Typs **IP-Adresse** eingerichtet wird, versucht das Framework automatisch lokale Abhängigkeiten zu finden, indem es die IP-Adressen-Variablen ignoriert und die Provider, die sich auf demselben Host befinden, anhand der verbleibenden Verbindungszeichenketten sucht.

So markieren Sie eine Variable als IP-Adresse

- Im Falle einer globalen Variablen

```
<DependencySignatures xmlns="http://www.hp.com/ucmdb/1-0-0/Dependencies">
   <VariableDeclarations>
      <Variable name="MY_VARIABLE" type=”IP Address” />
   </VariableDeclarations>
         …
</DependencySignatures>
```

- Im Falle einer Konzeptvariablen

```
<Concept name="IpEndpoint">
   <Properties>
      <KeyProperty name="PORT"/>
      <Property name="MY_VARIABLE" type=”IP Address” />
   </Properties>
</Concept>
```

Hinweis: Lokale Variablen können nicht den Typ "IP-Adresse" aufweisen.

Definieren des Deskriptors eines Benutzers

Um eine Benutzer-Provider-Abhängigkeit zu definieren sucht der Suchadapter nach Benutzern mit Konfigurationsdokumenten, die die Verbindungszeichenkette des Providers verwenden. Allerdings ist es manchmal sinnvoll, die bereitstellbaren Komponenten zu begrenzen, die als Benutzer eines bestimmten Service fungieren können. Dies gilt auch, wenn die Konfigurationsdokumente des Benutzers die Verbindungszeichenfolge enthalten oder, abhängig von den Eigenschaften der bereitstellbaren Komponente, über andere Ausgabevariablen verfügen.

Zu diesem Zweck ist es möglich, die bereitstellbare Komponente detailliert zu beschreiben und nicht nur den CI-Typ unter Verwendung des `<Descriptor>`-Elements anzugeben. Die bereitstellbare Komponente kann unter Verwendung folgender Elemente beschrieben werden:

- CI-Typ, d. h., die Abhängigkeiten sind nur relevant für bereitstellbare Komponenten vom Typ "J2EE-Applikation". (Erforderlich)
- Bedingungen für Zeichenkettenattribute, beispielsweise für bereitstellbare Komponenten vom Typ "J2EE-Applikation", die als Applikationsnamen "MyShop" verwenden. Die Bedingung kann nur auf Gleichheit testen. (Optional)

- Ein erforderlicher Verbundlink zu einem anderen CI. Wenn der Verbundlink im Deskriptor angegeben ist, muss die bereitstellbare Komponente über einen Verbundlink zu einem CI des jeweiligen Typs verfügen. (Optional)
- Bedingungen für Zeichenketteattribute des verbundenen CIs, wenn eine Bedingung festgelegt wurde (wie in der vorherigen Option). (Optional)
- Ein Pfad (TQL-Abfrage) zu dem Knoten, der die bereitstellbare Komponente enthält. Der Name der Abfrage wird mit dem Attribut **nodeToDeployableQuery** angegeben. Wenn der Knoten direkt über einen Verbundlink mit der bereitstellbaren Komponente verbunden ist, besteht keine Notwendigkeit eine TQL-Abfrage als Pfad anzugeben. Wenn der Knoten jedoch noch verwendet wird, um die bereitstellbare Komponente zu beschreiben (mit oder ohne Bedingungen), müssen Sie ihn weiterhin mithilfe des Tags `<ConnectedCICondition>` angeben. Wenn für die bereitstellbare Komponente an keiner Stelle der übergeordneten Hierarchie ein Knoten als Bereitstellungsort angegeben wurde, geben Sie ihn über das Attribut `hasContainingNode = “false”` an. Wenn Sie den Pfad angeben, stellen Sie sicher, auch das Attribut `hasContainingNode = "true"` hinzuzufügen. Weitere Informationen finden Sie unter "Definieren von TQL-Abfragen" auf Seite 101. (Optional)

Hinweis: Auf bereitstellbaren Deskriptoren oder verbundenen CIs wird nur der Attributtyp STRING unterstützt. Das Attribut kann nicht statisch sein und kann keine berechneten Attribute enthalten.

Beispiel für einen Deskriptor, der nur einen CI-Typ angibt:

```
<Deployable name="ApolloOnNode_2">
   <Descriptor cit="running_software">
   </Descriptor>
      …
```

Beispiel für einen Deskriptor mit einem Zeichenkettenattribut:

```
<Deployable name="ApolloOnCluster">
   <Descriptor cit="node">
      <Attribute name="default_gateway_ip_address_type" value="IPv6" />
   </Descriptor>
      …
```

Beispiel für einen Deskriptor mit einem verbundenen CI zum Testen der Gleichheit ohne Berücksichtigung der Groß-/Kleinschreibung:

```
<Deployable name="MyRunningSoftware">
   <Descriptor cit="running_software">
      <ConnectedCiCondition cit=”node” linkType=”composition”
isDirectionForward=”true”>
         <Attribute name=”name” value=”MyNode” operator=”equalIgnoreCase” />
      </ConnectedCiCondition>
   </Descriptor>
      …
```

Hinweis: Für ConnectedCiCondition ist es möglich nur linkType=”composition” und

isDirectionForward=”true” zu verwenden.

Definieren von Abhängigkeiten

Für jeden Benutzer können Sie mehrere Abhängigkeiten definieren. Jeder Abhängigkeit ist einem bestimmten Provider-CI-Typ zugeordnet. Während der Evaluierung der Abhängigkeitssignatur für ein Provider-CI werden alle Abhängigkeiten, die dem Provider CI-Typ zugeordnet sind, evaluiert. Jede Abhängigkeit verfügt über einen Suchausdruck aus einem oder mehreren Konfigurationsdokumenten. Wenn der Suchausdruck mit "True" evaluiert wird, gibt es eine Abhängigkeit (eine Benutzer-Provider-Beziehung) zwischen dem Benutzer und dem Provider. Auch wenn mehrere Abhängigkeiten zwischen dem gleichen Benutzer und Provider CI-Typ mit "True" evaluiert werden, wird nur eine Beziehung erstellt.

Ein Beispiel für eine Abhängigkeitssyntax ist:

```
<Deployable name="Websphere J2EE Application">
   <Descriptor cit="j2eeapplication"/>
   <Dependency name="J2EE Application to DB by JNDI" providerCiType="oracle"
scope="default">
      ...
```

Jede Abhängigkeit weist außerdem einen Gültigkeitsbereich auf. Ein Gültigkeitsbereich besteht aus den Benutzern, die unter allen Benutzern, die den Elementen des Deskriptors entsprechen, für diese bestimmt Abhängigkeit relevant sind. Weitere Informationen finden Sie unter "Definieren des Deskriptors eines Benutzers" auf Seite 76.

Erstellen eines Suchausdrucks

Hinweis: Auf bereitstellbaren Deskriptoren oder verbundenen CIs wird nur der Attributtyp STRING unterstützt. Das Attribut kann nicht statisch sein und kann keine berechneten Attribute enthalten.

Ein Suchausdruck besteht aus logischen Operatoren und Bedingungen. Unterstützte logische Operatoren sind "And" und "Or".

Bedingungen sind Ausdrücke, die mithilfe von Verbindungszeichenfolgen eines Providers und Konfigurationsdokumenten eines Benutzers evaluiert werden. Bedingungen unterscheiden sich bezüglich ihres Dateityps. So enthalten Konfigurationsdokumente für Eigenschaften andere Bedingungen als XML-Dokumente.

Hier handelt es sich um ein Beispiel für einen Suchausdruck für ein Konfigurationsdokument für Eigenschaften: Um eine Abhängigkeit zwischen einem Provider und einem Benutzer zu erstellen, nur wenn die IP-Adresse des Providers im Konfigurationsdokument **MyConfig.properties** des Benutzers unter dem Schlüssel IPADDRESS vorhanden ist. Das Abhängigkeitskonfigurationsdokument würde wie folgt aussehen:

```
<PropertiesConfigurationDocument name="MyConfig.properties"> (Type and name of
the configuration document)
   <Condition> (Beginning of the search expression)
```

```
      <Operator type="and"> (Operator)

         <KeyCondition key="IPADDRESS"> (Start a condition and state its type)
            <Values>
               <Value>${IP_ADDRESS}</Value> (Use a variable stating the
provider’s IP address)
            </Values>
         </KeyCondition>
      </Operator>
   </Condition>
</PropertiesConfigurationDocument>
```

Hinweis: Es muss mindestens ein `<Operator>`-Element in einem `<Condition>`-Element vorliegen, und zwar auch dann, wenn es logisch nicht notwendig ist, da es nur eine Suchbedingung gibt, wie im oben aufgeführten Beispiel.

Werfen wir einen Blick auf einen komplexeren Suchausdruck: Entweder ist die IP-Adresse des Providers im Konfigurationsdokument **MyConfig.properties** unter dem Schlüssel IPADDRESS oder der Hostname des Providers in der gleichen Datei unter dem Schlüssel HOST aufgeführt:

```
<PropertiesConfigurationDocument name="MyConfig.properties"> (Type and Name of
the configuration document)
   <Condition> (Beginning of the search expression)
      <Operator type="or"> (Operator)
         <KeyCondition key="IPADDRESS"> (Start a condition and state its type)
            <Values>
               <Value>${IP_ADDRESS}</Value> (Use a variable stating the
provider’s IP address)
            </Values>
         </KeyCondition>
         <KeyCondition key="HOST"> (Start another condition, under the same
operator)
            <Values>
               <Value>${HOSTNAME}</Value> (Use a variable stating the provider’s
host name)
            </Values>
         </KeyCondition>
      </Operator>
   </Condition>
</PropertiesConfigurationDocument>
```

Die logischen Operatoren können verschachtelt werden, um Bedingungen wie z. B. `(C1 AND (C2 OR C3))` zu erstellen. Die XML-Datei würde dann wie folgt aussehen:

```
<PropertiesConfigurationDocument name="MyConfig.properties"> (Type and Name of
the configuration document)
```

```
   <Condition>
      <Operator type="and">
         C1
         <Operator type=”or”>
            C2
            C3
         </Operator>
      </Operator>
   </Condition>
</PropertiesConfigurationDocument>
```

Wobei C1, C2 und C3 die XML-Ausschnitte der erforderlichen Suchbedingungen sind.

Es werden drei Dateitypen unterstützt, wobei jede einen eigenen Suchausdruckstyp enthält:

- PropertiesConfigurationDocument – eine Datei, in der jede Zeile das Format key=value verwendet. Weitere Informationen finden Sie unter "Konfigurationsdokumente für Eigenschaften" auf Seite 89.

  KeyCondition – eine Bedingung für den Wert einer bestimmtes Schlüssels
- XmlConfigurationDocument – eine XML-Datei. Weitere Informationen finden Sie unter "XML-Konfigurationsdokumente" auf Seite 92.

  XPathCondition – evaluiert eine XPath-Abfrage
- TextConfigurationDocument – beliebiges Text-Konfigurationsdokument. Weitere Informationen finden Sie unter "Textkonfigurationsdokumente" auf Seite 95.

  RegExp – evaluiert einen regulären Ausdruck

Weitere Informationen über Standardwerte finden Sie unter "Verwenden von Standardwerten für Variablen" auf der nächsten Seite.

Fehlerbehebung

- Unterschiedliche Konzepte im gleichen Knoten der Bedingungsstruktur sind nicht zulässig. Ebenso sind zwei Strukturzweige mit unterschiedlichen Konzepten unter dem gleichen Operator nicht zulässig.

  So ist beispielsweise die folgende Bedingungsdefinition ungültig:

```
<Condition>
    <Operator type="and">
        <XPathCondition>
            <XPath>/Setup/Configuration/Hostname[matches(@Name,
'.*?${Concept1.HOSTNAME}.*')]</XPath>
        </XPathCondition>
        <XPathCondition>
            <XPath>/Setup/Configuration/Port[matches(text(),
'${Concept2.PORT}')]</XPath>
        </XPathCondition>
    </Operator>
</Condition>
```

Verwenden von Standardwerten für Variablen

In einigen Fällen enthalten Konfigurationsdateien nicht den Wert einer bestimmten Verbindungszeichenfolge, wenn dieser einem Standardwert entspricht. Beim Definieren einer HTTP-URL wird der Portwert 80 wahrscheinlich nicht explizit in der URL angeführt. Das bedeutet, dass die Konfigurationsdatei eher http://www.hp.com/ enthält als http://www.hp.com:80/, obwohl beide Werte korrekt sind. Dies kann beim Schreiben einer Suchbedingung zu einem Problem führen, da die PORT-Variable den Wert 80 enthält und die Bedingung diesen Wert anfordert. Beim Testen der URL wird ein Fehler zurückgegeben, wenn der Wert nicht angegeben ist.

Um dieses Problem zu lösen, kann jede Variable optional einen Standardwert enthalten. Suchausdrücke können geändert werden, wenn der Wert der Variable mit dem Standardwert identisch ist.

Es gibt zwei Möglichkeiten einen Suchausdruck zu ändern:

- Eine Bedingung ignorieren, wenn der Wert einer Variablen mit Standardwert identisch ist.

  Verwenden Sie die Option eine Bedingung vollständig zu ignorieren, wenn mindestens eine Variable in der Suchbedingung mit dem Standardwert identisch ist. Wenn eine Bedingung ignoriert wird, gibt sie nicht "True" oder "False" zurück. Sie wird ignoriert, das heißt das Ergebnis des logischen Operators hängt von dem Operator und den übrigen Operanden ab. Beispiel:

  - Wenn eine Bedingung dem Muster **A AND B** entspricht und **A** und **B** andere Ausdrücke sind, wenn **A** ignoriert wird, werden die Ergebnisse mit den Ergebnissen für **B** identisch sein.
  - Für den Ausdruck: **A AND B AND C** wird das Ergebnis identisch sein mit **B AND C**, wenn **A** ignoriert wird.
  - Für den Ausdruck: **A OR B OR C** wird das Ergebnis identisch sein mit **B OR C**, wenn **A** ignoriert wird.

  In dem seltenen Fall, dass alle untergeordneten Bedingungen ignoriert werden, die Bedingung also vollständig ignoriert wird, wird als Ergebnis Konfigurationsdokumentbedingung "Bedingungen" zurückgegeben.

  Um eine Bedingung zu ignorieren, fügen Sie das Attribut **ignoreIfDefaultValue** zur Bedingung hinzu und verwenden Sie Variablen, die festlegen, dass diese Bedingung ignoriert wird. Beispiel:

  ```
  <KeyCondition key="serverName" ignoreIfDefaultValue="IPADDRESS">
  ```

- Verwenden Sie eine andere Bedingung, wenn der Wert einer Variablen mit dem Standardwert identisch ist.

  Wir kommen auf das Beispiel mit PORT 80 und der URL zurück. Verwenden Sie den folgenden regulären Ausdruck, um die URL zu testen:

  ```
  http://${DOMAIN}:${PORT}/
  ```

  Es ist offensichtlich, dass dieser reguläre Ausdruck nicht "True" zurückgeben wird, wenn eine Zeichenkette z. B. http://www.hp.com/ lautet, weil der Doppelpunkt (:) in der Zeichenkette nicht

vorhanden ist. Deshalb soll auch der folgende reguläre Ausdruck getestet werden, wenn die PORT-Variable den Wert **80** aufweist:

```
http://${DOMAIN}/
```

Bei diesem Szenario können Sie alternative Ausdrücke verwenden. Um einen alternativen Ausdruck zu definieren, definieren Sie mehrere Suchausdrücke unter den gleichen Bedingungen und geben an, welche verwendet werden soll, wenn die Variable dem Standardwert entspricht. Beispiel:

```
<KeyCondition key="url">
   <RegExp>http://${DOMAIN}:${PORT}/</RegExp>
   <RegExp alternativeFor=”PORT”>http://${DOMAIN}/</RegExp>
</KeyCondition>
```

Unter Verwendung des Attributs **alternativeFor** geben Sie an, dass anstelle des ursprünglichen Ausdrucks der alternative evaluiert wird, wenn die Variable dem Standardwert entspricht. Der ursprüngliche Ausdruck ist der Ausdruck ohne das Attribut **alternativeFor** bzw. der mit einem leeren **alternativeFor**-Attribut.

Für alle Variablen, die in einem Ausdruck enthalten und mit dem Standardwert identisch sind, müssen Sie einen alternativen Ausdruck definieren, um alle Kombinationen dieser Variablen abzudecken. Andernfalls erhalten Sie einen Kompilierungsfehler. Wenn es nur eine Variable gibt, benötigen Sie nur einen alternativen Ausdruck, wie im oben aufgeführten Beispiel dargestellt. Wenn es mehrere Variablen mit Standardwerten gibt, benötigen Sie mehrere alternative Ausdrücke.

Wenn beispielsweise die DOMAIN-Variable ebenfalls dem Standardwert entspricht, bedeutet die Verwendung der **alternativeFor**-Anweisung dass die folgenden alternativen Bedingungen vorhanden sind:

```
<KeyCondition key="url">
   <RegExp>http://${DOMAIN}:${PORT}/</RegExp>
   <RegExp alternativeFor=”PORT”>http://${DOMAIN}/</RegExp>
   <RegExp alternativeFor=”DOMAIN”>http://www.hp.com:${PORT}/</RegExp>
   <RegExp alternativeFor=”PORT, DOMAIN”>http://www.hp.com/</RegExp>
</KeyCondition>
```

Nur für eine dieser Kombinationen wird während der Laufzeit evaluiert.

Weitere Informationen zum Festlegen von Standardwerten für Variablen finden Sie unter "Standardwerte" auf Seite 75.

Angeben von Pfaden zu Konfigurationsdokumenten

Der Pfad zu einem Konfigurationsdokument verweist nicht auf die Datei im Host-Dateisystem, sondern auf den topologischen Pfad zwischen dem durch den Benutzer bereitstellbaren Dokument und dem Konfigurationsdokument.

Ist kein Pfad angegeben, wird standardmäßig davon ausgegangen, dass das Konfigurationsdokument durch einen Verbundlink mit dem bereitstellbaren Dokument verbunden ist. Mit anderen Worten wird davon ausgegangen, dass das bereitstellbare Komponenten-CI als Besitzer des Konfigurationsdokument-CIs fungiert.

So geben Sie einen anderen Pfad an:

1. Definieren Sie eine TQL-Abfrage wie unter "Definieren von TQL-Abfragen" auf Seite 101 beschrieben.
2. Verweisen Sie wie unten dargestellt auf die TQL-Abfrage aus dem Konfigurationsdokument unter Verwendung des `<DocumentCILocation>`-Elements:

```
<TextConfigurationDocument name="cmdb.properties">
   <DocumentCILocation>
      <ReferenceLocation>YourQueryName</ReferenceLocation>
   <DocumentCILocation>
   <Condition>
      ...
   </Condition>
</TextConfigurationDocument>
```

   In diesem Beispiel stellt "YourQueryName" einen Verweis auf den Namen der Abfrage im Abschnitt `<Queries>` dar.

So erstellen Sie die TQL-Abfrage zur Angabe des Pfads:

- Die TQL-Abfrage muss einen Pfad zwischen der bereitstellbaren Komponente und dem Konfigurationsdokument definieren. Hierbei muss es sich um einen einfachen Pfad handeln (keine Zyklen).
- Die TQL-Abfrage muss die beiden folgenden Endknoten mit den beiden bestimmten Namen (Groß-Kleinschreibung wird unterschieden) enthalten:
  - Deployable – das bereitstellbare Komonenten-CI im Pfad
  - Configuration_document – das CI im Pfad, das das Konfigurationsdokument angibt
- Bedingungen sind auf den Knoten **Deployable** und **Configuration_document** nicht zulässig.
- Die Kardinalität zwischen allen Knoten muss 1..1 sein.
- Es werden nur reguläre Link-Typen unterstützt. Es sind weder Verbund- oder Join-Beziehungen noch Unterdiagramme zulässig.

Konfigurationsdokumentüberschreibungen

Es gibt Situationen, in denen ein Konfigurationsdokument ein anderes überschreibt oder von anderen Konfigurationsdokumenten überschrieben wird. Wenn beispielsweise ein Konfigurationsdokument auf einem Host vorhanden ist, kann dieses Konfigurationsdokumente überschreiben, die auf einem Cluster vorhanden sind, zu dem der Host gehört. In solchen Fällen müssen Schlüsselwerte in allen Dateien gesucht werden, die möglicherweise die Werte überschreiben, um dann den Wert mit höchster Priorität auszuwählen. Im folgenden Beispiel hat das lokale Konfigurationsdokument auf dem Host eine höhere Priorität als das Dokument auf dem Cluster.

Um Dateiüberschreibungen in der Abhängigkeitssignatur zu definieren, verwenden Sie die folgende Syntax, um mehrere Pfade mit Priorität zu einem Konfigurationsdokument hinzuzufügen:

```
<PropertiesConfigurationDocument name="resources.xml">
```

```
   <DocumentCILocation>
      <ReferenceLocation priority="2">websphereas_resource_
configfiles</ReferenceLocation>
      <ReferenceLocation priority="3">j2ee_cluster_
configfiles</ReferenceLocation>
      <ReferenceLocation priority="1">j2eeapplication_
configfiles</ReferenceLocation>
   </DocumentCILocation>
   …
</PropertiesConfigurationDocument
```

Der Dateiname (**Ressourcen.xml** im oben genannten Beispiel) muss an allen Referenzspeicherorten identisch sein.

Beachten Sie, dass ein Referenzspeicherort eine TQL-Abfrage referenziert, die einen topologischen Pfad von der durch den Benutzer bereitstellbaren Komponente zum Konfigurationsdokument angibt. Das bedeutet auch, dieser Pfad muss zwischen der bereitstellbaren Komponente und allen anderen Speicherorten bestehen. Weitere Informationen zu `<DocumentCILocation>` finden Sie unter "Angeben von Pfaden zu Konfigurationsdokumenten" auf Seite 82.

Die Bedingung selbst wird nicht geändert, wenn Sie mehrere Pfade mit Prioritäten hinzufügen. Zur Laufzeit (wenn der Suchausdruck evaluiert wird) werden die richtigen, den Prioritäten entsprechenden Werte verwendet. Wenn beispielsweise in properties-Dateien der Schlüssel K sowohl in Priorität 1 als auch in Priorität 2 existiert (und eine Bedingung für seinen Wert eingerichtet wurde), wird die Bedingung nur für das Dokument mit Priorität 1 evaluiert. Wenn der Schlüssel K nur in Priorität 2 existiert, wird die Bedingung wird nur für das Dokument mit Priorität 2 evaluiert.

Weitere Informationen zu Bedingungen in properties-Dateien finden Sie unter "Konfigurationsdokumente für Eigenschaften" auf Seite 89.

Im XML-Dokumenten wird jeder XPath als ein Schlüssel betrachtet. Beispiel: `\Root\Element\@Attribute` bedeutet, das Attribut "Attribute" mit diesem Pfad ist ein Schlüssel, und sein Wert kann in verschiedenen Dateien überschrieben werden.

Wenn der XPath auch über konstante Bedingungen verfügt (eine Bedingung ohne Variablen), ist diese Bedingung Teil des Schlüssels. Beispiel: `\Root\Element[@name = ‘name’]\@Attribute`.

Wenn jedoch der XPath über nicht konstante Bedingungen verfügt, werden solche Bedingungen aus dem Schlüssel entfernt und als Teil des Wertes betrachtet. Für den Pfad `\Root\Element[@name = ${NAME}]\@Attribute` wäre der Schlüssel der Pfad `\Root\Element\@Attribute` und die Bedingung `Element[@name = ‘name’]` würde nur für das am höchsten priorisierte Dokument evaluiert, in dem dieser Schlüssel vorhanden ist.

Wenn ein XML-Dokument Prioritäten verwendet, muss undbedingt sichergestellt werden, dass die XPath-Bedingungen wie oben beschrieben erfüllt sind. Nur so kann fehlerhaftes und unerwartetes Verhalten vermieden werden.

Weitere Informationen zu XPath-Bedingungen finden Sie unter "XML-Konfigurationsdokumente" auf Seite 92.

Hinweis: Mehrere Referenzspeicherorte mit Prioritäten sind nur in Konfigurationsdokumenten des Typs "Property"und "XML" zulässig. Sie dürfen nicht in Dokumenten des Typs "Text" verwendet

werden.

Mehrere Konfigurationsdokumente mit derselben Priorität

Es ist möglich, verschiedenen `<ReferenceLocation>`-Elementen dieselbe Priorität zuzuweisen. In diesem Fall wird bei der Evaluierung einer Bedingung das Framework alle Dateien betrachten, die über dieselbe Priorität verfügen, und versuchen eine Übereinstimmung mit allen Dateien zu erzielen.

Die Bedingung wird mit "True" evaluiert, wenn mindestens eine festgelegte Anzahl Dateien mit "True" evaluiert wird. Die Anzahl der Dateien wird über das Attribut **samePriorityMatchAtLeast** definiert.

Beispiel:

```
<PropertiesConfigurationDocument name="resources.xml">
   <DocumentCILocation samePriorityMatchAtLeast=”1”>
      <ReferenceLocation priority="1">websphereas_resource_
configfiles</ReferenceLocation>
      <ReferenceLocation priority="1">j2ee_cluster_
configfiles</ReferenceLocation>
      <ReferenceLocation priority="1">j2eeapplication_
configfiles</ReferenceLocation>
   </DocumentCILocation>
…
</PropertiesConfigurationDocument>
```

Dies bedeutet, dass bei der Evaluierung einer Bedingung mindestens einer der referenzierten Dateispeicherorte auf eine Datei verweist, die mit **True** evaluiert wurde.

Hinweis: Derzeit wird für das Attribut **samePriorityMatchAtLeast** nur den Wert **1** unterstützt.

Über mehrere Dokumente definierte Abhängigkeiten

In vielen Fällen ist eine Suche über mehrere Konfigurationsdokumente erforderlich, um festzustellen, ob ein Benutzer einen Service anbietet, damit alle Verbindungszeichenfolgen eines Providers lokalisiert werden können.

Suche über mehrere nicht verbundene Konfigurationsdokumente

Die Konfiguratiosdokumente stehen möglicherweise nicht miteinander in Verbindung. Hierbei können einige der Verbindungszeichenfolgen in unterschiedlichen Dokumenten angezeigt werden. Ein Benutzer verfügt beispielsweise über zwei Konfigurationsdokumente: **A.conf** und **B.conf**. Die Verbindungszeichenfolge des Providers sind C1, definiert in **A.conf**, und C2, definiert in **B.conf**. Sowohl C1 und C2 müssen bestimmen, ob tatsächlich eine Abhängigkeit zwischen dem Benutzer und dem Provider besteht. In diesem Fall gibt es zwei erforderliche Suchausdrücke, jede in einem der Konfigurationsdokumente. Das Tag <Dependency> würde dann die folgende Struktur haben:

```
<Dependency name="dependency_name" providerCiType="webmodule" scope="my_scope">
   <PropertiesConfigurationDocument name="A.conf">
```

```
      <Condition>
         <Operator type="and">
            <KeyCondition key="C1">
               <Values>
                  <Value>${VAR_1}</Value>
               </Values>
            </KeyCondition>
         </Operator>
      </Condition>
   </PropertiesConfigurationDocument>
   <TextConfigurationDocument name="B.conf">
      <Condition>
         <Operator type="and">
            <RegExpCondition>
               <RegExp>C2?${VAR_2}</RegExp>
            </RegExpCondition>
         </Operator>
      </Condition>
   </TextConfigurationDocument>
</Dependency>
```

Hinweis:

- Es gibt keine Begrenzung der Anzahl der Dateien, die in einem `<Dependency>`-Element angezeigt werden können.
- Es können unterschiedliche Dateitypen verwendet werden (Eigenschaftsdatei, XML-Datei oder Textdatei).
- Die Reihenfolge der Dateien im `<Dependency>`-Element wird ignoriert. Die Dateien werden in beliebiger Reihenfolge geprüft.
- Die Suchausdrücke aller Dateien müssen den Wert **True** zurückgeben, damit die Abhängigkeit besteht.
- Da die Verbindungszeichenfolgen über mehrere Dateien verteilt sind, muss der Suchausdruck des Gültigkeitsbereichs breit genug sein, sodass mindestens eine der erforderlichen Dateien aus dem Gültigkeitsbereich zurückgegeben werden kann. Weitere Informationen finden Sie unter "Festlegen des Gültigkeitsbereichs für die Suche" auf Seite 97.

Um den Flow bei der Suche über mehrere Dateien zusammenzufassen, gehen Sie wie folgt vor:

1. Der Filtergültigkeitsbereich wird wie unter "Festlegen des Gültigkeitsbereichs für die Suche" auf Seite 97 beschrieben ausgeführt. Wenn keine der erforderlichen Dateien zurückgegeben wird, besteht keine Abhängigkeit zwischen dem Kunden und dem Provider.
2. Wenn mindestens einer der Dateien durch den Filter zurückgegeben wird, werden der Benutzer, der mit der Datei verbunden ist, sowie die übrigen erforderlichen Konfigurationen in die Data Flow Probe heruntergeladen.
3. Die Bedingungen werden in beliebiger Reihenfolge aus jeder Datei evaluiert.
4. Wenn alle Bedingungen den Wert **True** zurückgeben, besteht eine Abhängigkeit, andernfalls nicht.

Suche über mehrere Konfigurationsdokumente mit Abhängigkeiten

In anderen Fällen sind Konfigurationsdokumente eines Benutzers miteinander verbunden. So definiert die Datei **DBConnections.conf** beispielsweise mehrere Verbindungszeichenfolgen für mehrere Datenbanken und Schemas. Hierbei erhält jede Verbindungszeichenfolge einen Namen. In der Datei **MyApp.conf** wird einer dieser Verbindungsnamen verwendet. Dieser definiert, dass MyApp diese bestimmte Datenbank und dieses Schema verwendet.

Variablen des lokalen Gültigkeitsbereichs werden in der Konfigurationsdatei verwendet, um sicherzustellen, dass MyApp den Provider (DB) verwendet. Weitere Informationen finden Sie unter "Lokaler Gültigkeitsbereich" auf Seite 73.

Verwenden Sie die Anweisungen zum Einfügen in Variablen, um Werte in eine Variable einzufügen. Es gibt für jeden Dokumenttyp unterschiedliche Anweisungen:

- Eigenschaftsdateien – KeyVariable und KeyRegExpVariable. Weitere Informationen finden Sie unter "Konfigurationsdokumente für Eigenschaften" auf Seite 89.
- XML-Dateien – XPathVariable. Weitere Informationen finden Sie unter "XML-Konfigurationsdokumente" auf Seite 92.
- Textdateien – RegExpVariable. Weitere Informationen finden Sie unter "Textkonfigurationsdokumente" auf Seite 95.

In diesem Fall gibt es zwei erforderliche Suchausdrücke, jede in einer der Dateien. Das Tag <Dependency> hat somit die folgende Struktur:

```
<Dependency name="dependency_name" providerCiType="webmodule" scope="my_scope">
      <VariableDeclarations>
         <Variable name=”REFERENCE_NAME” />
      </VariableDeclarations>
   <PropertiesConfigurationDocument name="DBConnections.conf">
      <Condition>
         <Operator type="and">
            <KeyCondition key="C1">
               <Values>
                  <Value>${VAR_1}</Value>
               </Values>
            </KeyCondition>
         </Operator>
      </Condition>
      <Variables>
         <KeyVariable variable=”REFERENCE_NAME” key=”reference_name” />
      </Variables>
   </PropertiesConfigurationDocument>
   <TextConfigurationDocument name="MyApp.conf">
      <Condition>
         <Operator type="and">
            <RegExpCondition>
               <RegExp>C2?${REFERENCE_NAME}</RegExp>
            </RegExpCondition>
```

```
        </Operator>
      </Condition>
   </TextConfigurationDocument>
</Dependency>
```

Hinweis:

- Verwenden Sie das Element `<VariableDeclarations>`, um einen oder mehrere lokale Variablen zu definieren, die in der Abhängigkeit verwendet werden sollen.
- Das Element `<Variables>` wird verwendet, um Werte in die lokalen Variablen einzufügen. Der Abschnitt wird nur ausgeführt, wenn das Element `<Condition>` den Wert **True** zurückgibt.
- In diesem Beispiel wird `<KeyVariable>` verwendet, um den Wert des Schlüssels **reference_name** in die Variable REFERENCE_NAME einzufügen.
- Die Variable `${REFERENCE_NAME}` wird in der Bedingung **MyApp.conf** verwendet. Diese Variable ist abhängig von der Datei **DBConnections.conf**, und wird nur nach **DBConnections.conf** evaluiert und nur dann, wenn der Suchausdruck den Wert **True** zurückgibt.
- Zyklische Abhängigkeiten zwischen Dateien sind nicht zulässig.
- Wenn das Dokument **MyApp.conf** evaluiert wird, enthält die Variable `${REFERENCE_NAME}` bereits den Wert aus der Datei **DBConnections.conf**.
- Da die Verbindungszeichenfolgen über mehrere Dateien verteilt sind, muss der Suchausdruck des Gültigkeitsbereichs breit genug sein, sodass auch Dateien ohne Abhängigkeiten aus dem Gültigkeitsbereich zurückgegeben werden können. Weitere Informationen finden Sie unter "Festlegen des Gültigkeitsbereichs für die Suche" auf Seite 97.

Um den Flow bei der Suche über mehrere Dateien mit Abhängigkeiten zwischen ihnen zusammenzufassen, gehen Sie wie folgt vor:

1. Der Filtergültigkeitsbereich wird wie unter "Festlegen des Gültigkeitsbereichs für die Suche" auf Seite 97 beschrieben ausgeführt. Wenn keine der erforderlichen Dateien zurückgegeben wird oder keine Dateien von anderen Dateien abhängen, besteht keine Abhängigkeit zwischen dem Kunden und dem Provider.
2. Mindestens eine der Dateien, die keine Abhängigkeiten aufweist, wird durch den Filter zurückgegeben.
3. Der Benutzer, der mit der Datei verbunden ist, sowie die übrigen erforderlichen Konfigurationen werden in die Data Flow Probe heruntergeladen.

Die Anweisung zum Enfügen gibt möglicherweise keinen Wert zurück, z. B. wenn der Schlüssel, auf den sie sich bezieht nicht vorhanden ist. In diesem Fall wird der Variable ein leerer Wert zugewiesen. Dies ist jedoch nicht optimal. Wenn der erforderliche Wert nicht vorhanden ist, kann es sein, dass keine Abhängigkeit besteht, und es somit nicht sinnvoll ist, die Evaluierung fortzuführen. Verwenden Sie für dieses Szenario `allowNull=”false”` in der Anweisung zum Einfügen. Beispiel:

```
 <KeyVariable variable=”REFERENCE_NAME” key=”reference_name” allowNull=”false”
/>
```

Konfigurationsdokumente für Eigenschaften

Konfigurationsdokumente für Eigenschaften speichern eine Konfiguration im Format `Key=value`. Ein Konfigurationsdokument für Eigenschaften kann zum Beispiel folgendermaßen aufgebaut sein:

```
# My configuration document
Hostname=MyNode
```

wobei `Hostname` ein Schlüssel ist und `MyNode` der dazugehörige Wert.

"#" kennzeichnet eine Kommentarzeile. Kommentarzeilen werden beim Prüfen von Suchausdrücken ignoriert.

Um ein Konfigurationsdokument als Konfigurationsdokument für Eigenschaften zu kennzeichnen, verwenden Sie das Element `<PropertiesConfigurationDocument>`. Beispiel:

```
<Dependency name="history_db" providerCiType="cmdb" scope="default">
   <PropertiesConfigurationDocument name="cmdb.conf">
      ...
```

Definieren von Bedingungen

Es gibt mehrere Möglichkeiten zu prüfen, ob Verbindungszeichenkettenvariablen in properties-Dateien vorhanden sind. Für alle Bedingungstypen wird der Wert des Schlüssels, der über das **key**-Attribut angegeben wird, gegen andere Werte geprüft. Der Schlüssel ist immer ein konstanter Wert (Variablen sind unzulässig):

- Prüfen Sie, ob ein konstanter Wert als Schlüsselwert verwendet wird.

  ```
  <KeyCondition key="dal.datamodel.name">
     <Values>
        <Value>ConstantValue1</Value>
        <Value>ConstantValue2</Value>
     </Values>
  </KeyCondition>
  ```

  Die Bedingung gibt nur dann "True" zurück, wenn der Wert des Schlüssels einem der konstanten Werte entspricht. Die Groß-/Kleinschreibung wird dabei ignoriert.

- Prüfen Sie, ob eine Variable als Schlüsselwert verwendet wird.

  ```
  <KeyCondition key="dal.datamodel.name">
     <Values>
        <Value>${VARIABLE1}</Value>
        <Value>${VARIABLE2}</Value>
     </Values>
  </KeyCondition>
  ```

Die Bedingung gibt nur dann "True" zurück, wenn der Wert des Schlüssels dem Wert einer Variablen entspricht. Die Groß-/Kleinschreibung wird dabei ignoriert. Wenn die Variable eine Liste von Werten ist, werden alle Werte geprüft. Sobald der Schlüssel mit einem der Werte aus der Liste übereinstimmt, wird "True" zurückgegeben.

Hinweis: Sie können Variablen und konstante Werte in derselben Bedingung angeben.

- Prüfen Sie, ob der Wert eines Schlüssels mit einem regulären Ausdruck übereinstimmt.F

```
<KeyCondition key="dal.datamodel.name">
   <RegExp>
      ^SomeText${VARIABLE}MoreText$
   </RegExp>
</KeyCondition>
```

Der reguläre Ausdruck kann als Teil der Bedingung auch eine oder mehrere Variablen enthalten. Während der Laufzeit werden Variablen durch ihre tatsächlichen Werte ersetzt, wenn der konkrete Suchausdruck generiert wird.

Wenn eine Variable eine Werteliste enthält, wird ein Ausdruck erzeugt, der folgender Syntax entspricht:

```
^SomeText(Value1 | Value2  | Value3)MoreText$
```

In diesem Fall gibt die Bedingung "True" zurück, sobald einer der Werte aus der Liste vorhanden ist.

Beispiel für Bedingungen in einem Konfigurationsdokument für Eigenschaften

```
<PropertiesConfigurationDocument name="MyConfig.properties">
   <Condition>
      <Operator type="and">
         <KeyCondition key="TYPE">
            <Values>
               <Value>HTTP</Value>
               <Value>HTTPS</Value>
            </Values>
         </KeyCondition>
         <KeyCondition key="IPADDRESS">
            <Values>
               <Value>${IP_ADDRESS}</Value>
               <Value>${HOSTNAME}</Value>
            </Values>
         </KeyCondition>
         <KeyCondition key="SITE">
            <RegExp>
               //${SITE_NAME}//.*
            </RegExo>
         </KeyCondition>
```

```
      </Operator>
   </Condition>
</PropertiesConfigurationDocument>
```

Dies bedeutet, dass das Konfigurationsdokument für Eigenschaften mit der Bezeichnung **MyConfig.properties** folgende Komponenten enhalten muss:

- Ein Schlüssel mit der Bezeichnung TYPE enthält einen der folgenden Werte; HTTP oder HTTPS.
- Ein Schlüssel mit der Bezeichnung IPADDRESS enthält entweder die IP-Adresse des Providers (oder eine seiner IP-Adressen) oder seinen Hostnamen.
- Ein Schlüssel mit der Bezeichnung SITE beginnt mit "/"). Dann muss der SITE_NAME des Providers mit einem weiteren "/" und einer beliebigen Zeichenfolge folgen.

Einfügen von Variablenwerten

Es gibt zwei Möglichkeiten, den Wert eines Schlüssels oder einen Teil eines Werts in eine Variable zu extrahieren:

- Durch Einfügen eines Schlüsselwerts.
  - Verwenden Sie in einem dedizierten `<Variables>`-Abschnitt die folgende Syntax:

    ```
    <KeyVariable variable=”VARIABLE_NAME” key=”KEY_NAME” />
    ```

  - Verwenden Sie als Teil einer Suchbedingung die folgende Syntax:

    ```
    <KeyCondition key=”KEY_NAME”>
       <Values>
          <Value>REQUIRED_VALUE</Value>
       </Values>
       <Variables>
          <KeyVariable variable=”VARIABLE_NAME” />
       </Variables>
    </KeyCondition>
    ```

    Der Schlüssel der VARIABLE_NAME-Variable ist im Element `<KeyCondition>` definiert.

    Diese Syntax ist identisch wenn Sie `<RegExp>` anstelle von `<Values>` verwenden. In beiden Fällen, wird die Variable mit dem vollständigen Wert des Schlüssels eingegeben.

- Durch Einfügen eines Teils des Schlüsselwerts.
  - Verwenden Sie in einem dedizierten `<Variables>`-Abschnitt die folgende Syntax:

    ```
    <KeyRegExpVariable variable=”VARIABLE_NAME” key=”KEY_NAME”
    expression=”REGULAR_EXPRESSION” group=”GROUP_NUMBER” />
    ```

    In diesem Fall ist GROUP_NUMBER der Index (mit 0 beginnend) der Gruppe im regulären Ausdruck, der durch REGULAR_EXPRESSION definiert ist. In einem Eigenschaftendokument mit der Zeile

```
myKey = 123Value123
```

und der Anweisung

```
<KeyRegExpVariable variable=”VAR” key=”myKey” expression=”[0-9]*([a-zA-Z]*)
[0-9]*” group=”0” />
```

wird beispielsweise der Wert "value" in die VAR-Variable eingegeben.

Sie können Werte verwenden, die in derselben oder anderen Dateien definiert wurden. Beispiel:

```
<KeyRegExpVariable variable=”VAR” key=”myKey” expression=”${PREV_VAR}([a-
zA-Z]*)${PREV_VAR}” group=”0” />
```

Wenn die PREV_VAR-Variable den Wert "123" enthält, ruft die VAR-Variable den Wert "value" ab oder gibt den gesamten Wert eines Schlüssels ein. Gleich welche Option Anwendung findet, enthält die Variable so grundsätzlich einen einzelnen Wert.

- Verwenden Sie als Teil einer Suchbedingung die folgende Syntax:

```
<KeyCondition key=”KEY_NAME”>
   <RegExp>REGULAR_EXPRESSION</RegExp>
   <Variables>
      <KeyRegExpVariable variable=”VARIABLE_NAME” group=”GROUP_NUMBER” />
   </Variables>
</KeyCondition>
```

Der Schlüssel der VARIABLE_NAME-Variable ist im Element `<KeyCondition>` definiert.

Diese Option wird allerdings nur unterstützt, wenn `<RegExp>` verwendet wird und der reguläre Ausdruck, auf den sich das Attribut "group" der Variable bezieht, mit der Angabe in dem Element `<RegExp>` übereinstimmt.

XML-Konfigurationsdokumente

Mit einem XML-Konfigurationsdokument können Sie mithilfe von XPath-Abfragen problemlos Suchausdrücke für auf XML-Standard basierte Konfigurationsdokumente schreiben.

Um ein Konfigurationsdokument als XML-Konfigurationsdokument zu kennzeichnen, verwenden Sie das Element `<XmlConfigurationDocument>`. Beispiel:

```
<Dependency name="some_reference" providerCiType="webmodule" scope="default">
   <XmlConfigurationDocument name="web.xml">
      ...
```

Definieren von Bedingungen

Als Suchbedingungen in XML-Konfigurationsdokumenten sind nur XPath-2.0-Abfragen zulässig. Wenn

ein Knoten aus dem XPath ausgewählt wurde, wird als Bedingung der Wert **True** zurückgegeben. Andernfalls wird **False** zurückgegeben. Eine Abfrage kann Variablen in folgenden Speicherorten enthalten:

- In Element-Namen

```
/Root/${SITE_NAME}
```

Wenn eine Variable mehrere Werte enthält, führt das Framework mehrere XPath-Anweisungen aus. Wenn Sie beispielsweise `SITE_NAME` die Werte SITE1 und SITE2 enthält, generiert die Anweisung in diesem Beispiel die folgenden zwei konkreten Suchen:

```
\Root\SITE1
\Root\SITE2
```

- In Gleichheitsprüfungen

```
/Element[@att = ${VAR}]
oder
/Element/text() = ${VAR}
```

Gleichheit, wie in den Beispielen oben dargestellt, funktioniert nur für Variablen mit einzelnen Werten. Wenn die Möglichkeit besteht, dass eine Variable mehr als einen Wert enthält, verwenden Sie stattdessen die folgende Syntax:

```
/Element[${equals(@att, VAR)]
oder
/Element[${equals-ignore-case(text(), VAR}]
```

Hinweis:

- Verwenden Sie `${equals()}` zum Prüfen der Gleichheit unter Beachtung der Groß-/Kleinschreibung, oder `${equals-ignore-case()}` zum Prüfen der Gleichheit ohne Beachtung der Groß-/Kleinschreibung.
- Der erste Parameter sollte eine XPath-Funktion oder ein Attributname sein.
- Der zweite Parameter sollte immer ein Variablenname sein. Die Variablenname sollte ohne `${}` angezeigt werden.
- Diese Syntax kann für Variablen mit einem einzigen Wert oder mehreren Werten verwendet werden, und wird für alle Fälle empfohlen.
- Wenn mehrere Werte vorliegen und der Wert des ersten Parameters nur einem Wert der Variablen entspricht, wird der Wert **True** zurückgegeben.

- In regulären Ausdrücken

```
/datasources/mbean[matches(@name, '.*?,ip=${IPADDRESS}.*')]
```

Wenn eine Variable eine Werteliste enthält, wird ein Ausdruck entsprechend dem folgenden Ausdruck erzeugt:

```
/datasources/mbean[matches(@name, '.*?,ip=(Value1 | Value2 | Value3).*')]
```

Bei beiden Werten wird für die Bedingung der Wert **True** zurückgegeben.

Beispiel für Bedingungen in einem XML-Konfigurationsdokument

```
<XmlConfigurationDocument name="MyConfig.xml">
   <Condition>
      <Operator type="and">
         <XPathCondition>
            <XPath>\URL\Protocol[@name='HTTP' or @name='HTTPS']</XPath>
         </XPathCondition>
         <XPathCondition>
            <XPath>\URL\Host[${equals-ignore-case(@name, HOSTNAME)} or ${equals
(@name, IP_ADDRESS)}]</XPath>
         </XPathCondition >
         <XPathCondition>
            <XPath>\URL\Site[matches(text(), '//${SITE_NAME}//.*')]</XPath>
         </XPathCondition>
      </Operator>
   </Condition>
</XmlConfigurationDocument>
```

Das folgende Beispiel zeigt, dass das XML-Konfigurationsdokument **MyConfig.xml** die folgenden Elemente beinhalten muss:

- `\URL\Protocol\@name` muss entweder HTTP oder HTTPS sein.
- `\URL\Host\@name` muss entweder die IP-Adresse des Providers (oder eine seiner IP-Adressen) oder sein Hostname sein.
- `\URL\Site\text()` muss mit einem / beginnen. Dann muss der SITE_NAME des Providers mit einem weiteren / und einer beliebigen Zeichenfolge folgen.

Einfügen von Variablenwerten

Sie können XPath verwenden, um Textwerte abzufragen und diese in eine Variable einzufügen. Komplette Elemente können nicht in eine Variable eingefügt werden. Verwenden Sie hierzu in einem dedizierten `<Variablen>`-Abschnitt die folgende Syntax:

```
<XPathVariable variable="VARIABLE_NAME" xpath="XPATH_QUERY" />
```

Beispiele:

- Auswal eines Attributwerts

```
<XPathVariable variable="VAR" xpath="\Root\Element\@Att" />
```

wählt die Werte aus dem Attribut **Att** aus allen \Root\Element-Elementen in dem XML-Dokument aus.

- Auswahl eines Elementtextes

```
<XPathVariable variable=”VAR” xpath=”\Root\Element\text()” />
```

wählt den Text aller Elemente mit übereinstimmendem \Root\Element aus.

- Auswahl eines Attributwerts mit Bedingungen (unter Verwendung von Variablen aus dem gleichen oder einem anderen Konfigurationsdokument)

```
<XPathVariable variable=”VAR” xpath=”\Root\Element[${equals(@Att, VAR)}]
\@AnotherAtt” />
```

wählt den Wert von @AnotherAtt aus allen \Root\Elementen aus, in denen `@Att=${VAR}` ist.

Sie können XPath-Variablen als Teil von XPath-Bedingungen verwenden. In diesem Modus kann XPath auch einen relativen Pfad aus dem Ergebnisknoten der Bedingung enthalten – vorausgesetzt, es gibt ein Ergebnis (also die Bedingung den Wert **True** evaluiert hat). Verwenden Sie die folgende Syntax:

```
<XPathCondition>
   <XPath>XPATH_QUERY</XPath>
   <Variables>
      <XPathVariable variable="VARIABLE_NAME" relativePath="RELATIVE_XPATH_
QUERY" />
   </Variables>
</XPathCondition>
```

Beispiel:

```
<XPathCondition>
   <XPath>/${VAR_1}/chcpCodeToCharsetName[@name='test1' and matches(text(),
'.*?${OUTPUT_VAR2}.*')]</XPath>
   <Variables>
      <XPathVariable variable="OUTPUT_VAR1" relativePath="./@type" />
   </Variables>
</XPathCondition>
```

Vorausgesetzt, die XPathCondition hat den Wert **True** evaluiert und `<XPath>` hat einige XML-Knoten (ein Element in diesem Fall) ausgewählt, wird die Variable den Attributwert **type** des Knotens enthalten.

Es ist möglich, einen XPath für eine Variable zu definieren, der unabhängig von dieser ist. Definieren Sie hierzu einen XPath aus dem Stammverzeichnis des Dokuments, beginnend mit einem Schrägstrich ("/").

Nach dem Einfügen des XPath kann die Variable eine Liste mit Werten enthalten.

Textkonfigurationsdokumente

Eine Textkonfigurationsdokument ist ein Textdokument in einem Format, das dem Inhaltsentwickler bekannt ist. Es handelt sich dabei aber nicht um ein Konfigurationsdokument für Eigenschaften oder

XML-Dateien. Sie können reguläre Ausdrücke verwenden, um Suchausdrücke in Textkonfigurationsdokumenten zu formulieren.

Um ein Konfigurationsdokument als Textdatei zu kennzeichnen, verwenden Sie das Element `<TextConfigurationDocument>`. Beispiel:

```
<Dependency name="some_reference" providerCiType="oracle" scope="default">
   <TextConfigurationDocument name="tnsnames.ora">
      ...
```

Definieren von Bedingungen

Die einzigen Suchbedingungen, die in Textkonfigurationsdokumenten zulässig sind, sind reguläre Ausdrücke. Wenn das Muster übereinstimmt, wird die Bedingung "True" zurückgegeben.

Der reguläre Ausdruck kann als Teil der Bedingung auch eine oder mehrere Variablen enthalten. Während der Laufzeit werden Variablen durch ihre tatsächlichen Werte ersetzt, wenn die konkreten Suchausdrücke generiert werden.

Wenn eine Variable eine Werteliste enthält, wird ein Ausdruck entsprechend dem folgenden Ausdruck erzeugt:

```
^SomeText(Value1 | Value2 | Value3)MoreText$
```

Bei beiden Werten wird für die Bedingung der Wert **True** zurückgegeben.

Beispiel für Bedingungen in einem Textkonfigurationsdokument

```
<TextConfigurationDocument name="MyConfig.txt">
   <Condition>
      <Operator type="or">
         <RegExpCondition>
            <RegExp>^HTTPS?://(${HOSTNAME)/${SITE_NAME}/.*$</RegExp>
         </RegExpCondition>
         <RegExpCondition>
            <RegExp>^HTTPS?://(${IP_ADDRESS)/${SITE_NAME}/.*$</RegExp>
         </RegExpCondition>
      </Operator>
   </Condition>
</TextConfigurationDocument>
```

Das folgende Beispiel zeigt, dass das Textkonfigurationsdokument **MyConfig.txt** die folgenden Elemente beinhalten muss:

- `\URL\Protocol\@name` muss entweder HTTP oder HTTPS sein.
- `\URL\Host\@name` muss entweder die IP-Adresse des Providers (oder eine seiner IP-Adressen) oder sein Hostname sein.
- `\URL\Site\text()` muss mit einem / beginnen. Dann muss der SITE_NAME des Providers mit einem weiteren / und einer beliebigen Zeichenfolge folgen.

Einfügen von Variablenwerten

Sie können reguläre Ausdrücke mit Gruppen verwenden, um einen Wert in eine Variable für Textdokumente einzugeben. Die Gruppen markieren den Text, der eingegeben werden soll. Verwenden Sie in einem dedizierten `<Variables>`-Abschnitt die folgende Syntax:

```
<RegExpVariable variable=”VARIABLE_NAME” expression=”REGULAR_EXPRESSION”
group=”GROUP_NUMBER” />
```

In diesem Fall ist GROUP_NUMBER der Index (mit 0 beginnend) der Gruppe im regulären Ausdruck, der durch REGULAR_EXPRESSION definiert ist.

Für eine Datei mit der Zeile

```
This is part of a configuration document
```

und der Anweisung

```
<RegExpVariable variable=”VAR” expression=”(.*)configuration (.*)” group=”1” />
```

wird beispielsweise der Wert "document" in die VAR-Variable eingegeben. Sie können Variablen, die in unterschiedlichen Dokumenten definiert wurden, zusammen in einem Dokument verwenden. Beispiel:

```
<RegExpVariable variable=”VAR” expression=”.*${PREV_VAR} (.*)” group=”0” />
```

Wenn die PREV_VAR-Variable den Wert "configuration" enthält, wird in die VAR-Variable die Zeichenkette "document" eingegeben.

Sie können `<RegExpVariable>` als Bestandteil des `<RegExpCondition>`-Tags verwenden, indem Sie auf folgende Syntax zurückgreifen:

```
<RegExpCondition>
   <RegExp>REGULAR_EXPRESSION</RegExp>
   <Variables>
      <RegExpVariable variable="VARIABLE_NAME" group="GROUP_NUMBER" />
   </Variables>
</RegExpCondition>
```

Die regulären Ausdrücke, auf die sich das Attribut "group" der Variable bezieht, sind identisch mit den Angaben in dem Element `<RegExp>`.

Die eingegebene Variable wird immer mit einem einzelnen Wert eingegeben.

Festlegen des Gültigkeitsbereichs für die Suche

Der Zweck eines Gültigkeitsbereichs für eine Suche besteht darin, alle Benutzer zu finden, die potenziell Bestandteil einer Abhängigkeit mit einem bestimmten Providertyp sein können. Ein Gültigkeitsbereich hat zwei Verwendungsmöglichkeiten:

- Herausfiltern irrelevanter bereitstellbarer Komponenten aus UCMDB, sodass weniger Ergebnisse durchsucht werden müssen. (Erforderlich)

  Um dies zu erreichen wird versucht, eine Teilmenge der Verbindungszeichenketten des Providers in allen Konfigurationsdokumenten zu finden und die Suche nur auf die bereitstellbaren Komponenten zu beschränken, die mit diesen Konfigurationsdokumenten verbunden sind.

  Dieser Suchausdruck unterscheidet sich von anderen providerspezifischen Abhängigkeitssuchen, da sie alle potenziellen Benutzer so schnell wie möglich zurückgeben muss. Es geht hier nicht um die Fragestellung, ob eine providerspezifische Abhängigkeit vorhanden ist oder nicht.

  Die Syntax des Suchausdrucks entspricht ungefähr der Weise, wie Suchausdrücke in Konfigurationsdokumenten formuliert werden. Allerdings werden für diese Syntax nicht spezielle Bedingungen pro Dateityp definiert und es besteht auch keine Notwendigkeit, die Namen der Dateien anzugeben. Weitere Informationen finden Sie unter "Erstellen eines Suchausdrucks" auf Seite 78.

  Das Beispiel unten stellt den folgenden Ausdruck dar: `(x | ((y & z & w) & (h | g)))`.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ConfigurationDocumentSearchCondition xmlns="http://www.hp.com/ucmdb/1-0-
0/DependenciesDefaultSearch">
    <Operator type="Or">
        <Operator type="And">
            <Operand value="y"/>
            <Operand value="z"/>
            <Operand value="w"/>
            <Operator type="Or">
                <Operand value="h"/>
                <Operand value="g"/>
            </Operator>
        </Operator>
        <Operand value="x"/>
    </Operator>
</ConfigurationDocumentSearchCondition>
```

  y, y, z, w, h und g können konstante Werte, Variablen oder Konzeptvariablen sein.

  Die Werte (oder Variablenwerte) werden unter Verwendung dieses Ausdrucks in allen relevanten Konfigurationsdokumenten des Benutzers in UCMDB gesucht.

  Nur die Benutzer, die über solche Dateien verfügen, werden für die providerspezifische Suche weiterhin berücksichtigt und unter Verwendung der akkuraten Suchausdrücke evaluiert.

- Die bereitstellbaren Komponenten werden auf eine Teilmenge der Komponenten beschränkt, die mit dem Provider verbunden sind. Es werden zum Beispiel nur Abhängigkeiten gesucht, die Bestandteil der J2EE-Domäne des Providers sind. (Optional)

  Um dies zu erreichen wird eine TQL-Abfrage verwendet, um alle bereitstellbaren Komponenten zu finden, die auf beliebige Weise mit dem Provider verbunden sind. Weitere Informationen finden Sie unter "Definieren von TQL-Abfragen" auf Seite 101.

  Das Ergebnis der TQL-Abfrage (als bestimmtes Provider-CI dargestellt) definiert alle möglichen bereitstellbaren Komponenten, die möglicherweise eine Abhängigkeit mit dem Provider aufweisen. Die TQL-Abfrage unterliegt folgenden Regeln:

- Der Abfrageknoten für die bereitstellbare Komponente muss "Deployable" (Groß-Kleinschreibung wird beachtet) lauten. Die Ergebnisse dieses Abfrageknotens sind die möglichen bereitstellbaren Komponenten für den Gültigkeitsbereich.

- Die Komponenten mit der Bezeichnung "Deployable" stellt sowohl Benutzer- als auch Provider-Komponenten dar, die im selben Gültgkeitsbereich liegen können. Aus diesem Grund muss der CI-Typ des Abfrageknotens den bereitstellbaren CI-Typen für Benutzer und Provider übergeordnet sein.

- Es darf nur ein weiterer Abfrageknoten in der TQL-Abfrage enthalten sein (anderer Knoten als "Deployable"). Der Name des anderen Abfrageknotens ist frei wählbar. Dieser Knoten wird als Abfrageknoten für den Gültigkeitsbereich bezeichnet.

- Der Abfrageknoten "Deployable" und der Abfrageknoten für den Gültigkeitsbereich sind über einen Verbundlink verbunden, der alle verfügbaren Pfade zwischen diesen beiden Knoten enthält.

- Alle Gültigkeitsbereiche filtern implizit durch den Benutzer bereitstellbare Komponenten heraus, die sich nicht in derselben Routingdomäne wie der Provider befinden (vorausgesetzt der Provider wurde von einer bestimmten Routingdomäne ermittelt). Das Filtern anhand einer Routingdomäne erfordert den Deskriptor der bereitstellbaren Komponente, damit ein Pfad zum Knoten verfügbar ist. Weitere Informationen finden Sie unter "Definieren des Deskriptors eines Benutzers" auf Seite 76.

  Wenn es sich beispielsweise bei der durch den Provider bereitstellbaren Komponente eine eine aktive Software handelt, ist die Routingdomäne dieser aktiven Software identisch mit dem darin enthaltenen Knoten. Die Routingdomäne des Knotens wird durch die Routingdomäne seiner IP-Adressen bestimmt. Der Knoten kann sich auf mehr als eine Routingdomäne beziehen, wenn seine IP-Adressen aus unterschiedlichen Routingdomänen stammen. Wenn die durch den Provider bereitstellbare Komponente aber ein CI ist, das sich nicht auf einen Knoten oder eine IP-Adresse bezieht (z. B. ein Geschäftsservice), kann sie mit jeder durch den Provider bereitstellbaren Komponente verbunden werden, unabhängig von ihrer Routingdomäne.

Gehen Sie um Beispiel folgendermaßen vor, um einen Gültigkeitsbereich für eine J2EE-Domäne zu definieren:

1. Erstellen Sie einen bereitstellbaren Abfrageknoten des Typs "J2EE Deployed Object". Dies ist die Klasse ,sozusagen der kleinste gemeinsame Nenner unter allen CI-Typen, die bereitstellbare Komponenten in einer J2EE-Domäne darstellt. Eine bereitstellbare Komponente in diesem Gültigkeitsbereich kann zum Beispiel eine Komponente des Typs "Web Module" oder "J2EE Application" sein.
2. Erstellen Sie einen Gültigkeitsbereichs-Abfrageknoten des Typs "J2EE Domain".
3. Erstellen Sie einen Verbundlink zwischen den Abhängigkeits- und Gültigkeitsbereichs-Abfrageknoten mit allen Pfaden zwischen " J2EE Deployed Object" und "J2EE Domain". Der Verbundlink kann zum Beispiel Folgendes enthalten:
   - J2EE Application – Composition – J2EE Domain
   - Web Module – Composition – J2EE Application

Die TQL-Abfragedefinition muss in den Gültigkeitsbereich "XML" eingebunden sein.

Standardwerte der Variablen in einer Suche nach Gültigkeitsbereichen

Werfen wir einen Blick auf das folgende Beispiel:

- Sie verwenden die Variable PORT mit dem Standardwert 80. Dieser Standardwert wird unter Umständen in einigen Konfigurationsdateien nicht angezeigt.
- Sie verwenden `IP_ADDRESS AND PORT` als Suchausdruck für den Gültigkeitsbereich und für bestimmte Provider lautet der Wert der Variable PORT "80". Bereitstellbare Komponenten, die mit Konfigurationsdokumenten verbunden sind, die nicht den Standardwert "80" für den Port verwenden, werden bei dieser Suche nach dem Gültigkeitsbereich nicht zurückgegeben, obwohl das der Fall sein sollte, da im Suchausdruck angegeben wird, dass der Port in der Konfigurationsdatei enthalten sein muss.

Deshalb wird die Variable PORT aus dem Suchausdruck für den Gültigkeitsbereich entfernt, wenn sie den Standardwert verwendet. Ebenso wird die Variable in Abhängigkeits-Suchausdrücken ignoriert. Weitere Informationen finden Sie unter "Verwenden von Standardwerten für Variablen" auf Seite 81.

Wenn eine Variable einen anderen Wert als den Standardwert enthält, wird dieser Wert auf die übliche Weise in den Suchausdruck eingegeben.

Fehlerbehebung

Um zu testen, ob ein Gültigkeitsbereich die erwarteten Benutzer zurückgibt, können Sie die JMX-Methode **searchConfigurationDocuments** im Discovery Manager-Service auf dem UCMDB-Server verwenden. Der Parameter "searchString" sollte die Suchausdruck-XML-Datei sein. Weitere Informationen hierzu finden Sie unter "Festlegen des Gültigkeitsbereichs für die Suche" auf Seite 97. Die XML-Datei sollte nur konstante Werte und keine Variablen enthalten.

Diese XML-Datei kann auch aus dem Kommunikationsprotokoll des Providers in einem der Suchadapter abgerufen werden. Weitere Informationen finden Sie unter "Adapter für die Suche nach Abhängigkeiten" auf Seite 104.

Definieren von TQL-Abfragen

1. Fügen Sie die TQL-Abfrage-Definition der Abhängigkeitssignatur im Abschnitt `<Queries>` wie folgt hinzu:

```
...
<Queries>
   <Query name=”YourQueryName”>
      [TQL XML]
   </Query>
</Queries>
```

2. a. Verwenden Sie einen Texteditor zum Erstellen eines leeren Dokuments.
   b. Fügen Sie die folgenden Inhalte hinzu:

```
<tql:query xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition"
xmlns:ns3=
"http://www.hp.com/ucmdb/1-0-0/PolicyRuleDefinition" xmlns:tql=
"http://www.hp.com/ucmdb/1-0-0/TopologyQueryLanguage" name="<Query
Name>">

</tql:query>
```

   c. Erstellen Sie Ihre TQL-Abfrage in Modeling Studio. Weitere Informationen finden Sie unter "Modeling Studio" im *HP Universal CMDB – Modellierungshandbuch*.
   d. Exportieren Sie die TQL-Abfrage in die XML-Datei.
   e. Öffnen Sie das exportierte XML-Dokument, und kopieren Sie den gesamten Text in das Element `<resource>`.
   f. Fügen Sie diesen Text in das Element `<tql:query>` des Dokuments, das Sie in Schritt 1 erstellt haben.
   g. Kopieren Sie den gesamten Inhalt Ihres Textdokuments und fügen Sie ihn in das Element `<Query>` der Abhängigkeitssignaturdatei ein.

Hinweis: TQL-Abfragen sind in die Abhängigkeitssignatur-XML-Datei eingebettet, und nicht validiert. Wenn jedoch die Abfrage-XML-Datei selbst ungültig ist, wird eine Ausnahme während der Bereitstellung ausgelöst. Weitere Informationen finden Sie unter "Kompilierungsfehler" auf der nächsten Seite.

Packen und Bereitstellen mehrerer Abhängigkeitssignaturdateien

In UCMDB können mehrere Abhängigkeitssignaturdateien bereitgestellt werden. Bei der Suche nach Benutzer/Provider-Links werden alle bereitgestellten Dateien verwendet.

Abhängigkeitssignaturdateien sind Discovery-Konfigurationsdateien, die mit dem Präfix **dependencies/** und dem Suffix **.xml** gekennzeichnet sind, zum Beispiel **dependencies/JEE.xml**.

Jede Abhängigkeitssignaturdatei ist vollständig unabhängig. Beachten Sie Folgendes:

- Globale variable Definitionen können nur in der Datei verwendet werden, in der sie definiert sind. Es wird dringend empfohlen, so oft wie möglich die gleichen Namen für ähnliche Variablen zu vewenden. Wenn beispielsweise eine Variable eine IP-Adresse enthält und es zwei Abhängigkeitsdateien gibt, sollte in beiden Dateien eine Variable mit dem Namen IP_ADDRESS definiert sein. So können die unterschiedlichen Suchadapter ein einziges Zieldatenelement mit dem Namen IP_ADDRESS verwenden. Weitere Informationen finden Sie unter "Angeben von Variablenwerten" auf Seite 107.
- Konzeptdefinitionen können nur in der Datei verwendet werden, in der sie definiert sind. Es wird dringend empfohlen, so oft wie möglich die gleichen Konzeptnamen und Konzeptvariablennamen für Konzpete mit ähnlichem Zweck zu verwenden. Weitere Informationen finden Sie unter "Angeben der Werte der Konzeptvariablen" auf Seite 108.
- Bereitstellbare Komponenten können den gleichen Namen haben wenn sie in verschiedenen Dateien definiert sind. Sie werden als unterschiedlich bereitstellbare Komponenten behandelt.
- TQL-Abfragen können den gleichen Namen haben wenn sie in verschiedenen Dateien definiert sind. Sie werden aber als unterschiedliche TQL-Abfragen behandelt. Wenn in einer Abhängigkeitssignaturdatei über den Namen auf eine TQL-Abfrage verwiesen werden soll, muss eine TQL-Abfrage mit diesem Namen in der gleichen Datei vorhanden sein.
- Gültigkeitsbereiche können den gleichen Namen haben wenn sie in verschiedenen Dateien definiert sind. Sie werden als unterschiedliche Gültigkeitsbereiche behandelt. Wenn in einer Abhängigkeitssignaturdatei über den Namen auf einen Gültigkeitsbereich verwiesen werden soll, muss ein Gültigkeitsbereich mit diesem Namen in der gleichen Datei vorhanden sein.
- Bedingungsfunktionen können den gleichen Namen haben wenn sie in verschiedenen Dateien definiert sind. Wenn in einer Abhängigkeitssignaturdatei über den Namen auf eine Funktion verwiesen werden soll, muss eine Funktion mit diesem Namen in der gleichen Datei vorhanden sein.
- Standardwerte für Variablen und Konzeptvariablen sind spezifisch für die Dateien, in denen sie definiert sind. Zwei Variablen oder Konzeptvariablen mit dem gleichen Namen können in unterschiedlichen Dateien einen anderen Standardwert haben.
- Das Festlegen unterschiedlicher Schlüsseleigenschaften für das gleiche Konzept in unterschiedlichen Abhängigkeitssignaturdateien ist nicht zu empfehlen. Dies würde die Verwaltung sowie das korrekte Zuweisen von Werten aus unterschiedlichen Adaptern in ihre Zieldatenvariablen erschweren. Weitere Informationen finden Sie unter "Angeben der Werte der Konzeptvariablen" auf Seite 108.

Kompilierungsfehler

Kompilierungsvalidierungen

- Zwei oder mehrere bereitstellbare Komponenten mit dem gleichen Namen in der gleichen Datei.
- Zwei oder mehrere Gültigkeitsbereiche mit dem gleichen Namen in der gleichen Datei.
- Zwei oder mehrere TQL-Anfragen mit dem gleichen Namen in der gleichen Datei.
- Zwei oder mehrere Bedingungsfunktionen mit dem gleichen Namen in der gleichen Datei.
- Zwei Abhängigkeiten mit dem gleichen Namen in der gleichen bereitstellbaren Komponente.

- Verwenden eines Variablennamens in einer Konfigurationsdateibedingung, die nicht als globale Variable in der gleichen Datei oder als lokale Variable in der Abhängigkeit, der die Konfigurationsdatei zugewiesen ist, deklariert ist.
- Verwenden eines Variablennamens in einer Bedingungsfunktion, die nicht als globale Variable oder als Parameter für die Funktion deklariert ist.
- Zyklische Abhängigkeit zwischen Variablen (wenn beispielsweise die Anweisung zum Enfügen für die Variable VAR_A den Wert vom VAR_B verwendet und der Einfügewert für VAR_B verwendet den Wert der Variable VAR_A).
- Verweisen auf eine TQL-Abfrage, die in der gleichen Datei nicht vorhanden ist.
- Verweisen auf einen Gültigkeitsbereich, der in der gleichen Datei nicht vorhanden ist.
- Verweisen auf eine Bedingungsfunktion, die in der gleichen Datei nicht vorhanden ist.
- Das Verwenden einer Variable, die keinen Standardwert in einer "Ignorieren"-Anweisung oder einen alternativen Ausdruck enthält, löst einen Kompilierungsfehler aus. Weitere Informationen finden Sie unter "Verwenden von Standardwerten für Variablen" auf Seite 81.
- Wenn ein Suchausdruck Variablen mit Standardwerten enthält, müssen Sie alle Fälle verarbeiten, in denen eine beliebige Kombination dieser Variablen dessen Standardwert ausweisen. Dies kann entweder durch vollständiges Ignorieren des Ausdrucks für einige der Variablen oder durch Verwenden alternativer Ausdrücke erfolgen. Sie müssen über alternative Ausdrücke für alle Kombinationen von Variablen verfügen, die nicht in der "Ignorieren"-Anweisung enthalten sind. Andernfalls wird ein Kompilierungsfehler ausgegeben. Weitere Informationen finden Sie unter "Verwenden von Standardwerten für Variablen" auf Seite 81.
- Wenn eine TQL-Abfrage in einem `<DocumentCILocation>`-Tag verwendet wird, aber nicht den folgenden Regeln entspricht:
  - Die TQL-Abfrage muss einen Pfad zwischen der bereitstellbaren Komponente und dem Konfigurationsdokument definieren. Hierbei muss es sich um einen einfachen Pfad handeln (keine Zyklen).
  - Die TQL-Abfrage muss die beiden folgenden Endknoten mit den beiden bestimmten Namen (Groß-Kleinschreibung wird unterschieden) enthalten:
    - Deployable – das bereitstellbare Komonenten-CI im Pfad
    - Configuration_document – das CI im Pfad, das das Konfigurationsdokument angibt
  - Bedingungen sind auf den Knoten **Deployable** und **Configuration_document** nicht zulässig.
  - Die Kardinalität zwischen allen Knoten muss 1..1 sein.
  - Es werden nur reguläre Link-Typen unterstützt. Es sind weder Verbund- oder Join-Beziehungen noch Unterdiagramme zulässig.
- Wenn eine TQL-Abfrage in einem Gültigkeitsbereich verwendet wird, aber nicht den folgenden Regeln entspricht:
  - Der Abfrageknoten für die bereitstellbare Komponente muss "Deployable" (Groß-Kleinschreibung wird beachtet) lauten. Die Ergebnisse dieses Abfrageknotens sind die möglichen bereitstellbaren Komponenten für den Gültigkeitsbereich.

- Es darf nur ein weiterer Abfrageknoten in der TQL-Abfrage enthalten sein (anderer Knoten als "Deployable"). Der Name des anderen Abfrageknotens ist frei wählbar. Dieser Knoten wird als Abfrageknoten für den Gültigkeitsbereich bezeichnet.
- Der Abfrageknoten "Deployable" und der Abfrageknoten für den Gültigkeitsbereich sind über einen Verbundlink verbunden, der alle verfügbaren Pfade zwischen diesen beiden Knoten enthält.
- Die Komponenten mit der Bezeichnung "Deployable" stellt sowohl Benutzer- als auch Provider-Komponenten dar, die im selben Gültgkeitsbereich liegen können. Aus diesem Grund muss der CI-Typ des Abfrageknotens den bereitstellbaren CI-Typen für Benutzer und Provider übergeordnet sein.

- Wenn ein Textkonfigurationsdokument mehrere `< ReferenceLocation>`-Tags mit unterschiedlichen Prioritäten enthält. Weitere Informationen finden Sie unter "Konfigurationsdokumentüberschreibungen" auf Seite 83.

Adapter für die Suche nach Abhängigkeiten

Das Framework für die Suche nach Abhängigkeiten muss über einen dedizierten Adapter ausgeführt werden. Die Zuständigkeiten des Adapters sind:

- Erfassen aller relevanten Verbindungszeichenketten für einen Provider.
- Ausführen des Frameworks für die Suche nach Abhängigkeiten.
- Weiterleiten der Suchergebnisse (die Abhängigkeiten) an UCMDB.

Jeder Adapter verarbeitet einen Providertyp. Ein Adapter verarbeitet beispielsweise JAR-Provider.

Sie können auch mehrere Adapter zum Verarbeiten desselben Providertyps verwenden. In diesem Fall müssen Sie jedoch sicherstellen, dass jeder Provider pro Adapter nur einmal ausgeführt. Nur so erhalten Sie konsistente Ergebnisse.

Ebenso wie bei anderen Adaptern, müssen Sie nach Erstellung des Adapters für die Suche nach Abhängigkeiten einen Discovery-Job erstellen, um die Adapterlogik auszuführen.

Dieser Abschnitt umfasst:

Erstellen eines Abhängigkeitssuchadapters

1. Wählen Sie **Data Flow Management > Adapterverwaltung**.
2. Klicken Sie auf **Neu**  und wählen Sie **Neuer Adapter** aus.
3. Geben Sie die Details des Adapters ein und klicken Sie auf **OK**.
4. Klicken Sie im Ausschnitt **Ressourcen** mit der rechten Maustaste auf den neu erstellten Adapter, und wählen Sie **Adapter-Quelle bearbeiten**.

5. Ersetzen Sie die Zeile:

```
<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.DynamicServic
e">
```

durch

```
<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.WorkflowServic
e">
```

6. Ersetzen Sie die Zeile:

```
<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.DynamicService
Params" ignoreMissingReconciliationRules="false" enableRecording="false"
enableAging="true" useDefaultValueForAging="false"
autoDeleteOnErrors="success" recordResult="false" />
```

durch

```
<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.WorkflowServic
eParams" patternType="workflow_adapter" enableAging="true"
ignoreMissingReconciliationRules="false" enableRecording="false"
autoDeleteOnErrors="success" recordResult="false"
maxThreadRuntime="86400000" useDefaultValueForAging="false">
   <Workflow>
      <steps>
         <step name="Dependencies Discvoery" failure-policy="mandatory">
            <module
type="java">com.hp.ucmdb.discovery.probe.agents.probemgr.accuratedependencie
s.processing.DependenciesDiscoveryWorkflowStep</module>
            <timeoutParking>
               <initialTimeout>180000</initialTimeout>
               <retriesThreshold>12</retriesThreshold>
               <multipleBy>2</multipleBy>
               <maxRetry>10</maxRetry>
               <timeoutThreshold>10800000</timeoutThreshold>
            </timeoutParking>
         </step>
      </steps>
      <finalStep />
      <libraryScripts />
   </workflow>
</params>
```

7. Bereiten Sie ein Skript vor, das die Ergebnisse der Abhängigkeitssignatursuche verarbeitet. Weitere Informationen finden Sie unter "Schreiben eines Jython-Skripts" auf Seite 111.
8. Fügen Sie den folgenden Schritt dem Workflow-Adapter hinzu:

```
<step name="Default Search Result Awaiting" failure-policy="mandatory">
   <module type="jython">[Your Jython Script Name]</module>
   <noParking />
</step>
```

9. Klicken Sie im Ausschnitt **Eingabe** auf **CI-Typ auswählen**  und wählen Sie den Provider-CI-Typ für diesen Adapter aus.
10. Klicken Sie auf die Schaltfläche **Eingabeabfrage bearbeiten**  , und bereiten Sie die Eingabe-TQL-Abfrage vor. Weitere Informationen finden Sie unter "Definieren einer Eingabe-TQL-Abfrage und von Zieldaten" auf der nächsten Seite.
11. Klicken Sie im Ausschnitt **Discovery-CITs** auf  , und fügen Sie den Provider-Typ, alle möglichen Benutzertypen und den Provider/Benutzer-Link-Typ hinzu.
12. Wenn Sie fertig sind, klicken Sie auf **Speichern**.

Definieren eines Benutzer-Provider-Adapters

Die Suchfunktion wird über Discovery-Adapter gesteuert. Jeder Adapter verarbeitet die Suche für einen bestimmten Satz von Verbindungszeichenketten eines Provider-Typs. Die Eingabe-TQL-Abfrage des Adapters ruft alle CIs ab, die solche Verbindungszeichenketten enthalten und gibt Werte in die Variablen der Verbindungszeichenketten und in die Konzepte ein. Dazu werden die Zieldaten des Triggers verwendet. Weitere Informationen finden Sie unter "Entwickeln von Jython-Adaptern" auf Seite 37.

Diese Adapter sind Workflow-Adapter. Sie führen mehrere Schritte zur Durchführung der Suche aus, um Benutzer-Provider-Beziehungen zu ermitteln:

1. Der Adapter führt die Logik zur Zusammenstellung eines konkreten Suchausdrucks für den Konfigurationsdokumentfilter des Gültigkeitsbereichs aus. Dies erfolgt durch die Eingabe der Werte der Verbindungszeichenketten in die globalen Variablen der Abhängigkeitssignatur und durch das Instanziieren von Konzepten.
2. Der Adapter übergibt den konkreten Suchausdruck des Gültigkeitsbereichs an die UCMDB-Solr-Engine, die auf dem UCMDB-Server ausgeführt wird.
3. Der Adapter richtet eine TQL-Abfrage an UCMDB, um aller erforderlichen Informationen abzurufen (z. B. Konfigurationsdokumente, bereitstellbare Deskriptoren), sodass die Suche der Abhängigkeitssignatur ausgeführt werden kann.
4. Der Adapter lädt den Inhalt der erforderlichen Konfigurationsdokumente herunter und speichert ihn in der Data Flow Probe.
5. Der Adapter führt dann die Abhängigkeitssuchen durch, die für den Provider und die Benutzer relevant sind, die anhand des Gültigkeitsbereichs in den Schritten 2 und 3 gefunden wurden.
6. Ein Jython-Skript gibt dann die Ergebnisse der Abhängigkeitssuchen zurück.

Definieren einer Eingabe-TQL-Abfrage und von Zieldaten

Zum Erstellen eines Suchausdrucks aus dem Gültigkeitsbereich und den Bedingungen einer Abhängigkeitssignatur muss jede Variable und jedes Konzept, das im Suchausdruck erwähnt wird durch eine konkrete Verbindungszeichenfolge ersetzt werden (zurückgegeben von der TQL-Abfrage). Eine solche Ersetzung erfolgt auf Basis adapterspezifischer Zuordnungen, die Sie definieren.

Die Eingabe-TQL-Abfrage für einen Abhängigkeitszuordnungsadapter muss folgende Elemente definieren:

- Die Trigger-CIs für die Suche nach dem Provider (der SOURCE-Abfrageknoten).
- Alle Zeichenfolgen, die für den bestimmten Provider erforderlich sind.

Diese TQL-Abfragen sollten alle Verbindungszeichenfolgen zurückgeben, die über die Topologie der CIs verteilt sind, und den Provider identifizieren, der einen Service liefert. Wenn beispielsweise der Provider ein Oracle RAC ist, sollte die TQL-Abfrage alle CIs zurückgeben, die eine Verbindungszeichenfolge enthalten, die ein Benutzer benötigt, um eine Verbindung zu dem RAC herzustellen, und dessen Dienste nutzen zu können. Aus diesem Grund sollte die Layout-Definition der TQL-Abfrage jedes Attribut ausgeben, das eine solche Zeichenfolge enthält. So sollte beispielsweise die TQL-Abfrage für ein RAC die IP-Adresse und und den Port zurückgeben, über den die jede Instanz des RAC zugänglich ist, sowie den RAC-Service-Namen.

Variablen und Konzepte müssen mithilfe der Zieldatendefinition des Adapters Attributen von Abfrage-Knoten in der Eingabe-TQL-Abfrage zugewiesen werden. Jedes Pattern-Element sollte einen eindeutigen Namen haben, um eine korrekte Zuordnung zu ermöglichen.

Wie für jeden Adapter, wird die Eingabe-TQL-Abfrage für einen Suchadapter für jeden beliebigen Trigger ausgeführt, der mit der Trigger-TQL-Abfrage für den mit diesem Adapter verbundenen Job übereinstimmt.

Bei Verwendung der Adapter für Service Discovery, können Sie die Trigger nicht nur auf Trigger beschränken, die vom Server ermittelt wurden (dies geschieht automatisch), sondern auch auf Trigger, die mit der Geschäftsservice-CI (über einige Pfade) verbunden sind. Um dies zu erreichen, definieren Sie die Eingabe-TQL-Abfrage, damit sie einen Pfad zwischen dem Trigger und der Geschäftsservice-CI enthält (in der Regel einige Verbundlink), und geben Sie dem Abfrageknoten der Geschäftsservice-CI den Namen "SERVICE" (Groß-/Kleinschreibung beachten). Wenn das Framework einen "SERVICE"-Abfrageknoten identifiziert, werden die Ergebnisse automatisch auf den Service (oder die Services) beschränkt, zu welchem die Trigger-CI gehört.

Angeben von Variablenwerten

Jeder Suchadapter muss Werte für alle globalen Variablen und Konzepte festlegen, die in Suchbedingungen in der Abhängigkeitssignatur für Abhängigkeiten angezeigt werden, die der Adapter identifizieren soll. Diese Werte werden in der Verbindungszeichenfolge des Providers angegeben (weitere Informationen finden Sie unter "Variablen und Konzepte" auf Seite 72). Sie werden verwendet, um Benutzer für den Provider (den Trigger) zu finden.

Hinweis: Wenn eines der Zuordnungsattribute leer ist, lassen Sie es wie nachfolgend dargestellt:

```
CONTEXT_ROOT = ${WEB_MODULE.j2eemanagedobject_contextroot:}
```

In diesem Beispiel ist CONTEXT_ROOT das Attribut, das einen leeren Wert in der Abhängigkeitssignatur erhalten und ignoriert wird.

So legen Sie den Wert für eine Variable fest:

1. Fügen Sie einen Zieldatenwert ein, der genau den gleichen Namen wie die Variable hat (Groß-/Kleinschreibung ist zu beachten).
2. Legen Sie für den Zieldatenwert einen hartcodierten Wert oder eine Variable fest.

Weitere Informationen finden Sie unter "Entwickeln von Jython-Adaptern" auf Seite 37.

Angeben der Werte der Konzeptvariablen

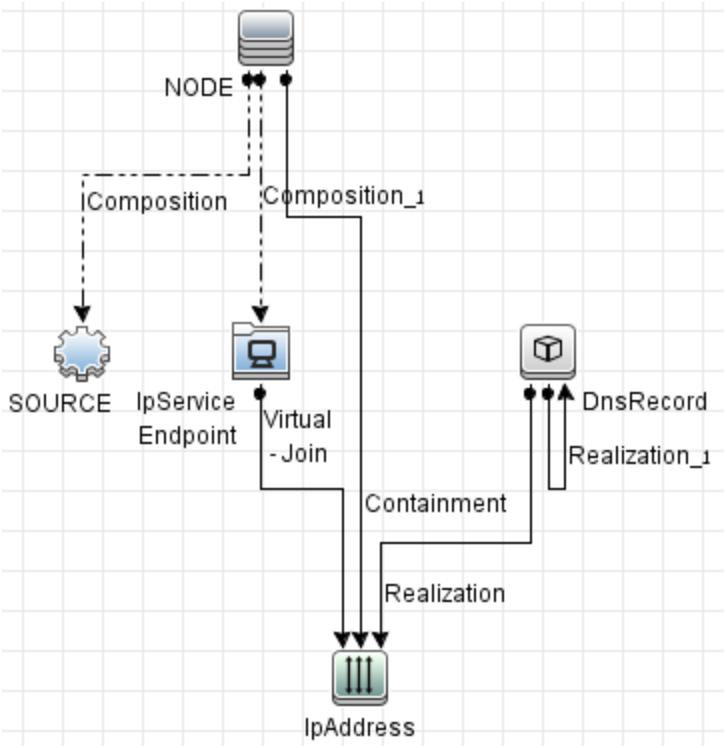
Es ist ein Konzept erforderlich, das einen Satz untrennbarer, eng verzahnter Verbindungszeichenfolgen enthält. Die Eingabe-TQL-Abfrage kann mehrere untrennbare Verbindungszeichenfolgensätze zurückgeben. Beispiel: Eine aktive Softwareinstanz lauscht an mehreren IP-Port-Paaren. Für jeden dieser Sätze sollte eine separate Konzeptinstanz instanziiert werden. Das Schlüsselattribut des Konzepts wird verwendet, um Konzeptinstanzen zu trennen. Der Schlüssel verweist auf ein Musterelement der TQL-Abfrage. Für jede CI-Instanz, die für das Schlüsselmusterelement zurückgegeben wird, wird eine neue Konzeptinstanz erstellt. Auf jede dieser CI-Instanzen wird als **Schlüssel CI-Instanz** verwiesen.

Hinweis: TQL-Abfragen können nicht mehrere Zuordnungsdefinitionen für das gleiche Konzept enthalten.

Die Verbindungszeichenfolgen für jede Konzeptinstanz sollten aus deren entsprechender Schlüssel-CI-Instanz und aus den CIs, die mit der Schlüssel-CI verbunden sind, stammen. Nachfolgend ist ein Beispiel für eine Abhängigkeitssignaturdatei mit dieser Konzeptdefinition aufgeführt:

```
<Concept name="IpEndpoint">
   <Properties>
      <KeyProperty name="PORT"/>
      <Property name="IPADDRESS"/>
      <Property name="DNS"/>
   </Properties>
</Concept>
```

Die Eingabe-TQL-Abfrage des Adapters sieht wie folgt aus:



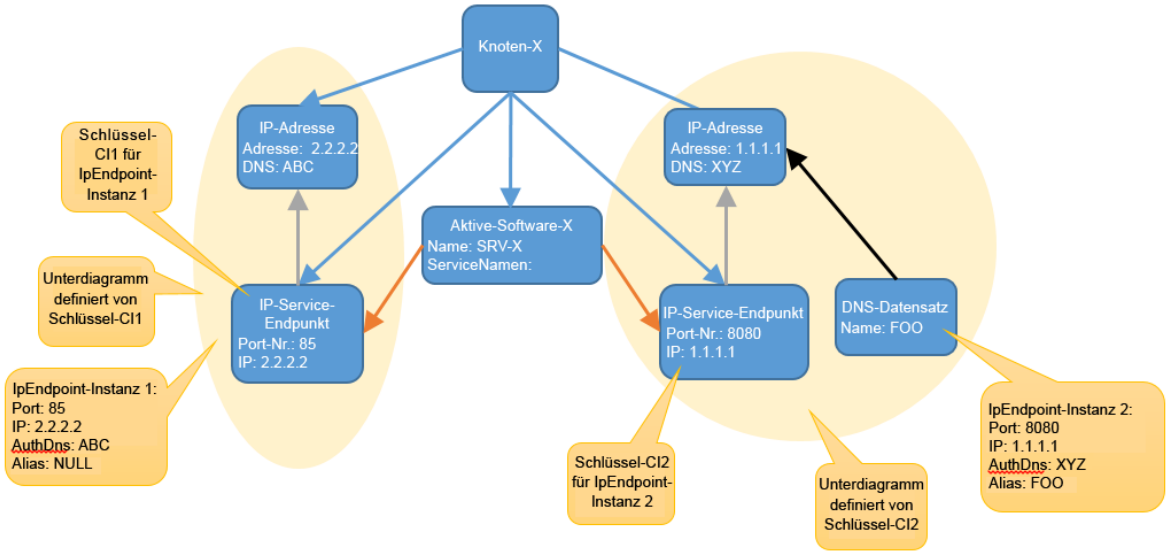
Um Konzepte im Adapter zu instanziieren, muss jede Kozeptvariable, ähnlich wie Variablen, einer Zieldatenvariable zugeordnet werden. Weitere Informationen finden Sie unter "Angeben von Variablenwerten" auf Seite 107.

Die Zieldaten des Adapters sehen wie folgt:

```
IpEndpoint.PORT = SOURCE.NODE.IpServiceEndpoint.name
IpEndpoint.IPADDRESS = SOURCE.NODE.IpServiceEndpoint.IpAddress.name
IpEndpoint.DNS = SOURCE.NODE.IpServiceEndpoint.IpAddress.DnsRecord.name
```

Das Abfrageelement **IpServiceEndpoint** ist also die Schlüssel-CI für das Konzept **IpEndpoint** in diesem Adapter.

In der unten abgebildeten Beispielgrafik einer TQL-Abfrage (das Ergebnis der TQL-Abfrage fürdie Ausführung von Software X), es ist möglich, zwei Schlüssel-CIs zu erkennen. Die Verbindungdzeichenfolgen zum Füllen der Attribute für jede **IpEndpoint**-Konzeptinstanz stammen aus dem Unterdiagramm jeder Schlüssel-CI. So repräsentiert jede Konzeptinstanz einen anderen Satz untrennbarer Verbindungszeichenfolgen.



Im Gegensatz zur Zuordnung von Variablen müssen Sie bei der Zuordnung von Konzeptvariablen den Pfad von der Eingabe-TQL-Abfrage aus dem Abfrageknoten des Triggers (dieser trägt immer den Namen SOURCE) zu dem Abfrageknoten, an den die Variable gebunden sein wird, angeben. Darüber hinaus müssen Sie, sobald die Schlüsselvariable konfiguriert ist, den Pfad der Schlüsselvariable allen anderen Konzeptvariablen als Präfix voranstellen.

Diesem Beispiel sind folgende Informationen zu entnehmen:

- Der Name der Zieldatenvariable ist der Name des Konzepts, gefolgt von dem Namen der Konzeptvariablen.
- Sie können mehrere Konzepte im gleichen Adapter zuordnen, indem Sie den Namen jedes Konzepts in den Zieldatenvariablen angeben.
- Der Name des Pfads beginnt immer mit "SOURCE".
- Den Pfaden von Konzeptvariablen, die nicht die Schlüsselvariable sind, wird der Pfad zur Schlüsselvariable als Präfix vorangestellt.
- Einen Pfad besteht lediglich aus den Namen von Abfrageknoten, die Links sind nicht enthalten. Zwischen zwei im Pfad angegebenen Abfrageknotennamen muss jedoch mindestens ein Link vorhanden sein.

Wenn ein Pfad einen oder mehrere der folgenden Elemente enthält, kann der Trigger während seiner Verteilungsphase nicht ausgelöst werden:

- Ein Abfrageknotenname, der in der Eingabe-TQL-Abfrage nicht vorhanden ist.
- Zwei benachbarte Abfrageknotennamen, die in der Eingabe-TQL-Abfrage nicht verknüpft sind.
- Ein Attributname, der nicht Teil des Ergebnis-CI-Typs ist.

Wenn es mehrere Ergebnis-CIs für einen Abfrageknoten gibt, der nicht der Schlüssel-Abfrageknoten ist, wird die Zieldatenvariable eine Liste aller die Eigenschaftswerte aus diesen CIs sein. Eine Liste kann auf

Basis von Zieldatenvariablen erstellt werden, die einen Pfad enthalten, der in keiner anderen Zieldatenvariable für das gleiche Konzept enthalten ist. Andernfalls wird der Trigger während seiner Verteilungsphase nicht ausgelöst. Im obigen Beispiel kann die Zieldatenvariable `IpEndpoint.IPADDRESS` keine Liste enthalten, weil ihr Pfad in der Zieldatenvariable `IpEndpoint.DNS` enthalten ist. `IpEndpoint.DNS` kann eine Liste enthalten, da sein Pfad in keiner anderen Variable enthalten ist. Dies bedeutet, dass es nur eine Ergebnis-CI aus dem Abfrageknoten **IP-Address** geben kann, die mit einem Ergebnis aus dem Abfrageknoten **IpServiceEndpoint** verknüft ist. Dies sollte berücksichtigt werden wenn Sie Ihre TQL-Abfrage und die Pfade planen.

Selbstlinks in der TQL-Abfrage generieren eine eine Liste mit ähnlichen Ergebnissen wie eine TQL-Abfrage mit mehreren Ergebnis-CIs in einem Abfrageknoten, der nicht der Schlüssel-Abfrageknoten ist. Aus diesem Grund gelten die gleichen Einschränkungen auch für Abfrageknoten mit Selbstlinks.

Weitere Informationen finden Sie unter "Variablen und Konzepte" auf Seite 72.

Schreiben eines Jython-Skripts

Der finale Schritt bei der Erstellung eines Abhängigkeitszuordnungsdapters ist es, ein Jython-Skript zu schreiben, um die Ergebnisse zu protokollieren

Um auf die Ergebnisse der Abhängigkeitssuche zugreifen zu können, müssen Sie diese mithilfe der folgenden Code-Zeilen aus dem Workflow State abrufen:

```
workflowState = Framework.getWorkflowState()
searchResult = workflowState.getProperty
(DependenciesDiscoveryConsts.DEPENDENCIES_DISCOVERY_RESULT)
```

Die Variable **searchResult** ist garantiert nicht-null wenn der Suchschritt erfolgreich abgeschlossen wurde. Es wird empfohlen, den Suchschritt im Workflow als obligatorisch zu definieren, sodass der Jython-Skript-Schritt bei einem fehlgeschlagenen Suchschritt nicht ausgeführt wird. Die Variable wird ein Objekt des Typs **com.hp.ucmdb.discovery.probe.agents.probemgr.accuratedependencies.search.ConsumerDeployableSearchResult** enthalten.

Verwenden Sie dieses Objekt, um die folgenden Schritte durchzuführen:

1. Überprüfen aller vom Benutzer bereitstellbarer Komponenten, die durch die Abhängigkeitssuche gefunden wurden. Beachten Sie: Bei der Suche werden die Verbindungszeichenfolgen des Providers als Eingabe verwendet und alle vom Benutzer bereitgestellten Komponenten mit Abhängigkeit zu dem Provider zurückgegeben.
2. Für jeden Benutzer gibt es möglicherweise mehr als eine Abhängigkeitssignatur. Sie sollten diese auch iterieren. Jede Abhängigkeitssignatur kann unterschiedliche Ausgabevariablenwerte enthalten. Tatsächlich stehen die Werte aller Variablen, die in einer Abhängigkeitssignatur verwendet wurden (globale, lokale oder Konzeptvariablen) dem Jython-Skript in der Suchergebnissen zur Verfügung.
3. Erstellen des Object State Holders (OSH), der den Link zwischen dem Benutzer und dem Provider enthält. Sie können auch die Details des Providers aus dem Suchergebnis-Objekt abrufen.
4. Hinzufügen des OSH zu dem Ergebnis-Vektor (OSHV), und Senden der Ergebnisse an UCMDB.

Nachfolgend ist ein Ausschnitt aus einem Jython-Skript aufgeführt, in dem der oben genannten Workflow ausgeführt wird:

```
# Get the search results object from the Workflow State
workflowState = Framework.getWorkflowState()
searchResult = workflowState.getProperty
(DependenciesDiscoveryConsts.DEPENDENCIES_DISCOVERY_RESULT)

# Prepare the OSHV that will contain the dependencies
oshv = ObjectStateHolderVector()
dependencyCount = 0

# Retrieve the provider OSH
providerServiceOsh = searchResult.getProviderDeployable().getDeployable()

# Loop through all the consumer deployable components
for index in range(0, searchResult.size()):
   deployable = searchResult.get(index)

   # Get the consumer deployable component’s OSH
   deployableOsh = deployable.getDeployable()

   # Create an OSH for the dependency relationship between the consumer and
provider
   consumerProviderLink = modeling.createLinkOSH(‘consumer_provider’,
deployableOsh, providerServiceOsh)

   references = []

   # Iterate through all of the dependencies that were found between the
consumer and the provider in deployable.getDependencies():

      # Extract the name of the dependency as it appears in the dependency
signature file
      dependencyName = dependency.getDependencyName()

      # Get the values of all the variables that were used by the dependency
      variables = dependency.getExportVariables()

      # Aggregate the values of the variable named REFERENCE
      # Note: The value of a variable can be a list if the variable contains
multiple values
      dependencyNames.append(dependencyName)
      for var in variables:
         varName = var.getName()
         values = var.getValues()
         if varName.lower() == REFERENCES:
            references += list(values)
```

```
   reference = references and ','.join(references)
   if reference:
      consumerProviderLink.setAttribute(REFERENCES, reference)

   # Add the link to the results OSHV
   oshv.add(consumerProviderLink)

# Send the result to the UCMDB
Framework.sendObjects(oshv)
```

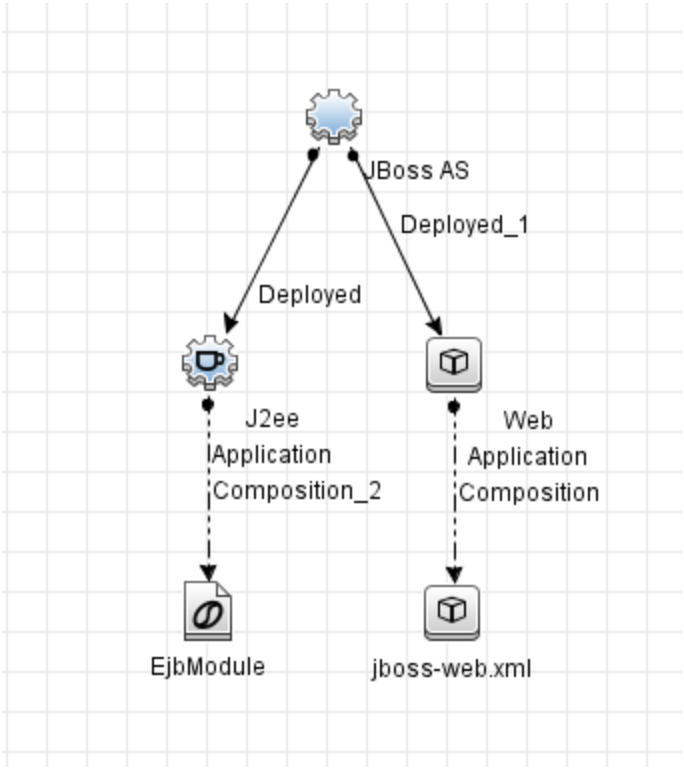
Hinweis: Die Serviceumfangregeln (ein Satz von Regeln für die Discovery-Regel-Engine) werden auf der Basis von Informationen ausgeführt, die vom Skript gesendet werden. Aus diesem Grund ist es wichtig, so viele Informationen wie möglich über die Beziehung, den Benutzer und den Provider zu senden. Es wird empfohlen, die Beziehung mit dem Benutzer- und Provider-OSH zu erstellen, der aus dem Suchergebnis-Objekt abgerufen wurd, wie oben angezeigt. Dieser OSH enthält alle erforderlichen Informationen, damit die Serviceumfangregeln (und andere Regel-Engine-Regeln) ordnungsgemäß funktionieren.

Adaptereinschränkungen

- Adapter zur Abhängigkeitszuordnung werden für Data Flow Probes nicht unterstützt, die im separaten Modus installiert sind.
- Wenn der UCMDB-Server neu gestartet wird, können Trigger für die Abhängigkeitszuordnung von Adapter-Jobs wegen einer Zeitüberschreitung fehlschlagen. Die Trigger sollten bei der nächsten geplanten Ausführung erfolgreich ausgeführt werden.

Vollständiges Beispiel

Dieser Abschnitt enthält ein vollständiges Beispiel zur Entwicklung einer Abhängigkeitssignatur und eines Adapters, die die Abhängigkeiten zwischen einer J2EE-Applikation und einer Webapplikation finden, die auf dem gleichen JBoss AS bereitgestellt werden.



Dieser Abschnitt umfasst die folgenden Themen:

Entwicklungs-Workflow

Nachfolgend ist der allgemeine Workflow für die Entwicklung einer Abhängigkeitssignatur und eines Adapters aufgeführt:

1. Den Provider-CI-Typ für die Signatur identifizieren, die Sie entwickeln werden. In diesem Beispiel wird der Provider-Typ die J2ee Application sein.
2. Entscheiden über die vom Benutzer bereitstellbaren Komponenten, die für den ausgewählten Provider relevant sind. Da die Signatur für die Konfigurationsdokumente des Benutzers spezifisch ist, erhält jeder Benutzer seine eigenen Abhängigkeitssignaturen.

   In diesem Beispiel wählen wir eine vom Benutzer bereitstellbare Komponente aus – eine Webapplikation in JBoss.
3. Entscheiden, ob diese Signaturen zu einer vorhandenen Abhängigkeitssignaturdatei hinzugefügt werden oder eine neue Datei erstellt wird.

   Es besteht kein funktionaler Unterschied zwischen den beiden Methoden. Wählen Sie die Option aus, die die Wartung der Abhängigkeitssignaturen möglichst effizient vereinfacht. Wenn beispielsweise eine vom Benutzer bereitstellbare Komponente in einer der Abhängigkeitssignaturdateien bereits vorhanden ist, ist dessen Wartung wahrscheinlich einfacher, wenn Sie diese neue Abhängigkeit der vorhandenen breitzustellenden Komponente hinzufügen. In diesem Beispiel erstellen wir eine neue Abhängigkeitssignaturdatei, die die neue Abhängigkeitssignatur enthält.
4. Überprüfen, ob einen neuer Adapter erforderlich ist, oder der Provider bereits durch einen

vorhandenen Adapter abgedeckt ist. Beachten Sie, dass die Trigger-CI der Suchadapter der Provider-CI-Typ ist. Folgendes sollte überprüft werden:

a. Gibt es einen bestehenden Suchadapter, der diesen CI-Typ bereits als Trigger-CI verwendet?

   Falls nicht, müssen Sie einen solchen Adapter erstellen. Dies werden in diesem Beispiel tun.

b. Falls ein Suchadapter vorhanden ist, werden alle erforderlichen Verbindungszeichenkettenvariablen und -konzepte den Zieldaten zugeordnet?

   Falls nicht, müssen Sie den vorhandenen Adapter aktualisieren und diese Variablen den Zieldaten hinzufügen. Beachten Sie, dass die Zieldaten aus der Eingabe-TQL-Abfrage extrahiert werden. Dies bedeutet, dass die Eingabe-TQL-Abfrage möglicherweise geändert werden muss, um die neuen Informationen liefern zu können.

5. Bereiten Sie einen Job für den Adapter vor, sofern noch nicht vorhanden.
6. Stellen Sie sicher, dass die Trigger-TQL-Abfrage auch Ihre Provider als Trigger ausgibt.

Entwickeln von Abhängigkeitssignaturen

Entwickeln von Abhängigkeitssignaturen zwischen einer JBoss J2EE Application und einer Web Application.

1. Verstehen, welche Verbindungszeichenfolgen zum Erstellen der Abhängigkeit zwischen dem Benutzer und dem Provider erforderlich sind. In diesem Beispiel ist die erforderliche Verbindungszeichenkette der JNDI-Name des EJB aus EJB-Modulen der Provider-J2EE Application.
2. Alle Verbindungszeichenketten sollten als globale Variablen oder Konzepte definiert sein. Die Werte dieser Variablen werden durch den Adapter eingefügt. Bevor diese Variablen in der Abhängigkeitssignaturdatei (einer neuen oder bereits vorhandenen) definiert werden, überprüfen Sie in allen Dateien, ob ähnliche Variablen und Konzepte bereits vorhanden ist. Falls dies der Fall ist, geben Sie Ihren Variablen, Konzepten und Konzeptvariablen dieselben Namen wie sie die vorhandene Elemente tragen. Dadurch wird die Entwicklung des Adapters einfacher, da die Anzahl der einzufügenden globalen Variablen nicht wächst. In diesem Beispiel fügen wir diese Definitionen einer neuen Datei hinzu, sodass die Datei wird folgendermaßen aussehen wird:

```
<VariableDeclarations>
    <Variable name="EJB_JNDI_NAME"/>
</VariableDeclarations>
```

3. Analysieren Sie die Konfigurationsdateien und identifizieren Sie:
   - Den Speicherplatz der einzelnen Verbindungszeichenfolgen in den einzelnen Konfigurationsdateien.
   - Den Typ jeder Datei: Eigenschaftsdatei, XML-Datei oder andere Textdatei.
   - Die Beziehung zwischen den einzelnen Dateien (bei mehreren Dateien).

   In diesem Beispiel definiert JBoss Web Application die EJB-Referenz in der Konfiguration **jboss-web.xml** wie folgt:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
    <!-- A reference to an EJB in the same server with a custom JNDI binding
-->
    <ejb-ref>
        <ejb-ref-name>ejb/BHome</ejb-ref-name>
        <jndi-name>someapp/ejbs/beanB</jndi-name>
    </ejb-ref>
    <!-- A reference to an EJB in an external server -->
    <ejb-ref>
        <ejb-ref-name>ejb/RemoteBHome</ejb-ref-name>
        <jndi-name>jnp://otherserver/application/beanB</jndi-name>
    </ejb-ref>
</jboss-web>
```

Der Wert im Abschnitt `<ejb-ref-name>` ist die Referenz der Verbindungszeichenkette. Da dies eine XML-Datei ist, können wir den folgenden XPath-Ausdruck verwenden, um eine Übereinstimmung mit dem JNDI-Namen des EJB zu erreichen:

```
//jboss-web/ ejb-ref/ ejb-ref-name[matches(., '^${EJB_JNDI_NAME}$')]
```

4. Sie müssen einen Gültigkeitsbereich definieren, der einen Standardsuchausdruck enthält, wodurch die durch die Abhängigkeitszuordnung erforderliche Konfigurationsdatei gefunden werden kann. In diesem Beispiel ist das Schlüsselwort der JNDI-Name selbst. Nachfolgend wird der Gültigkeitsbereich definiert:

```
<ScopeDefinitions>
    <Scope name="JBoss_EJB_Same_Cell">
        <ConfigurationDocumentContentFilter>
            <Operator type="and">
                <Operand value="${EJB_JNDI_NAME}"/>
            </Operator>
        </ConfigurationDocumentContentFilter>
    </Scope>
</ScopeDefinitions>
```

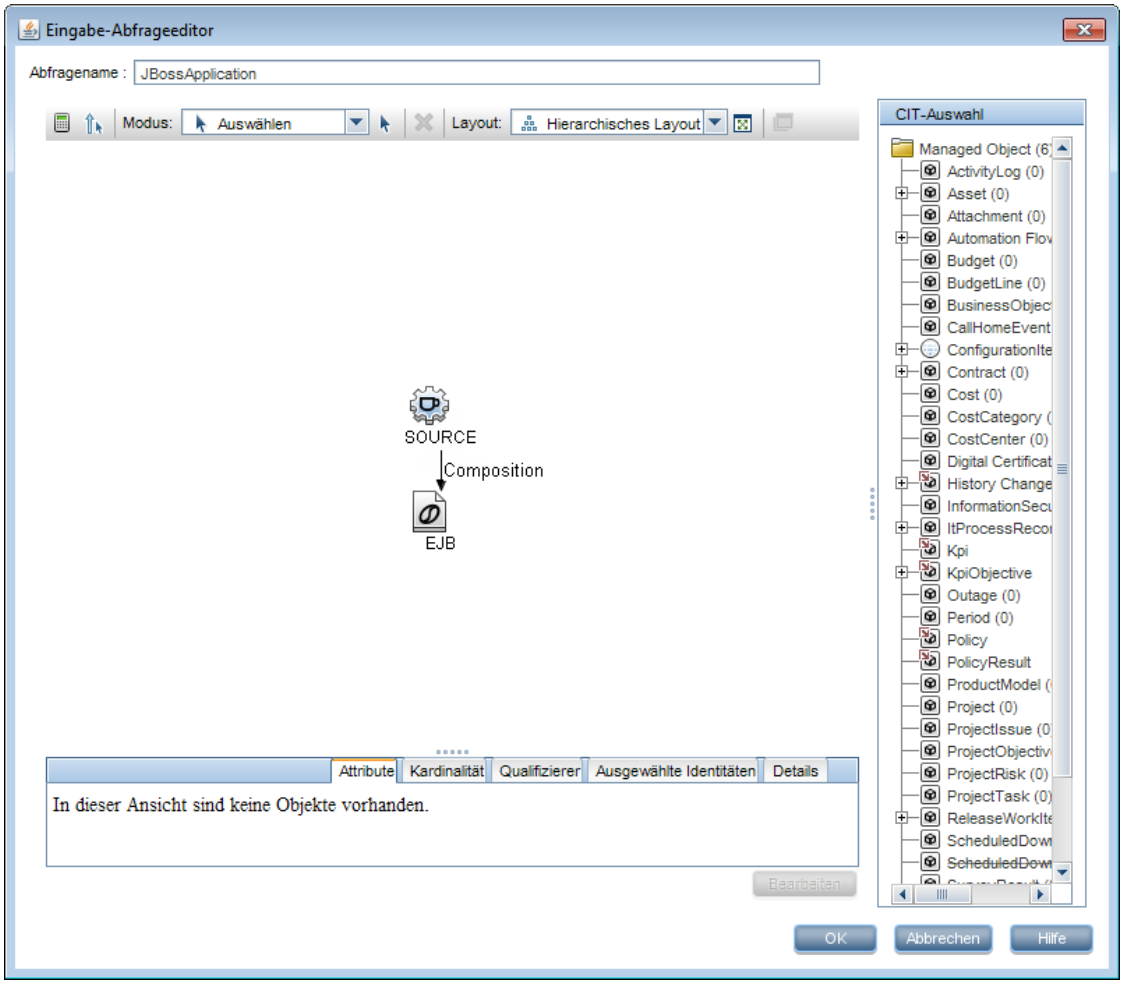
Die vollständige Abhängigkeitssignatur sieht wie folgt aus:

```
<?xml version="1.0"?>
<DependencySignatures xmlns="http://www.hp.com/ucmdb/1-0-0/Dependencies">
    <VariableDeclarations>
        <Variable name="EJB_JNDI_NAME"/>
    </VariableDeclarations>
    <Deployable name="JBoss J2EE Application to Web Application by JNDI" >
        <Descriptor cit="webapplication"/>
        <Dependency name="J2EE Application with Internal EJB"
providerCiType="j2eeapplication" scope="JBoss_EJB_Same">
```

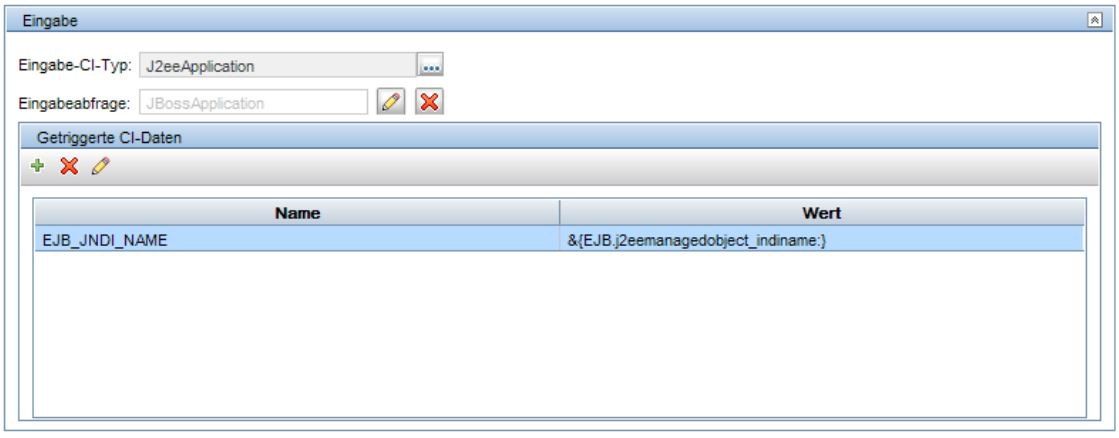
```
            <XmlConfigurationDocument name="jboss-web.xml">
                <Condition>
                    <Operator type="or">
                        <XPathCondition>
                            <XPath>//ejb-ref/jndi-name[matches(., '^${EJB_JNDI_
NAME}$', 'i')]</XPath>
                        </XPathCondition>
                    </Operator>
                </Condition>
            </XmlConfigurationDocument>
        </Dependency>
    </Deployable>
    <ScopeDefinitions>
        <Scope name="JBoss_EJB_Same_Cell">
            <ConfigurationDocumentContentFilter>
                <Operator type="and">
                    <Operand value="${EJB_JNDI_NAME}"/>
                </Operator>
            </ConfigurationDocumentContentFilter>
        </Scope>
    </ScopeDefinitions>
</DependencySignatures>
```

Entwickeln des Adapters

1. Erstellen Sie einen neuen Workflow-Adapter, wie unter "Erstellen eines Abhängigkeitssuchadapters" auf Seite 104 beschrieben.
2. Stellen Sie den Trigger-CI-Typ auf den Provider-Typ ein. In diesem Beispiel ist J2EE Application der Provider.
3. Definieren Sie die Eingabe-TQL-Abfrage für den Adapter, um die erforderlichen CIs und ihre Attribute anzuzeigen. In diesem Beispiel müssen wir die Trigger-J2EE-Application-CI zusammen mit EJBModule-CIs auslösen, die zum Provider J2EE Application gehören.



4. Weisen Sie die globalen Variablen, die in den Signaturen für die erforderlichen Verbindungszeichenkettenvariablen und -konzepte definiert sind, den Trigger-CI-Daten zu. In diesem Beispiel wir weisen Sie das Attribut j2eemanagedobject_jndiname aus der EJBModule-CI den Zieldaten EJB_JNDI_NAME zu.



5. Fügen Sie im Ausschnitt **Workflow-Schritte** die folgenden Workflow-Definition ein:

```
<Workflow>
    <steps>
        <step name="Accurate Dependency Search" failure-policy="mandatory">
            <module type="jython">DependenciesDiscovery.py</module>
            <timeoutParking>
                <initialTimeout>60000</initialTimeout>
                <retriesThreshold>1</retriesThreshold>
                <multipleBy>1</multipleBy>
                <maxRetry>20</maxRetry>
                <timeoutThreshold>60000</timeoutThreshold>
            </timeoutParking>
        </step>
    </steps>
    <finalStep>
        <module type="jython">AccurateDependencyMapping.py</module>
    </finalStep>
    <libraryScripts />
</workflow>
```

   Sie können die Zeitüberschreitungsdauer durch Konfigurieren des Paramenters **timeoutParking** ändern.

6. Erstellen Sie eine Trigger-TQL-Abfrage für den neuen Adapter.
7. Fügen Sie die Jobdefinition in **ServiceDiscoveryActivityType** wie folgt ein:

```
<ServiceDiscoveryActivityType id="top-down" displayName="Top-down">
    <JobsDefinitions>
       …
        <job id=" JBoss Application to Web Application " displayName=" JBoss
Application to Web Application ">
            <patternId>JBossApplication2WebApplication</patternId>
            <triggers>
                <trigger>jboss_application_trigger</trigger>
            </triggers>
            <parameters/>
        </job>
       …
    </JobsDefinitions>
</ServiceDiscoveryActivityType>
```

Kapitel 5: 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.

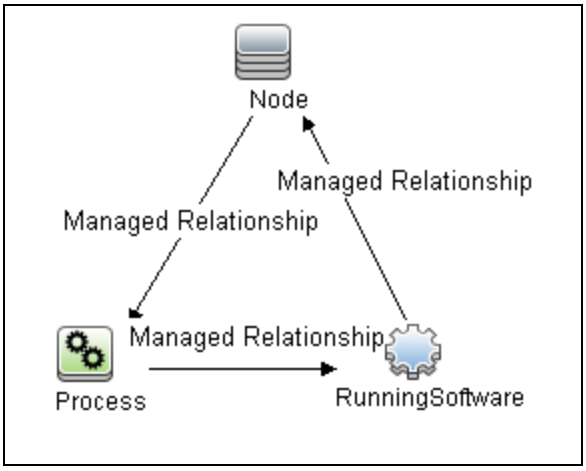
TQL-Abfragen für allgemeine Datenbankadapter

Für Auffüllungsjobs muss jedes erforderliche CI-Layout im Dialogfeld **Layouteinstellungen** von Modeling Studio geprüft werden. Weitere Informationen finden Sie unter Query Node/Relationship Properties Dialog Box 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:

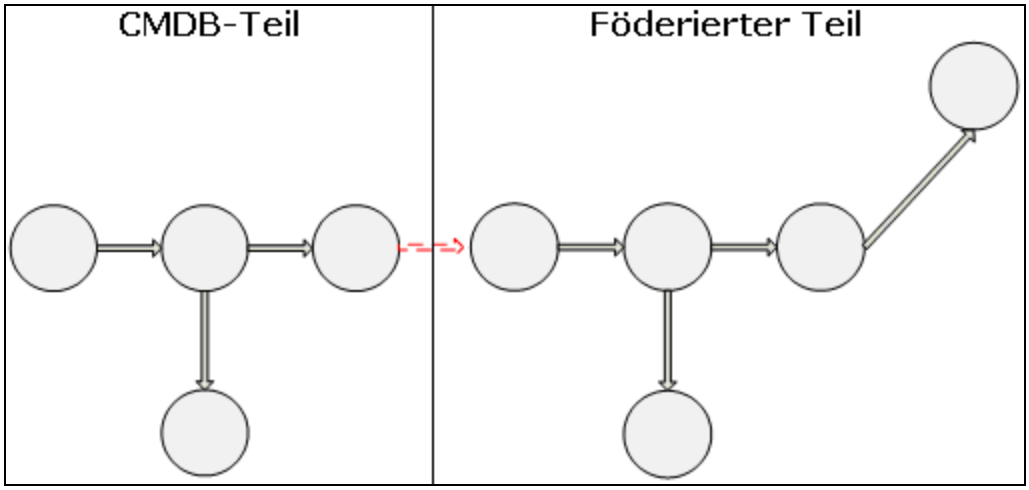
- Unterdiagramme werden nicht unterstützt.
- Verbundbeziehungen werden nicht unterstützt.
- Zyklen oder Zyklusteile werden nicht unterstützt.

  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 126.
- 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 192.

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.

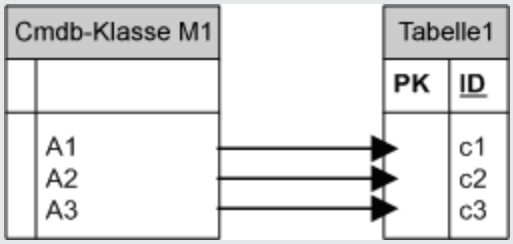
Dieser Abschnitt umfasst außerdem die folgenden Themen:

Beispiele für eine objektrelationale Zuordnung

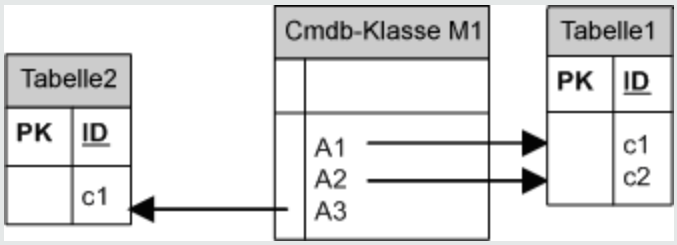
Die folgenden Beispiele beschreiben eine objektrelationale Zuordnung:

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.

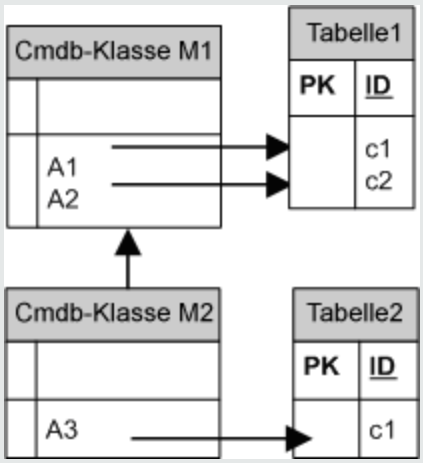


Beispiel für die Zuordnung einer CMDB-Klasse zu zwei Datenbanktabellen:



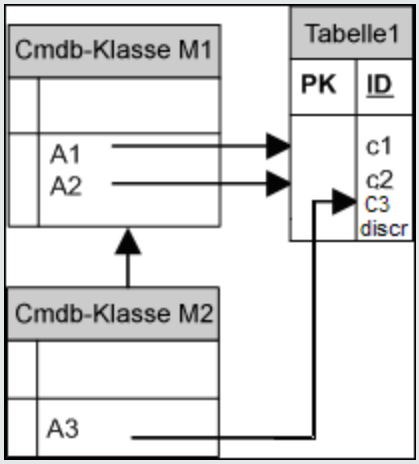
Beispiel für Vererbung:

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.



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 das Definieren von Zuweisungen 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" unten
- "Erstellen eines CI-Typs" auf Seite 128
- "Erstellen einer Beziehung" auf Seite 128

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 123.

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 123.

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.

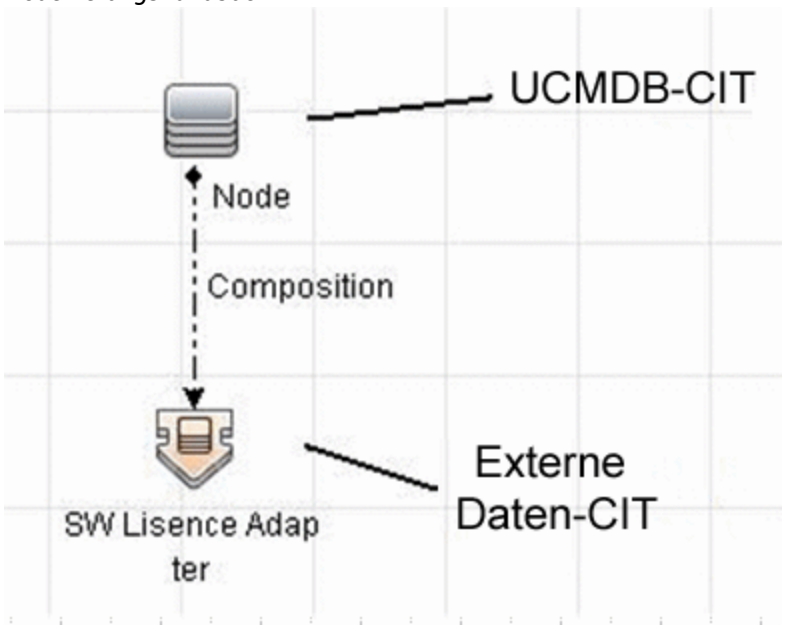
   a. Greifen Sie in UCMDB auf CIT Manager zu und erstellen Sie einen neuen CI-Typ. Weitere Informationen finden Sie unter Erstellen eines CI-Typs im *HP Universal CMDB – Modellierungshandbuch*.

   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.

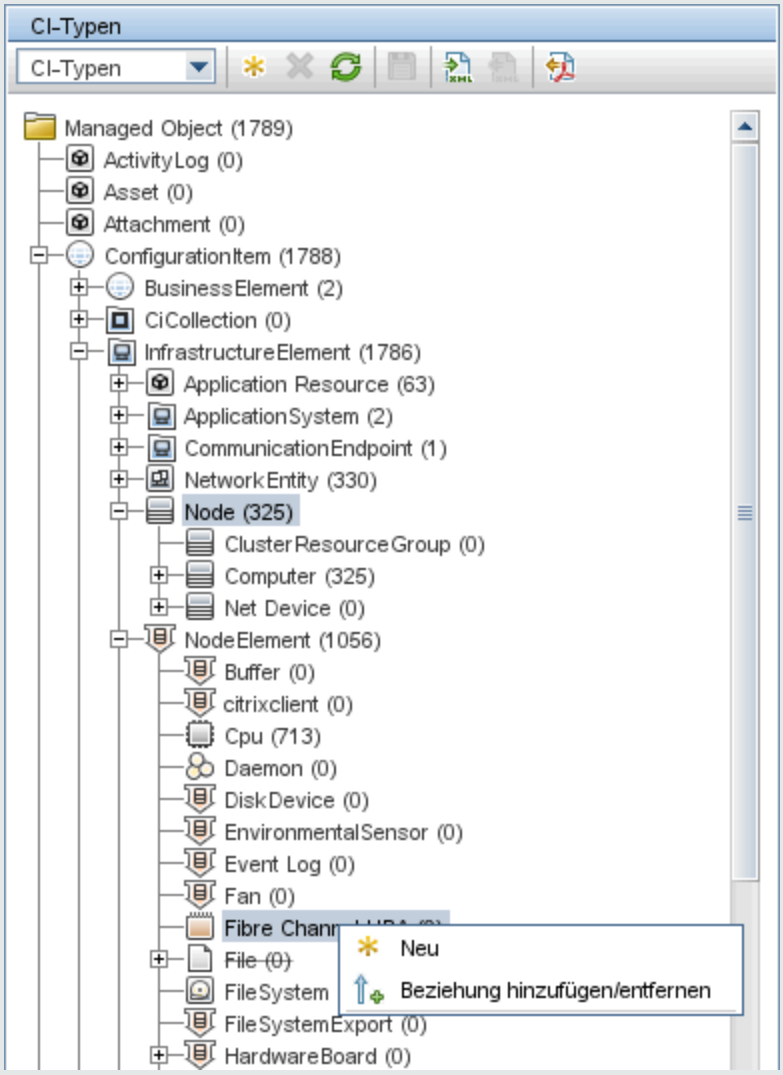
   Fügen Sie passende, gültige Beziehungen zum neuen CIT hinzu. Weitere Informationen finden Sie unter Dialogfeld "Beziehung hinzufügen/entfernen" im *HP Universal CMDB – Modellierungshandbuch*.



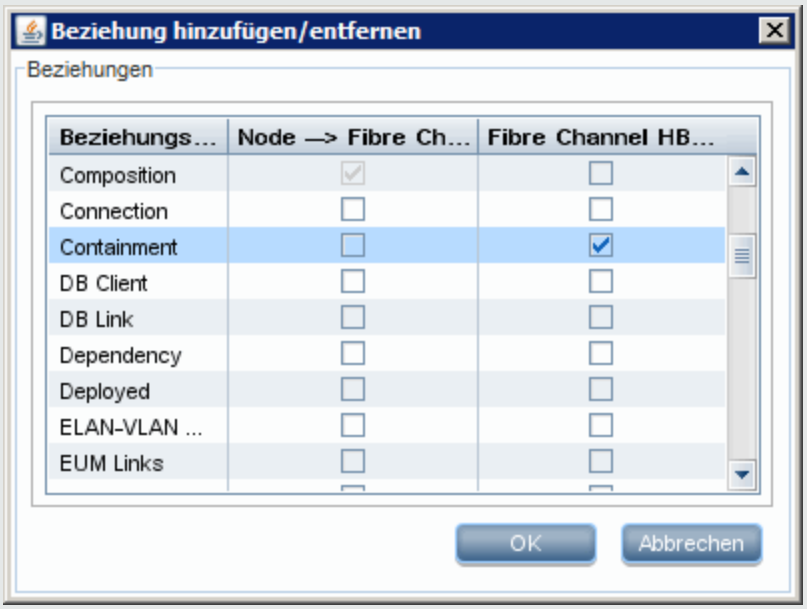
Hinweis: In diesem Stadium können Sie die föderierten Daten noch nicht sehen und die externen Daten noch nicht auffüllen, da Sie noch keine Methode zum Einbinden der Daten definiert haben.

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 Anzeigelabel 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>
  ```

  Um wieder zur neuen Zuordnungs-Engine zurückzukehren, setzen Sie das Element auf den folgenden Wert zurück:

  ```
  <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" unten.
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 130).

   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

Die vereinfachte (minimale) Methode ist eine Methode für die Erstellung der Zuordnungsdatei **simplifiedConfiguration.xml**, die vom Adapter verwendet wird. Diese Methode ermöglicht eine grundlegende Auffüllung oder Föderation eines einzelnen CIT.

Die Anweisungen in diesem Abschnitt beschreiben eine Methode zur Zuordnung des Klassenmodells für bestimmte CI-Typen in der CMDB zu einem RDBMS.

Alle in diesem Abschnitt erwähnten Konfigurationsdateien befinden sich im Package **db-adapter.zip** im Ordner **C:\hp\UCMDB\UCMDBServer\content\adapters**, den Sie unter "Vorbereiten des Adapter-Packages" auf Seite 130 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

Konfigurieren der Datei "adapter.conf"

So 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**.

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:

1. "Erstellen der Datei "simplifiedConfiguration.xml"" unten
2. "Erstellen der TQL" auf der nächsten Seite
3. "Erstellen eines Integrationspunkts" auf Seite 136

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>
   ```

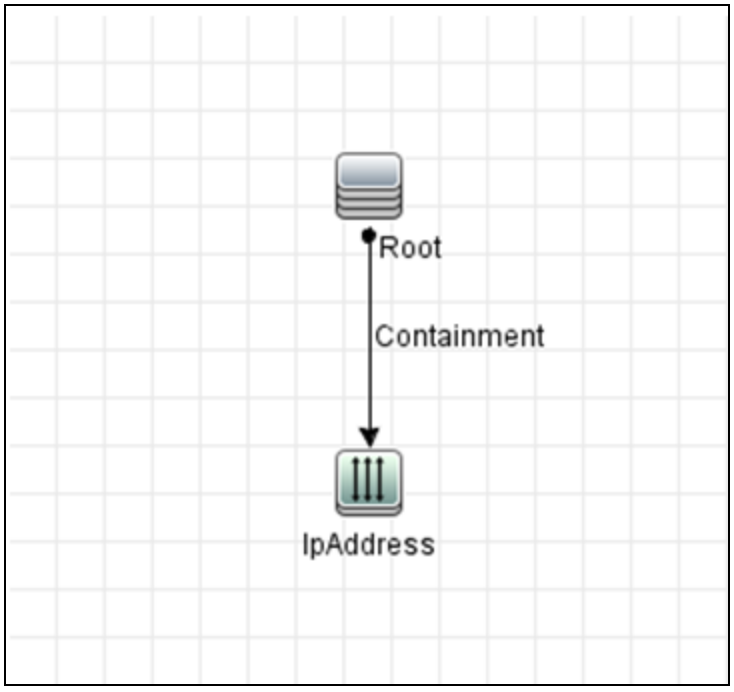
Der Inhalt der Datei **simplifiedConfiguration.xml** wird nun wie folgt angezeigt:

```
<?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="simpleNode">
       <primary-key column-name="host_id"/>
   <reconciliation-by-two-nodes connected-node-cmdb-class-name="ip_address" cmdb-
link-type="containment">
       <or is-ordered="true">
       <attribute cmdb-attribute-name="name" column-name="host_name" ignore-
case="true"/>
       <attribute cmdb-attribute-name="data_note" column-name="note" ignore-
case="true"/>
          <connected-node-attribute cmdb-attribute-name="name" column-name="ip_
address"/>
       </or>
       </reconciliation-by-two-nodes>
   </cmdb-class>
</generic-db-adapter-config>
```

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 die CI-Typen **Node** und **IpAddress** mit einer Containment-Beziehung.
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 **Bestimmte Attribute** unter **Bedingung für 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 **Bestimmte Attribute** unter **Bedingung für 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. Navigieren 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 bei der Vorbereitung des Adapter-Package extrahiert haben. Weitere Informationen finden Sie unter "Vorbereiten des Adapter-Packages" auf Seite 130.

Diese Aufgabe umfasst folgende Schritte:

- "Konfigurieren der Datei "orm.xml"" unten
- "Konfigurieren der Datei "reconciliation_rules.txt" " auf Seite 140

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.

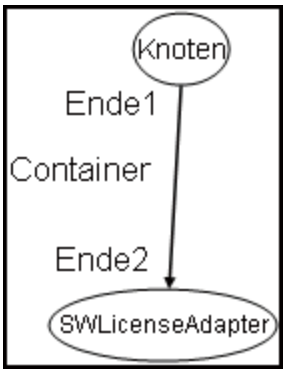
   Diese Datei enthält standardmäßig eine Vorlage, die Sie zum Zuordnen der benötigten Anzahl von CITs und Beziehungen verwenden können.

   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:

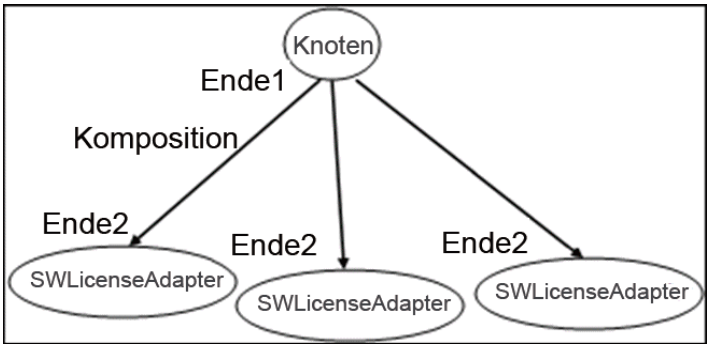
   - 1:1:

   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>
```

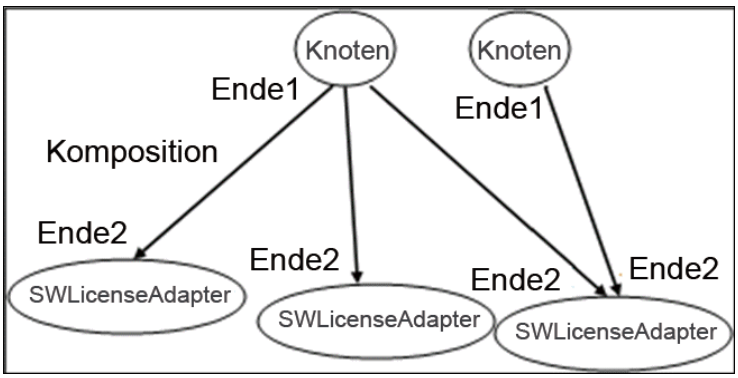
- n:1:



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 160.

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 128 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_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 169.

Implementieren eines Plugins

In diese Aufgabe wird beschrieben, wie Sie einen allgemeinen DB-Adapter mit Plugins implementieren und bereitstellen.

Hinweis: Stellen Sie vor dem Schreiben eines Plugins für einen Adapter sicher, dass Sie die erforderlichen Schritte unter "Vorbereiten des Adapter-Packages" auf Seite 130 durchgeführt haben.

1. Option 1 - Schreiben eines Java-basierten Plugins
   a. 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**

   b. 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

Plugin-Typ	Schnittstellenname	Methode
Abrufen unterstützter Abfragen	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Ändern der TQL-Abfragedefinition und der Ergebnisse	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Ändern der Layoutanforderung für CIs	FcmdbPluginGetCIsLayout	getCisLayout
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>
```

c. 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**.

d. 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.

2. Option 2 - Schreiben eines Groovy-basierten Plugins

a. Erstellen Sie im Menü **Adapterverwaltung** unter den Konfigurationsdateien des Adapter-Package eine Datei im Groovy-Code (MyPlugin.groovy).

b. Implementieren Sie in der Groovy-Klasse die entsprechenden Schnittstellen. Die Schnittstellen sind in der Datei **db-interfaces.jar** definiert (siehe Tabelle oben).

3. 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.

4. 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 "Adapterverwaltung" 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*.

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 Topology Reports Overview 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 190.

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.

In dieser Aufgabe wird beschrieben, 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 der Version 1.2.2.20100217-2310 oder höher.

Diese Aufgabe umfasst folgende Schritte:

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 28 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 Port-Nummer 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 muss 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 169.

    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 148 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" unten 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 169.

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 – Modellierungshandbuch*.

    b. Erstellen Sie einen Integrationspunkt mit dem Adapter, den Sie im Schritt "Erstellen eines neuen Integrationspunkts" oben 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 147.

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:

Allgemeine Konfiguration

- **adapter.conf.** Die Konfigurationsdatei des Adapters. Weitere Informationen finden Sie unter "Die Datei "adapter.conf"" auf der nächsten Seite.

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 Seite 154.

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 156.
- **reconciliation_types.txt**. Enthält die Abstimmungsregeln. Weitere Informationen finden Sie unter

"Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 169.

- **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 171.
- **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 172.
- **Replication_config.txt.** Diese Datei enthält eine kommagetrennte Liste der CI- und Beziehungstypen, deren Eigenschaftsbedingungen vom Replizierungs-Plugin unterstützt werden. Einzelheiten hierzu finden Sie unter "Die Datei "replication_config.txt"" auf Seite 173.
- **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 173.

Hibernate-Konfiguration

- **persistence.xml.** Wird zum Überschreiben von Hibernate-Standardkonfigurationen verwendet. Weitere Informationen finden Sie unter "Die Datei "Persistence.xml"" auf Seite 174.

Aktivieren temporärer Tabellenunterstützung für den Adapter

Durch die Aktivierung temporärer Tabellen kann der Adapter effizienter mit der Remote-Datenbank arbeiten, sodass die Datenbank und das Netzwerk entlastet werden, wodurch wiederum die Leistung verbessert wird.

Um die temporäre Tabellenunterstützung im allgemeinen Datenbankadapter zu aktivieren, müssen folgende Bedingungen erfüllt sein:

- Die zum Herstellen der Verbindung mit der Datenbank angegebenen Anmeldeinformationen enthalten die Berechtigung zum Erstellen, Modifizieren und Löschen temporärer Tabellen.
- Die folgenden Einstellungen müssen in der Konfigurationsdatei **adapter.config** konfiguriert sein:

  temp.tables.enabled=true

  performance.enable.single.sql=true

Hinweis: Temporäre Tabellen werden nur für Microsoft SQL und Oracle unterstützt.

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, 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.
- **stringlist.delimiter.of.<CIT-Name>.<Attributname>=<delimiter>**. Um ein Attribut vom Typ **stringlist** zu einer Datenbankspalte im allgemeinen Datenbankadapter zuzuordnen, muss das Attribut einer Zeichenkettenspalte zugeordnet werden, die eine Liste verketteter Werte enthält. Beispiel: Wenn Sie das Attribut **policy_category** dem CI-Typ **policy** zuordnen möchten und die Zeichenkettenspalte eine Liste der folgenden Werten enthält: value1##value2##value3 (die eine Liste der 3 Werte value1, value2, value3 definieren), verwenden Sie die folgende Einstellung: **stringlist.delimiter.of.policy.policy_category=##.**
- **temp.tables.enabled=true**. Ermöglicht die Verwendung temporärer Tabellen zur Verbesserung der Leistung. Nur verfügbar, wenn **performance.enable.single.sql** aktiviert ist (wird nur in Microsoft SQL und Oracle unterstützt). Eventuell sind bestimmte Berechtigungen im Datenbankserver erforderlich.
- **temp.tables.min.value=50**. Legt die Anzahl der Bedingungswerte (oder IDs) fest, die für die Verwendung temporärer Tabellen erforderlich sind.

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"

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

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 im Internet 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/-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 aufweisen: `<Ende_1>_<Link_Name>_<Ende_2>`. Beispielweise definiert `node_composition_`

Elementname/-pfad	Beschreibung	Attribute
		`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. **Verwendung:** 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`. **Verwendung:** 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 Klammern eingeschlossen werden. **Verwendung:** Erforderlich **Typ:** Zeichenkette
secondary-table (entity-mappings > entity)	Dieses Element kann zur Definition einer sekundären Tabelle für die DB-Entität verwendet werden. Die Tabelle	**Name:** name **Beschreibung:** Der Name der sekundären Tabelle. Wenn der Name der Tabelle nicht das zugehörige

Elementname/-pfad	Beschreibung	Attribute
	muss über eine 1:1-Beziehung mit der primären Tabelle verbunden werden. Sie können mehrere sekundäre Tabellen definieren. Optional.	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. **Verwendung:** 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. **Verwendung:** 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. **Verwendung:** 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. • 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. **Verwendung:** Erforderlich **Typ:** Zeichenkette
discriminator-value	Dieses Element definiert den Typ	

Elementname/-pfad	Beschreibung	Attribute
(entity-mappings > entity)	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. **Verwendung:** 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. **Verwendung:** Erforderlich **Typ:** Zeichenkette
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. **Verwendung:** 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

Elementname/-pfad	Beschreibung	Attribute
		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. **Verwendung:** Erforderlich **Typ:** Entweder `Ende1` oder `Ende2` 2. **Name:** target-entity **Beschreibung:** Der Name der Entität, auf die sich das Ende bezieht. **Verwendung:** 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. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** name **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. **Verwendung:** 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**.

Konfigurieren einer spezifischen "orm.xml"-Datei für jede Remote-Produktversion

Es besteht die Möglichkeit, eine spezifische **orm.xml**-Datei zu konfigurieren, sodass der Adapter für eine bestimmte Remote-Produktversion eine spezifische **orm.xml**-Datei verwendet. Wenn beispielsweise der Remote-Datenspeicher zwei Produktversionen (x und y) aufweist, kann für jede Version eine andere Entitätenzuordnung konfiguriert werden.

So konfigurieren Sie eine spezifische orm.xml-Datei pro Remote-Produktversion:

1. Fügen Sie einen Parameter namens **version** zur Datei **adapter.xml** hinzu und geben die möglichen Versionswerte als **valid-values** ein.
2. Erstellen Sie im Adapter-Package unter dem Ordner **META-INF** einen Ordner namens **VersionOrm**.
3. Erstellen Sie im Ordner **VersionOrm** eine Datei mit der Bezeichnung **orm.xml** für jede spezifische Version. Der Dateiname sollte das Versionspräfix enthalten. Wenn beispielsweise die Version den Namen **x** hat, sollte der Dateiname **x_orm.xml** lauten.

Hinweis: Die Datei **orm.xml** im Ordner **META-INF** wird für jede Remote-Produktversion geladen, und zwar unabhängig davon, ob Sie eine spezifische **orm.xml**-Datei für eine Remote-Produktversion erstellen. Sie kann Entitäten beinhalten, die für alle Versionen auf die gleiche Weise zugeordnet sind.

Die Datei "reconciliation_types.txt"

Ab UCMDB 10.00 ist die Datei **reconciliation_types.txt** nicht mehr relevant. Für die Abstimmung kann jeder CI-Typ verwendet werden. Die Föderations-Engine führt die Zuordnung automatisch aus.

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 `Knotennamen UND/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.FcmdbDalTransformerTo ExternalDB** 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 176.

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. Einzelheiten hierzu finden Sie unter "Plugins" auf Seite 181.

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>
```

Herstellen einer Datenbankverbindung mittels NT-Authentifizierung

Es ist möglich, eine Verbindung zu einem MS SQL-Server herzustellen, der NT-Authentifizierung erfordert. Hierzu wird ein Treiber benötigt, der die Domäne analysieren kann (ein jTDS-JDBC-Treiber).

Die Authentifizierung erfolgt entsprechend den vorgegebenen Parametern (Domäne, Benutzername, Kennwort) und nicht mit den NT-Anmeldeinformationen des aktuell aktiven Prozesses.

1. Bearbeiten Sie in der Datei **persistence.xml** die untenstehenden Eigenschaften wie folgt:

```
<!--The driver class name"/-->
<property name="hibernate.connection.driver_class"
value="net.sourceforge.jtds.jdbc.Driver"/>
<property name="hibernate.connection.url" value="jdbc:jtds:sqlserver://[host
name]:[port];DatabaseName=[database name];domain=[the domain]"/>
<!--DB login credentials"/-->
<property name="hibernate.connection.username" value="[username]"/>
<property name="hibernate.connection.password" value="[password]"/>
```

2. Speichern Sie die JDBC-Treiberdatei unter **<Installationsordner der Probe>\lib**.
3. Starten Sie die Probe neu.

Konfigurieren der Datei persistence.xml für die SCCM-Integration für die Verwendung der NTLM-Authentifizierung

Hinweis: Die Informationen in diesem Abschnitt beziehen sich nur auf die SCCM-Integration.

Damit die SCCM-Integration die NTLM-Authentifizierung verwenden kann, konfigurieren Sie die Datei **persistence.xml** wie folgt:

1. Speichern Sie die JDBC-Treiberdatei unter **<Installationsordner der Probe>\lib**.

   Beispielsweise können Sie die Datei **jtds-1.3.1.jar** aus http://sourceforge.net/projects/jtds/files/ im Ordner **DataFlowProbe\lib** ablegen.
2. Starten Sie den Server und die Probe.
3. Wählen Sie in UCMDB **Datenflussverwaltung > Adapterverwaltung > SCCMAdapter**.
4. Wählen Sie im Ausschnitt **Ressourcen** die SCCM-Adapter-Konfigurationsdatei im Ordner **Packages > SCCMAdapter > Configuration Files**.
5. Nehmen Sie in der Datei **adapter.conf** folgende Einstellung vor: **dal.use.persistence.xml=true**.
6. Fügen Sie in der Datei **persistence.xml** folgende Inhalte hinzu:

```
<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">
    <persistence-unit name="GenericDBAdapter">
```

```
        <properties>
            <!-- added to fix: org.hibernate.HibernateException:
'hibernate.dialect' must be set when no Connection avalable -->
            <property name="hibernate.dialect"
value="org.hibernate.dialect.HSQLDialect"/>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>

            <!--The driver class name"/-->
            <property name="hibernate.connection.driver_class"
value="net.sourceforge.jtds.jdbc.Driver"/>
            <property name="hibernate.connection.url"
value="jdbc:jtds:sqlserver://<DB_host>:<port>;DatabaseName=<DB_
name>;domain=<domain_name> "/>
        </properties>
    </persistence-unit>
</persistence>
```

Hinweis: Ersetzen Sie den markierten Teil mit Ihrer Verbindungs-URL.

7. In der Datei **persistence.xml** ist kein Benutzer oder Kennwort erforderlich.
8. Navigieren Sie zu **Datenflussverwaltung > Integration Studio** und klicken Sie auf die Schaltfläche **Neuer Integrationspunkt**  .
9. Geben Sie die Werte für erforderliche Felder ein.

   Falls die ID der Anmeldeinformationen eingegeben werden muss, führen Sie die folgenden Schritte durch:

   a. Wählen Sie im Dialogfeld **Anmeldeinformationen auswählen** im linken Protokoll-Ausschnitt die Option **Generic DB Protocol (SQL)** aus.

   b. Klicken Sie im rechten Ausschnitt der Anmeldeinformationen auf die Schaltfläche **Neue Verbindungsdetails für ausgewählten Protokolltyp erstellen**  .

   c. Wählen Sie im Dialogfeld für die neuen Verbindungsdetails als Datenbanktyp die Option **MicrosoftSQLServerNTLM**.

   d. Geben Sie eine Port-Nummer ein.

   e. Geben Sie einen Benutzernamen im folgenden Format ein: **Domäne\Benutzername**.

   f. Geben Sie ein Kennwort ein.

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 Wertes oder eines 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.CmdbToAdapterBytes ToStringTransformer.

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.**

Der benutzerdefinierte Konverter

Sie haben die Möglichkeit, einen eigenen benutzerdefinierten Konverter (Transformator) zu schreiben. So können Sie einen exakt auf Ihre Anforderungen zugeschnittenen Konverter erstellen.

Es gibt zwei Möglichkeiten, einen benutzerdefinierten Konverter zu schreiben:

1. Schreiben eines kompilierten Java-Konverters
   a. Erstellen Sie ein Java-Projekt in einer Java-IDE (wie z. B. Eclipse, Intellij oder Netbeans).
   b. Fügen Sie die Dateien **federation-api.jar** und **db-interfaces.jar** zu Ihrem Klassenpfad hinzu.
   c. Erstellen Sie eine Java-Klasse, die die folgenden Schnittstellen implementiert (von der Datei **db-interfaces.jar**):
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   d. Kompilieren Sie das Projekt und erstellen Sie eine JAR-Datei.
   e. Speichern Sie die JAR-Datei im Package des Adapters (unter adapterCode\<Adapter-ID>)
   f. Stellen Sie das Package bereit.
   g. Fügen Sie den Klassennamen des neuen Konverters zur Datei **transformations.txt** hinzu.
2. Schreiben eines (skriptbasierten) Groovy-Konverters

   Das ursprüngliche GDBA-Package enthält das Beispiel **GroovyExampleTransformer.groovy**.
   a. Erstellen Sie im Package des Adapters (unter adapterCode\<Adapter-ID>) eine Groovy-Datei. Dies können Sie direkt über das Menü **Adapterverwaltung** machen.
   b. Erstellen Sie eine Groovy-Klasse, die die folgenden Schnittstellen implementiert (von der Datei **db-interfaces.jar**):
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   c. Fügen Sie den Klassennamen des neuen Groovy-Konverters zur Datei **transformations.txt** hinzu.

   Hinweis: Groovy ist eine Skriptsprache, die Java erweitert. Regulärer Java-Code ist auch gültiger Groovy-Code.

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 Implementieren und Bereitstellen von Plugins finden Sie unter "Implementieren eines Plugins" auf Seite 141.

Konfigurationsbeispiele

In diesem Abschnitt finden Sie Beispiele für Konfigurationen.

Dieser Abschnitt umfasst die folgenden Themen:

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>
            <oder>
                <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 156.

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_rules.txt"

Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 169.

```
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">
            <oder>
                <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">
            <oder>
                <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_rules.txt"

Weitere Informationen finden Sie unter "Die Datei "reconciliation_rules.txt" (für Abwärtskompatibilität)" auf Seite 169.

```
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 156.

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">
            <oder>
                <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" unten
- "Protokollspeicherorte" auf der nächsten Seite

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 123.

Externe Referenzen

Weitere Informationen zur JavaBeans 3.0-Spezifikation finden Sie unter http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

Fehlerbehebung und Einschränkungen – Entwickeln von allgemeinen Datenbankadaptern

In diesem Abschnitt werden die Fehlerbehebung und die Einschränkungen für den allgemeinen Datenbankadapter beschrieben.

Allgemeine Einschränkungen

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.

Kapitel 6: 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" auf der nächsten Seite
- "Datenpush" auf Seite 196
- "Auffüllung" auf Seite 197

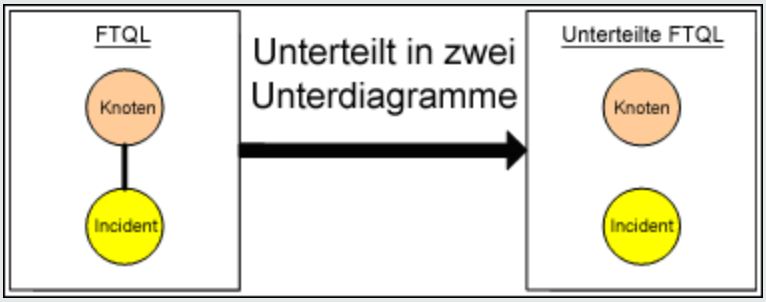
Föderation nach dem On-the-Fly-Prinzip

Föderierte TQL-Abfragen ermöglichen das Abrufen von Daten von jedem externen Daten-Repository ohne die Daten zu replizieren.

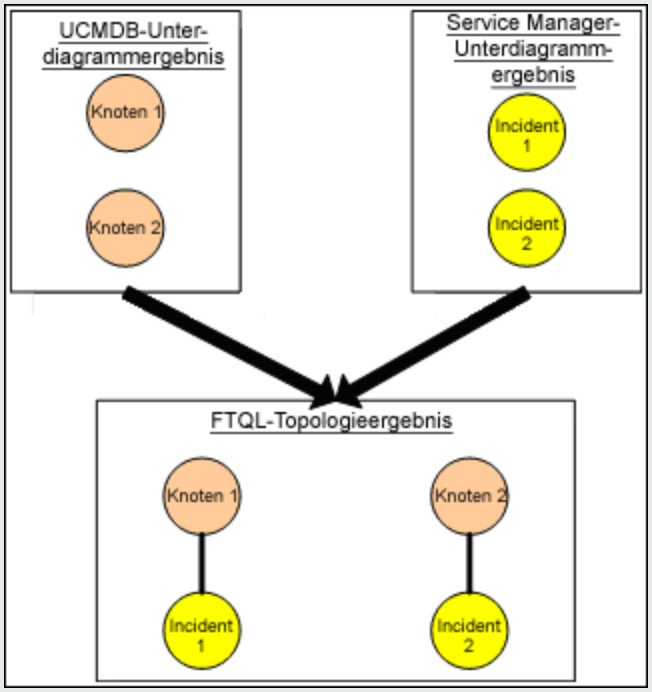
Eine föderierte TQL-Abfrage verwendet Adapter, die externe Daten-Repositorys darstellen, um entsprechende externe Beziehungen zwischen den CIs von verschiedenen externen Daten-Repositorys und den CIs von UCMDB zu erstellen.

Beispiel für einen Föderationsfluss nach dem On-the-Fly-Prinzip:

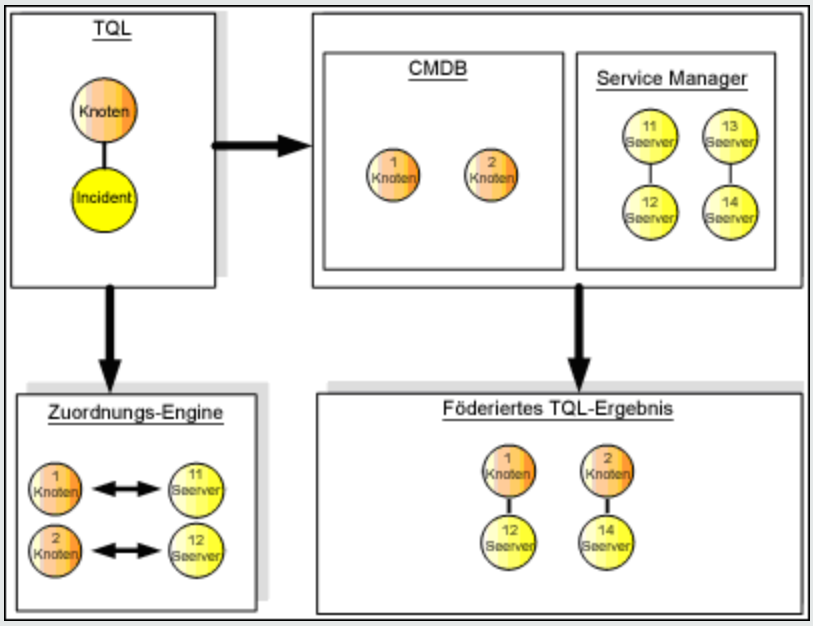
1. Federation Framework unterteilt eine föderierte TQL-Abfrage in mehrere Unterdiagramme, wobei sich alle Knoten in einem Unterdiagramm auf das gleiche Daten-Repository beziehen. Jedes Unterdiagramm ist über eine virtuelle Beziehung mit den anderen Unterdiagrammen verbunden (enthält aber selbst keine 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.

Instanzbasierter Auffüllungs-Flow

Wenn der Adapter für die Unterstützung eines instanzbasierten Flusses definiert wurde (unter Verwendung des unter "Tags und Eigenschaften für die XML-Konfiguration" auf Seite 220 beschriebenen Tags **<instance-based-data>**), sucht die Auffüllungs-Engine automatisch nach entfernten CIs in der Instanz und entfernt diese aus der UCMDB (wobei davon ausgegangen wird, dass das Löschen für den spezifischen Auffüllungs-Job zulässig ist). Jede Instanz muss über ein Stamm-CI verfügen, das in der TQL-Definition mit dem Namen**Root** angegeben ist. Jedes Mal, wenn ein Stamm-CI übergeben wird, wird die gesamte Instanz (d. h. alle mit dem Stamm-CI verbundenen CIs) mit der Version verglichen, die zuletzt an die UCMDB gesendet wurde, und alle CIs, die mit dem Stamm verbunden waren, jetzt aber nicht mit ihm verbunden sind, werden aus der UCMDB gelöscht. Damit der Adapter den instanzbasierten Fluss korrekt unterstützt, muss jede Änderung, die in der gesamten Instanz an einem CI oder Attribut vorgenommen wird, dazu führen, dass die gesamte Instanz erneut an die UCMDB gesendet wird.

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" unten
- "Hilfsmethoden des Adapters" auf der nächsten Seite

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" unten
- "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.

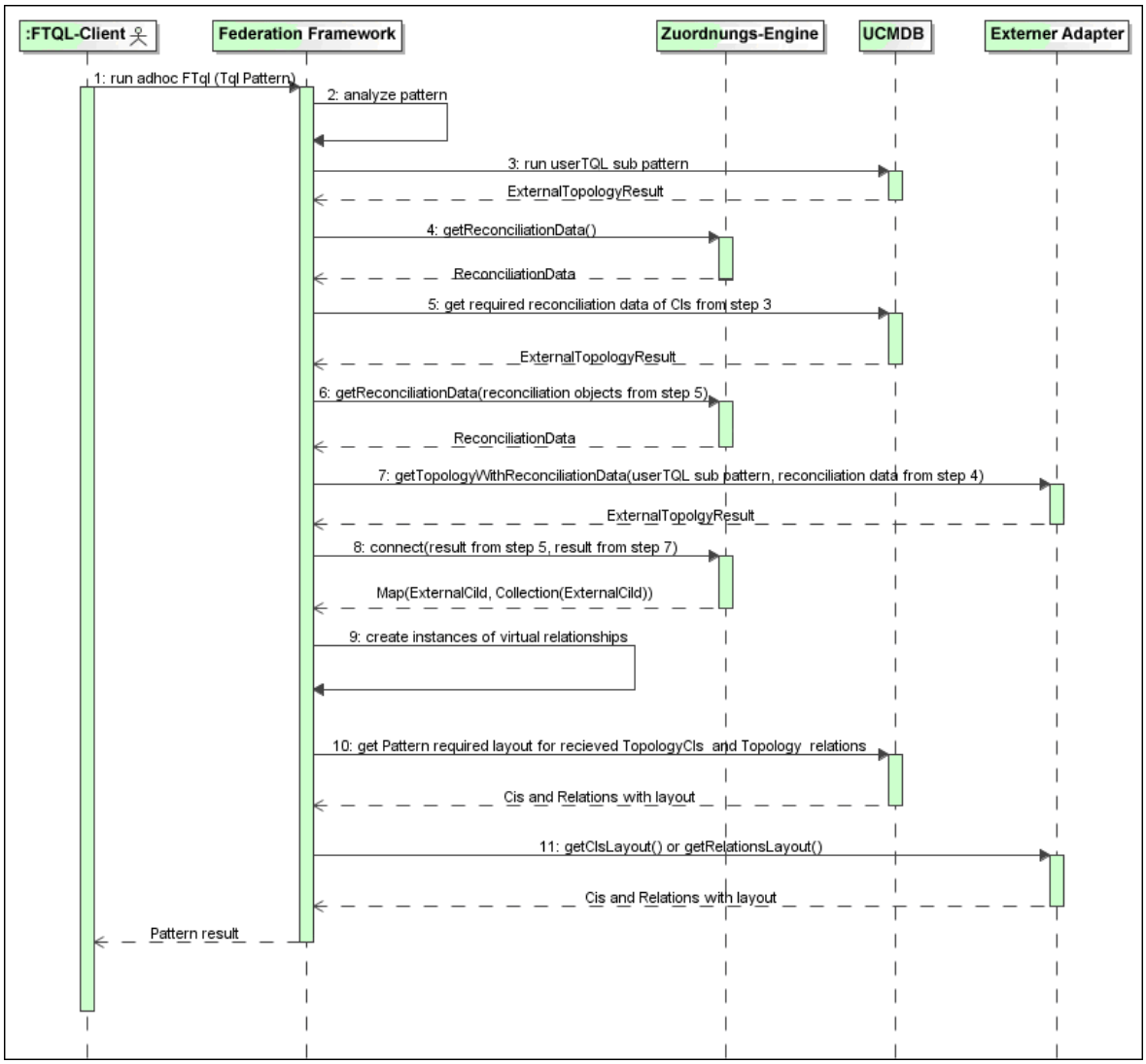
Dieser Abschnitt umfasst die folgenden Themen:

- " Berechnung beginnt am Serverende" auf der nächsten Seite
- "Berechnung beginnt am Ende des externen Adapters" auf Seite 203
- "Beispiel für den Federation Framework-Fluss für föderierte TQL-Abfragen" auf Seite 204

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.

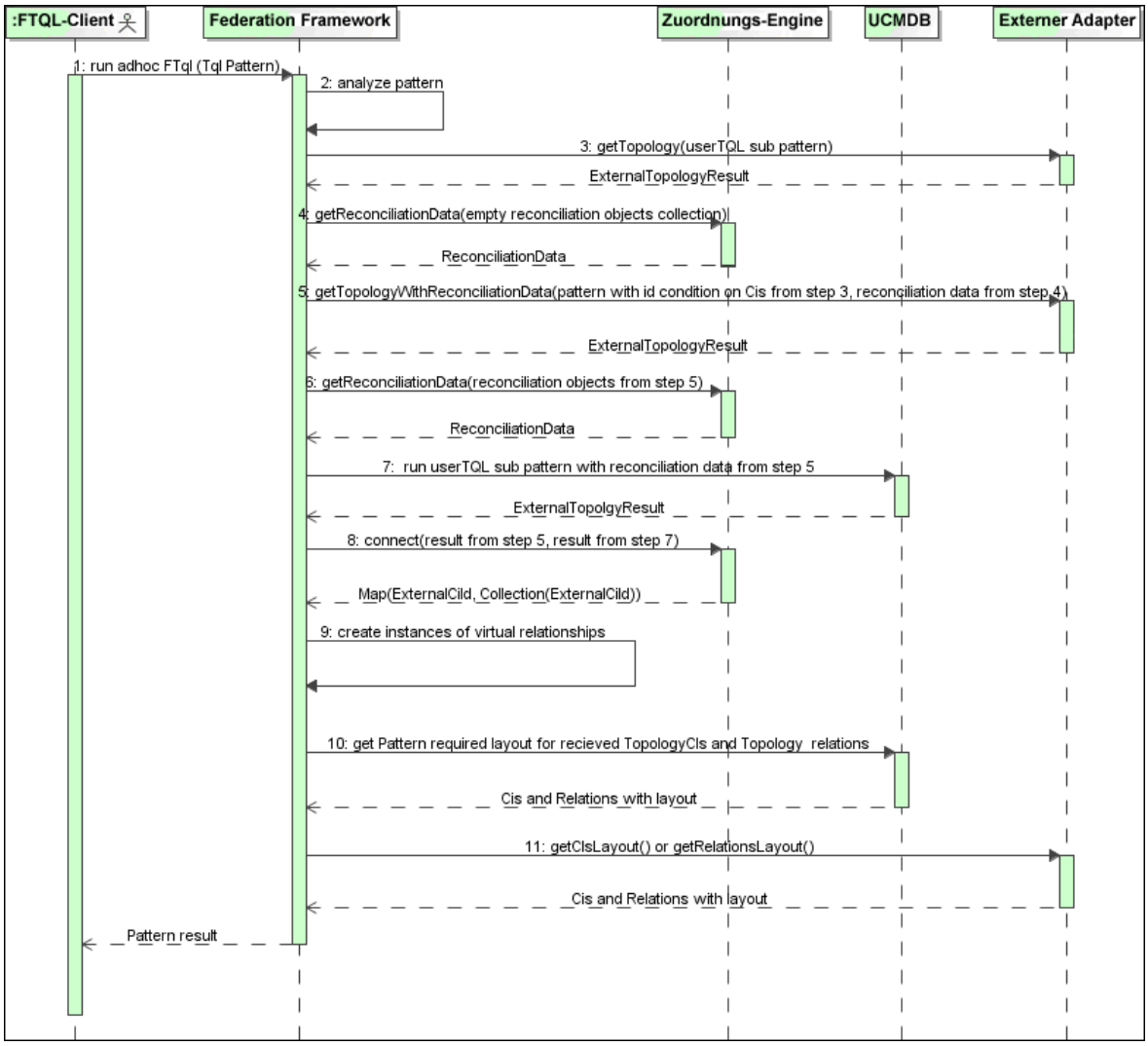


Die Zahlen in der Abbildung werden im Folgenden erläutert:

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.

Zahl	Erklärung
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: • **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" oben von UCMDB empfangen wurden.
6	Federation Framework ruft die Zuordnungs-Engine auf, um die Abstimmungsdaten abzurufen. Im Gegensatz zu Schritt "3" oben empfängt die Zuordnungs-Engine nun die Abstimmungsobjekte von Schritt "5" oben 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" oben 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" oben empfangen wurde. Der Parameter `secondResult` ist das Ergebnis der externen Topologie, das vom externen Adapter in Schritt "7" oben 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:

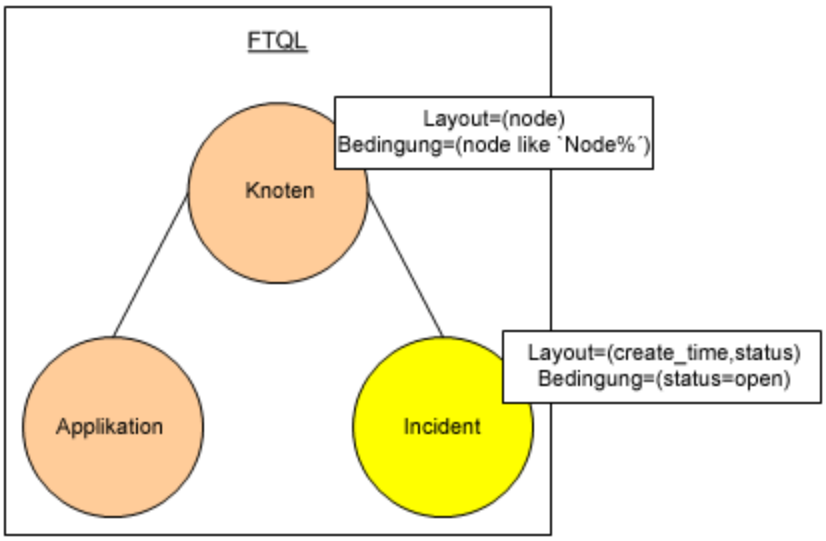
Zahl	Erklärung
1	Federation Framework empfängt einen Aufruf für die Berechnung einer föderierten TQL.
2	Federation Framework analysiert den Adapter, sucht die virtuelle Beziehung und teilt die ursprüngliche TQL in zwei Unteradapter – einen für UCMDB und einen für das externe Daten-Repository.
3	Federation 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.

Zahl	Erklärung
4	Nach dem Empfang der Topologieergebnisse ruft Federation Framework die entsprechende Zuordnungs-Engine mit der aktuellen virtuellen Beziehung auf und 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

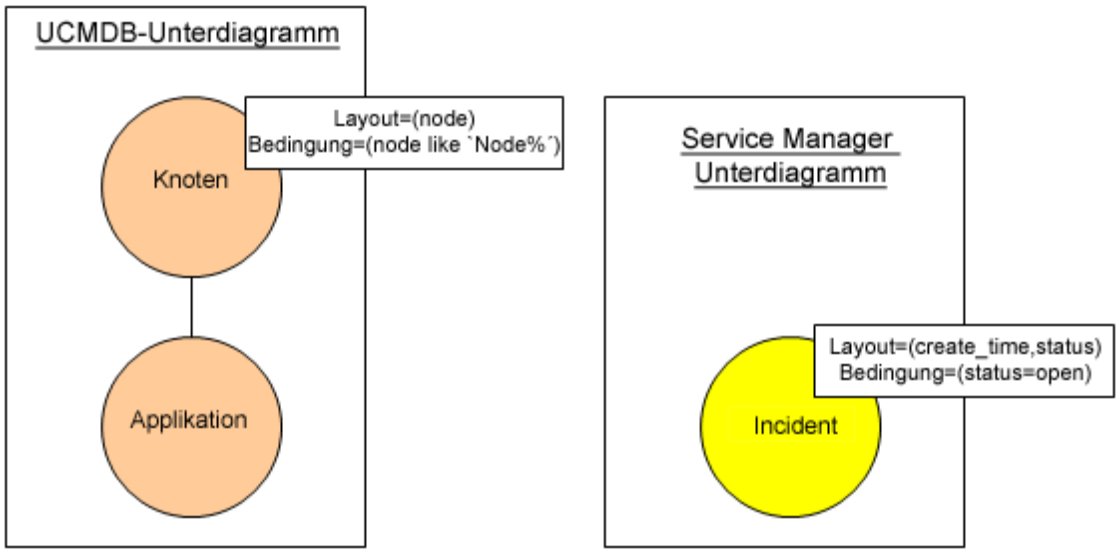
In diesem Beispiel wird erklärt, wie Sie alle offenen Vorfälle bei bestimmten Knoten anzeigen. Das ServiceCenter-Daten-Repository ist das externe Daten-Repository. Die Knoteninstanzen sind in UCMDB gespeichert und die Vorfallinstanzen in ServiceCenter. Es wird davon ausgegangen, dass die

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.

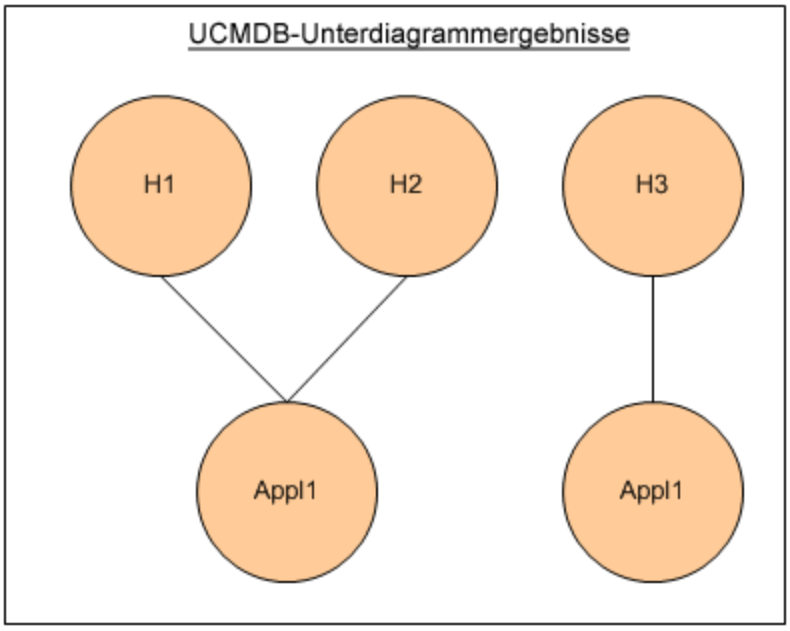


Hinweis: Für die Attributföderation wird die Methode **getTopology** des Adapters aufgerufen. Die Abstimmungsdaten werden in der Benutzer-TQL (in diesem Fall das CI-Element) angepasst.

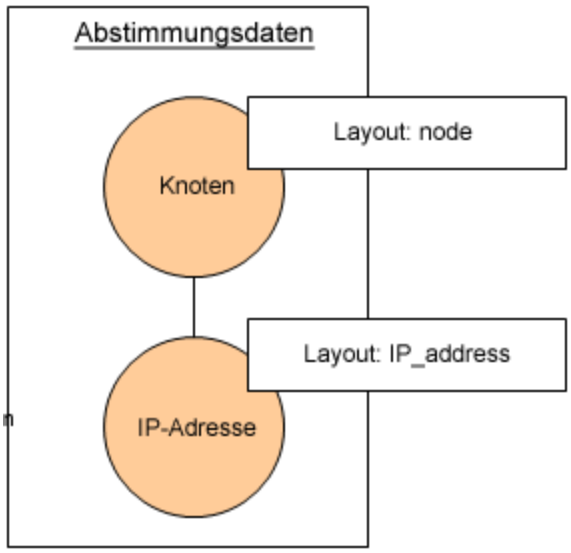
1. Nach der Analyse des Adapters erkennt Federation Framework die virtuelle Beziehung zwischen dem `Knoten` und dem `Vorfall` und unterteilt die föderierte TQL-Abfrage in zwei Unterdiagramme:



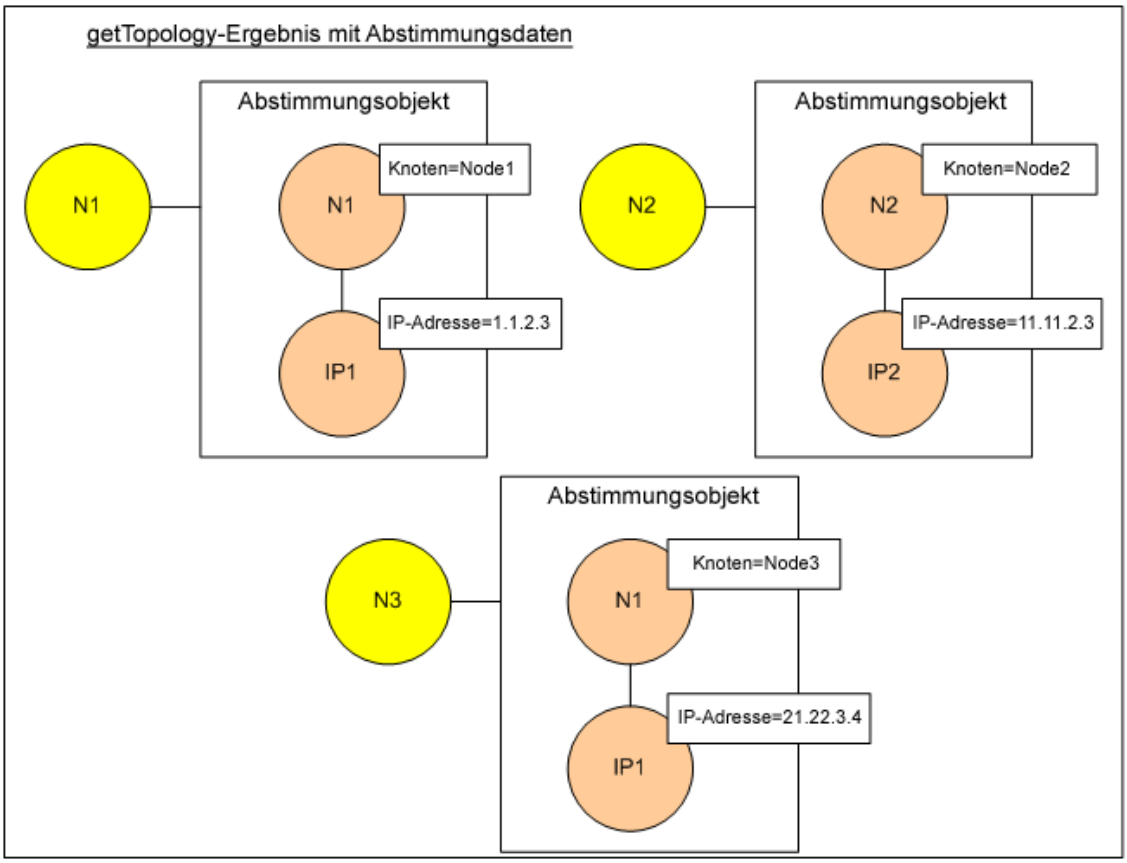
2. Federation Framework führt das UCMDB-Unterdiagramm aus, um die Topologie anzufordern, und empfängt die folgenden Ergebnisse:



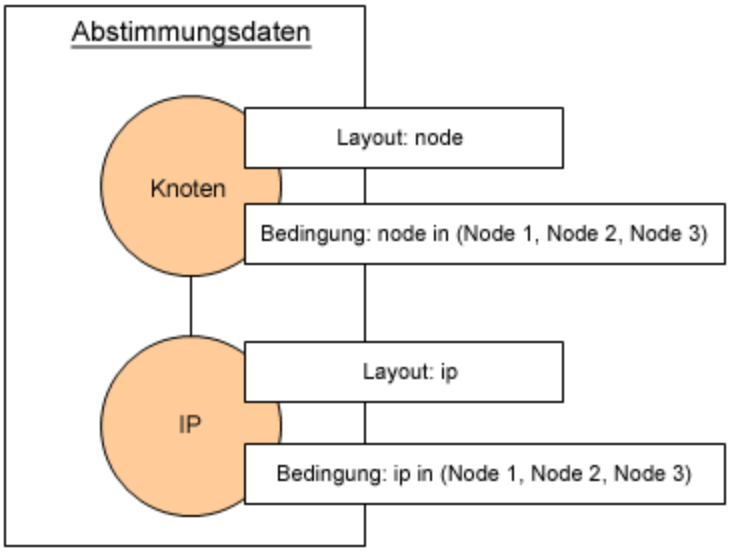
3. Federation Framework fordert von der entsprechenden Zuordnungs-Engine die Abstimmungsdaten für das erste Daten-Repository (UCMDB) an, das die Informationen zum Herstellen einer Verbindung zwischen den von den beiden Daten-Repositorys empfangenen Daten enthält. Die Abstimmungsdaten in diesem Fall lauten wie folgt:



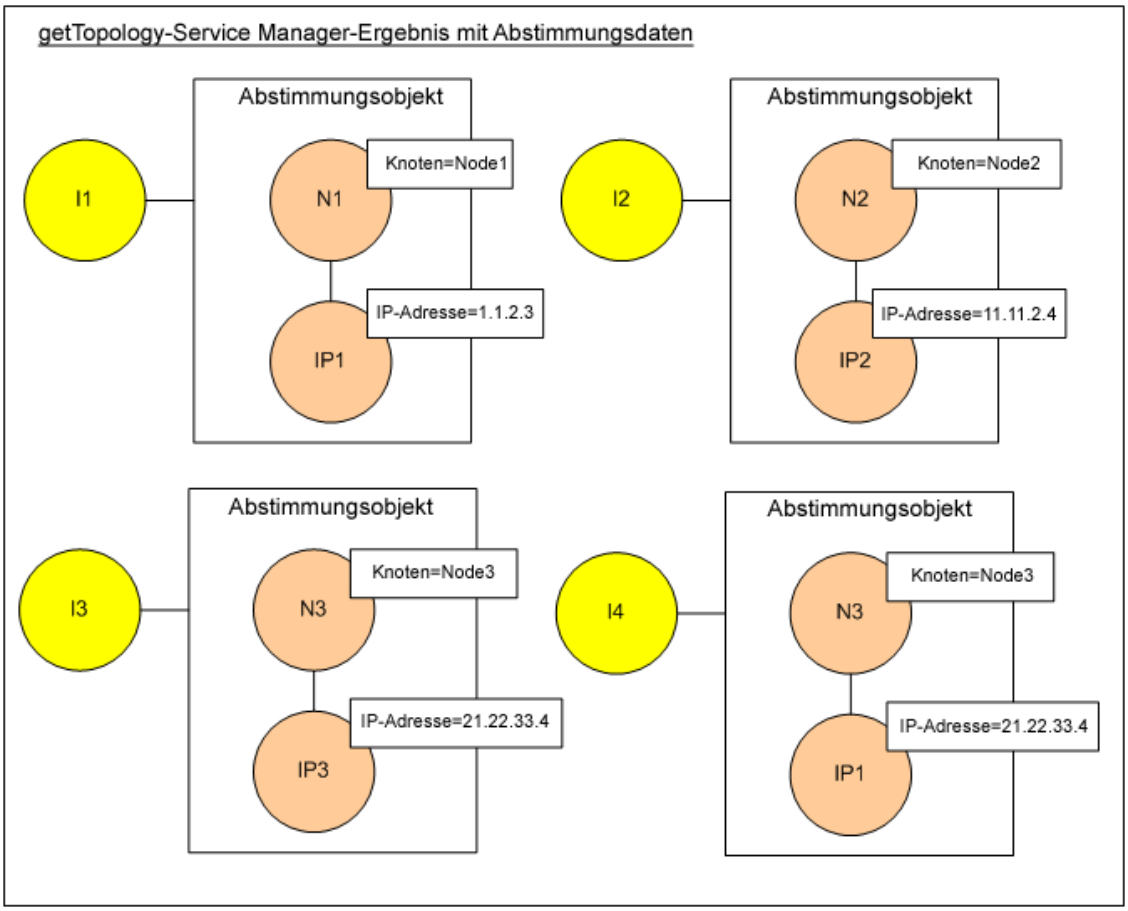
4. Federation Framework erstellt eine Ein-Knoten-Topologieabfrage mit den Knoten- und ID-Bedingungen vom vorherigen Ergebnis (node in H1, H2, H3) und führt diese Abfrage mit den erforderlichen Abstimmungsdaten bei UCMDB aus. Das Ergebnis enthält die für die ID-Bedingung relevanten Knoten-CIs und das entsprechende Abstimmungsobjekt für jedes CI:



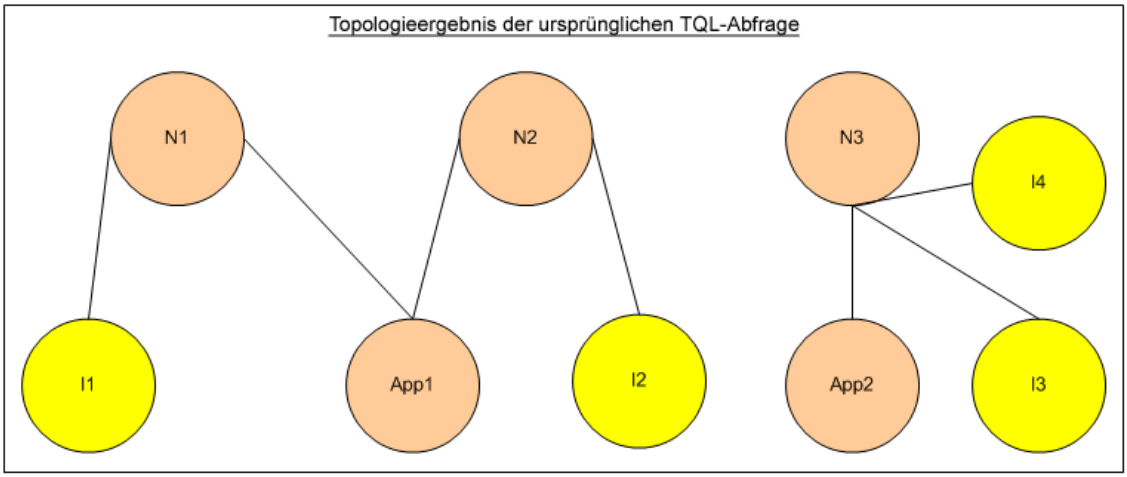
5. Die Abstimmungsdaten für ServiceCenter sollten eine Bedingung für `node` und `ip` enthalten, die von den von UCMDB empfangenen Abstimmungsobjekten abgeleitet wird:



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

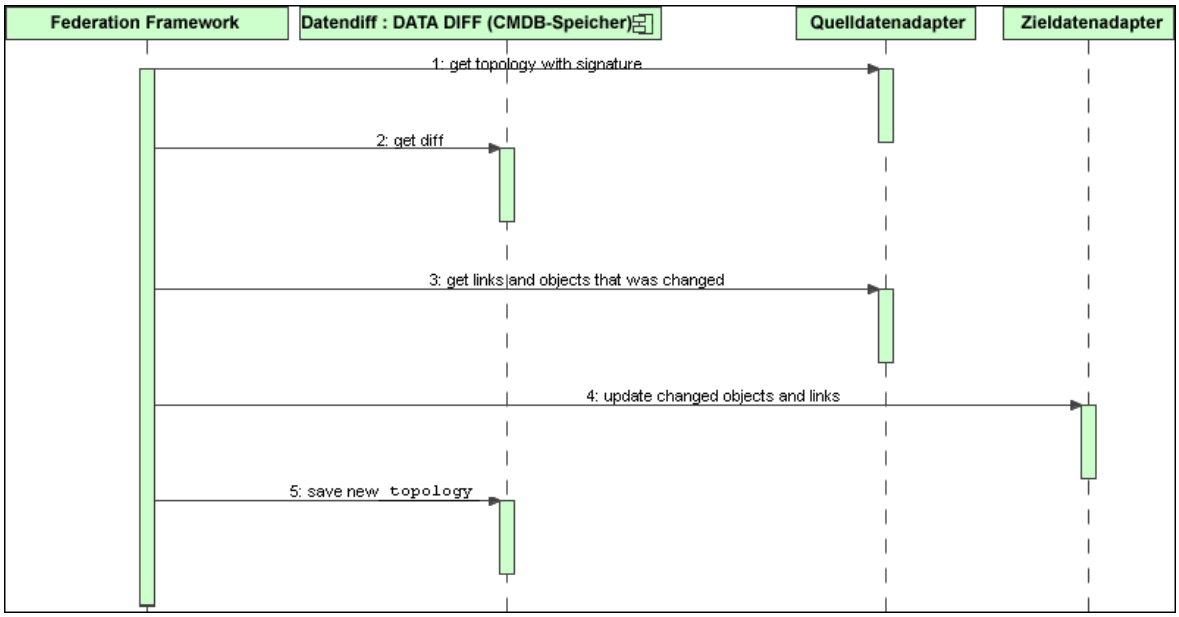
Dieser Abschnitt umfasst die folgenden Themen:

Definitionen und Begriffe

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 **SingleNode-Topologieschnittstelle** wird verwendet, wenn der Adapter keine externen Beziehungen unterstützt, d. h. der Adapter erhält grundsätzlich keine Anforderungen mit mehr als einem externen CI. Die Abstimmungsdaten, die für die Durchführung des Vorgangs benötigt werden, können als komplexe Abfrage beschrieben werden (siehe unten unter SingleNodeFederationTopologyReconciliationAdapter).

  Alle SingleNode-Schnittstellen werden zur Vereinfachung des Workflows erstellt. Verwenden Sie für komplexere Abfragen die **FederationTopologyAdapter**-Schnittstelle.
- Eine **FederationTopologyAdapter-Schnittstelle** wird zur Definition von Adaptern verwendet, die komplexe föderierte Abfragen unterstützen. Die Abstimmungsanforderung in diesen Adaptern ist Teil des **QueryDefinition**-Parameters.

  Die Föderations-Engine verwendet die Abstimmungsdaten, um die föderierten Daten mit den richtigen lokalen CIs zu verbinden. Abstimmungsdaten können in mehreren Anforderungen abgerufen werden (die Berechnung erfolgt entsprechend den Ergebnissen rekursiv). In diesem Fall empfängt der Adapter eine Anforderung, die nur Abstimmungsdaten enthält..

SingleNode-Schnittstellen

Die folgenden Schnittstellen weisen unterschiedliche Arten von Abstimmungsdaten auf:

- **SingleNodeFederationIdReconciliationAdapter.** Verwenden Sie diese Schnittstelle, wenn der Adapter eine **Einzelknoten-TQL** unterstützt und die Abstimmung zwischen den Daten-Repositorys anhand der ID berechnet wird.
- **SingleNodeFederationPropertyReconciliationAdapter.** Verwenden Sie diese Schnittstelle, wenn der Adapter eine **Einzelknoten-TQL** unterstützt und die Abstimmung zwischen den Daten-Repositorys anhand der Eigenschaft eines CI erfolgt.

- **SingleNodeFederationTopologyReconciliationAdapter.** Verwenden Sie diese Schnittstelle, wenn der Adapter eine **Einzelknoten-TQL** unterstützt und die Abstimmung zwischen den Daten-Repositorys anhand der Topologie erfolgt. Der Adapter sollte den Fall unterstützen, wo das Abfrageelement leer ist und nur die Abstimmungstopologie angefordert wird.

DataAdapter-Schnittstellen

- **FederationTopologyAdapter.** Verwenden Sie diesen Adapter, um komplexe föderierte TQL-Abfragen zu unterstützen. Ermöglicht die größte Vielfalt. Der Adapter sollte den Fall unterstützen, wo die Abfragedefinition nur Abstimmungsdaten beschreibt.
- **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.

RessourceReporting-Schnittstellen

Die folgenden Schnittstellen ermöglichen es dem Adapter, die Ressourcen zu melden, die konfiguriert werden können, um das Verhalten des Adapters anzupassen. Auf diese Weise können Sie diese Ressourcen direkt über Integration Studio bearbeiten. Diese Schnittstellen sollten zusätzlich zu den oben aufgeführten Adapterschnittstellen verwendet werden.

- **PopulationQueriesResourcesLocator.** Legt fest, welche Ressourcen für jede spezifische Auffüllungsabfrage bearbeitet werden können.
- **PushQueriesResourceLocator.** Legt fest, welche Ressourcen für jede Datenpush-Abfrage bearbeitet werden können.
- **GeneralResourcesLocator.** Legt fest, welche allgemeinen Ressourcen in diesem Adapter bearbeitet werden können.

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.
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 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**  und wählen Sie **Neuer Adapter** aus.
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.BaseDestinationDa
ta">
<!-- 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 220.

4. Definieren der unterstützten Klassen

   Definieren Sie die unterstützten Klassen entweder im Adaptercode, 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.

   Wenn der Adapter **getTopologyWithReconciliationData** implementiert und die Adapterfunktionen die Möglichkeit beinhalten als Ausgangspunkt verwendet zu werden, sollte der Adapter auch die Anforderungstopologie mit Abstimmungsdaten ohne Bedingungen unterstützen (nur Typ). In diesem Fall sollte der Adapter die vollständigen Abstimmungsdaten der gefundenen Ergebnisse zurückgeben.

   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.

Ein Bestandteil der Föderations-API ist die DataAdapterEnvironment-Schnittstelle. Diese Schnittstelle stellt die Umgebung des Datenadapters dar. Sie enthält die Umgebung-API, die für die ordnungsgemäße Funktion des Adapters erforderlich ist. Weitere Informationen zur DataAdapterEnvironment-Schnittstelle finden Sie unter "DataAdapterEnvironment-Schnittstelle" auf Seite 222.

6. Definieren der Abstimmungsregeln oder Implementieren der Zuordnungs-Engine

   Wenn Ihr Adapter föderierte TQL-Abfragen unterstützt, haben Sie zwei Möglichkeiten, um eine Zuordnungs-Engine zu definieren:

   - Sie können die Standard-Zuordnungs-Engine von verwenden, die für die Zuordnung die internen CMDB-Abstimmungsregeln nutzt. Wenn Sie diese Engine verwenden möchten, lassen Sie das XML-Tag **<default-mapping-engine>** leer.
   - 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*.

Erstellen eines Beispieladapters

Im Folgenden wird beschrieben, wie Sie einen Beispieladapter erstellen. Diese Aufgabe umfasst folgende Schritte:

- "Auswählen der Adapterlogik" unten
- "Laden des Projekts" unten

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.
	<start-point-adapter>	Gibt an, dass dieser Adapter der Ausgangspunkt für die Berechnung der TQL-Abfrage ist.
	<one-node-topology>	Die Fähigkeit, föderierte Abfragen mit einem föderierten Abfrageknoten durchzuführen.
	<pattern-topology>	Die Fähigkeit, komplexe Abfragen zu föderieren.
	<support-replicatioin-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.
	<instance-based-data>	Dieses Tag gibt an, dass der Adapter einen instanzbasierten Auffüllungsfluss unterstützt.
	<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
	<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.

DataAdapterEnvironment-Schnittstelle

OutputStream openResourceForWriting(String resourceName) throws FileNotFoundException;

Diese Methode öffnet eine Ressource mit einem bestimmten Namen zum Beschreiben. Sie wird zum Speichern persistenter Daten für die Integration verwendet. Es empfiehlt sich, diese Methode zu verwenden, anstatt zu versuchen, Dateien unter Verwendung von Java-Methoden zu laden. Der Benutzer sollte sicherstellen, dass der Stream geschlossen wird, wenn der Schreibvorgang beendet ist. **close()/flush()** speichert die Ressource. Diese Methode erstellt eine Laufzeitressource (die Dateien aus dem Adapter-Package werden nicht überschrieben).

Parameter

- **resourceName:** Der Name der abzurufenden Ressource. Dieser Name muss bei allen Integrationen des gleichen Adapters eindeutig sein.

Rückgabewert

Gibt einen Stream zum Beschreiben zurück.

Ausnahmen

- Diese Methode löst die Ausnahme *FileNotFoundException* aus, wenn der Ressourcentyp eine Datei ist und die Datei nicht vorhanden ist oder wenn die Ressource ein Verzeichnis und keine reguläre Datei ist oder wenn die Ressource aus einem anderen Grund nicht zum Lesen geöffnet werden kann.
- Diese Methode löst die Ausnahme *SecurityException* aus, wenn ein Security Manager vorhanden ist, dessen Methode *checkRead* den Zugriff auf die Datei verweigert.

InputStream openResourceForReading(String resourceName) throws FileNotFoundException;

Diese Methode öffnet eine Ressource mit einem bestimmten Namen zum Lesen. Sie wird zum Lesen persistenter Daten für die Integration verwendet. Es empfiehlt sich, diese Methode zu verwenden, anstatt zu versuchen, ein Datei unter Verwendung von Java-Methoden zu laden. Der Benutzer sollte sicherstellen, dass der Stream geschlossen wird, wenn der Lesevorgang beendet ist. Zunächst wird versucht, Dateien aus dem Adapter-Package zu laden. Werden keine Dateien gefunden, wird versucht, eine zur Laufzeit erstellte Ressource von *DataAdapterEnvironment.openResourceForWriting(String)* zu laden. Die Laufzeitressourcen können mittels JMX (von der Probe bzw. dem Server) angezeigt werden.

Parameter

- **resourceName:** Der Name der abzurufenden Ressource. Dieser Name muss bei allen Integrationen des gleichen Adapters eindeutig sein.

Rückgabewert

Gibt einen Stream zum Lesen zurück.

Ausnahmen

- Diese Methode löst die Ausnahme *FileNotFoundException* aus, wenn der Ressourcentyp **file** lautet und die Datei nicht vorhanden ist oder wenn die Ressource ein Verzeichnis und keine reguläre Datei ist oder wenn die Ressource aus einem anderen Grund nicht zum Lesen geöffnet werden kann.
- Diese Methode löst die Ausnahme *SecurityException* aus, wenn ein Security Manager vorhanden ist, dessen Methode *checkRead* den Lesezugriff auf die Datei verweigert.

Properties openResourceAsProperties(String propertiesFile) throws IOException;

Diese Methode öffnet eine Ressource mit einem bestimmten Namen und lädt sie als *Properties*-Struktur. Sie wird zum Lesen persistenter Daten für die Integration verwendet. Es empfiehlt sich, diese Methode zu verwenden, anstatt zu versuchen, die **.properties**-Dateien unter Verwendung von Java-Methoden zu laden. Zunächst wird versucht, Dateien aus dem Adapter-Package zu laden. Werden keine Dateien gefunden, wird versucht, eine zur Laufzeit erstellte Ressource von *DataAdapterEnvironment.openResourceForWriting(String)* zu laden. Die Laufzeitressourcen können mittels JMX (von der Probe bzw. dem Server) angezeigt werden.

Parameter

- **propertiesFile:** Der Name der abzurufenden Ressource. Dieser Name muss bei allen Integrationen des gleichen Adapters eindeutig sein.

Rückgabewert

Gibt den in **Properties** repräsentierten Dateiinhalt zurück.

Ausnahmen

- Diese Methode löst die Ausnahme *FileNotFoundException* aus, wenn der Ressourcentyp **file** lautet und die Datei nicht vorhanden ist oder wenn die Ressource ein Verzeichnis und keine reguläre Datei ist oder wenn die Ressource aus einem anderen Grund nicht zum Lesen geöffnet werden kann.
- Diese Methode löst die Ausnahme *SecurityException* aus, wenn ein Security Manager vorhanden ist, dessen Methode *checkRead* den Lesezugriff auf die Datei verweigert.
- Diese Methode löst die Ausnahme *IOException* aus, wenn die Eigenschaftendatei nicht in das Objekt *Properties* konvertiert werden konnte.

String openResourceAsString(String resourceName) throws IOException;

Diese Methode öffnet eine Ressource mit einem bestimmten Namen und lädt sie als Zeichenkette. Sie wird zum Lesen persistenter Daten für die Integration verwendet. Es empfiehlt sich, diese Methode zu verwenden, anstatt zu versuchen, Dateien unter Verwendung von Java-Methoden zu laden.

Zunächst wird versucht, Dateien aus dem Adapter-Package zu laden. Werden keine Dateien gefunden, wird versucht, eine zur Laufzeit erstellte Ressource von *DataAdapterEnvironment.openResourceForWriting(String)* zu laden. Die Laufzeitressourcen können mittels JMX (von der Probe bzw. dem Server) angezeigt werden.

Parameter

- **resourceName:** Der Name der abzurufenden Ressource. Dieser Name muss bei allen Integrationen des gleichen Adapters eindeutig sein.

Rückgabewert

Gibt den im Zeichenkettenformat repräsentierten Dateiinhalt zurück.

Ausnahmen

- Diese Methode löst die Ausnahme *FileNotFoundException* aus, wenn der Ressourcentyp **file** lautet und die Datei nicht vorhanden ist oder wenn die Ressource ein Verzeichnis und keine reguläre Datei ist oder wenn die Ressource aus einem anderen Grund nicht zum Lesen geöffnet werden kann.
- Diese Methode löst die Ausnahme *SecurityException* aus, wenn ein Security Manager vorhanden ist, dessen Methode *checkRead* den Lesezugriff auf die Datei verweigert.
- Diese Methode löst die Ausnahme *IOException* aus, wenn ein E/A-Fehler auftritt.

public void saveResourceFromString(String relativeFileName, String value) throws IOException;

Diese Methode empfängt eine Zeichenkette und speichert sie als Ressource. Sie wird zum Speichern persistenter Daten für die Integration verwendet. Es empfiehlt sich, diese Methode zu verwenden, anstatt zu versuchen, Dateien unter Verwendung von Java-Methoden zu speichern. Diese Methode konvertiert die Zeichenkette in einen Stream und speichert diesen in der Ressource. Sie erstellt eine Laufzeitressource, kann aber nicht die Dateien aus dem Adapter-Package überschreiben. Die Laufzeitressourcen können mittels JMX (von der Probe bzw. dem Server) angezeigt werden.

Parameter

- **relativeFileName:** Der Name der abzurufenden Ressource. Dieser Name muss bei allen Integrationen des gleichen Adapters eindeutig sein.
- **value:** Die Zeichenkette, die als Ressource gespeichert werden soll.

Ausnahmen

Diese Methode löst die Ausnahme **IOException** aus, wenn ein E/A-Fehler auftritt.

boolean resourceExists(String resourceName);

Diese Methode prüft, ob der angegebene Ressourcenname vorhanden ist. Sie sucht nach Dateien aus dem Adapter-Package und nach zur Laufzeit erstellten Ressourcen von *DataAdapterEnvironment.openResourceForWriting(String)*.

Parameter

- **resourceName:** Der Name der abzurufenden Ressource. Dieser Name muss bei allen Integrationen des gleichen Adapters eindeutig sein.

Rückgabewert
Gibt **True** zurück, wenn *resourceName* vorhanden ist.

boolean deleteResource(String resourceName);

Diese Methode löscht die angegebene Ressource aus den persistenten Daten. Sie löscht eine Laufzeitressource, kann aber keine Dateien aus dem Adapter-Package löschen. Die Laufzeitressourcen können mittels JMX (für die Probe bzw. den Server) angezeigt werden.

Parameter

- **resourceName:** Der Name der zu löschenden Ressource. Dieser Name muss bei allen Integrationen des gleichen Adapters eindeutig sein.

Rückgabewert

Gibt **True** zurück, wenn die Ressource erfolgreich gelöscht wurde.

Collection<Zeichenkette> listResourcesInPath(String path);

Diese Methode ruft eine Liste der Ressourcen im angegebenen Ressourcenpfad ab. Sie sucht nach Dateien aus dem Adapter-Package und nach zur Laufzeit erstellten Ressourcen von *DataAdapterEnvironment.openResourceForWriting(String)*. Die Laufzeitressourcen können mittels JMX (für die Probe bzw. den Server) angezeigt werden.

Parameter

- **path:** Der Ressourcenpfad. Beispiel: "META-INF/myfiles/"

Rückgabewert

Gibt eine Liste der Ressourcen im Pfad zurück.

DataAdapterLogger getLogger();

Ruft den Logger ab, der vom Adapter verwendet werden soll. Dieser Logger wird für die Protokollierung von Ereignissen in Ihrem Adapter verwendet.

Rückgabewert

Gibt den Logger zurück, der von der DataAdapter-Schnittstelle verwendet wird.

DestinationConfig getDestinationConfig();

Diese Methode ruft die Zielkonfiguration der Integration ab. Diese Konfiguration beinhaltet alle Verbindungs- und Ausführungseinstellungen für die Integration.

Rückgabewert

Gibt die Zielkonfiguration **DestinationConfig** des Adapters zurück.

int getChunkSize();

Diese Methode ruft die Chunk-Größe für die Auffüllung ab, die für diese Integration angefordert wurde.

Rückgabewert

Gibt die Chunk-Größe für die Auffüllung zurück.

int getPushChunkSize();

Diese Methode ruft die Chunk-Größe für Datenpush ab, die für diese Integration angefordert wurde.

Rückgabewert

Gibt die Chunk-Größe für Datenpush zurück.

ClassModel getLocalClassModel();

Diese Methode ruft ein Klassenmodell zum Abfragen von Informationen über das Klassenmodell der lokalen UCMDB ab. Sie liefert ein aktualisiertes Klassenmodell. Sobald das Objekt **ClassModel** zurückgegeben wurde, wird es nicht mehr mit weiteren Klassenmodelländerungen aktualisiert. Um ein aktualisiertes Klassenmodell abzurufen, verwenden Sie diese Methode erneut.

Rückgabewert

Gibt das Klassenmodell der UCMDB zurück.

CustomerInformation getLocalCustomerInformation();

Diese Methode ruft Kundeninformationen für den Kunden ab, der den Adapter ausführt.

Rückgabewert

Gibt Kundeninformationen für den Kunden zurück, der den Adapter ausführt.

Object getSettingValue(String name);

Diese Methode ruft eine bestimmte Adaptereinstellung ab.

Parameter

name: Der Name der Einstellung.

Rückgabewert

Gibt den Einstellwert zurück.

Map<Zeichenkette, Objekt> getAllSettings();

Diese Methode ruft alle Adaptereinstellungen ab.

Rückgabewert

Gibt die Adaptereinstellungen zurück.

boolean isMTEnabled();

Diese Methode prüft, ob die Serverumgebung Mandantenfähigkeit (Multiple Tenancy, MT) unterstützt.

Rückgabewert
Gibt **true** zurück, wenn die Serverumgebung MT unterstützt, andernfalls wird **false** zurückgegeben.

String getUcmdbServerHostName();

Diese Methode gibt den lokalen UCMDB Server-Hostnamen zurück.

Rückgabewert

Gibt den lokalen UCMDB Server-Hostnamen zurück.

Kapitel 7: Entwickeln von Push-Adaptern

Dieses Kapitel umfasst folgende Themen:

Entwickeln und Bereitstellen von Push-Adaptern

Generische Push-Adapter stellen eine allgemeine Plattform zur Verfügung, die eine schnelle Entwicklung von Integrationen ermöglicht, um UCMDB-Daten per Push an externe Daten-Repositorys (Datenbanken und Applikationen von Drittanbietern) zu übertragen. Generische Push-Adapter werden entsprechend dem Protokoll kategorisiert, das für den Datenpush verwendet wird. Weitere Informationen zum Datenpush über XML unter Verwendung des generischen XML-Push-Adapters finden Sie unter "SQL-Abfragen für generische XML-Push-Adapter" auf Seite 241. Weitere Informationen zum Datenpush über einen Webservice unter Verwendung des generischen Webservice-Push-Adapters finden Sie unter "Generischer Webservice-Push-Adapter" auf Seite 241.

Die Entwicklung einer benutzerdefinierten Integration auf der Basis eines generischen Push-Adapters erfordert Folgendes:

- Erstellen eines neuen Adapter-Package aus den Vorlagendateien des entsprechenden generischen Push-Adapters. Weitere Informationen finden Sie unter "Erstellen eines Adapter-Package" auf der nächsten Seite.
- Zuordnungen zwischen den CI-Linktypen von UCMDB und den externen Datenelementen. Die Zuordnungen werden als XML-Datei gespeichert und an jedes externe Daten-Repository angepasst. Weitere Informationen finden Sie unter "Erstellen von Zuordnungen" auf Seite 232.
- Ein Jython-Skript, um die Datenelemente per Push in das externe Daten-Repository zu übertragen. Weitere Informationen finden Sie unter "Schreiben von Jython-Skripts" auf Seite 236.
- Zusätzliche adapterspezifische Schritte. Hierzu zählen beispielsweise das Auswählen des Pfads für die Datei, die für den XML-Push-Adapter geschrieben werden soll, oder das Erstellen eines Datenempfängers für den Webservice-Push-Adapter.

Erstellen eines Adapter-Package

Um einen neuen MDR-spezifischen Push-Adapter zu erstellen, sollten Sie eine Kopie des generischen Adapters erstellen und dann die Kopie bearbeiten, um den Adapter als Adapter für ein bestimmtes Push-Ziel anzupassen.

Generische Adapter-Packages finden Sie an einem der folgenden Speicherorte:

- Generischer XML- Push-Adapter: **hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip**
- Generischer Webservice-Adapter: **hp\UCMDB\UCMDBServer\content\adapters\web-service-push-adapter.zip**

So erstellen Sie einen neuen Push-Adapter von einem generischen Push-Adapter:

1. Extrahieren Sie den Inhalt der ausgewählten Package-ZIP-Datei in einen Arbeitsordner.
2. Überprüfen Sie die folgenden Verzeichnisse zur Vorbereitung der Umbenennungs- und Ersetzungsphase:
   - **adapterCode.** Enthält das Verzeichnis, das für das **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase-Verzeichnis** bereitgestellt wird. Die JAR-Dateien, die hier bereitgestellt werden, bewirken keinen automatischen Neustart der Probe und erscheinen auch nicht automatisch im Klassenpfad der Probe.
   - **discoveryConfigFiles:** Enthält die Zuordnungsdefinitionen des Adapters und verweist auf das richtige Jython-Skript (**push.properties**).
   - **discoveryPatterns:** Enthält die XML-Definition des Adapters, der auf dem UCMDB-Server bereitgestellt wird.
   - **discoveryScripts:** Enthält die Jython-Skripts des Adapters, über die die Verbindung zum Datenspeicher eines Drittanbieters hergestellt und der Datenpush durchgeführt wird.
   - **discoveryResources:** Enthält die Datei **UCMDBDataReceiver.jar**, die die Java-Integrationsklassen für den Webservice enthält.

   Hinweis: Wenn Sie dieses Package bereitstellen, wird die Probe neu gestartet, um diese **JAR-Datei** in den Klassenpfad der Probe aufzunehmen. Zusätzlich zur Bereitstellung des Package sind keine weiteren Aktionen erforderlich.
3. Nehmen Sie die folgenden Änderungen innerhalb der entpackten Adapter-Verzeichnisstruktur vor:
   a. **discoveryConfigFiles\<Name_Ihres_Push-Adapters>:** Benennen Sie das Verzeichnis "PushAdapter" oder "XMLtoWebService" um in den Namen des neuen Push-Adapters (z. B. "MeinPushAdapter").
   b. **discoveryConfigFiles\<Name_Ihres_Push-Adapters>\push.properties:** Führen Sie in der Datei **push.properties** die folgenden Schritte durch:
      - Ändern Sie den Namen **jythonScript.name** in den Namen des Jython-Skripts, das von dem neuen Push-Adapter verwendet werden soll (z. B. **pushToMyService.py**).
      - Aktualisieren Sie den Namen der Zuordnungsdatei, die von dem neuen Push-Adapter

verwendet werden soll (z. B. **myPushAdapter_mappings**). Fügen Sie nicht die Erweiterung **.xml** hinzu. Dies geschieht automatisch.

c. **discoveryPatterns\<Name_des_Push-Adapters>.xml:** Benennen Sie diese Datei um und geben Sie ihr den Namen der XML-Datei, die die Definition des neuen Adapters enthält (z. B. **my_push_adapter.xml**).

d. **discoveryPatterns\<Ihr_Push-Adapter>.xml:** Aktualisieren Sie diese Datei wie folgt:

- Für das XML-Element **<pattern>:** Legen Sie die Attribute **id** und **description** entsprechend fest. Beispiel:

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

  wird geändert in:

  ```
  <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">
  ```

- Für das XML-Element **<parameters>:** Aktualisieren Sie die untergeordneten Elemente entsprechend den Anforderungen Ihres Adapters. Standardmäßig werden die folgenden untergeordneten Elemente verwendet, um einen Push-Adapter zu definieren. Diese Werte werden zugewiesen, wenn nach der Konfiguration des Adapters der Integrationspunkt in Integration Studio definiert wird. Aktualisieren Sie die Parameterliste, damit sie die erforderlichen Verbindungsattribute widerspiegelt. Entfernen Sie das Attribut **probeName** nicht.
  - **host:** Der Name des Servers, der als Host für den Webservice fungiert.
  - **port:** Der Port, der den UCMDB-Datenempfängerdienst abhört.
  - **Web Service Push Adapter: uri** - Der Rest des URLs, um die Endpunktadresse für den Dienst des Datenempfängers zu bilden.
  - **probeName:** Legt fest, bei welcher Data Flow Probe der Push-Job ausgeführt wird.
- Für das XML-Element **<integration>:** Ändern Sie den Wert des untergeordneten Elements **<category>** in etwas anderes als "Generic" Standardmäßig werden Integrationsadapter, die zur Kategorie "Generic" gehören, nicht in Integration Studio angezeigt. Wenn Sie eine Integration mit einem Datenspeicher eines Drittanbieters durchführen, setzen Sie diesen Wert auf "Third Party". Wenn Sie eine Integration mit einem HP BTO-Produkt durchführen, setzen Sie diesen Wert auf "HP BTO Products".

e. **adapterCode\PushAdapter:** Benennen Sie den Ordner in die im vorherigen Schritt verwendete Adapter-ID um (z. B. **adapterCode\MyPushAdapter**).

f. **discoveryScripts\<Ihr_Jython_Push_Skript>.py:** Erstellen Sie eine Datei mit dem Namen, der in der Eigenschaft **push.properties jythonScript.name** definiert ist. Die Datei **discoveryScript** enthält ein Skript, 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 236). Wenn Sie das Skript umbenennen, sollte die Eigenschaft **jythonScript.name** in **adapterCode\<Adapter_ID>\push.properties** entsprechend aktualisiert werden.

- XML-Push-Adapter: **pushScript.py**
- Webservice-Push-Adapter: **XMLtoWebService.py**

g. **tql\<Ihre_Integrations_TQLs>:** Speichern Sie die TQL-XML-Definition Ihrer Integrations-TQLs wie ein reguläres Package in diesem Verzeichnis. Alle TQLs in diesem Ordner werden zum Zeitpunkt der Bereitstellung des Adapter-Package bereitgestellt.

h. **discoveryConfigFiles\<Name_Ihres_Push-Adapters>\mappings:** Erstellen Sie eine XML-Zuordnungsdatei pro TQL, die Sie in Ihrer Integration verwenden möchten. Beachten Sie, dass der Push-Adapter die Transformationen in der Zuordnungsdatei auf die Ergebnisse der Integrations-TQLs anwendet und die Daten dann in drei Parametern (**addResult** ,**updateResult** und **deleteResult**) einer Ad-hoc-Aufgabe an die Data Flow Probe sendet.

i. **adapterCode\<Adapter_ID>\mappings:** Ersetzen Sie die Datei **mappings.xml** durch die Zuordnungsdateien, die Sie vorbereitet haben (weitere Informationen hierzu finden Sie unter "Erstellen von Zuordnungen" auf der nächsten Seite).

   XML-Push-Adapter: Dieses Zuordnungsbeispiel entspricht dem Beispiel der Tabellen, die in ORACLE in der Datei **sql_queries** erstellt wurden.

   Wenn Sie für jede TQL-Methode eine Zuordnungsdatei verwenden möchten, weisen 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.

4. Wenn Sie alle oben aufgeführten Änderungen vorgenommen haben, erstellen Sie eine **ZIP**-Datei, indem Sie die in Schritt 3 oben angegebenen Ordner und Dateien auswählen (z. B. **Mein_Push_Adapter.zip**).
5. Stellen Sie die neu erstellte **ZIP**-Datei über den Package Manager auf dem UCMDB-Server bereit (navigieren Sie zu **Verwaltung > Package Manager**).
6. Erstellen Sie unter **Datenflussverwaltung > Integration Studio** einen Integrationspunkt und definieren Sie die Integrations-TQLs, die der Integrationspunkt verwendet. Legen Sie einen Zeitplan für den automatischen Datenpush fest.

Fehlerbehebung

Beim Erstellen eines neuen Push-Adapters ist es sehr wichtig, dass beim Umbenennen und Ersetzen keine Fehler gemacht werden, da sich jeder Fehler auf den Adapter auswirken kann. Das Package muss entpackt und erneut verpackt werden, um als UCMDB-Package zu fungieren. Beispiele hierzu bieten die vordefinierten Packages. Im Folgenden sind einige häufig auftretende Fehler aufgeführt:

- Aufnehmen eines zusätzlichen Verzeichnisses oberhalb der Package-Verzeichnisse in die ZIP-Datei.
  Lösung: Verpacken Sie das Package in demselben Verzeichnis, in dem sich auch die Package-Verzeichnisse befinden, z. B. **discoveryResources**, **adapterCode** usw. Nehmen Sie keine weitere übergeordnete Verzeichnisebene in die ZIP-Datei auf.
- Auslassen einer wichtigen Umbenennung eines Verzeichnisses, einer Datei oder einer Zeichenkette in einer Datei.
  Lösung: Befolgen Sie die Anweisungen in diesem Abschnitt sehr sorgfältig.
- Falsche Schreibweise bei einer wichtigen Umbenennung eines Verzeichnisses, einer Datei oder einer Zeichenkette in einer Datei.
  Lösung: Ändern Sie Ihre Benennungskonvention nicht nachträglich, nachdem Sie bereits mit der

Umbenennung begonnen haben. Wenn Sie feststellen, dass Sie den Namen ändern müssen, beginnen Sie noch einmal von vorn und versuchen Sie nicht, den Namen nachträglich zu korrigieren, da hier ein hohes Fehlerrisiko besteht. Zur weiteren Reduzierung des Fehlerrisikos empfiehlt es sich, die Funktion "Suchen und Ersetzen" zu verwenden anstatt Zeichenketten manuell zu ersetzen.

- Bereitstellen von Adaptern mit den gleichen Dateinamen wie andere Adapter, insbesondere in den Verzeichnissen **discoveryResources** und **adapterCode**.
  Lösung: Eventuell verwenden Sie eine UCMDB-Version mit einem bekannten Problem, das verhindert, dass Zuordnungsdateien und andere Adapter, die sich in der gleichen UCMDB-Umgebung befinden, den gleichen Namen haben. Wenn Sie versuchen, ein Package mit doppelten Namen bereitzustellen, schlägt die Package-Bereitstellung fehl. Dieses Problem kann auch dann auftreten, wenn sich die Dateien in verschiedenen Verzeichnissen befinden. Weiterhin kann dieses Problem unabhängig davon auftreten, ob sich die Duplikate im Package oder in zuvor bereitgestellten Packages befinden.

An dieser Stelle können Sie mithilfe des neuen Adapters, den Sie soeben bereitgestellt haben, einen neuen Push-Adapter-Job in Integration Studio erstellen.

TQL-Best-Practices für Push-Adapter

1. Erstellen Sie in den Strukturen "TQL" und "Ansicht" eine Ordnerstruktur und legen Sie dort alle neuen TQLs und Ansichten ab. Verwenden Sie eine Benennungskonvention.
2. Sofern es sich nicht um eine kleine TQL handelt, empfiehlt es sich, die TQL, die die größte Ähnlichkeit aufweist, zu kopieren.
3. Nehmen Sie jeweils nur eine Änderung vor. Ferner ist es ratsam, nach jeder Änderung zu speichern, zu testen und eine Vorschau anzuzeigen. Wiederholen Sie die Schritte, bis Sie mit dem Ergebnis zufrieden sind.

Erstellen von Zuordnungen

Die Rohdaten des TQL-Ergebnisses haben die Form des UCMDB-Klassenmodellschemas. Wahrscheinlich verwendet der Benutzer ein anderes Datenmodell. Der Push-Adapter stellt einen Zuordnungsmechanismus zur Verfügung, um die Daten in ein Format umzuwandeln, das besser für den allgemeinen Gebrauch geeignet ist. Zuordnungen führen sowohl direkte als auch komplexe Transformationen durch, von direkten, auf Benennungen beruhenden Konvertierungsfunktionen bis hin zu Aggregations- und Referenzierungsfunktionen mit über- und untergeordneten Elementen.

Die Zuordnungsspezifikation finden Sie im Abschnitt "Zuordnen der Dateireferenz" auf Seite 260. Verwenden Sie die Referenz beim Erstellen einer Zuordnungsdatei.

Hinweis: Die Datei mit den Adaptereigenschaften verweist auf den Namen der Zuordnungsdatei. In den Adapterkonfigurationsdateien implementiert der Adapter eine Ordnerstruktur unter Verwendung des Adapternamens. Sie müssen diesen Ordner umbenennen, um bei der Implementierung eines Adapters die von Package Manager geforderte Eindeutigkeit zu gewährleisten.

Erstellen einer Zuordnungsdatei

1. Beginnen Sie mit einer Standardzuordnungsdatei.
2. Stellen Sie den Adapter bereit und führen Sie ihn einmal aus.
3. Beobachten Sie die Ergebnisse.
4. Identifizieren und notieren Sie die notwendigen Änderungen.
5. Nehmen Sie die im vorherigen Schritt identifizierten Änderungen vor. Die folgende Auflistung kann als Orientierungshilfe für die Reihenfolge der Änderungen dienen.
   a. Beginnen Sie mit dem oberen, nicht-transformativen Abschnitt. Stellen Sie nach jeder Änderung sicher, dass der Adapter ausgeführt werden kann.
   b. Ändern Sie den Abschnitt "Quell-CIs" in die UCMDB-Namen im TQL-Ergebnis.
   c. Ordnen Sie zuerst die Schlüssel zu.
   d. Fügen Sie dann alle direkten Zuordnungen hinzu.
   e. Fügen Sie die komplexen Zuordnungen hinzu.
   f. Fügen Sie die Link-Zuordnungen hinzu.

Wiederholen Sie die Schritte 2 - 5, bis die zugeordneten Daten zum Gebrauch geeignet sind. Wählen Sie das entsprechende Package mit dem generischen Adapter aus, das Sie als Grundlage für den neuen Push-Adapter verwenden möchten.

Die Zuordnungsdateien funktionieren für alle Arten von Push-Adaptern gleich. Der generische XML-Push-Adapter schreibt die zugeordneten Ergebnisse in eine Datei. Der generische Webservice-Push-Adapter sendet die XML-Ergebnisse an einen Datenempfänger. Weitere Informationen finden Sie unter "Generischer Webservice-Push-Adapter" auf Seite 241.

Vorbereiten der Zuordnungsdateien

Hinweis: Sie können alle CIs und Beziehungen ohne Zuordnung direkt aus der CMDB abrufen und 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 262.

Diese Aufgabe umfasst folgende Schritte:

- "Erstellen der Datei "mappings.xml"" auf der nächsten Seite
- "Zuordnen von CIs" auf der nächsten Seite
- "Zuordnen von Links" auf Seite 235

1. Erstellen der Datei "mappings.xml"

   Die Zuordnungsdateistruktur wird wie folgt erstellt (verwenden Sie eine bestehende Datei als Vorlage):

```
<?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 ist (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, sodass sämtliche Links des Typs und alle geerbten Links auf die gleiche Weise zugeordnet werden:

```
<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, sodass nur Links des entsprechenden Typs verarbeitet werden. Links von geerbten Typen werden nicht verarbeitet, sofern ihr Typ nicht ebenfalls zugeordnet wurde (auf eine der beiden Arten):

```
<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 37.

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')
```

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 Port-Nummer, 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 dem Schritt zum Bearbeiten der Datei **discoveryPatterns\push_adapter.xml** unter "Erstellen eines Adapter-Package" auf Seite 229), verwenden Sie den folgenden Code:

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

Beispiel:

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

Dieser Abschnitt umfasst folgende Themen:

Arbeiten mit den Ergebnissen der Zuordnung

Generische Push-Adapter erstellen 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 `externaIld` 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`.

Es ist auch möglich, den Push-Status für jedes CI und jeden Link wie folgt zu melden:

1. `updateStatus = ReplicationActionDataFactory.createUpdateStatus();`

   wobei `updateStatus` eine Instanz der Klasse `UpdateStatus` ist, die die Status der CIs und Links enthält.

2. Fügen Sie einen Status zu `updateStatus` hinzu, indem Sie die Methode `reportCIStatus` oder die Methode `reportRelationStatus` aufrufen.

   Beispiel:

   ```
   status = ReplicationActionDataFactory.createStatus(Severity.FAILURE, 'Failed',
   ERROR_CODE_CI, errorParams,Action.ADD);

   updateStatus.reportCIStatus(externalId, status);
   ```

   wobei `ERROR_CODE_CI` die Nummer der Fehlermeldungen angibt wie sie in der Adapterdatei **properties.errors** angezeigt werden (weitere Informationen zur Datei **properties.errors** finden Sie unter "Konventionen für das Schreiben von Fehlermeldungen" auf Seite 64) und `errorParams` die Parameter enthält, die an die Meldung übergeben werden. Weitere Informationen finden Sie in der Java-Dokumentation **ReplicationActionDataFactory**.

3. Gehen Sie wie folgt vor, um ein Push-Ergebnis mit den Statusangaben zu erstellen:

   ```
   updateResult = DataPushResultsFactory.createDataPushResults(objectMappings,
   linkMappings, updateStatus);

   return updateResult
   ```

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: Wenn `datatype="BYTE"` ist, dann 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()`. Wenn es gibt Unterschiede im Standardgebietsschema zwischen dem Server und der Probe gibt, wird die Rekonstruktion entsprechend dem Standardgebietsschema des Servers durchgeführt.

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

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.

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 Seite 229.

In dieser Aufgabe richten Sie den Push-Adapter für die Durchführung der differenziellen Synchronisierung ein.

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.

SQL-Abfragen für generische XML-Push-Adapter

Im Adapter-Package enthält die Datei **sql_queries**, die sich unter **adapterCode > PushAdapter > sqlTablesCreation** befindet, die erforderlichen Abfragen, 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.

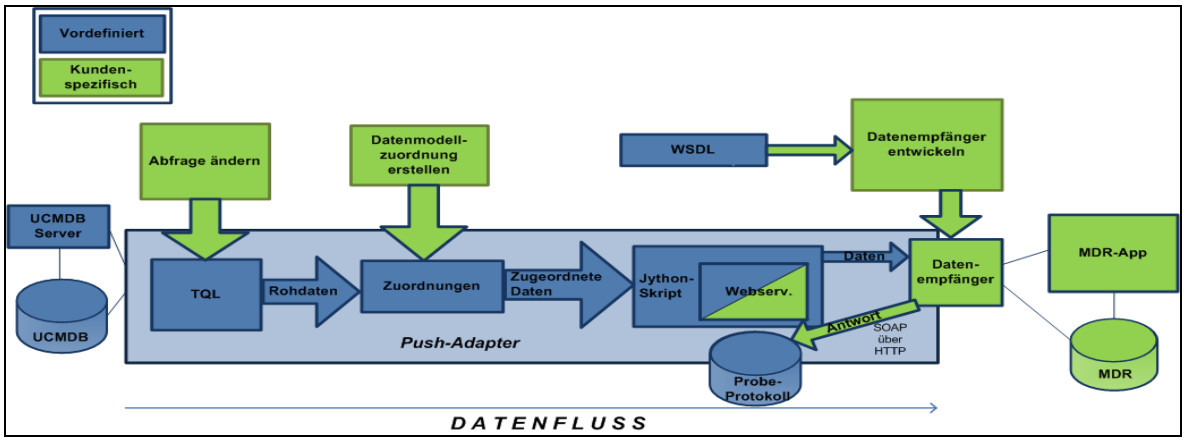
Generischer Webservice-Push-Adapter

Der generische Webservice-Push-Adapter stellt einen von der UCMDB initiierten Push von SOAP-Nachrichten bereit, die Abfragedaten für einen Webservice-Datenempfänger enthalten. Die zugeordneten Ergebnisse werden als standardmäßige SOAP-Nachrichten über das HTTP POST-Protokoll an den Datenempfänger gesendet. Der Datenempfänger muss die vom Push-Adapter produzierten SOAP-Nachrichten verstehen. Um die Entwicklung des entsprechenden Datenempfängers zu erleichtern, wird eine WSDL-Datei mit diesem Push-Adapter bereitgestellt.

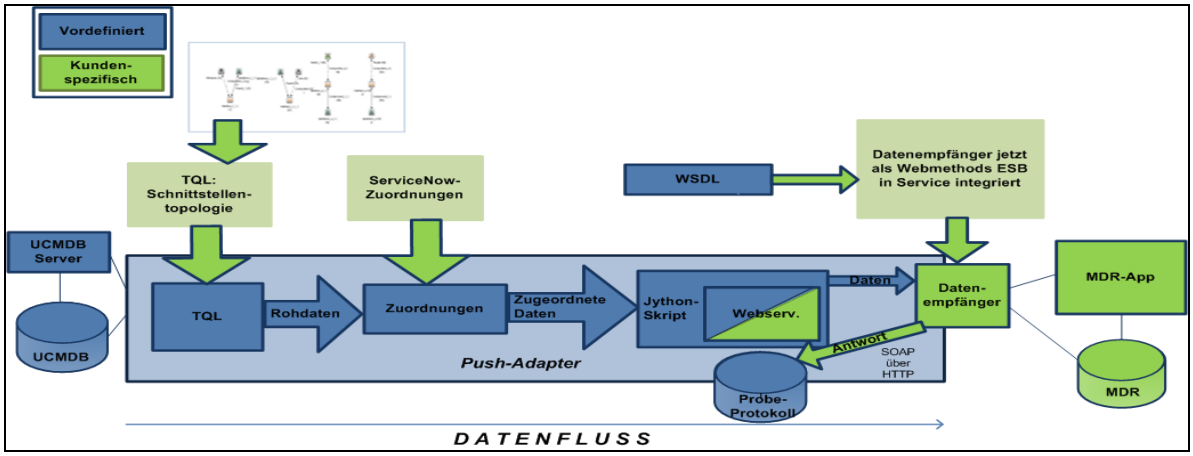
Die benutzerdefinierte Verarbeitung des Antwort-XML-Skripts zur SOAP-Nachricht kann im Jython-Skript erfolgen.

Um das Format der eingehenden zugeordneten Daten zu verstehen, sollte der Entwickler des Datenempfängers mit dem Entwickler der Zuordnungsdatei kommunizieren. Eine **.xsd**-Datei wird derzeit nicht mit dieser Version des Webservice-Push-Adapters zur Verfügung gestellt. Folglich müssen die Daten in einer die eingehenden Daten reflektierenden Form verarbeitet werden, d. h. in einer Kombination der ursprünglichen TQL und der angewendeten Zuordnungen.

Die Funktionen des Webservice-Push-Adapters für die Übertragung der Daten per Push an den Client sind nachfolgend dargestellt. Die grün hervorgehobenen Elemente wurden vom Client angepasst oder bereitgestellt, um den Adapter für ein bestimmtes Push-Ziel zu implementieren. Die blau hervorgehobenen Elemente sind vordefinierte Komponenten.

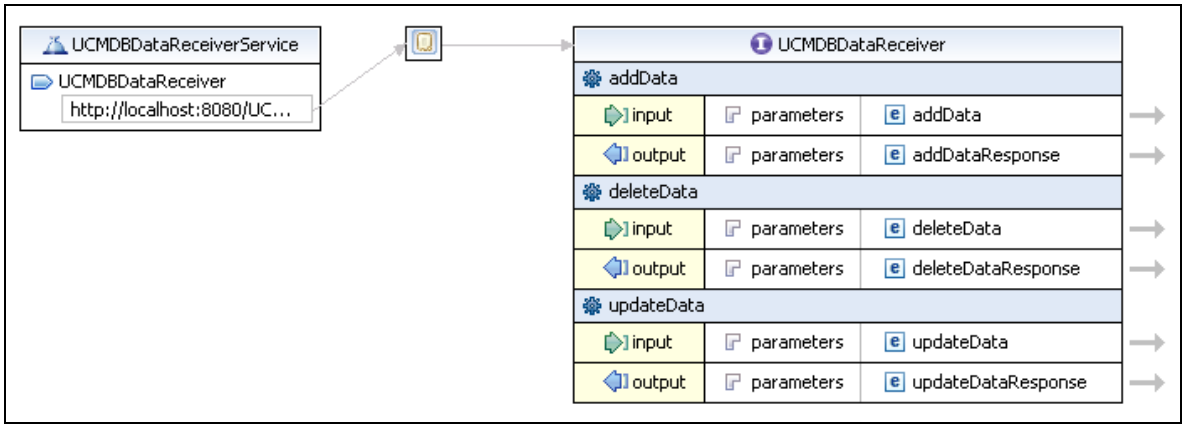


Im Folgenden ist ein Beispiel für eine Implementierung des generischen Webservice-Push-Adapters bei einem MDR-spezifischen Push-Adapter unter Verwendung eines Enterprise Service Bus (ESB) abgebildet:



WSDL

Der Client-Entwickler erhält eine WSDL-Datei, um einen Datenempfänger zu erstellen, der über einen Webservice mit dem UCMDB-Push-Adapter kommunizieren kann. Die Datei **UCMDBDataReceiver.wsdl** beschreibt die SOAP-Nachrichten, die für die Datenkommunikation von der UCDMB zum Datenempfänger verwendet werden. Das Entwurfsdiagramm von der WSDL-Datei ist hier abgebildet:



Der Datenempfänger (der eigentlich ein Server oder "Dienstendpunkt" in der SOAP-Terminologie ist) sollte drei Methoden implementieren: **addData**, **deleteData** und **updateData**, entsprechend den Datasets, die von der UCMDB per Push übertragen werden. Die HTTP-Header enthalten das richtige **SOAPAction**-Schlüsselwort, das den Typ der gesendeten Daten angibt. Der Datenempfänger ist für die Implementierung der Geschäftslogik und die Verarbeitung der Daten zuständig.

Der standardmäßige WSDL-URL lautet wie folgt:

- http://localhost:8080/UCMDBDataReceiver/services/UCMDBDataReceiver?wsdl

Nach der Implementierung durch den Datenempfänger könnte der URL folgendermaßen aussehen:

- http://testWSPAserver:4444/MyCo.IT.SvcMgt.ws.us:provider/UCMDBDataReceiver?wsdl

Abgesehen von der Endung "?wsdl" ist der URL des Webservice identisch mit dem WSDL-URL.

Die Quelle für die WSDL ist unten aufgeführt:

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions targetNamespace="http://ucmdb.hp.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://ucmdb.hp.com" xmlns:intf="http://ucmdb.hp.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!--WSDL created by Apache Axis version: 1.4 Built on Apr 22, 2006 (06:55:48
PDT)-->

    <wsdl:types>

        <schema elementFormDefault="qualified"
        targetNamespace="http://ucmdb.hp.com"
        xmlns="http://www.w3.org/2001/XMLSchema">

            <element name="addData">

                <complexType>

                    <sequence>
```

```
                    <element name="xmlAdded" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="addDataResponse">
                <complexType/>
            </element>
            <element name="deleteData">
                <complexType>
                    <sequence>
                    <element name="xmlDeleted" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="deleteDataResponse">
                <complexType/>
            </element>
            <element name="updateData">
                <complexType>
                    <sequence>
                    <element name="xmlUpdate" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="updateDataResponse">
                <complexType/>
            </element>
        </schema>
    </wsdl:types>

    <wsdl:message name="addDataRequest">
```

```
        <wsdl:part element="impl:addData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="deleteDataResponse">
        <wsdl:part element="impl:deleteDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="updateDataResponse">
        <wsdl:part element="impl:updateDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="deleteDataRequest">
        <wsdl:part element="impl:deleteData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="addDataResponse">
        <wsdl:part element="impl:addDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="updateDataRequest">
        <wsdl:part element="impl:updateData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:portType name="UCMDBDataReceiver">
        <wsdl:operation name="addData">
            <wsdlsoap:operation soapAction="addDataRequest"/>
            <wsdl:input message="impl:addDataRequest" name="addDataRequest">
            </wsdl:input>
            <wsdl:output message="impl:addDataResponse" name="addDataResponse">
            </wsdl:output>
        </wsdl:operation>
```

```
        <wsdl:operation name="deleteData">

            <wsdlsoap:operation soapAction="deleteDataRequest"/>

            <wsdl:input message="impl:deleteDataRequest"
            name="deleteDataRequest">

            </wsdl:input>

            <wsdl:output message="impl:deleteDataResponse"
            name="deleteDataResponse">

            </wsdl:output>

        </wsdl:operation>

        <wsdl:operation name="updateData">

            <wsdlsoap:operation soapAction="updateDataRequest"/>

            <wsdl:input message="impl:updateDataRequest"
            name="updateDataRequest">

            </wsdl:input>

            <wsdl:output message="impl:updateDataResponse"
            name="updateDataResponse">

            </wsdl:output>

        </wsdl:operation>

    </wsdl:portType>

    <wsdl:binding name="UCMDBDataReceiverSoapBinding"
    type="impl:UCMDBDataReceiver">

        <wsdlsoap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />

        <wsdl:operation name="addData">

            <wsdl:input name="addDataRequest">

                <wsdlsoap:body use="literal" />

            </wsdl:input>

            <wsdl:output name="addDataResponse">

                <wsdlsoap:body use="literal" />

            </wsdl:output>

        </wsdl:operation>

        <wsdl:operation name="deleteData">

            <wsdl:input name="deleteDataRequest">
```

```
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="deleteDataResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="updateData">
            <wsdl:input name="updateDataRequest">
                <wsdlsoap:body use="literal" />
            </wsdl:input>
            <wsdl:output name="updateDataResponse">
                <wsdlsoap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="UCMDBDataReceiverService">
        <wsdl:port binding="impl:UCMDBDataReceiverSoapBinding"
        name="UCMDBDataReceiver">
            <wsdlsoap:address
            location="http://localhost:8080/UCMDBDataReceiver/services/
            UCMDBDataReceiver"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Antwortbehandlung

Der Datenempfänger sollte eine Zeichenkette in der Struktur **addDataResponse**, **deleteDataResponse** oder **updateDataResponse** zurückgeben. Der Adapter übergibt die Antwortdaten unverarbeitet an die Datei **probeMgr-adaptersDebug.log** der Probe. Der Empfänger kann beliebige Zeichenkettendaten zurückgeben. Die Antworten werden in ein SOAP-kompatibles XML-Skript eingebettet. Im Jython-Skript können Sie **SOAPMessage** und zugehörige Java-Klassen verwenden, um die Antwortnachrichten zu analysieren. Im Folgenden sehen Sie ein Beispiel für eine Antwortnachricht vom Datenempfänger:

```
<2012-03-16 15:47:38,080> [INFO ] [Thread-110] - XMLtoWebService.py:addData
received response:
```

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<intf:addDataResponse xmlns:intf="http://ucmdb.hp.com">
    <xml>&lt;result&gt;&lt;status&gt;error&lt;/status&gt;
    &lt;message&gt;Error publishing config item changes&lt;/message&gt;
    &lt;/result&gt;</xml>
</intf:addDataResponse>
</soapenv:Body>
```

Es handelt sich bei dieser Nachricht um eine Fehlermeldung **<Error publishing config item changes>**. Die Antwort des Datenempfängers kann jedoch beliebigen Inhalts sein. In diesem Fall ist die Antwort eine Fehlermeldung, weil dies die Absicht des Entwicklers war, und der Push-Adapter erwartet eine Antwort, die auf eine erfolgreiche oder eine fehlgeschlagene Durchführung hinweist. Der Inhalt können Abstimmung-IDs aller erfolgreich hinzugefügten CIs oder Fehlermeldungen für bestimmte CIs sein. Die Anpassung des GWSPA könnte die Analyse der Antwortnachricht und das Ergreifen von Maßnahmen wie das erneute Senden bestimmter CIs oder das Durchführen von Protokollierungen umfassen.

Testen der WSDL

Das SOAPUI Eclipse-Plugin wird zum Testen der Webservice-Schichten während der Entwicklung verwendet. Sie können SOAPUI bei der Anpassung eines Webservice zu Hilfe nehmen. SOAPUI bietet eine integrierte Entwicklungsumgebung (Integrated Development Environment, IDE), um den Aufbau, das Senden und das Empfangen von SOAP-Nachrichten zu testen. Aus der SOAPUI-Perspektive hat die WSDL auf den Seiten 243-247 die folgende Beispielnachricht generiert:

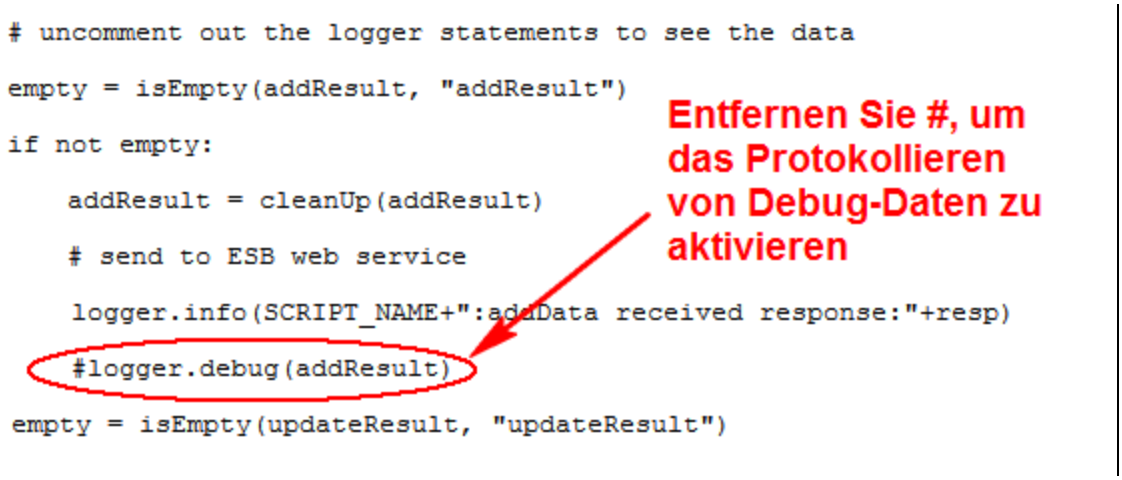
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ucm="http://ucmdb.hp.com">
    <soapenv:Header/>
    <soapenv:Body>
        <ucm:addData>
            <ucm:xmlAdded>?</ucm:xmlAdded>
        <ucm:addData>
    </soapenv:Body>
</soapenv:Envelope>
```

Das "**?**" oben im Element **xmlAdded** ist der Speicherort der Daten, der von der Integration des Webservice-Push-Adapters bereitgestellt wird.

Beobachten der Ergebnisse

Wenn der Push-Adapter normal, d. h. im Nicht-Debug-Modus, arbeitet, werden die Daten erst dann in eine Datei geschrieben, wenn das endgültige Ergebnis geschrieben wird (die TQL-Zwischenergebnisse und die Ergebnisse der zugeordneten Daten werden normalerweise nicht in der Protokolldatei

angezeigt). Die Ergebnisse können jedoch in die Debug-Datei der Probe geschrieben werden, indem die Kommentierung der **logger.debug**-Anweisungen (durch Entfernen des "#"-Zeichen) im Abschnitt **DiscoveryMain** wie nachfolgend gezeigt aufgehoben wird:




```
# uncomment out the logger statements to see the data

empty = isEmpty(addResult, "addResult")

if not empty:

    addResult = cleanUp(addResult)

    # send to ESB web service

    logger.info(SCRIPT_NAME+":addData received response:"+resp)

    #logger.debug(addResult)

empty = isEmpty(updateResult, "updateResult")
```

Stellen Sie sicher, dass die Logger-Anweisung in der gleichen Spalte beginnt wie die vorangehenden und nachfolgenden Zeilen. Jython berücksichtigt Einrückungen und wenn die Einrückung aller Zeilen nicht korrekt ist, schlägt das Skript fehl.

Die Debug-Protokolldatei **probeMgr-adaptersDebug.log** auf der Probe zeigt den Inhalt der Ausgabe:

```
<2011-12-07 14:02:23,019> [INFO ] [Thread-273] - XMLtoWebService.py started

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - ESB Push parameters:

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - Wshost=harpy.trtc.com

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] - WShostport=5555

<2011-12-07 14:02:23,019> [DEBUG] [Thread-273] -
WSuri=ws/DtITServiceManagement.esla.v1.ws.provider:UMDBDataReceiver

<2011-12-07 14:02:23,019> [INFO ] [Thread-273] - URL is
http://harpy.trtc.com:5555/ws/DtITServiceManagement.esla.v1.ws.
provider:UMDBDataReceiver

<2011-12-07 14:02:23,035> [DEBUG] [Thread-273] - Connected to
http://harpy.trtc.com:5555/ws/DtITServiceManagement.esla.v1.ws.
provider:UMDBDataReceiver

<2011-12-07 14:02:23,035> [ERROR] [Thread-273] - sending results

<2011-12-07 14:02:23,035> [DEBUG] [Thread-273] - <?xml version="1.0"
encoding="UTF-8"?>

<root>

    <data>

        <objects>
```

```
<Object mode="" name="u_imp_ip_switch" operation="add"
mamId="9e8c2f6bdfe4b7d0864c79e70833902c">

    <field name="Correlation ID" key="true" datatype="char"
    length="">9e8c2f6bdfe4b7d0864c79e70833902c</field>

    <field name="name" key="false" datatype="char" length="">nma_
    09sw</field>

    <field name="location" key="false" datatype="char" length="" />

    <field name="u_chassis_vendor_type" key="false" datatype="char"
    length="">ciscoCat2960-24TT</field>

    <field name="serial_number" key="false" datatype="char"
    length="" />

    <field name="ram" key="false" datatype="char" length="" />

    <field name="os_version" key="false" datatype="char" length=""
    />

</Object>
```

Ändern des Jython-Skripts

XMLtoWebService.py

Das vom Webservice-Push-Adapter verwendete Jython-Skript weist große Ähnlichkeit mit dem XML-Push-Adapter auf. Das Skript verwendet die im Adapter enthaltene Datei **UCMDBDataReceiver.jar**. Es implementiert die **SendDataToReceiver()**-Methode. **SendDataToReceiver()** verwendet drei Parameter:

1. Aktion (Hinzufügen, Aktualisieren oder Löschen)
2. Den URL des Datenempfängers
3. Die Daten

Der Hinzufügungsblock sieht beispielsweise folgendermaßen aus: **SendDataToReceiver("add", URL, addResult)**

Alle Webservice- und SOAP-Schichten werden eingebettet. Der URL ist die Dienstendpunktadresse des UCMDB-Datenempfängers. Dieser URL ist identisch mit dem URL, der zum Abrufen der WSDL über das Suffix "?wsdl" verwendet wird.

Die Quelle des Jython-Skripts ist im Folgenden aufgeführt. Die Zeilen des Webservice-Integrationswrappers sind grün hervorgehoben.

```
####################################

# script: XMLtoWebService.py

####################################
```

```
# This jython script accepts TQL data results (adds, updates, and deletes) from
the Integration adapter.

# and sends it to a web service. The web service is called UCMDBDataReceiver.

# A web service client of this name must be addressable at the URL provided by
the parameters.

# The SendDataToReceiver.jar exposes the SendDataToReceiver function, as well as
the service locator.

# examples of the service locator are in the testconnection section.

# regular expressions

import re

# logging

import logger

# web service interface

from com.hp.ucmdb import SendDataToReceiver

from com.hp.ucmdb.SendDataToReceiver import locateService

from com.hp.ucmdb.SendDataToReceiver import SendData

################################################

########      VARIABLES             ##########

################################################

SCRIPT_NAME = "XMLtoWebService.py"

logger.info(SCRIPT_NAME+" started")

def cleanUp(str):


    # replace mode=""

    str = re.sub("mode=\"\w+\"\s+", "", str)


    # replace mamId with id

    str = re.sub("\smamId=\"", " id=\"", str)


    # replace empty attributes

    str = re.sub("[\n|\s|\r]*<field name=\"\w+\" datatype=\"\w+\" />", "", str)
```

```
    # replace targetRelationshipClass with name
    str = re.sub("\stargetRelationshipClass=\"", " name=\"", str)

    # replace Object with object with name
    str = re.sub("<Object mode=\"", "<object mode=\"", str)
    str = re.sub("<Object operation=\"", "<object operation=\"", str)
    str = re.sub("<Object name=\"", "<object name=\"", str)
    str = re.sub("</Object>", "</object>", str)

    # replace field to attribute
    str = re.sub("<field name=\"", "<attribute name=\"", str)
    str = re.sub("</field>", "</attribute>", str)

    #logger.debug("String = %s" % str)
    #logger.debug("cleaned up")

    return str
def isEmpty(xml, type = ""):
    objectsEmpty = 0
    linksEmpty = 0

    m = re.findall("<objects />", xml)
    if m:
        #logger.warn("\t[%s] No objects found" % type)
        objectsEmpty = 1

    m = re.findall("<links />", xml)
    if m:
        #logger.warn("\t[%s] No links found" % type)
        linksEmpty = 1
```

```
    if objectsEmpty and linksEmpty:
        return 1
    return 0
##############################################
########      MAIN                 ##########
##############################################
def DiscoveryMain(Framework):
    #fix this for web service export
    errMsg = "UCMDBDataReceiver Service not found."
    testConnection = Framework.getTriggerCIData("testConnection")
    # Get Web Service Push variables
    WShostName = Framework.getTriggerCIData("Host Name")
    WShostport = Framework.getTriggerCIData("Protocol Port")
    WSuri = Framework.getTriggerCIData("URI")

    logger.info(SCRIPT_NAME+":ESB Push parameters:")
    logger.info("Host Name="+WShostName)
    logger.info("Protocol Port="+WShostport)
    logger.info("URI="+WSuri)
    URL = "http://"+WShostName+":"+WShostport+"/"+WSuri
    logger.info("URL="+URL)
    if testConnection == 'true':
        # locate the service
        test_receiver = SendDataToReceiver()
        locator = test_receiver.locateService(URL)
        #locator = locateService(URL)
        if(locator):
            logger.info(SCRIPT_NAME+":Test connection was successful")
            return
        else:
            raise Exception, errMsg
```

```
        return
# do same thing here if not just a test connection -
receiver = SendDataToReceiver()
locator = receiver.locateService(URL)
if(locator):
    logger.info(SCRIPT_NAME+":Connected to "+URL)
else:
    logger.error(SCRIPT_NAME+":no locator")
    raise Exception, errMsg
    return

# get add/update/delete result objects from the Framework
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
logger.debug(deleteResult)

# get referenced data - unused in this adapter implementation
#addRefResult = Framework.getTriggerCIData('referencedAddResult')
#updateRefResult = Framework.getTriggerCIData('referencedUpdateResult')
#deleteRefResult = Framework.getTriggerCIData('referencedDeleteResult')
# uncomment out the logger statements to see the data
empty = isEmpty(addResult, "addResult")
if not empty:
    addResult = cleanUp(addResult)
    # send to ESB web service
    logger.info(SCRIPT_NAME+":sending addData Result")
    rcvr = SendDataToReceiver()
    resp = rcvr.SendData("add", URL, addResult)
    logger.info(SCRIPT_NAME+":addData received response:"+resp)
    #logger.debug(addResult)
```

```
    empty = isEmpty(updateResult, "updateResult")
    if not empty:
        updateResult = cleanUp(updateResult)
        # send to ESB web service
        #logger.debug(updateResult)
        logger.info(SCRIPT_NAME+":sending updateData Result")
        rcvr = SendDataToReceiver()
        resp = rcvr.SendData("update", URL, updateResult)
        logger.info(SCRIPT_NAME+":received response:"+resp)


    empty = isEmpty(deleteResult, "deleteResult")
    if not empty:
        deleteResult = cleanUp(deleteResult)
        # send to ESB web service
        #logger.debug(deleteResult)
        logger.info(SCRIPT_NAME+":sending deleteData Result")
        rcvr = SendDataToReceiver()
        resp = rcvr.SendData("delete", URL, deleteResult)
        logger.info(SCRIPT_NAME+":received response:"+resp)
    logger.info(SCRIPT_NAME+" ended")
```

Anpassen der Verarbeitung der Antwortnachricht

Der Datenempfänger sollte eine Zeichenkette zurückgeben, die die gewünschte Antwort oder den gewünschten Status enthält. Der Webservice-Push-Adapter übergibt die Antwort standardmäßig an das Informationsebenenprotokoll der Probe. Die Antwortnachricht ist ein SOAP-formatiertes XML-Skript, das die zurückgegebene(n) Antwortzeichenkette(n) enthält. Der Empfänger kann beliebige Daten zurückgeben, wie beispielsweise gruppierte oder einzelne Fehler- oder Erfolgsmeldungen. Wenn eine weitere Verarbeitung gewünscht wird, kann die Antwort vom Jython-Skript des Adapters verarbeitet werden. Es ist keine Java-Programmierung erforderlich.

Ein Beispiel für eine zurückgegebene Antwortmeldung, gesendet mithilfe des folgenden Codes:

```
// stub example for building your own UCMDBDataData Receiver
public class UCMDBDataReceiver {
```

```
public String addData (String xmlAdd){
System.out.println(xmlAdd); // do something with the data
// send back a response message based on what you did
String tr = new String("a test response from addData!");
return tr;
}
```

sehen Sie hier:

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <addDataResponse xmlns="http://ucmdb.hp.com">
        <addDataReturn>a test response from addData!</addDataReturn>
    </addDataResponse>
</soapenv:Body>
```

Ändern des Datenempfängers

Ein Java-Client kann die Klassen implementieren, die in der Datei **UCMDBDataReceiver.jar** enthalten sind, und den Webservice genauso aufrufen wie ein Jython-Skript. Außerdem können auch nicht eingebettete Methoden aufgerufen werden. Für die Klassen **UCMDBDataReceiver.jar** ist eine Java-Dokumentation vorhanden. Der unten aufgeführte Quellcode zeigt, wie Sie diese grundlegenden Methoden verwenden, um die Daten in eine SOAP-Nachricht einzubetten und über HPPT an den Empfänger zu senden.

Der erste Schritt besteht darin, ein **UCMDBDataReceiverServiceLocator**-Objekt zu erstellen. Anschließend wird das Objekt **UCMDBDataReceiverEndPointAddress** dem URL des Datenempfängers zugewiesen.

Zum Senden der Daten wird die **getUCMDBDataReceiver**-Methode des Lokators aufgerufen, um ein **UCMDBDataReceiver**-Objekt zu erstellen. Das **UCMDBDataReceiver**-Objekt implementiert die Methoden, um die Hinzufügungs-/Änderungs-/Löschungsdaten zu senden. Es gibt drei identische Codeblöcke, um jeden Anforderungstyp zu verarbeiten.

Der Quellcode für die Klasse **SendDataToReceiver** ist unten aufgeführt. Die hervorgehobenen Objekte und Methoden sind die wesentlichen zu verwendenden Elemente.

```
/**
* Test SendData for the UCMDB Data Receiver for the UCMDB Web Service Push
Adapter
*/
package com.hp.ucmdb;
```

```
import com.hp.ucmdb.SendDataToReceiver;

/**

* TestSendData can be used to verify the SOAP classes are working.

* TestSendData creates a SendDataToReceiver class and invokes its SendData
method.

* a response String is returned.

* The test URL is typically appended with "?wsdl" to get the WSDL of the
service.

*/

public class TestSendData {

    /**

    * @param args - test SOAP message.

    *  optional arguments [0] a test string [1] a service endpoint URL of a
    Data Receiver.

    * the default URL is sent the incoming argument as a test message.

    * the default URL is
    "http://localhost:8080/UCMDBDataReceiver/services/UCMDBDataReceiver".

    * If any errors are encountered, TestClient will attempt to throw
    exceptions.

    */

    public static void main(String[] args) {

        // use test message if supplied, otherwise supply a default test string

        String teststring = new String("Test SOAP message from
        UCMDBDataReceiver TestSendData.");

        if(args.length > 0) {

            teststring = args[0];

        }

        // use test URL if supplied, otherwise supply the default URL

        String URL = new String("");

        if(args.length > 1) {

            URL = args[1];

        }

        // return response
```

```
        String response = new String("");
        // perform the tests
        try{
            if(URL.equals("")) {
                UCMDBDataReceiverServiceLocator locator = new
                UCMDBDataReceiverServiceLocator();
                UCMDBDataReceiver receiver = locator.getUCMDBDataReceiver();
                URL = locator.getUCMDBDataReceiverAddress();
                System.out.println("TestClient: tested
                URL="+locator.getUCMDBDataReceiverAddress());
                System.out.println("TestClient: receiver="+receiver.toString
                ());
            }
            SendDataToReceiver sdtr = new SendDataToReceiver();
            // this sends a test push and gets a response message
            response = sdtr.SendData("add", URL, args[0]);
            System.out.println("Response received was:"+response);
        } catch(Exception e){
            System.out.println("TestClient: Remote Error:");
            e.printStackTrace();
        }
    }
}
```

Der Quellcode ist auch in der Datei **UCMDBDataReceiver.jar** für die anderen Klassen enthalten:

- TestClient.java
- UCMDBDataReceiver.java
- UCMDBDataReceiverProxy.java
- UCMDBDataReceiverService.java
- UCMDBDataReceiverServiceLocator.java
- UCMDBDataReceiverSoapBindingStub.java

Die Quelle wurde in der Eclipse-IDE erzeugt und dann geändert. Beim Ändern des UCMDB-Codes ist Vorsicht geboten, da ein Großteil automatisch generiert wird, um die SOAP-Spezifikation einzuhalten.

Java-Dokumentation

Mit dem generischen Webservice-Push-Adapter wird eine vollständig kommentierte **Java-Dokumentation** bereitgestellt. Die **Java-Dokumentation** ist im Dokumentordner **javadoc** gespeichert. Beginnen Sie mit der Datei **index.html**. Die Übersichtsseite ermöglicht den Zugriff auf die Dokumentation für alle Klassen und Methoden in der SDK.

Alle Klassen

- **SendDataToReceiver:** API für den Webservice-Wrapper
- **TestClient:** Testclient zum Überprüfen der Konnektivität zu einem Dienstendpunkt
- **UCMDBDataReceiver:** Webservice-Wrapper

Der Rest wird vom Webservice-Builder automatisch generiert:

- UCMDBDataReceiverProxy
- UCMDBDataReceiverService
- UCMDBDataReceiverServiceLocator
- UCMDBDataReceiverSoapBindingStub

Übersicht

Die grundlegende Verwendung des SDK, einschließlich Quellcodebeispiele, wird in der Dokumentation im Package erläutert. Diese **Java-Dokumentation** ist für den UCMDB-Webservice-Push-Adapter vorgesehen. Die API kann von Jython oder Java aufgerufen werden.

Das SDK enthält zwei Quellenbeispiele: **TestClient** und **SendDataToReceiver**. **TestClient** bietet einen sehr begrenzten Test des antwortenden lokalen Client. **SendDataToReceiver** ist die Hauptklasse, die zum Senden von Daten an einen Webservice verwendet wird.

Verwenden Sie dieses SDK (vor allem die beigefügte WSDL) zuerst, um einen UCMDB-Datenempfänger für die Kommunikation mit diesem Webservice zu implementieren. Verwenden Sie das SDK dann, um einen Push-Adapter in der UCMDB zu erstellen und die UCMDB TQL-Ergebnisdaten per Push an den Datenempfänger zu übertragen. Die grundlegende Verwendung dieser API wird im Folgenden sowohl mit Jython- als auch mit Java-Implementierungen beschrieben.

Implementieren von "SendDataToReceiver()"

SendDataToReceiver() bettet alle Funktionen in eine einzige Methode ein:

- Jython: `SendDataToReceiver("add",yourURL,"Hello!")`
- Java: `SendDataToReceiver("add",yourURL,"Hello!");`

Alternativ können Sie wie im folgenden Beispiel gezeigt ein **SendDataToReceiver**-Objekt (beispielsweise zum Bearbeiten anderer Einstellungen) erstellen und dann die Methode **SendData** separat aufrufen:

- Jython:

```
rcvr = SendDataToReceiver()

responseMsg = rcvr.SendData(“add”, yourURL, “Hello!”)
```

- Java:

```
SendDataToReceiver rcvr = new SendDataToReceiver();

String responseMsg = rcvr.SendData(“add”, yourURL, “Hello!”);
```

Oder gehen Sie folgendermaßen vor, wenn Sie eine schrittweise Vorgehensweise bevorzugen:

1. Erstellen Sie ein neues **UCMDBDataReceiverServiceLocator( )**-Objekt x und legen Sie die Endpunktadresse des Objekts später fest, wie es im folgenden Beispiel gezeigt wird:
   - **Jython:**

     ```
     x = UCMDBDataReceiverServiceLocator()

     x.setUCMDBDataReceiverEndPointAddress(URL)
     ```

   - **Java:**

     ```
     UCMDBDataReceiverServiceLocator x = new UCMDBDataReceiverServiceLocator();

     x. setUCMDBDataReceiverEndPointAddress(URL);
     ```

2. Erstellen Sie dann wie folgt einen UCMDB-Datenempfänger:
   - **Jython:** `y = x.getUCMDBDataReceiver()`
   - **Java:** `UCMDBDataReceiver y = x.getUCMDBDataReceiver();`
3. Senden Sie anschließend die Daten wie folgt über den SOAP-Webservice:
   - **Jython:**
     - `y.addData(yourData)`
     - oder `y.updateData(yourData)`
     - oder `y.deleteData(yourData)`
   - **Java:**
     - `y.addData(yourData);`
     - oder `y.updateData(yourData);`
     - oder `y.deleteData(yourData);`
4. Es kann erforderlich sein, die Konnektivität zu testen. Wenn der Test erfolgreich war, verwenden Sie das gleiche Lokatorobjekt, um **UCMDBDataReceiver** für die Datenübertragung zurückzugeben.

Die Klassen enthalten keine Destruktoren und führen keine Speicherverwaltung durch.

Zuordnen der Dateireferenz

Verwenden von Zuordnungen

Eine Zuordnung muss für jedes Zielattribut in der transformierten XML-Ausgabe erstellt werden. Die Zuordnungen geben an, wo und wie die Daten abgerufen werden sollen. Wenn sich die Daten in einem anderen entsprechenden Attribut in der UCMDB befinden, wird eine direkte Zuordnung verwendet.

Um Daten von mehreren Attributen oder Attribute von den Attributen eines über- oder untergeordneten CI des UCMDB-CI abzurufen, können komplexe Zuordnungen erforderlich sein. Das unten aufgeführte Zuordnungsschema zeigt alle möglichen Zuordnungen.

Die Zuordnungsdatei ist eine XML-Datei, die definiert, welche CI-/Beziehungstypen in der UCMDB welchen CI-/Beziehungstypen im Zieldatenspeicher zugeordnet werden. Das Format wird im Folgenden ausführlich erläutert. Die Zuordnungsdatei steuert, welche CI- und Beziehungstypen und welche Attribute per Push übertragen werden.

Für jedes Attribut, das per Push an das Ziel-MDR übertragen werden soll, ist ein Zuordnungseintrag vorhanden. Jeder Zuordnungseintrag kann aus einem oder mehreren Attribut(en) in den UCMDB-Push-Rohdaten bestehen. Zuordnungseinträge ermöglichen eine vollständig granulare Steuerung der endgültigen Struktur und der Benennung der Daten, die per Push an das Ziel-MDR übertragen werden sollen.

Direkte Zuordnungen

Zuordnungen transformieren ein Datenmodell in ein anderes (in diesem Fall von der UCMDB in das Push-Ziel-MDR). Transformationen können einfach sein, wie beispielsweise im Fall einer 1:1-Beziehung zwischen dem UCMDB-Attribut und dem Ziel. Sie unterscheiden sich nur durch den Namen und vielleicht durch den Typ.

Bei den meisten Attributzuordnungen handelt es sich um direkte Zuordnungen. Zum Beispiel kann der Servername "ServerX" in UCMDB als CI des Typs **unix** mit dem Attributnamen **primary_server_name** und dem Typ **string** mit einer Länge von 50 repräsentiert werden. Das Datenmodell des Ziel-MDR kann dieselbe logische Entität als CI-Typ **linux** mit dem Attributnamen **hostname** und dem Typ **char[]** mit einer Länge von maximal 250 angeben. Direkte Zuordnungen können alle oben genannten Typen von Übersetzungsaufgaben erfüllen.

Nachfolgend finden Sie ein Beispiel für eine direkte Zuordnung:

```
 <target_attribute name="dns_domain" datatype="char">

<map type="direct" source_attribute="domain_name" />

</target_attribute>
```

Diese direkte Zuordnung ordnet das UCMDB-Attribut **dns_domain** dem Attribut **domain_name** im Ziel-Datenmodell zu.

Verwenden Sie den Datentyp **char** ungeachtet des tatsächlichen Datentyps, sofern es nicht unbedingt erforderlich ist, den tatsächlichen Datentyp zu verwenden.

Komplexe Zuordnungen

Komplexere Zuordnungen ermöglichen zusätzliche Transformationen:

- Um Attributwerte von mehreren CIs zu einem Ziel-CI zuzuordnen.
- Um Attribute von untergeordneten CIs (mit der Beziehung **container_f** oder einer enthaltenen Beziehung) dem übergeordneten CI im Zieldatenspeicher zuzuordnen. Ein Beispiel wäre die Einstellung eines Werts mit der Bezeichnung **Number of CPUs** bei einem Zielhost-CI. Ein weiteres Beispiel könnte die Einstellung des Werts **Total Memory** bei einem Zielhost-CI sein (durch Addition der Arbeitsspeicherwerte aller Speicher-CIs eines Host-CI in der UCMDB).
- Um Attribute von übergeordneten CIs (mit der Beziehung **container_f** oder einer enthaltenen Beziehung) dem CI des Zieldatenspeichers zuzuordnen. Ein Beispiel wäre die Einstellung eines Werts mit der Bezeichnung **Container Server** bei einem Zielattribut eines CI mit der Bezeichnung **Installed Software**, indem der Wert vom übergeordneten Host des Software-CI in der UCMDB abgerufen wird.

Nachfolgend sehen Sie ein Beispiel für eine komplexe Zuordnung, bei der zwei durch Kommata voneinander getrennte Quellattribute verwendet werden, um das Zielattribut **os** zu erstellen:

```
<target_attribute name="os" datatype="char">
    <map type="compoundstring">
        <source_attribute name="discovered_os_name" />
        <constant value="," />
        <source_attribute name="host_osinstalltype" />
    </map>
</target_attribute>
```

Umkehren der Linkrichtungen

Es ist möglich, dass die UCMDB Daten enthält, deren Struktur von Quelle zu Quelle unterschiedlich ist. Zum Beispiel kann die Beziehung zwischen einem IpAddress-CI und einem Interface-CI eine **übergeordnete Beziehung** sein, wie es bei der Integration von HP Network Node Manager der Fall sein kann. Oder es kann sich um einen **Containment-Link** handeln, wie er häufig von Universal Discovery erstellt wird. Darüber hinaus verlaufen diese Links in entgegengesetzten Richtungen zueinander.

Es ist derzeit nicht möglich, die Richtung von Links in der Zuordnungsdatei umzukehren. Durch die Umkehr der Variablen **_end1** und **_end2** wird entweder die Reihenfolge der Daten in der transformierten XML-Datei geändert oder der Link fehlt in den Quelldaten.

Eine mögliche Lösung für dieses Problem besteht darin, wie nachfolgend beschrieben eine Enrichment-Regel zu definieren:

1. Der TQL-Teil des Enrichments ist eine Untergruppe einer vom Push-Adapter verwendeten TQL. Diese TQL wählt alle Links aus, die entgegengesetzt zu der in der transformierten XML-Datei angegebenen Richtung verlaufen.
2. Der Enrichment-Teil definiert einen neuen Link in der richtigen Richtung und des gewünschten Typs.
3. Enrichment wird aktiviert und erstellt dann die richtigen Links.
4. Die TQL des Integrationsjobs verweist nun auf den Enrichment-Link anstatt auf den ursprünglichen Link.
5. Die Zuordnungen vom Typ <link> im Push-Adapter verweisen dann ebenfalls auf den Enrichment-Link und erzeugen eine Reihe von Links, die im Hinblick auf den Typ und die Richtung konsistent sind.

Schema der Zuordnungsdatei

Elementname/-pfad	Beschreibung	Attribute
Integration	Definiert die Zuordnungsinhalte der Datei. Abgesehen von der Anfangszeile und	

Elementname/-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. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** Versionen **Beschreibung:** Version(en) der Quelldaten-Repositorys **Verwendung:** Erforderlich **Typ:** Zeichenkette 3. **Name:** vendor **Beschreibung:** Anbieter des Quelldaten-Repository **Verwendung:** Erforderlich **Typ:** Zeichenkette
target (integration > info)	Definiert die Informationen über das Zieldaten-Repository.	1. **Name:** type **Beschreibung:** Name des Quelldaten-Repository. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** Versionen **Beschreibung:** Version(en) des Quelldaten-Repository. **Verwendung:** Erforderlich **Typ:** Zeichenkette 3. **Name:** vendor **Beschreibung:** Anbieter des Quelldaten-Repository **Verwendung:** Erforderlich **Typ:** Zeichenkette
targetcis (integration)	Container-Element für alle CIT-Zuordnungen.	
source_ci_type_tree (integration > targetcis)	Definiert einen Quell-CIT und alle davon geerbten	1. **Name:** name **Beschreibung:** Name des Quell-CIT.

Elementname/-pfad	Beschreibung	Attribute
	CI-Typen.	**Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** mode **Beschreibung:** Der für den aktuellen CI-Typ erforderliche Aktualisierungstyp. **Verwendung:** 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. **Verwendung:** 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 > source_ci_type	Definiert einen Ziel-CIT.	1. **Name:** name **Beschreibung:** Name des Ziel-CI-Typs. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** schema

Elementname/-pfad	Beschreibung	Attribute
Oder integration > targetcis > source_ci_type_tree)		**Beschreibung:** Der Name des Schemas, das zum Speichern dieses CI-Typs am Ziel verwendet wird. **Verwendung:** Nicht erforderlich **Typ:** Zeichenkette 3. **Name:** namespace **Beschreibung:** Gibt den Namespace dieses CI-Typs beim Ziel an. **Verwendung:** 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 > targetrelations > link > targetprimarykey) Oder integration > targetrelations >	Identifiziert ein Primärschlüsselattribut. Nur erforderlich, wenn **mode** = **update** oder **insert_else_update**.	

Elementname/-pfad	Beschreibung	Attribute
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. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** datatype **Beschreibung:** Datentyp des Attributs des Ziel-CIT. **Verwendung:** Erforderlich **Typ:** Zeichenkette 3. **Name:** length **Beschreibung:** Für Daten vom Typ "Zeichenkette/Zeichen" die Ganzzahlgröße des Zielattributs. **Verwendung:** 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 integration > targetcis > source_ci_type_tree > target_attribute) Oder (integration >	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 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

Elementname/-pfad	Beschreibung	Attribute
targetrelations > link > target_attribute Oder integration > targetrelations > source_link_type_tree > target_attribute)		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 integration > targetrelations > source_link_type_tree > target_attribute > map) Nur gültig für Zuordnungen des Typs **childattr**	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. **Verwendung:** 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 die Summe aller numerischen eingeschlossenen Werte zurück. • **average** – Gibt einen numerischen Durchschnitt aller eingeschlossenen Werte zurück. • **min** – Gibt den niedrigsten eingeschlossenen Wert (numerisch/Zeichen) zurück. • **max** – Gibt den höchsten eingeschlossenen Wert (numerisch/Zeichen) zurück.
source_child_ci_type	Gibt an, von welchem	1. **Name.** name

Elementname/-pfad	Beschreibung	Attribute
(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**.	verbundenen CI das untergeordnete Attribut genommen wird.	**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 integration > targetcis > source_ci_type_tree > target_attribute > map Oder (integration > targetrelations > link > target_attribute > map Oder integration > targetrelations > source_link_type_tree	Erlaubt die Ausschlussfilterung der dem Quell-CI untergeordneten CIs auf Basis von Attributwerten. Wird mit dem Unterelement aggregation verwendet, um Granularität darüber zu erreichen, welche untergeordneten Attribute dem Attributwert des Ziel-CIT zugeordnet sind. Optional.	1. **Name.** minlength **Beschreibung.** Schließt Zeichenketten aus, die kürzer als der angegebene Wert sind. **Verwendung:** Nicht erforderlich **Typ.** Ganzzahl 2. **Name.** maxlength **Beschreibung.** Schließt Zeichenketten aus, die länger als der angegebene Wert sind. **Verwendung:** Nicht erforderlich **Typ.** Ganzzahl 3. **Name.** minvalue **Beschreibung.** Schließt Zahlen aus, die kleiner als der angegebene Wert sind. **Verwendung:** Nicht erforderlich **Typ.** Numerisch 4. **Name.** maxvalue **Beschreibung.** Schließt Zahlen aus, die größer als der angegebene Wert sind. **Verwendung:** Nicht erforderlich **Typ.** Numerisch

Elementname/-pfad	Beschreibung	Attribute
> target_attribute > map) Nur gültig für Zuordnungen des Typs **childatt**		
targetrelations (integration)	Container-Element für alle Beziehungszuordnungen. 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 **Verwendung:** Erforderlich **Typ.** Zeichenkette 3. **Name:** nameSpace **Beschreibung:** Der Namespace für den Link, der beim Ziel erstellt wird. **Verwendung:** Nicht erforderlich **Typ:** Zeichenkette 4. **Name:** mode **Beschreibung:** Der für den aktuellen Link erforderliche Aktualisierungstyp. **Verwendung:** Erforderlich **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

Elementname/-pfad	Beschreibung	Attribute
		Beschreibung: CI-Typ von Ende 1 der Quellbeziehung. **Verwendung:** Erforderlich **Typ:** Zeichenkette 6. **Name:** source_ci_type_end2 **Beschreibung:** CI-Typ von Ende 2 der Quellbeziehung. **Verwendung:** 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. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** target_link_type **Beschreibung:** Name der Zielbeziehung. **Verwendung:** Erforderlich **Typ:** Zeichenkette 3. **Name:** nameSpace **Beschreibung:** Der Namespace für den Link, der beim Ziel erstellt wird. **Verwendung:** Nicht erforderlich **Typ:** Zeichenkette 4. **Name:** mode **Beschreibung:** Der für den aktuellen Link erforderliche Aktualisierungstyp. **Verwendung:** Erforderlich **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

Elementname/-pfad	Beschreibung	Attribute
		Quellbeziehung **Verwendung:** Erforderlich **Typ:** Zeichenkette 6. **Name:** source_ci_type_end2 **Beschreibung:** CI-Typ von Ende 2 der Quellbeziehung **Verwendung:** 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. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** superclass **Beschreibung:** Name der Superklasse des CI-Typs von Ende 1. **Verwendung:** Nicht erforderlich **Typ:** Zeichenkette
target_ci_type_end2 (integration > targetrelations > link Oder integration > targetrelations > source_link_type_tree)	CI-Typ von Ende 2 der Zielbeziehung.	1. **Name:** name **Beschreibung:** Name des CI-Typs von Ende 2 der Zielbeziehung. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** superclass **Beschreibung:** Name der Superklasse des CI-Typs von Ende 2. **Verwendung:** Nicht erforderlich **Typ:** Zeichenkette

Schema der Zuordnungsergebnisse

Elementname/-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.	

Elementname/-pfad	Beschreibung	Attribute
Object (root > data > objects)	Beschreibt den Aktualisierungsvorgang für ein einzelnes Objekt und alle seine Attribute.	1. **Name:** name **Beschreibung:** Der Name des CI-Typs. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** mode **Beschreibung:** Der für den aktuellen CI-Typ erforderliche Aktualisierungstyp. **Verwendung:** 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: 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. **Verwendung:** 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	1. **Name:** name **Beschreibung:** Der Name des Felds. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** key **Beschreibung:** Gibt an, ob dieses Feld ein Schlüssel für das Objekt ist. **Verwendung:** Erforderlich

Elementname/-pfad	Beschreibung	Attribute
	Endes erscheint als Objekt (unter <objects>).	**Typ:** Boolescher Wert 3. **Name:** datatype **Beschreibung:** Der Typ des Felds. **Verwendung:** Erforderlich **Typ:** Zeichenkette 4. **Name:** length **Beschreibung:** Für Daten vom Typ "Zeichenkette/Zeichen" ist dies die Ganzzahlgröße des Zielattributs. **Verwendung:** 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. **Verwendung:** Erforderlich **Typ:** Zeichenkette 2. **Name:** targetParent **Beschreibung:** Der Typ des ersten Endes des Links (übergeordnetes Element). **Verwendung:** Erforderlich **Typ:** Zeichenkette 3. **Name:** targetChild **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. **Verwendung:** 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

Elementname/-pfad	Beschreibung	Attribute
		Beschreibung: Die mit diesem CI durchzuführende Operation. **Verwendung:** 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. **Verwendung:** Erforderlich **Typ:** Zeichenkette

Anpassung

In diesem Abschnitt werden einige der grundlegenden Verfahren für häufig durchgeführte Anpassungen von Push-Adaptern beschrieben.

Hinzufügen eines Attributs

1. Stellen Sie sicher, dass das Attribut im TQL-Ergebnis enthalten ist.
2. Fügen Sie die Attributzuordnung im korrekten CI-Zuordnungsabschnitt zur Zuordnungsdatei hinzu.
3. Stellen Sie sicher, dass der Datenempfänger für den Empfang des zusätzlichen Attributs in den Daten vorbereitet ist.

Entfernen eines Attributs

Um ein Attribut zu entfernen, müssen Sie es aus der Zuordnungsdatei entfernen. Außerdem sollten Sie das Attribut aus der TQL entfernen, wenn es nicht mehr im Ergebnis oder als bedingter Knoten verwendet wird.

Hinzufügen eines CI-Typs

1. Fügen Sie den CI-Typ zur TQL hinzu.
2. Stellen Sie sicher, dass der CI-Typ und die zugehörigen Attributdaten im TQL-Ergebnis angezeigt werden (verwenden Sie die Funktionen **Berechnen** und **Vorschau**).
3. Fügen Sie die Zuordnung des CI-Typs in der Zuordnungsdatei hinzu. Kopieren Sie die Zuordnungen eines anderen CI-Typs, um die Erstellung eines neuen CI-Typs zu beschleunigen.
4. Ändern Sie den Namen und die Attributzuordnungen der kopierten XML-Datei entsprechend dem neuen CI-Typ und seinen Attributen. Weitere Informationen zu den verfügbaren Zuordnungstypen finden Sie unter "Zuordnen der Dateireferenz" auf Seite 260.

Entfernen eines CI-Typs

1. Entfernen Sie den CI-Typ aus der TQL.
2. Entfernen Sie den Zuordnungsabschnitt für den CI-Typ in der Zuordnungsdatei.

Hinzufügen von Links

1. Stellen Sie sicher, dass die beiden Endpunkt-CIs in den Daten vorhanden sind.
2. Stellen Sie sicher, dass der Link, den Sie hinzufügen müssen, ein gültiger Link ist (überprüfen Sie den Link im CIT Manager).
3. Fügen Sie die Linkelemente in den Beziehungsabschnitt der XML-Zuordnungsdatei ein.

Entfernen von Links

1. Entfernen Sie den Linkabschnitt des Links, den Sie entfernen möchten, in der Zuordnungsdatei.
2. Wenn möglich, entfernen Sie den Link aus der TQL (sofern hierdurch nicht die Effizienz oder Funktion der TQL beeinträchtigt wird).

Kapitel 8: Entwickeln von allgemeinen Adaptern

Dieses Kapitel umfasst folgende Themen:

Instanzsynchronisation

Die Operationen für den allgemeinen Adapter des Typs "Push" und "Auffüllung" arbeiten mit Instanzdaten. Weitere Informationen über die Konzepte von *Instanz* und *Stamm* finden Sie unter "Instanzbasierter Auffüllungs-Flow" auf Seite 198 und "Umsetzen des Datenpush unter Verwendung des allgemeinen Adapters" unten.

Umsetzen des Datenpush unter Verwendung des allgemeinen Adapters

Datenpush verwendet das vorhandene Framework **Erweiterter generischer Push-Adapter** mit kleineren Änderungen im XML-Schema.

Hinweis: Der generische Adapter arbeitet in Instanzmodus (d. h. er funktioniert nicht mit einzelnen CI-Typen, aber mit Sammlungen von CIs, die unter einem Haupt-CI zusammengefasst sind). Weitere Informationen finden Sie unter "Instanzbasierter Auffüllungs-Flow" auf Seite 198.

Folgende XML-Schema-Änderungen waren erforderlich, um bidirektionale Zuordnungssemantiken anpassen zu können:

- Der Tag **<targetcis>** wurde umbenannt in **<target_entities>**.
- Der Tag **<source_instance_type>** wurde umbenannt in **<source_instance>**.
- Der Tag **<target_ci_type>** wurde umbenannt in **<target_entity>**.

- Der Tag **<for-each-source-ci>** wurde umbenannt in **<for-each-source-entity>**.
- Das Kopfzeilenattribut **versions** wurde umbenannt in **version**, und erfordert keine Dezimalziffer mehr.

Dieser Abschnitt enthält Informationen zu Übertragung von Daten mithilfe des Frameworks **GenericAdapter**:

Push – Übersicht

Der generische Adapter funktioniert auf Datenstrukturen, die für das TQL-Abfrageergebnis stehen. Jeder Adapter, der über das generische Adapter-Framework 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 das generische Adapter-Framework 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 Data Push im *HP Universal CMDB – Handbuch zur Datenflussverwaltung*.

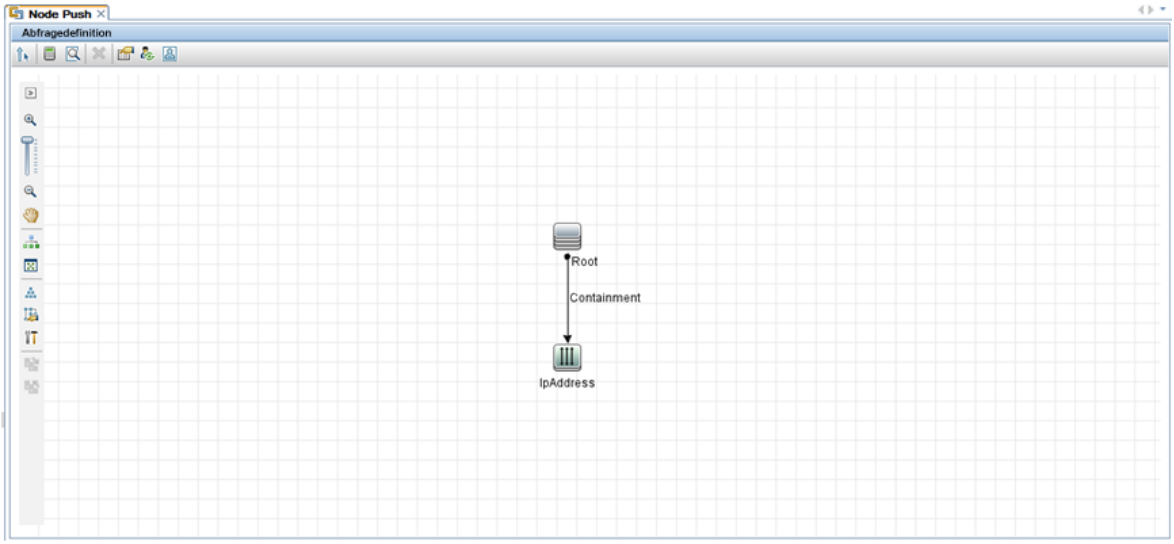
Die Entwicklung erweiterter Push-Adapter beinhaltet zwei grundlegende Schritte:

1. Implementieren der PushAdapterConnector-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:

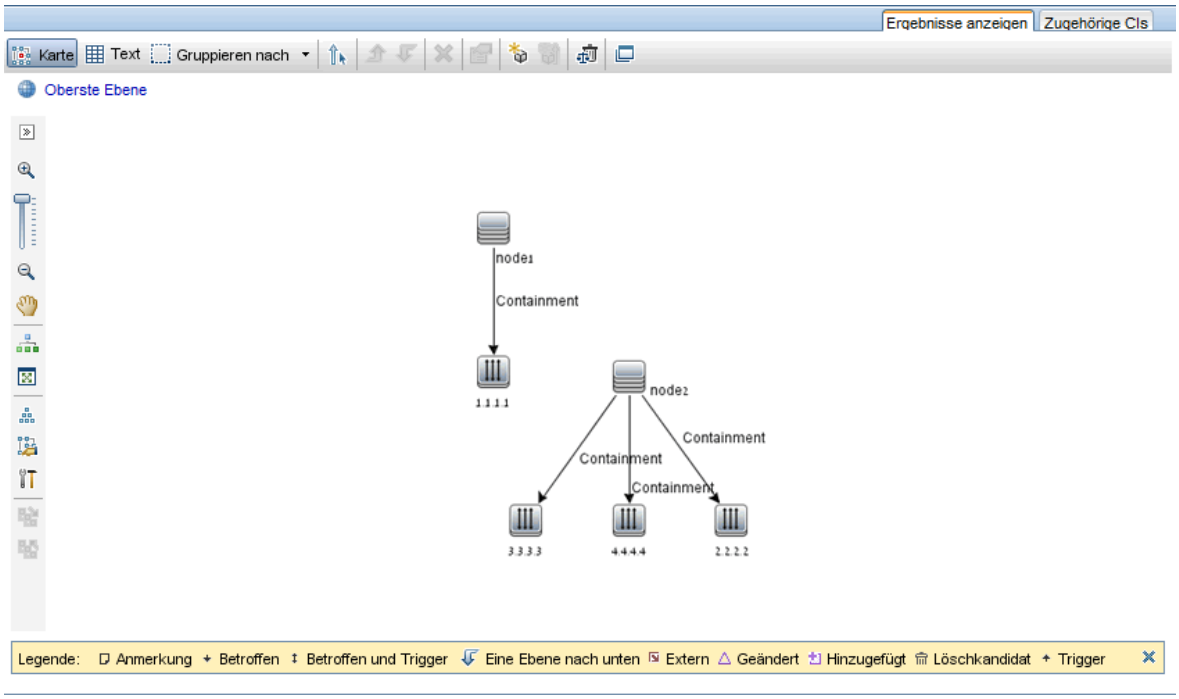


In der Zuordnungsdatei erstellen wir zwei Ziel-CI-Typen: **Computer** und **IP**. **Computer** weist eine Variable und zwei Attribute auf. **IP** besitzt ein Attribut.

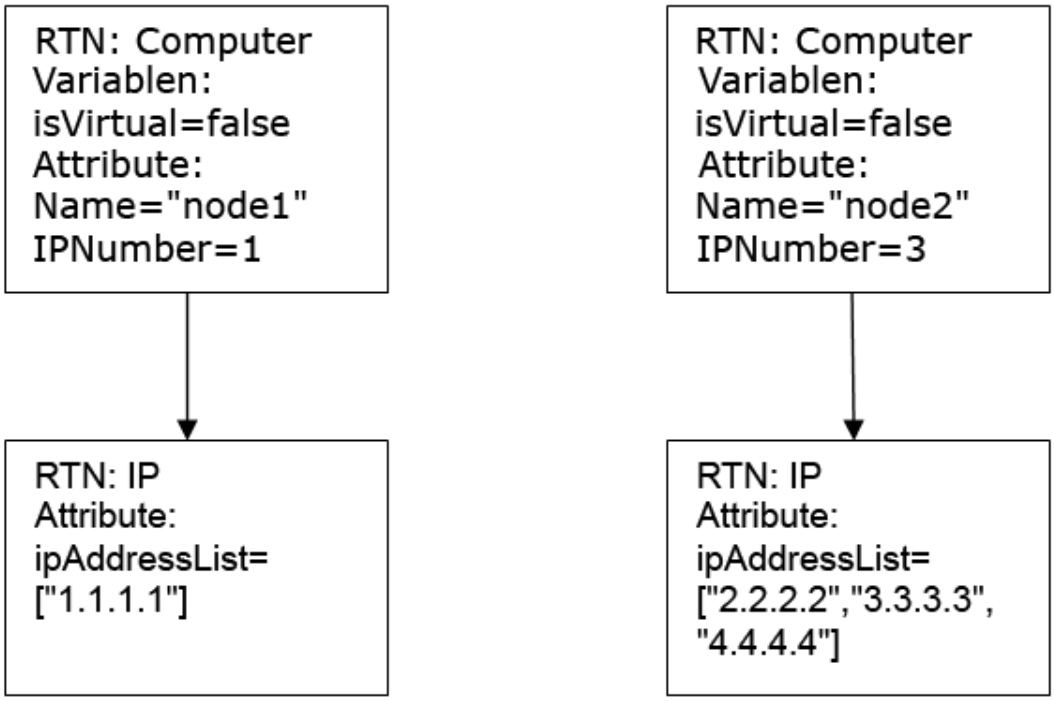
Die XML-Zuordnungsdatei lautet wie folgt:

```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../generic-adapter.xsd">
    <info>
        <source name="UCMDB" version="10.20" vendor="HP"/>
        <target name="PushProduct" version="9.3" vendor="HP"/>
    </info>

    <import>
        <scriptFile path="mappings.scripts.PushFunctions"/>
    </import>

    <target_entities>
        <source_instance query-name="Node Pusy" root-element-name="Root">
            <target_entity 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_entity name="IP">
                    <target_mapping name="IpAddressList" datatype="STRING_LIST" value="Root.IpAddress*.getAt('name')"/>
                </target_entity>
            </target_entity>
        </source_instance>
    </target_entities>

</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 PushAdapterConnector 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" unten.

Groovy Traveler

Der Zugriff auf die TQL-Abfrageergebnisse erfolgt folgendermaßen:

- **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_entity

Dieser Tag wird für die RTN-Erstellung verwendet.

Das Attribut **name** repräsentiert den Namen von **target_entity**: 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_entity**-Elemente werden nicht zum RTN hinzugefügt. In diesem Beispiel möchten wir keine **target_entity**-Instanz erstellen, für die das Attribut **root_iscandidatefordeletion** in UCMDB auf **true** gesetzt ist.

Das **target_entity**-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_entity**-Element als untergeordnetes Element eines anderen **target_entity**-Elements zu definieren. Dies geschieht wie folgt:

```
<target_entity name="Portfolio">
<variable name="vSerialNo" datatype="STRING" value="Root['global_id']"/>
<target_mapping name="CMDBId" datatype="STRING" value="globalId"/>
<target_entity name=Asset">
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
</target_entity>
</target_entity>
```

Der RTN **Portfolio** weist einen untergeordneten RTN mit der Bezeichnung **Asset** auf.

for-each-source-entity

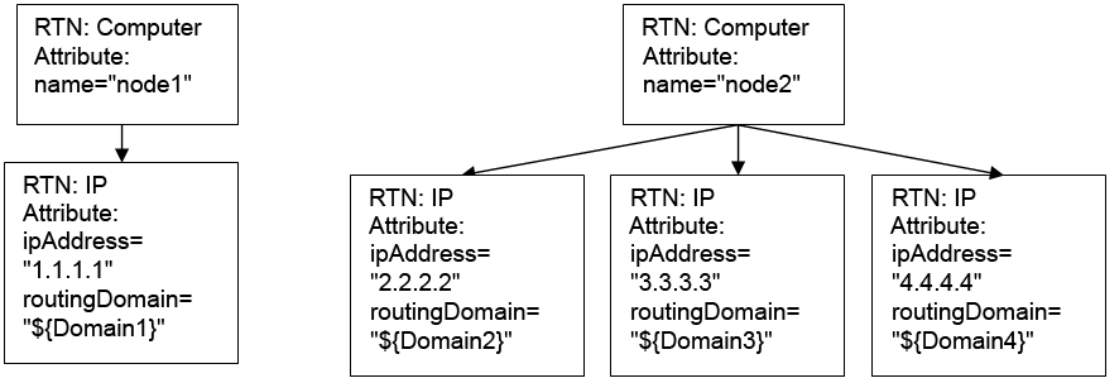
Dieser Tag listet die spezifischen CIs der Stamminstanz auf. Er weist die folgenden Felder auf:

- **source-entities=" "** – 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_entity name="Computer">
    <target_mapping name="name" datatype="STRING" value="Root['name']"/>
    <for-each-source-entity count-index="i" var-name="currIP" source-entities="Root.IpAddress">
        <target_entity name="IP">
            <target_mapping name="ipAddress" datatype="STRING" value="Root.IpAddress[i]['name']"/>
            <target_mapping name="routingDomain" datatype="STRING" value="currIP['routing_domain']"/>
        </target_entity>
</target_entity>
```

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 PushAdapterConnector-Schnittstelle

Die Implementierung sollte die folgenden grundlegenden Schritte unterstützen:

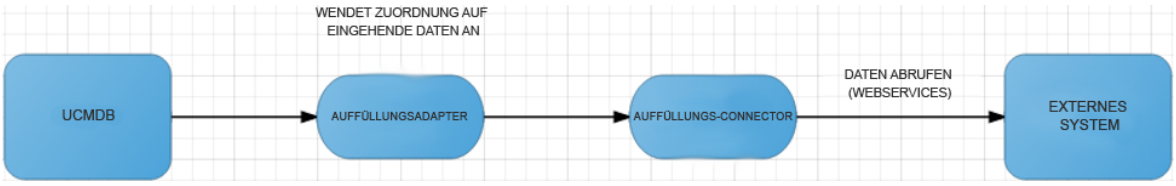
```
public class PushExampleAdapter implements PushAdapterConnector

{

public UpdateResult pushTreeNodes(PushConnectorInput input) 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 the status of data that was
actually pushed, detailed status reports on failed IDs, and actions actually
performed on successful ids.
// 2. handle the data:
// a. handle data to add. Can be retrieved by:
input.getResultTreeNodes.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.
}
}
```

Umsetzen des Datenauffüllung unter Verwendung des allgemeinen Adapters

Dieser Abschnitt umfasst die folgenden Themen:

Architektur des Auffüllungs-Framework



Der Mechanismus ist ähnlich wie der des Push-Adapter-Frameworks, d. h. dass ein Benutzer dieses Auffüllungs-Adapter-Frameworks eine Zuordnungsdatei und eine Connector-Implementierung bereitstellen, und diese in einem UCMDB-Adapter-Package zusammenfassen muss.

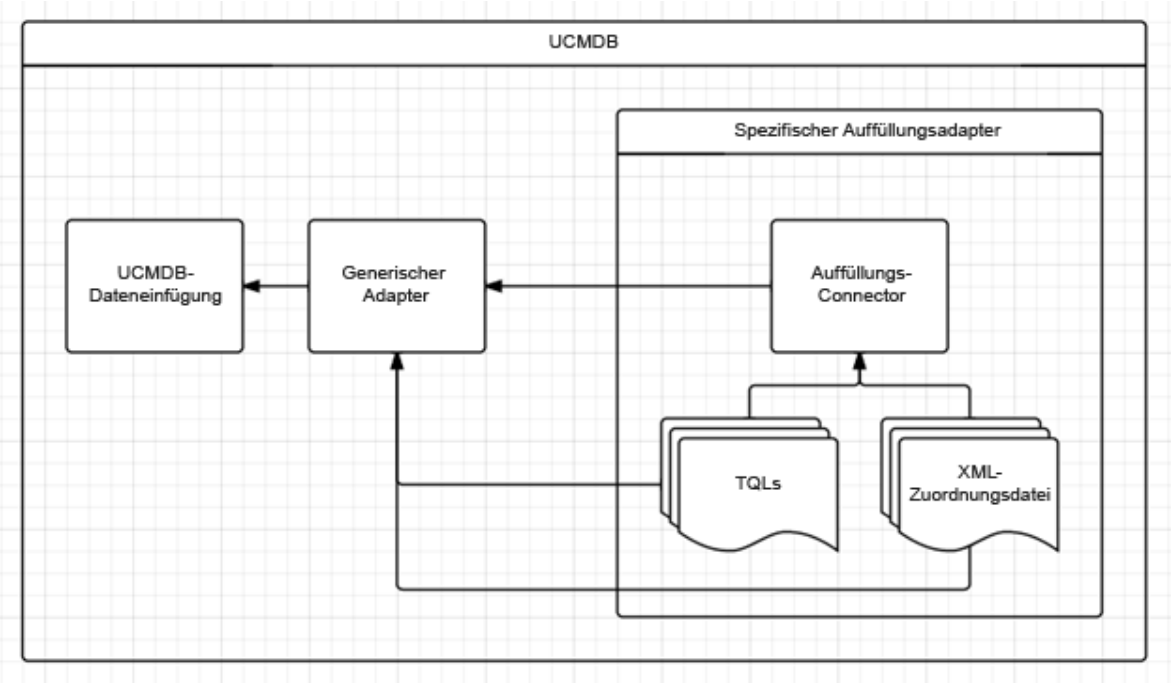
Der Vorgangsfluss enthält die folgenden Schritte

1. Der UCMDB-Benutzer löst die Auffüllungsvorgang über die UI aus.
2. Der Befehl an den Auffüllungsadapter gesendet.
3. Der Auffüllungsadapter ruft den Auffüllungs-Connector an, und empfängt die Daten in Chunks.
4. Der Auffüllungsadapter wendet die definierte Zuordnung von Daten aus jedem Chunk an, und leitet diese an den UCMDB-Server weiter.

An der Auffüllung beteiligte Hauptelemente

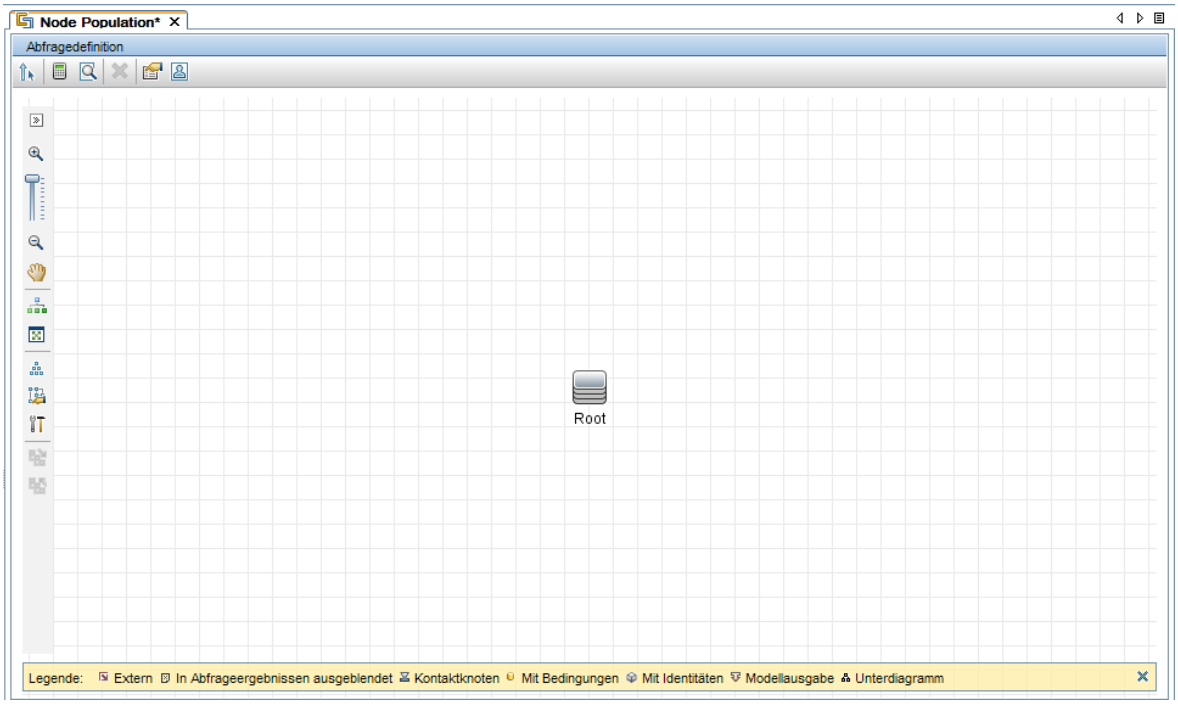
Die an der Auffüllung beteiligten Hauptelemente sind:

- TQL-Abfragen, die die Daten angeben, die in UCMDB aufgefüllt werden
- XML-Zuordnungsdateien, die angeben wie die vom Connector zurückgegebenen Daten UCMDB zugeordnet werden
- erforderliche Daten
- der Auffüllungs-Connector, der für das Abrufen von externen Systemdaten verantwortlich ist und sie wieder an den allgemeinen UCMDB-Adapter zurückgibt



TQL-Abfrage für das Auffüllen

Die Rolle einer TQL-Abfrage für das Auffüllen dient zur Angabe der Daten, die in UCMDB aufgefüllt werden. Beispielsweise wird die TQL in der folgenden Abbildung verwendet, um Knoteninstanzen in UCMDB einzubinden.



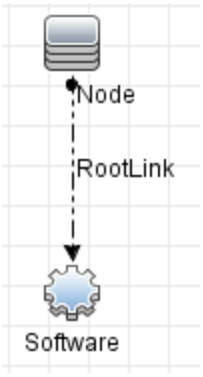
Der Auffüllungs-Connector ist zum Verständnis der TQL-Abfrage für das Auffüllen und die Bereitstellung der erforderlichen Daten aus dem externen System verantwortlich.

Zuordnungsdateien für das Auffüllen

Die XML-Zuordnungsdateien haben den gleichen Zweck wie für Push-Operationen, nur in umgekehrter Richtung. Diese Zuordnungsdateien beschreiben, wie die durch den Connector zurückgegebenen Daten den UCMDB-Daten zugeordnet werden.

Die hier bereitgestellten Informationen sind relevant für die Auffüllungs-Zuordnung, und noch nicht für den erweiterten generischen Push-Adapter aufgeführt.

Unten sehen Sie ein Beispiel für eine Zuordnung für UCMDB-Knoten und aktive Software. Die erste Abbildung zeigt die Auffüllungs-TQL-Abfrage für Knoten und Software. Die zweite Abbildung zeigt die Auffüllungs-Zuordnung für die Knoten und Software.




```
<target_entities>
    <!--The query name must match the one selected in the UI-->
    <source_instance query-name="Nodes And Software Population" root-element-name="PC">
        <!-- need to match case in UCMDB TQL -->
        <target_entity name="RootLink">
            <target_mapping name="name" datatype="STRING" value="PC['name'] + 'has' + PC.Programs[0]['name']"/>
        </target_entity>
        <target_entity name="Node" type="Util.getNodeType(PC)">
            <target_mapping name="name" datatype="STRING" value="PC['name']"/>
            <target_mapping name="description" datatype="STRING" value="PC['description']"/>
        </target_entity>
        <target_entity name="Software">
            <target_mapping name="name" datatype="STRING" value="PC.Programs[0]['name']"/>
            <target_mapping name="root_container_name" datatype="STRING" value="PC['name']"/>
            <target_mapping name="product_name" datatype="STRING" value="'vmware_hypervisor'"/>
        </target_entity>
    </source_instance>
</target_entities>
```

In diesem Auffüllungsjob werden Daten aus einem externen System in Form des ResultTreeNode (RTN) PC abgerufen. Die ResultTreeNode-API wurde durch den erweiterten generischen Push-Adapter eingeführt und ist in der Datei **push-interfaces.jar** im Ordner **lib** auf dem UCMDB-Server gespeichert. Weitere Informationen finden Sie unter "Umsetzen des Datenpush unter Verwendung des allgemeinen Adapters" auf Seite 276.

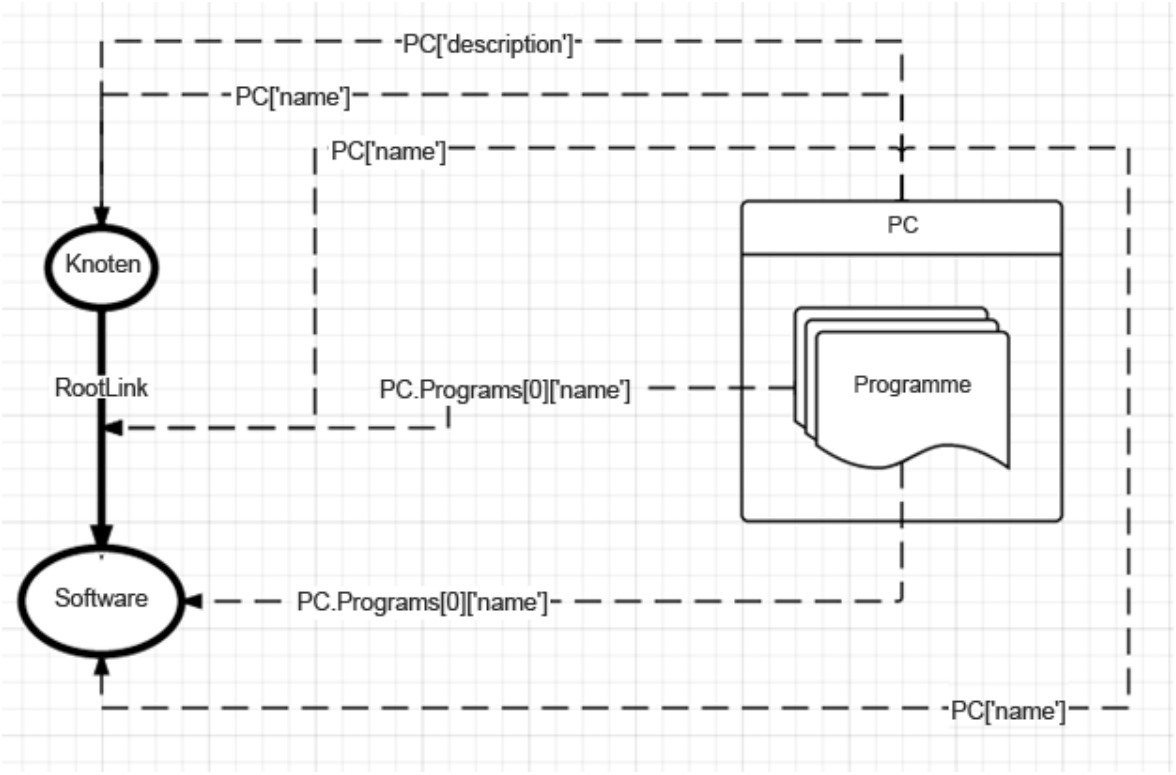
Der PC-RTN enthält allgemeine Knoten-Informationen in Form von Attributen sowie eine eingebettete Programm-Entität, die Software-Typ-Entitäten mit den relevanten Attributen enthält.

Eine PC-Instanz wird 3 Entitäten in UCMDB zugeordnet:

- Einem Knoten-CI
- Einer Aktive Software-CI
- Einer Verbundlink-CI

Weitere Informationen über das Format der PC-Instanz finden Sie unter "Ausgabe der Aufüllungsanforderung" auf Seite 296

Die Art und Weise, wie Connector-Daten UCMDB-Daten zugeordnet werden, ist in der folgenden Abbildung dargestellt:



Analysieren wir nun die Schlüsselzeilen:

```
<!--The query name must match the one selected in the UI-->
<source_instance query-name="Nodes And Software Population" root-element-name="PC">
```

Die Definition **source_instance** gibt an, dass Entitäten in UCMDB einfügen. Die UCMDB-Topologie, welche diese Entitäten guppiert, wird durch die Auffüllungs-TQL-Abfrage für Knoten und Software definiert. Darüber hinaus ist die durch den Connector zurückgegebene Datenstruktur, die zum Erstellen der UCMDB-Daten verwendet werden, eine **ResultTreeNode**-Struktur namens **PC**.

```
<target_entity name="RootLink">
    <target_mapping name="name" datatype="STRING" value="PC['name'] + 'has' + PC.Programs[0]['name']"/>
</target_entity>
```

Der Tag **target_entity** gibt an, dass an dieser Stelle eine neue UCMDB-Entität beginnt, und diese Entität dem Element **RootLink** in der Auffüllungs-TQL-Abfrage für Knoten und Software entspricht. Diese Angabe umfasst auch den UCMDB-CI-Typ der neuen Entität.

Die Entität **RootLink**, die erstellt wird, wird ein Attribut **name** enthalten, dessen Wert etwa wie folgt lauten wird: **Computer_22 has MySQL Server**.

In diesem Beispiel für eine Zuordnung wird eine manuelle Link-Auffüllung verwendet. Es wird jedoch der automatische Ansatz empfohlen, beschrieben unter "Automatische Link-Auffüllung" auf der nächsten Seite.

Das Auffüllungsattribut type

```
<target_entity name="Node" type="Util.getNodeType(PC)">
```

Beachten Sie, dass die Entität **Knoten** über ein Attribut **type** verfügt. Dieser **type** zeigt den genauen CI-Typ an, den diese Entität in UCMDB haben wird. Das **type**-Attribut ist nicht obligatorisch, da die Standardeinstellung der Entität aus dem TQL-Element abgerufen wird, auf welches sie sich bezieht (in diesem Fall den Knoten). Um jedoch mehrere Instanzen des UCMDB-Knoten-CI-Typs zurückzugeben, und einige dieser Windows-Instanzen sind während andere Unix-Instanzen sind, kann mithilfe des Attributs **type** der genaue UCMDB-Erstellungstyp festgelegt werden. In diesem Fall erstellen wir eine Funktion **getNodeType** innerhalb der Skriptdatei **Util**, welche als Eingabe die PC-Struktur empfängt und die gültige UCMDB-CI-Typ-ID ("unix" für Unix und "nt" für Windows) zurückgibt.

Hinweis: Das Attribut **target_entity type** ist nur verfügbar im Kontext eines Auffüllungsflusses; sein Wert muss zudem ein gültiger Groovy-Ausdruck sein.

Die Erstellung der Software-Entität erfolgt auf die gleiche Weise.

Automatische Link-Auffüllung

Im Zuordnungsbeispiel im Kapitel "Zuordnungsdateien für das Auffüllen" wurde dargestellt, was ist erforderlich ist, um einen aufgefüllten Link explizit zuzuordnen. Eine zugeordnete Zielentität (wie z. B. die unten abgebildete) muss für jedes einzelne TQL-Link-Element vorhanden sein.

```
<target_entity name="RootLink">
    <target_mapping name="name" datatype="STRING" value="PC['name'] + 'has' + PC.Programs[0]['name']"/>
</target_entity>
```

Dank des GenericAdapter-Verfahrens zur automatischen Link-Auffüllung müssen die TQL-Link-Elemente nicht mehr mit den Zuordnungsabschnitten verknüpft werden, wie oben gezeigt. Das Framework generiert eine Link-CI-Instanz des Typs, der in der TQL mit leeren Eigenschaften angegebenen ist. Dieser Vorgang wird für alle Links in der Auffüllungs-TQL-Abfrage ausgeführt.

Die erstellten Beispiel-Zuordnungsergebnisse in einer Link-CI des Typs Komposition mit einem Link, beenden (end1 und end2) die CI-Instanzen des Knotens und der aktiven Software.

Sie sollten die manuelle Link-Auffüllung verwenden, wenn der aufzufüllende Link folgendes erfordert:

- Einen dynamischen Link-Typ (unter Verwendung des Attributs **type**)
- Link-Eigenschaften

Manuelle Link-Auffüllung

Der generische Adapter erreicht die Auffüllung von Links durch die Definition (Zuordnung) der drei Entitäten, die für einen Link erforderlich sind:

- Die Entität **Link**
- Die End 1-Entität des Links
- Die End 2-Entität des Links

Analysieren wir nun das Zuordnungsbeispiel aus dem Kapitel "Zuordnungsdateien für das Auffüllen" auf Seite 287. In diesem Fall füllen wir drei Entitätstypen in UCMDB auf: Knoten, Aktive Software sowie den

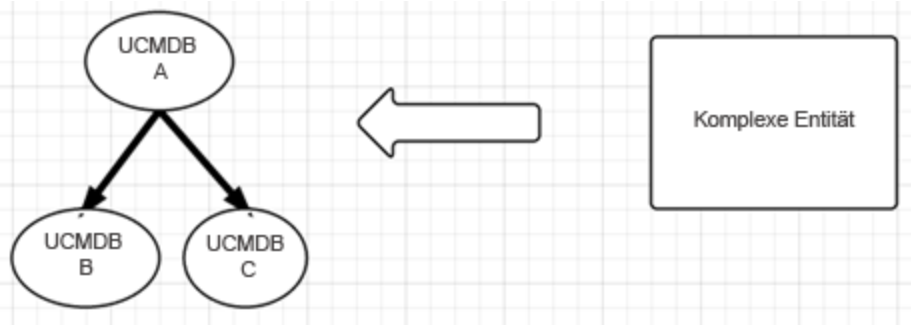
Verbundlink zwischen diesen beiden Entitäten. Da wir einen Link auffüllen möchten (den Verbundlink namens **RootLink**), müssen wir auch die beiden Link-Enden zuordnen. Der TQL-Abfrage ist zu entnehmen, dass das Entitäten Node (end 1) und Software (end 2) eine Zuordnung erfordern. Das Framework **GenericAdapter** liest die Link-Struktur, indem der Elementsname und die Definition der erstellten Entität in der TQL-Abfrage ausgelesen werden. Da die Auffüllung auch Instanzen von Knoten und aktiver Software zurückgeben muss, ist die erforderliche Zuordnung des Endes bereits vorhanden.

Typen der Link-Auffüllung

Es gibt zwei Typen von Link-Auffüllungs-Situationen:

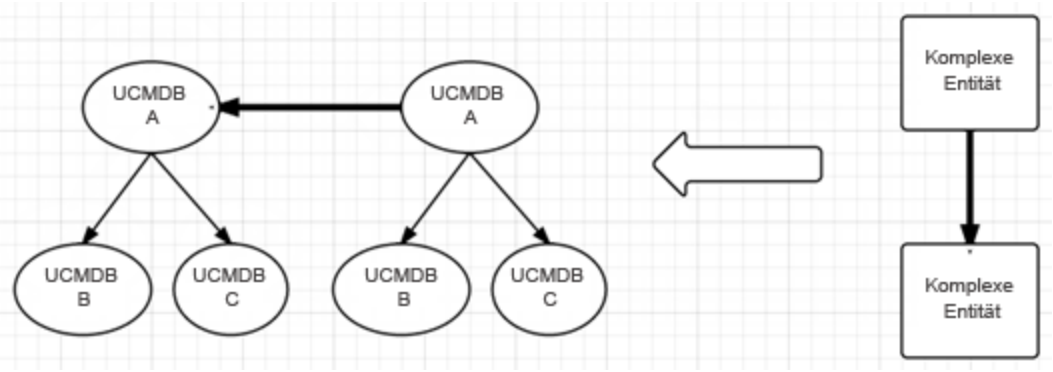
- Auflösung einer komplexen externen Entität in mehrfach verknüpften UCMDB-Entitäten

  In diesem Fall wird eine komplexe externe Entität, wie z. B. PC, in die UCMDB-Typen **Knoten** und **Aktive Software** konvertiert, welche durch eine Verbundbeziehung verknüpft werden müssen. Dieser Typ von Link ist nur im UCMDB-Kontext vorhanden.



- Links zwischen komplexen externen Entitäten

  In diesem Fall müssen wir eine Verbindung zwischen zwei komplexen externen Entitäten, wie zum Beispiel PCs, erstellen.



Auffüllungs-Connector

Der Auffüllungs-Connector ist für das Abrufen von externen Systemdaten zuständig. Diese Daten werden im festgelegten API-Format an den generischen Adapter weitergeleitet (**ResultTreeNode**), anschließend den UCMDB-Datenstrukturen zugeordnet und mittels des Dateneingabeprozesses in UCMDB eingefügt.

Ähnlich wie der Push-Connector kann der Auffüllungs-Connector sowohl in Java als auch in Groovy implementiert werden, und muss den Auffüllungs-Connector Java Interface implementiert haben (in der Abbildung unten dargestellt).

Um den Auffüllungs-Connector zu konfigurieren, fügen Sie die folgende Zeile in die Adapterkonfigurations-XML-Datei ein:

```
<adapter-settings>
    <adapter-setting name="PopulationConnector.class.name">com.hp.ucmdb.connector.dummy.DummyPopulationConnector</adapter-setting>
```

```
public interface PopulationAdapterConnector extends GenericConnector, DynamicMappingConnector {

    /**
     * Executes a population request and provides the entities that will be populated in the format of
     * {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode}
     * Used for both population and federation with the flow type flag which is present in the
     * {@link com.hp.ucmdb.adapters.population.connector.PopulationConnectorInput}
     *
     * @param input population request
     * @return a population response
     * @throws DataAccessException
     */
    PopulationConnectorOutput populate(PopulationConnectorInput input) throws DataAccessException;

    /**
     * Returns the collection of queries the current connector supports for population
     * Note: this method is independent to the adapter life cycle (i.e. start/shutdown methods) and must work at all times.
     * <p/>
     * The method also supports return of queries with their folders path: E.g.
     * "Folder/Secondary Folder/Query Name 1"
     * "Folder/Secondary Folder/Query Name 2"
     *
     * @param env the adapter environment
     * @return a collection of the supported queries
     */
    Collection<String> getPopulationQueries(DataAdapterEnvironment env);

    /**
     * Returns the collection of queries the current connector supports for federation
     * Note: this method is independent to the adapter life cycle (i.e. start/shutdown methods) and must work at all times.
     * <p/>
     * The method also supports return of queries with their folders path: E.g.
     * "Folder/Secondary Folder/Query Name 1"
     * "Folder/Secondary Folder/Query Name 2"
     *
     * @param env the adapter environment
     * @return a collection of the supported queries
     */
    Collection<String> getFederationQueries(DataAdapterEnvironment env);

    /**
     * Update target id (global id) for each source object.
     *
     * @param idMapping mapping between source id (external id) and target id (string)
     */
    void updateGlobalIDsFromTarget(FCmdbExtenalToTargetIdMappingSet idMapping);

    /**
     * This methods reports population queries resources used by the adapter.<br>
     * This allows editing these resources directly from the Integration Studio.
     *
     * @param input  the requested information
     * @param output the returned resources
     * @see com.hp.ucmdb.federationspi.adapter.resource.PushQueriesResourceLocator
     */
    void locatePopulationQueriesResources(DataAdapterEnvironment env, LocatePopulationQueriesResourcesInput input,
                                          LocatePopulationQueriesResourcesOutput output);

    /**
     * Returns the collection of classes the current adapter supports for query.<br>
     * Notes:<br>
     * 1. This method is independent of the adapter life cycle (i.e. start/shutdown methods).<br>
     * 2. If adapter configuration (xml) defines supported classes, this method doesn't need to be implemented (return null).<br>
     *
     * @param env the Adapter env
     * @return collection of the supported classes
     * @throws DataAccessException in case of an error
     */
    Collection<SupportedClassConfig> getSupportedClasses(DataAdapterEnvironment env);
}
```

Die erste Methode, `populate`, ist die Haupt-Connector-Methode für das Abrufen von Daten aus dem externen System. Diese Methode empfängt als Eingabe eine Auffüllungs-TQL-Abfrage und gibt die Ergebnisse im generischen *ResultTreeNode*-Format zurück. Weitere Informationen finden Sie unter "Umsetzen des Datenpush unter Verwendung des allgemeinen Adapters" auf Seite 276. Neben den wichtigsten Business-Daten gibt der Connector auch Status- und Chunk-Informationen zurück.

Die zweite Methode, `getSupportedQueries`, zeigt die TQLs an, die durch den Auffüllungs-Connector unterstützt werden.

Die dritte und vierte Methoden werden eher bei erweiterten Anwendungsfälle eingesetzt: die Rückgabe der IDs der Auffüllungsdaten und die Lokalisierung der relevanten Auffüllungsressourcen innerhalb des Adapters für eine bestimmte Abfrage. Weitere Informationen zu diesen APIs finden Sie in der Datei **push-interfaces.jar**.

Eingabe der Auffüllungsanforderung

Eine Auffüllungsanforderung wird durch ein **QueryDefinition**-Objekt definiert, das die UCMDB-Auffüllungsabfrage beschreibt. Der Auffüllungs-Connector ist für das Lesen dieses Abfragesobjekts und dessen Übersetzung in die Abfragesprache des externen Systems zuständig.

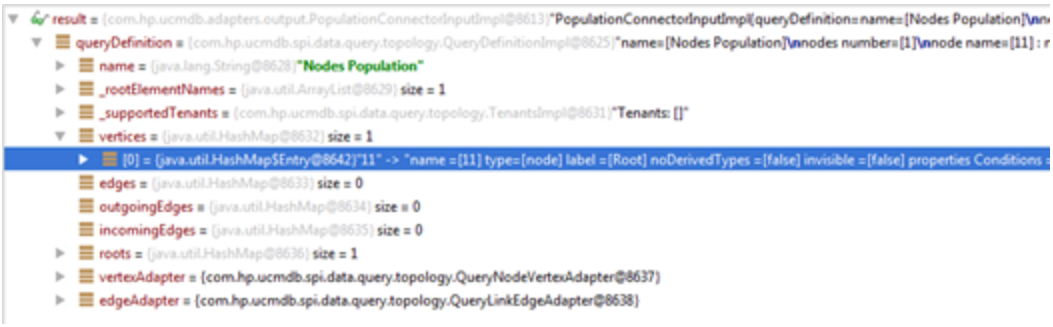
```
 * @author Sergiu Indrie
 * @since 10.20
 */
public interface PopulationConnectorInput {

    QueryDefinition getQueryDefinition();

    /**
     * Indicates the required {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode} structure
     * that the population result must return.
     *
     * For the target mapping <target_mapping name="lMaxMemory" datatype="LONG" value="Root.VMWare_Host_Resource['vm_memory_limit']"/>
     * the resulting tree node should have the following structure: VMWare_Host_Resource will be a child of Root and vm_memory_limit will
     * be an attribute of VMWare_Host_Resource
     *
     * @return a map containing simplified result trees
     * @see PopulationConnectorOutput#getResultTreeNodes()
     */
    Map<String, ResultTreeNodeStructure> getResultTreeNodeStructure();

    /**
     * Returns the flow type of the operation.
     * Can be POPULATION or FEDERATION
     *
     * @return the flow type
     */
    FederationTopologyAdapterInput.FlowType getFlowType();

    /**
     * Returns the date from the last sync
     *
     * @return the date
     */
    Date getFromDate();
```

Neben dem Objekt **QueryDefinition** gibt es folgende Objekte:

- **getResultTreeNodeStructure** – gibt die erforderliche Struktur an, die das Auffüllungsergebnis zurückgeben muss.
- **getFlowType** – wird verwendet, um festzustellen, ob die Anforderung an den Connector vom Typ POPULATION oder FEDERATION ist.

getFromDate – gibt das Datum der letzten Synchronisierung an. Wenn das Datum Null ist, wird FULL POPULATION ausgeführt, andernfalls wird Diff POPULATION ausgeführt wird (wenn der Flow-Typ FEDERATION ist, wird die getFromDate-Methode immer Null zurückgegeben).

In der folgenden Abbildung wird ein Beispiel für eine Auffüllungsanforderung dargestellt:



In diesem Beispiel enthält die Anforderung die Abfrage **Nodes Population**. Die Abfrage enthält nur ein TQL-Element vom Typ **Node**.

ResultTreeNodeStructure

Um Ihren **PopulationAdapterConnector** zu implementieren, müssen Sie die UCMDB-Auffüllungs-TQL lesen, verstehen, was UCMDB abruft, und die Ergebnisse über externe System-Entitäten bereitstellen. So kann UCMDB beispielsweise alle Knoten abrufen, die mit Geschäftsservice-Instanzen in Ihrem externen System verknüpft sind, und es könnte sein, dass das Equivalent für einen Computer im externen System **PC** ist, und mit einer **Service**-Entität verknüpft ist. Daher muss Ihr Auffüllungs-Connector Instanzen von **PC** zurückgeben, die mit Instanzen von **Service** verknüpft sind. In diesem Fall wird die Zuordnung in etwa wie folgt aussehen:

```
<target_entities>
    <source_instance query-name="Nodes And BS Population" root-element-name="PC">
        <!-- Node -->
        <target_entity name="Root">
            <target_mapping name="name" datatype="STRING" value="PC['name']"/>
        </target_entity>

        <!-- Business Service -->
        <target_entity name="BusinessService">
            <target_mapping name="name" datatype="STRING" value="PC.Service['name']"/>
            <target_mapping name="description" datatype="STRING" value="PC.Service['description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Um die **PC**-Instanzen zurückzugeben, die mit **Service**-Instanzen verknüft sind, geben wir in diesem Fall eine **PC**-RTN zurück, die **Service** als einen untergeordneten Knoten enthält. Wir hätten jedoch die Zuordnung auch im Format eines **Service**-RTN mit einem untergeordneten **PC** wie Folgendem erstellen können (wodurch die Zuordnung ungültig geworden wäre):

```
<target_entities>
    <source_instance query-name="Nodes And BS Population" root-element-name="Service">
        <!-- Node -->
        <target_entity name="Root">
            <target_mapping name="name" datatype="STRING" value="Service.PC['name']"/>
        </target_entity>

        <!-- Business Service -->
        <target_entity name="BusinessService">
            <target_mapping name="name" datatype="STRING" value="Service['name']"/>
            <target_mapping name="description" datatype="STRING" value="Service['description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Daher muss zur Unterstützung der Entwicklung des Auffüllungs-Connectors die durch den generischen Adapter gesendete Auffüllungs-Anforderung auch die RTN-Struktur der in der Zuordnungsdatei verwendeten Daten enthalten. Diese gibt dem implementierenden Connector das erforderliche Format des zurückgegebenen RTNs an.

Im ersten Fall lautet die **ResultTreeNodeStructure** wie folgt:

PC
- name

  Service
  - name
  - description

Im zweiten Fall lautet die **ResultTreeNodeStructure** wie folgt:

Service
- name
- description

  PC
  - name

Ausgabe der Auffüllungsanforderung

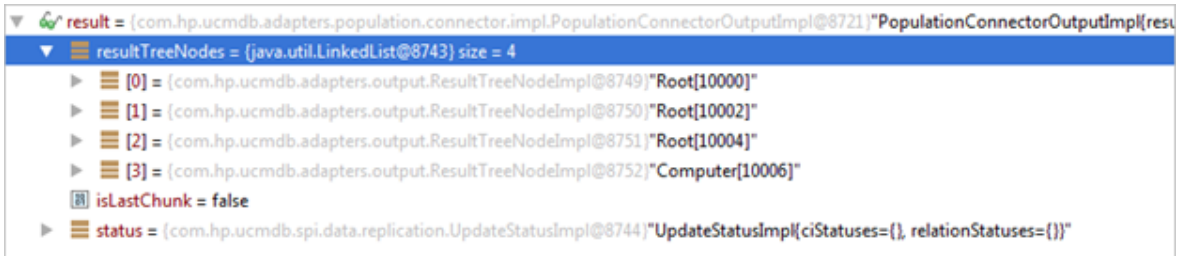
Nach der Verarbeitung einer Auffüllungsanforderung muss der Auffüllungs-Connector ein `PopulationConnectorOutput` zurückgeben.

```
public interface PopulationConnectorOutput {

    /**
     * The result trees representing the external entities in {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode} format.
     *
     * @return a list of result trees or an empty list
     */
    List<ResultTreeNode> getResultTreeNodes();

    /**
     * Adds a result tree to the population output object.
     *
     * @param resultTreeNode the result tree to add
     */
    void addResultTreeNode(ResultTreeNode resultTreeNode);

    /**
     * This method indicates if the result received is the last chunk of data.
     *
     * @return true if this is the last chunk, false otherwise.
     */
    boolean isLastChunk();

    /**
     * Set whether this result object is the last chunk or not.
     */
    void setLastChunk(boolean isLastChunk);

    /**
     * Returns the status information about the population result.
     */
    UpdateStatus getStatus();

    /**
     * Set the population result status.
     */
    void setStatus(UpdateStatus status);

}
```

Dieses Ausgabeobjekt enthält folgende Daten:

- Die abgefragten Daten im `ResultTreeNode`-Format.
- Statusinformationen (erforderlich im Fall eines Fehlers)
- Chunk-Informationen

In der folgenden Abbildung ist ein Beispiel für eine Auffüllungsantwort dargestellt:



In der Antwort oben hat der Connector vier Dateninstanzen (entsprechend dem UCMDB-Node), einen leeren Status (der einen erfolgreichen Ablauf anzeigt) sowie ein Kennzeichen zurückgegeben, dass anzeigt, dass es sich hierbei nicht um den letzten Chunk handelt.

Modi des Auffüllungsadapters

Das UCMDB Adapter Framework ermöglicht zwei Typen von Auffüllungsadaptern:

- Standard-Auffüllungsadapter
  - Bei diesem Adapter fehlt der Tag **<changes-source/>** in der Adapter-XML-Datei
  - Dieser Adapter gibt immer vollständige Abfragedaten aus externen System zurück. In diesem Fall ist das UCMDB Probe Framework zuständig für die Bestimmung des Unterschieds zwischen zwei aufeinanderfolgenden Läufen. Das Probe Framework erreicht dies durch den Vergleich des vorherigen Ergebnisses für die vorliegende Abfrage mit dem aktuellen Ergebnis, sowie durch die Berechnung der Unterschiede. Die vollständige Auffüllung wird erreicht, indem das aktuelle Abfrageergebnis nicht verglichen wird und als finales Ergebnis behandelt wird. Dieser Workflow setzt voraus, dass die Auffüllungsdaten nicht durch einen "Von Datum"-Filter gefiltert sind, da hierdurch ein Datenvergleich keine Bedeutung hätte.
- **changes-source**-Auffüllungsadapter
  - Dieser Adapter wird durch die Verwendung des Tags **<changes-source/>** in der Adapter-XML-Datei konfiguriert:

```
<adapterInfo>
    <adapter-capabilities>
        <support-federated-query/>
        <support-replicatioin-data>
            <source>
                <changes-source/>
                <push-back-ids/>
                <re-populate/>
            </source>
            <target/>
        </support-replicatioin-data>
```

  - Die changes-source-Adapter ist zuständig für die Berechnung des Unterschiedes zwischen zwei aufeinanderfolgenden Läufen.

Löschen von CIs bei Verwendung eines changes-source-Adapters

Wenn Sie einen changes-source-Auffüllungsadapter verwenden, ist Ihr Adapter für die explizite Löschung von CIs zuständig. Dies erfolgt mithilfe des Zuordnungsdatei-XML-Attributs **is-deleted**, das einen gültigen Groovy-Ausdruck akzeptiert.

Beispiel: In der nachfolgenden Zuordnungsdatei gibt der Auffüllungs-Connector **Service**-Instanzen zurück. Obwohl die Instanzen immer noch gültig sind, wurden einige CIs, die Teil dieser **Service**-Instanzen waren, gelöscht. Um das Löschen dieser CIs anzuzeigen, müssen Sie das Attribut **is-deleted** der Zuordnung **BusinessService** verwenden.

```
<target_entities>
    <source_instance query-name="Nodes And BS Population" root-element-name="Service">
        <!-- Node -->
        <target_entity name="Root">
            <target_mapping name="name" datatype="STRING" value="Service.PC['name']"/>
        </target_entity>

        <!-- Business Service -->
        <target_entity name="BusinessService" is-deleted="Functions.isOlderThanThreeMonths()">
            <target_mapping name="name" datatype="STRING" value="Service['name']"/>
            <target_mapping name="description" datatype="STRING" value="Service['description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Explizite Zuordnung externer IDs

Kann es Situationen geben, in denen die Auffüllungsdaten (CIs) eine durch den Connector/Adapter überwachte **ExternalId** haben müssen. Befolgen Sie hierzu das folgende Zuordnungskonstrukt:

```
<target_entities>
    <!--The query name must match the one selected in the UI-->
    <source_instance query-name="Node with ID" root-element-name="Computer">
        <!-- need to match case in UCMDB TQL -->
        <target_entity name="Root">
            <!--This is how the RTN External ID is set-->
            <variable name="external_id_obj" datatype="STRING" value="Computer['external_id_obj']"/>
            <!--RTN Attributes-->
            <target_mapping name="name" datatype="STRING" value="Computer.Asset[0]['name']"/>
            <target_mapping name="description" datatype="STRING" value="Computer['name']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

In diesem Fall erhält das Stamm-CI eine ExternalId, die auf Connector-Ebene erstellt und auf dem Computer gespeichert wurde ['external_id_obj']. Der Erstellung der ExternalId kann auch auf Zuordnungsebene erfolgen, unter Verwendung eines Groovy-Skripts.

Hinweis: Das Verfahren der expliziten Erstellung einer externen ID überschreibt das **type**-Attribut `target_entity`. Daher wird beim Erstellen einer externen ID mit der Zuordnungsskriptdatei oder innerhalb des Connectors das **type**-Attribut ignoriert, und als endgültiger UCMDB-Typ des aufgefüllten CIs wird der UCMDB-Typ verwendet, der im Objekt **ExternalId** festgelegt ist.

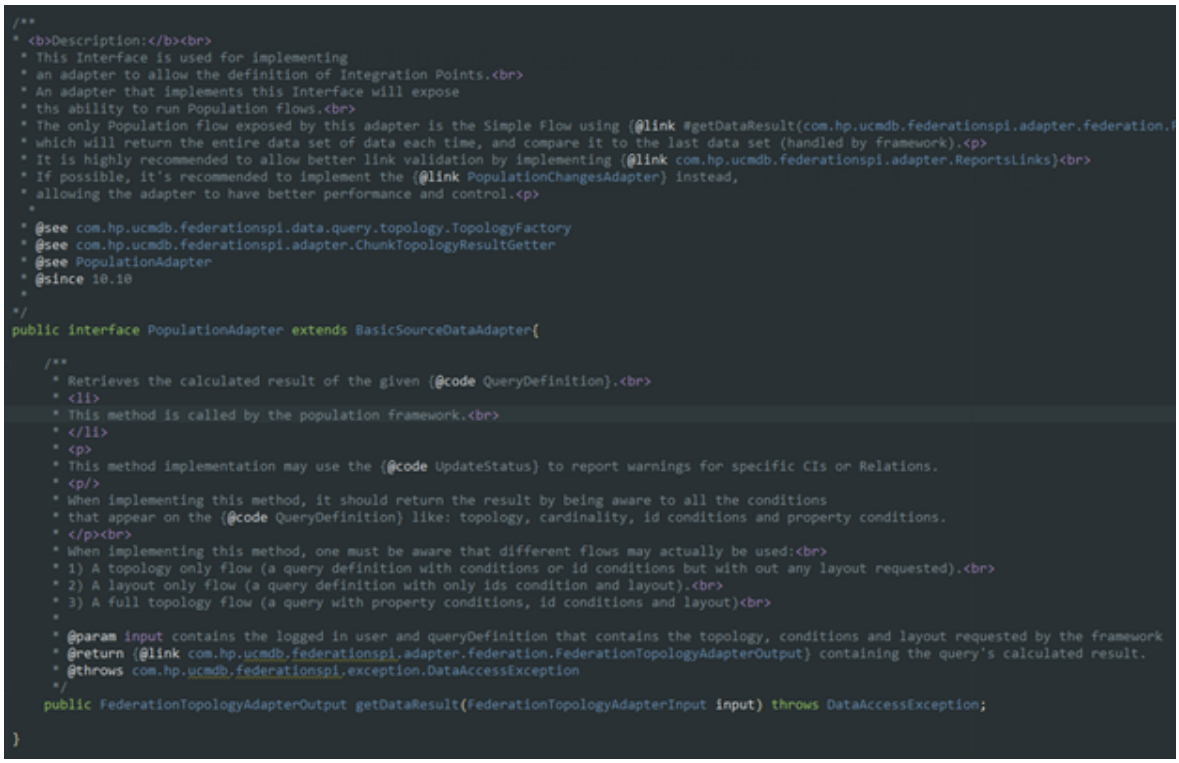
Pushback von globalen IDs

Es gibt Szenarien, in denen aufgefüllte CIs in UCMDB auch im externen System auf dem aktuellen Stand gehalten werden müssen. In einem sochen Szenario ermöglicht das Framework **GenericAdapter** die Aktivierung eines Pushback von IDs. Um diese Funktion zu verwenden, wird ein Rückruf für alle CIs durchgeführt, die in UCMDB aufgefüllt wurden, der den Auffüllungsadapter über die zugewiesene globale ID für jedes CI informiert.

Um diese Funktion zu aktivieren, fügen Sie die nachstehend markierte Zeile der Adapterkonfigurations-XML-Datei hinzu:

```
<adapterInfo>
    <adapter-capabilities>

        <support-federated-query>
        <supported-classes>
            <supported-class is-derived="true" all-attributes-supported="true" name="node" is-reconciliation-supported="true"/>
            <supported-class is-derived="true" all-attributes-supported="true" name="business_service" is-reconciliation-supported="true"/>
            <supported-class is-derived="true" all-attributes-supported="true" name="incident" is-reconciliation-supported="true"/>
        </supported-classes>
        </support-federated-query>
        <support-replicatioin-data>
            <source>
                <push-back-ids/>
                <instance-based-data/>
                <population-queries-resources-locator/>
            </source>
            <target>
                <instance-based-data>true</instance-based-data>
                <push-queries-resources-locator/>
            </target>
        </support-replicatioin-data>
        <general-resources-locator/>
    </adapter-capabilities>
```

Außerdem müssen Sie die Methode `PopulationAdapterConnector` wie folgt implementieren:



```
/**
 * <b>Description:</b><br>
 * This Interface is used for implementing
 * an adapter to allow the definition of Integration Points.<br>
 * An adapter that implements this Interface will expose
 * ths ability to run Population flows.<br>
 * The only Population flow exposed by this adapter is the Simple Flow using {@link #getDataResult(com.hp.ucmdb.federationspi.adapter.federation.
 * which will return the entire data set of data each time, and compare it to the last data set (handled by framework).<p>
 * It is highly recommended to allow better link validation by implementing {@link com.hp.ucmdb.federationspi.adapter.ReportsLinks}<br>
 * If possible, it's recommended to implement the {@link PopulationChangesAdapter} instead,
 * allowing the adapter to have better performance and control.<p>
 *
 * @see com.hp.ucmdb.federationspi.data.query.topology.TopologyFactory
 * @see com.hp.ucmdb.federationspi.adapter.ChunkTopologyResultGetter
 * @see PopulationAdapter
 * @since 10.10
 *
 */
public interface PopulationAdapter extends BasicSourceDataAdapter{

    /**
     * Retrieves the calculated result of the given {@code QueryDefinition}.<br>
     * <li>
     * This method is called by the population framework.<br>
     * </li>
     * <p>
     * This method implementation may use the {@code UpdateStatus} to report warnings for specific CIs or Relations.
     * <p/>
     * When implementing this method, it should return the result by being aware to all the conditions
     * that appear on the {@code QueryDefinition} like: topology, cardinality, id conditions and property conditions.
     * </p><br>
     * When implementing this method, one must be aware that different flows may actually be used:<br>
     * 1) A topology only flow (a query definition with conditions or id conditions but with out any layout requested).<br>
     * 2) A layout only flow (a query definition with only ids condition and layout).<br>
     * 3) A full topology flow (a query with property conditions, id conditions and layout)<br>
     *
     * @param input contains the logged in user and queryDefinition that contains the topology, conditions and layout requested by the framework
     * @return {@link com.hp.ucmdb.federationspi.adapter.federation.FederationTopologyAdapterOutput} containing the query's calculated result.
     * @throws com.hp.ucmdb.federationspi.exception.DataAccessException
     */
    public FederationTopologyAdapterOutput getDataResult(FederationTopologyAdapterInput input) throws DataAccessException;

}
```

Umsetzen der Datenföderation unter Verwendung des allgemeinen Adapters

Eine Föderation von Daten kann wie folgt erreicht werden:

Ansatz der Föderationszuordnung

Eine Föderationszuordnung wird durch eine Zuordnung der Unter-TQLs erreicht, die vom UCMDB Federation Framework für die Verarbeitung einer Föderationsanforderung verwendet werden. Die allgemeine Idee ist, dass bei dem Empfang einer Föderationsanforderung durch den generischen Adapter folgendes geschieht:

1. Analyse der dynamischen Föderations-TQL-Abfrage und Vergleich mit einer Liste der statischen Föderations-TQL-Abfragen, die durch den Föderations-Connector bereitgestellt werden.
2. Es wird eine statische TQL-Abfrage-Übereinstimmung vorgenommen. Diese TQL-Abfrage wird verwendet, um die erforderliche Zuordnung für die vorliegende Föderationsanforderung zu identifizieren, und das Eingabeargument der RTN-Struktur (ein Java-Objekt, das die für den Connector erforderliche Knotentruktur veranschaulicht) zu erstellen, das dem Föderations-Connector bereitgestellt wird. (Weitere Informationen finden Sie in der Datei **push-interfaces.jar**)
3. Senden des Föderationsaufruf mit dem TQL-Argument an den Connector.
4. Zuordnen der durch den Connector gesendeten RTN-Strukturen (auf die gleiche Weise wie für die Auffüllung). Siehe "Umsetzen des Datenauffüllung unter Verwendung des allgemeinen Adapters" auf Seite 285.

Zuordnung des Föderations-Links

Die Zuordnung des Föderations-Links erfolgt wie auch die Zuordnung des Auffüllungs-Links automatisch. Siehe "Automatische Link-Auffüllung" auf Seite 290.

Föderations-API des allgemeinen Adapters

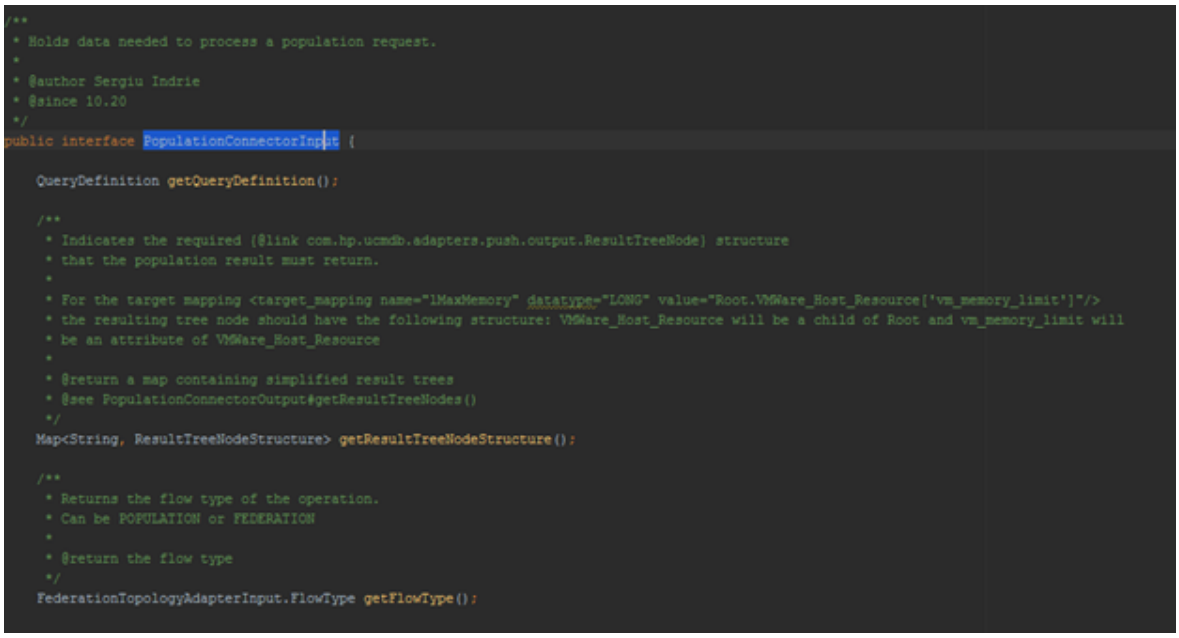
Die Föderations-API des allgemeinen Adapters ist der Auffüllungs-API des generischen Adapters sehr ähnlich. Der Grund hierfür ist, dass die Java-Schnittstelle des allgemeinen Föderationsadapters identisch mit der Schnittstelle des allgemeinen Auffüllungsadapters ist.

```java
/**
* <b>Description:</b><br>
 * This Interface is used for implementing
 * an adapter to allow the definition of Integration Points.<br>
 * An adapter that implements this Interface will expose
 * ths ability to run Population flows.<br>
 * The only Population flow exposed by this adapter is the Simple Flow using {@link #getDataResult(com.hp.ucmdb.federationspi.adapter.federation.
 * which will return the entire data set of data each time, and compare it to the last data set (handled by framework).<p>
 * It is highly recommended to allow better link validation by implementing {@link com.hp.ucmdb.federationspi.adapter.ReportsLinks}<br>
 * If possible, it's recommended to implement the {@link PopulationChangesAdapter} instead,
 * allowing the adapter to have better performance and control.<p>
 *
 * @see com.hp.ucmdb.federationspi.data.query.topology.TopologyFactory
 * @see com.hp.ucmdb.federationspi.adapter.ChunkTopologyResultGetter
 * @see PopulationAdapter
 * @since 10.10
 *
 */
public interface PopulationAdapter extends BasicSourceDataAdapter{

    /**
     * Retrieves the calculated result of the given {@code QueryDefinition}.<br>
     * <li>
     * This method is called by the population framework.<br>
     * </li>
     * <p>
     * This method implementation may use the {@code UpdateStatus} to report warnings for specific CIs or Relations.
     * <p/>
     * When implementing this method, it should return the result by being aware to all the conditions
     * that appear on the {@code QueryDefinition} like: topology, cardinality, id conditions and property conditions.
     * </p><br>
     * When implementing this method, one must be aware that different flows may actually be used:<br>
     * 1) A topology only flow (a query definition with conditions or id conditions but with out any layout requested).<br>
     * 2) A layout only flow (a query definition with only ids condition and layout).<br>
     * 3) A full topology flow (a query with property conditions, id conditions and layout)<br>
     *
     * @param input contains the logged in user and queryDefinition that contains the topology, conditions and layout requested by the framework
     * @return {@link com.hp.ucmdb.federationspi.adapter.federation.FederationTopologyAdapterOutput} containing the query's calculated result.
     * @throws com.hp.ucmdb.federationspi.exception.DataAccessException
     */
    public FederationTopologyAdapterOutput getDataResult(FederationTopologyAdapterInput input) throws DataAccessException;

}
```

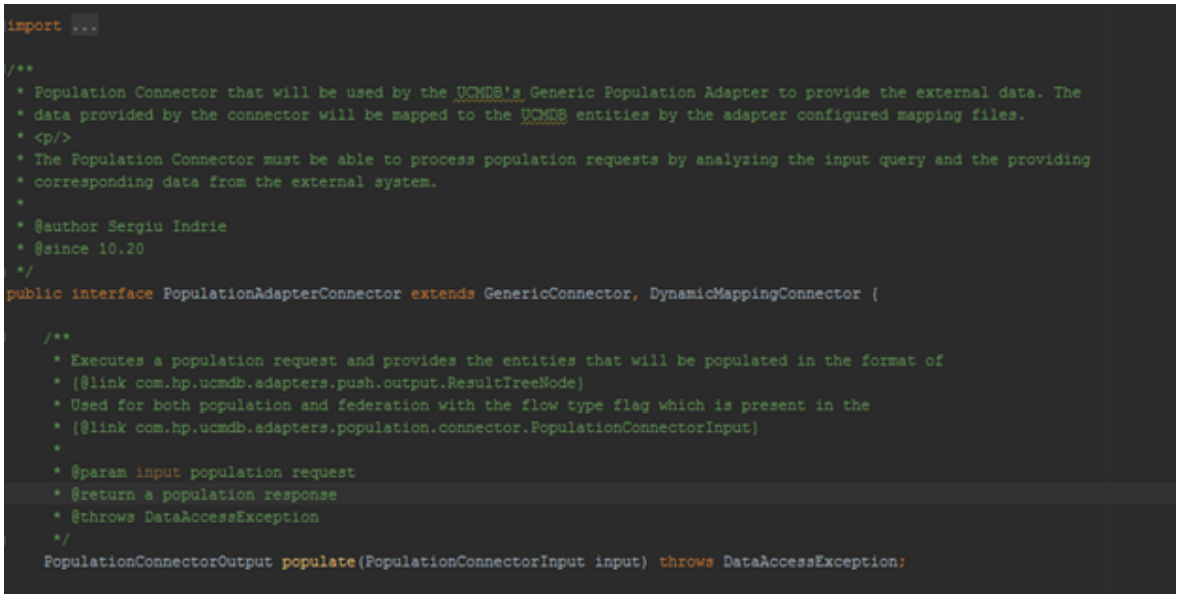
```java
import ...

/**
  * <b>Description:</b><br>
  * This Interface is used for implementing
  * an adapter to allow the definition of Integration Points.<br>
  * An adapter that implements this Interface will expose to run Federation flows.<br>
  * The adapter can report status per CI during the flow with {@link com.hp.ucmdb.federationspi.data.replication.UpdateStatus}.<br>
  * It is highly recommended to allow better link validation by implementing {@link com.hp.ucmdb.federationspi.adapter.ReportsLinks}
  * </p>
  *
 */
public interface FederationTopologyDataAdapter extends FTqlDataAdapter {

    /**
     * Retrieves the calculated result of the given {@code QueryDefinition}.<br>
     * <li>
     * This method is called by the federation framework when the user query includes any
     * federated elements(node or link).<br>
     * </li>
     * <p>
     * This method implementation may use the {@code UpdateStatus} to report warnings for specific CIs or Relations.
     * <p/>
     * When implementing this method, it should return the result by being aware to all the conditions
     * that appear on the {@code QueryDefinition} like: topology, cardinality, id conditions and property conditions.
     * </p><br>
     * When implementing this method, one must be aware that different flows may actually be used:<br>
     * 1) A topology only flow (a query definition with conditions or id conditions but with out any layout requested).<br>
     * 2) A layout only flow (a query definition with only ids condition and layout).<br>
     * 3) A full topology flow (a query with property conditions, id conditions and layout)<br>
     *
     * @param input contains the logged in user and queryDefinition that contains the topology, conditions and layout requested by th
     * @return {@link FederationTopologyAdapterOutput} containing the query's calculated result.
     * @throws com.hp.ucmdb.federationspi.exception.DataAccessException
     */
    public FederationTopologyAdapterOutput getDataResult(FederationTopologyAdapterInput input) throws DataAccessException;

}
```

Connector-Schnittstelle für die Föderation des allgemeinen Adapters

Föderationsanforderungen verwenden die gleiche Methode, die für Auffüllungsanforderungen verwendet wird, sodass die gleiche Implementierung des Auffüllungs-Connectors verwendet werden kann. In der PopulationConnectorInput Java-Klasse wurde ein neues Attribut mit der Bezeichnung **FlowType** hinzugefügt. Das Attribut **FlowType** kann zwei Werte haben, FEDERATION oder POPULATION. Basierend auf diesem Attribut kennt der allgemeine Adapter den Anforderungstyp.



```
/**
 * Holds data needed to process a population request.
 *
 * @author Sergiu Indrie
 * @since 10.20
 */
public interface PopulationConnectorInput {

    QueryDefinition getQueryDefinition();

    /**
     * Indicates the required {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode} structure
     * that the population result must return.
     *
     * For the target mapping <target_mapping name="lMaxMemory" datatype="LONG" value="Root.VMWare_Host_Resource['vm_memory_limit']"/>
     * the resulting tree node should have the following structure: VMWare_Host_Resource will be a child of Root and vm_memory_limit will
     * be an attribute of VMWare_Host_Resource
     *
     * @return a map containing simplified result trees
     * @see PopulationConnectorOutput#getResultTreeNodes()
     */
    Map<String, ResultTreeNodeStructure> getResultTreeNodeStructure();

    /**
     * Returns the flow type of the operation.
     * Can be POPULATION or FEDERATION
     *
     * @return the flow type
     */
    FederationTopologyAdapterInput.FlowType getFlowType();
```



```
import ...

/**
 * Population Connector that will be used by the UCMDB's Generic Population Adapter to provide the external data. The
 * data provided by the connector will be mapped to the UCMDB entities by the adapter configured mapping files.
 * <p/>
 * The Population Connector must be able to process population requests by analyzing the input query and the providing
 * corresponding data from the external system.
 *
 * @author Sergiu Indrie
 * @since 10.20
 */
public interface PopulationAdapterConnector extends GenericConnector, DynamicMappingConnector {

    /**
     * Executes a population request and provides the entities that will be populated in the format of
     * {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode}
     * Used for both population and federation with the flow type flag which is present in the
     * {@link com.hp.ucmdb.adapters.population.connector.PopulationConnectorInput}
     *
     * @param input population request
     * @return a population response
     * @throws DataAccessException
     */
    PopulationConnectorOutput populate(PopulationConnectorInput input) throws DataAccessException;
```

Unterstützte Föderationsabfragen

Die Föderations- und Auffüllungsabfragen befinden sich in verschiedenen Ordnern. Die Java-Schnittstelle **PopulationAdapterConnector** bietet die folgenden beiden Methoden für die Angabe der unterstützten Auffüllungs- und Föderationsabfragen:

- **getPopulationQueries** – Gibt die Sammlung der Abfragen zurück, die der aktuelle Connector für die Auffüllung unterstützt.
- **getFederationQueries** – Gibt die Sammlung der Abfragen zurück, die der aktuelle Connector für die Föderation unterstützt.

```
/**
 * Returns the collection of queries the current connector supports for population
 * Note: this method is independent to the adapter life cycle (i.e. start/shutdown methods) and must work at all times.
 * <p/>
 * The method also supports return of queries with their folders path: E.g.
 * "Folder/Secondary Folder/Query Name 1"
 * "Folder/Secondary Folder/Query Name 2"
 *
 * @param env the adapter environment
 * @return a collection of the supported queries
 */
Collection<String> getPopulationQueries(DataAdapterEnvironment env);

/**
 * Returns the collection of queries the current connector supports for federation
 * Note: this method is independent to the adapter life cycle (i.e. start/shutdown methods) and must work at all times.
 * <p/>
 * The method also supports return of queries with their folders path: E.g.
 * "Folder/Secondary Folder/Query Name 1"
 * "Folder/Secondary Folder/Query Name 2"
 *
 * @param env the adapter environment
 * @return a collection of the supported queries
 */
Collection<String> getFederationQueries(DataAdapterEnvironment env);
```

Einrichten der Föderation

Dieser Abschnitt umfasst:

Konfigurieren von Adaptereinstellungen

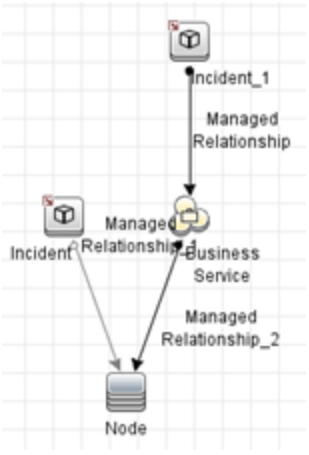
Für eine bestimmte TQL-Abfrage muss der generische Adapter alle Knoten aus dieser TQL-Abfrage im Tag **<supported-classes>** deklarieren. Beispiel: Wenn die TQL-Abfrage die Form eines Vorfalls hat, der mit einem Knoten verknüpft ist, müssen Sie sowohl den Knoten als auch den Incident als unterstützte Klassen in der Adaptereinstellungs-XML-Datei des Adapters deklarieren. Diese befindet sich in der Package-ZIP-Datei im Ordner **discoveryPatterns**.


```
<supported-classes>
    <supported-class is-derived="true" all-attributes-supported="true" name="node" is-reconciliation-supported="true"/>
    <supported-class is-derived="true" all-attributes-supported="true" name="incident" is-reconciliation-supported="true"/>
</supported-classes>
```

Einrichten von Föderationsabfragen aus Protokolldateien

Einige Voraussetzungen müssen erfüllt sein, bevor das Federation Framework verwendet werden kann. Das Federation Framework erstellt mehrere Anforderungen mit unterschiedlichen TQL-Abfragen an den Adapter, um die erforderlichen Daten abzurufen. Für jede einzelne TQL-Abfrage, die das Federation Framework sendet, muss ähnlich wie bei dem Auffüllungsfluss eine dynamische TQL-Abfrage im Adapter erstellt werden.

Der Unterschied zwischen Föderation und Auffüllung ist, dass die Föderations-TQL-Abfragen dynamisch durch das Framework gesendet werden, so dass sie im Vorfeld nicht bekannt sind. Beispiel einer TQL-Abfrage:



Die Framework sendet die folgenden Abfragen an den Adapter:

- Eine Abfrage nur mit dem Vorfall
- Eine Abfrage mit den Vorfall, verknüpft mit dem Knoten, mit einer Beziehung der Typverbindung
- Eine Abfrage mit den Vorfall, verknüpft mit dem Knoten, mit einer Beziehung der Typmitgliedschaft
- Eine Abfrage mit den Vorfall, verknüpft mit dem Geschäftsservice, mit einer Beziehung der Typverbindung
- Eine Abfrage mit den Vorfall, verknüpft mit dem Geschäftsservice, mit einer Beziehung der Typmitgliedschaft

Hinweis: Alle Abfragen, die die Föderations-Engine an den Adapter sendet und die gespeichert werden müssen, haben den Namen **User mapping union FTQL**.

Nachdem die Ergebnisse der Abfrage **User mapping union FTQL** verarbeitet wurden, werden andere Aufrufe zum Abrufen der Attribute der Objekte vorgenommen. Diese Aufrufe enthalten eine Abfrage mit dem Namen **objects layout**. Die Föderations-Engine versucht, alle Attribute für ein CI abzurufen. Der Connector muss diese jedoch nicht alle bereitstellen; es ist ausreichend, nur diejenigen zurückzugeben, die für die Zuordnungsdatei erforderlich sind.

Die gleiche Abfrage wird mit unterschiedlichen Beziehungen gesendet, da in der TQL-Abfrage ein Link vom Typ **managed_relationship** zwischen Knoten und Vorfall/Geschäftsservice und Vorfall besteht, die einzigen gültigen Links für eine Verknüpfung dieser CI-Typen jedoch Verbindung und Mitgliedschaft sind.

```
<tql:link from="incident_12" to="node_10050" class="connection" name="connection_1" id="1"/>
<tql:link from="incident_12" to="node_1000050" class="connection" name="connection_2" id="2"/>
<tql:link from="datacenter_20050" to="node_10050" class="composition" name="composition_30050" id="30050"/>
```

```
<tql:link from="incident_12" to="node_10050" class="membership" name="membership_1" id="1"/>
<tql:link from="incident_12" to="node_1000050" class="membership" name="membership_2" id="2"/>
<tql:link from="datacenter_20050" to="node_10050" class="composition" name="composition_30050" id="30050"/>
```

Durch die Verwendung dieses Ansatzes müssen wir nur eine statische TQL mit einem generischen Link vom Typ **managed_relationship** definieren, anstatt zwei nahezu identische TQLs mit unterschiedlichen Link-Typen zu definieren.

```
2014-08-07 16:43:55,231 [AdHoc:AD_HOC_TASK_PATTERN_ID-179-1407419035224] TRACE -  >> Received federation call with the following query:
2014-08-07 16:43:55,231 [AdHoc:AD_HOC_TASK_PATTERN_ID-179-1407419035224] TRACE -  >>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tql:query name="User mapping union FTQL" is-live="true" priority="low" xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRuleD
    <tql:node class="incident" name="incident_16" id="16">
        <tql:where>
            <tql:links>
                <tql:or>
                    <tql:link-ref name="membership_1"/>
                    <tql:link-ref name="membership_2"/>
                </tql:or>
            </tql:links>
        </tql:where>
    </tql:node>
    <tql:node class="business_service" name="business_service_10050" id="10050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="name"/>
                    <tql:list type="string">
                        <tql:string>MyBusServ</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="name"/>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:node class="business_service" name="business_service_1000050" id="1000050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="global_id"/>
                    <tql:list type="string">
                        <tql:string>183ad8038405644e67aba201334714ea</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:link from="incident_16" to="business_service_10050" class="membership" name="membership_1" id="1"/>
    <tql:link from="incident_16" to="business_service_1000050" class="membership" name="membership_2" id="2"/>
</tql:query>
```

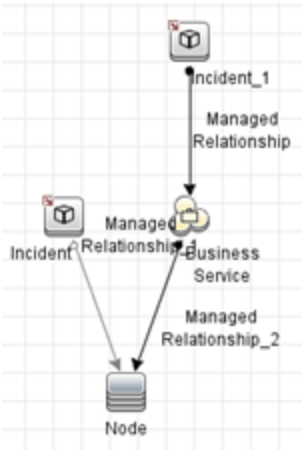
Diese statischen TQL-Abfragen müssen durch den Adapter bereitgestellt werden. Um die Entwicklung des Adapters zu unterstützen, werden die TQL-Abfragen, die von Federation Framework gesendet werden, in die Datei **fcmdb.push.mapping.log** geschrieben (mit aktivierter Protokollebene "TRACE"). Um den Entwicklungsaufwand zu verfeinfachen, verwenden Sie die Einstellung `<adapter-setting name="dev.mode">true</adapter-setting>`. Wenn diese Einstellung nach Ausführung einer Föderationsabfrage auf **True** gesetzt ist, erstellt das Framework automatisch ein leeres Föderationsergebnis für die aktuelle nicht abgestimmte TQL-Abfrage.

Beispiel für eine Föderationsabfrage aus "fcmdb.push.mapping.log":

```
2014-08-07 16:43:55,231 [AdHoc:AD_HOC_TASK_PATTERN_ID-179-1407419035224] TRACE -  >> Received federation call with the following query:
2014-08-07 16:43:55,231 [AdHoc:AD_HOC_TASK_PATTERN_ID-179-1407419035224] TRACE -  >>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tql:query name="User mapping union FTQL" is-live="true" priority="low" xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRuleD
    <tql:node class="incident" name="incident_16" id="16">
        <tql:where>
            <tql:links>
                <tql:or>
                    <tql:link-ref name="membership_1"/>
                    <tql:link-ref name="membership_2"/>
                </tql:or>
            </tql:links>
        </tql:where>
    </tql:node>
    <tql:node class="business_service" name="business_service_10050" id="10050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="name"/>
                    <tql:list type="string">
                        <tql:string>MyBusServ</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="name"/>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:node class="business_service" name="business_service_1000050" id="1000050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="global_id"/>
                    <tql:list type="string">
                        <tql:string>183ad8038405644e67aba201394714ea</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:link from="incident_16" to="business_service_10050" class="membership" name="membership_1" id="1"/>
    <tql:link from="incident_16" to="business_service_1000050" class="membership" name="membership_2" id="2"/>
</tql:query>
```

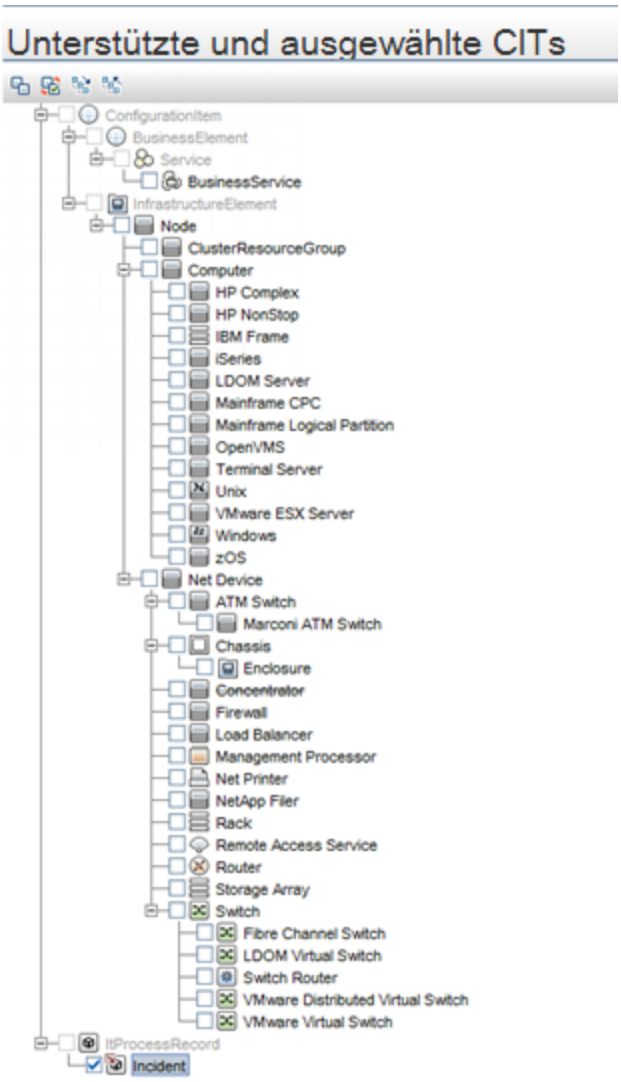
Beispiel zum Einrichten der Föderation

Im Beispiel wird die folgende Föderations-TQL verwendet:



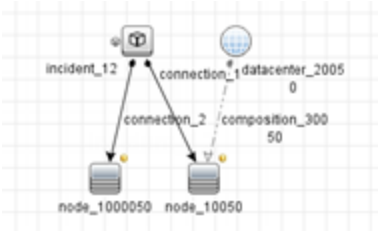
Für diese TQL-Abfrage muss der Adapter die unterstützten Klassen in die Adaptereinstellungs-XML-Datei deklarieren. Diese befindet sich in der Adapter-Package-ZIP-Datei im Ordner **discoveryPatterns**. Die unterstützten Klassen sind **node**, **incident** und **business_service**.

Im Integration Studio muss der Vorfall auf der Registerkarte **Föderation** ausgewählt werden, wie unten abgebildet:

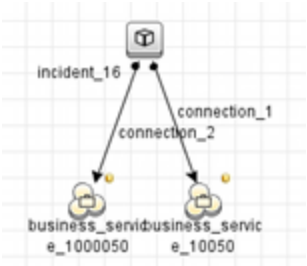


Für diese TQL-Abfrage müssen die folgenden drei TQL-Abfragen in den Adapter vorliegen:

Incident



Mit Knoten verknüpfter Incident



Mit business_service verknüpfter Incident

Weitere Informationen zum Abrufen statischer TQLs finden Sie unter "Einrichten von Föderationsabfragen aus Protokolldateien" auf Seite 304. Obwohl diese TQL-Abfragen Bedingungen enthalten, die abhängig sind von den vorhandenen Daten in UCMDB, sollte sich dies nicht auf die Struktur der TQL-Abfragen oder die Art der Durchführung der Zuordnung auswirken.

Für jede dieser TQL-Abfragen ist eine Zuordnungsdatei im Adapter erforderlich:

- Incident

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resource:XmlResourceWrapper xmlns:ns1="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns2="http://www.hp.com/ucmdb/1-0-0/PolicyRuleDefiniti
    <resource xsi:type="tql:Query" name="SM_Incident" is-active="false" priority="low" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<tql:node class="incident" name="incident_12" id="12"/>
    </resource>
    <resourceBundle>integration_tqls_bundle</resourceBundle>
</resource:XmlResourceWrapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../generic-adapter.xsd">
    <!-- add scheme reference -->
    <info>
        <source name="Dummy" version="10.0" vendor="HP"/>
        <target name="UCMDB" version="10.20" vendor="HP"/>
    </info>

    <!--
    This mapping converts the Root entities received from the population connector to a "node" item in UCMDB.
    The output of this mapping must match the indicated query (TQL), "Dummy Query"
    -->

    <target_entities>
        <!--The query name must match the one selected in the UI-->
        <source_instance query-name="SM_Incident" root-element-name="incident_12">
            <!-- need to match case in UCMDB TQL -->
            <target_entity name="incident_12">
                <target_mapping name="name" datatype="STRING" value="incident_12['name']"/>
                <target_mapping name="description" datatype="STRING" value="incident_12['description']"/>
                <target_mapping name="reference_number" datatype="STRING" value="incident_12['reference_number']"/>
            </target_entity>
        </source_instance>
    </target_entities>

</integration>
```

- Mit Knoten verknüpfter Incident

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resource:XmlResourceWrapper xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRuleDefinition" xmlns:resou
    <resource xsi:type="tql:Query" name="SM_Node" is-active="false" priority="low" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<tql:node class="node" name="node_10050" id="10050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="name"/>
                    <tql:list type="string">
                        <tql:string>mynode</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
            <tql:links>
                <tql:link-ref name="composition_30050" min-occurs="0"/>
                <tql:link-ref name="connection_1"/>
            </tql:links>
        </tql:where>
        <tql:content>
            <tql:properties...>
        </tql:content>
    </tql:node>
    <tql:node class="node" name="node_1000050" id="1000050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="global_id"/>
                    <tql:list type="string">
                        <tql:string>9b360a1f42eaf7cff793feb57c88f098</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content...>
    </tql:node>
    <tql:node class="incident" name="incident_12" id="12">
        <tql:where>
            <tql:ids>
                <tql:id>GA%0Aincident%0A3%0Adescription%3DSTRING%3Dtest_incident2%0Aname%3DSTRING%3DIncident2%0Areference_number%3DSTRING%3D101%0A</tql:id>
                <tql:id>GA%0Aincident%0A3%0Adescription%3DSTRING%3Dtest_incident%0Aname%3DSTRING%3DIncident%0Areference_number%3DSTRING%3D100%0A</tql:id>
            </tql:ids>
            <tql:links>
                <tql:or>
                    <tql:link-ref name="connection_1"/>
                    <tql:link-ref name="connection_2"/>
                </tql:or>
            </tql:links>
        </tql:where>
    </tql:node>
    <tql:node class="datacenter" name="datacenter_20050" id="20050"/>
    <tql:link from="incident_12" to="node_10050" class="managed_relationship" name="connection_1" id="1"/>
    <tql:link from="incident_12" to="node_1000050" class="managed_relationship" name="connection_2" id="2"/>
    <tql:link from="datacenter_20050" to="node_10050" class="composition" name="composition_30050" id="30050"/>
    </resource>
    <resourceBundle>integration_tqls_bundle</resourceBundle>
</resource:XmlResourceWrapper>
```

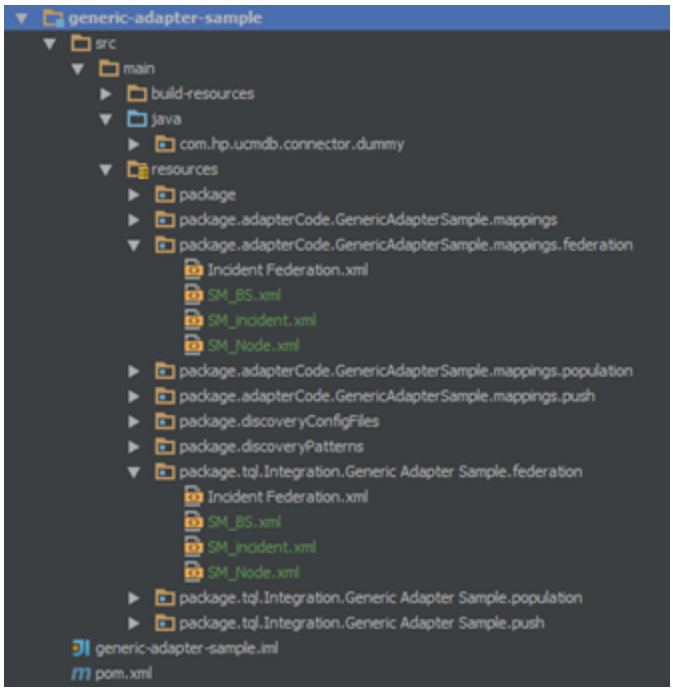
```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../generic-adapter.xsd">
    <!-- add scheme reference -->
    <info>
        <source name="Dummy" version="10.0" vendor="HP"/>
        <target name="UCMDB" version="10.20" vendor="HP"/>
    </info>
    <target_entities>
        <!--The query name must match the one selected in the UI-->
        <source_instance query-name="SM_Node" root-element-name="Computer">
            <!-- need to match case in UCMDB TQL -->
            <target_entity name="node_1000050">
                <target_mapping name="name" datatype="STRING" value="Computer['name']"/>
            </target_entity>


            <for-each-source-entity count-index="i" source-entities="Computer.incident_12" var-name="currIP">
                <target_entity name="incident_12">
                    <target_mapping name="name" datatype="STRING" value="currIP['name']"/>
                    <target_mapping name="description" datatype="STRING" value="currIP['description']"/>
                    <target_mapping name="reference_number" datatype="STRING" value="currIP['reference_number']"/>
                </target_entity>
            </for-each-source-entity>


        </source_instance>
    </target_entities>
</integration>
```

- Incident an business_service

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resource:XmlResourceWrapper xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRu
    <resource xsi:type="tql:Query" name="SM_BS" is-active="false" priority="low" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tql:node class="incident" name="incident_16" id="16">
        <tql:where>
            <tql:links>
                <tql:or>
                    <tql:link-ref name="connection_1"/>
                    <tql:link-ref name="connection_2"/>
                </tql:or>
            </tql:links>
        </tql:where>
    </tql:node>
    <tql:node class="business_service" name="business_service_10050" id="10050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="name"/>
                    <tql:list type="string">
                        <tql:string>MyBusServ</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="name"/>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:node class="business_service" name="business_service_1000050" id="1000050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="global_id"/>
                    <tql:list type="string">
                        <tql:string>183ad8038405644e67aba201334714ea</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
        <tql:link from="incident_16" to="business_service_10050" class="managed_relationship" name="connection_1" id="1"/>
        <tql:link from="incident_16" to="business_service_1000050" class="managed_relationship" name="connection_2" id="2"/>
    </resource>
    <resourceBundle>integration_tqls_bundle</resourceBundle>
</resource:XmlResourceWrapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../generic-adapter.xsd">
    <!-- add scheme reference -->
    <info>
        <source name="Dummy" version="10.0" vendor="HP"/>
        <target name="UCMDB" version="10.20" vendor="HP"/>
    </info>
    <target_entities>
        <!--The query name must match the one selected in the UI-->
        <source_instance query-name="SM_BS" root-element-name="BS">
            <!-- need to match case in UCMDB TQL -->
            <target_entity name="business_service_10050">
                <target_mapping name="name" datatype="STRING" value="BS['name']"/>
            </target_entity>
            <for-each-source-entity count-index="i" source-entities="BS.incident_16" var-name="currIP">
                <target_entity name="incident_16">
                    <target_mapping name="name" datatype="STRING" value="currIP['name']"/>
                    <target_mapping name="description" datatype="STRING" value="currIP['description']"/>
                    <target_mapping name="reference_number" datatype="STRING" value="currIP['reference_number']"/>
                </target_entity>
            </for-each-source-entity>
        </source_instance>
    </target_entities>
</integration>
```

Diese TQL-Abfragen und Zuordnungsdateien müssen im Adapter vorhanden sein, wie nachfolgend gezeigt:



Abstimmung

Bei Verwendung des Frameworks **GenericAdapter** zum Auffüllen oder Föderieren von Daten müssen die CIs immer die erforderlichen Abstimmungsdaten enthalten, um in UCMDB akzeptiert zu werden. Beim Auffüllen von CI-Typen, wie z. B. aktiver Software, die einen Container-CI-Typ erfordern, müssen Sie sicherstellen, dass Sie die erforderlichen Container-Felder (z. B. **root_container_name** und **product_name**) und die Container-CI (zum Beispiel **Node**) aufgefüllt werden. Um CIs aufzufüllen, die von einem Root-Container abhängig sind, müssen das CI, sein Root-Container, und der Link zwischen ihnen die im gleichen Schritt erstellt werden (entweder mit einer expliziten Link-Auffüllung oder einer automatischen Link-Auffüllung zwischen den beide CIs).

Darüber hinaus sollten Sie bei der Zuordnung der aufgefüllten/föderierten CIs die Zuordnung des Attributs **global_id** berücksichtigen, da dieses einerseits die UCMDB-Abstimmungs-Engine entscheidend unterstützt und andererseits eine genaue CI-Abstimmung garantiert.

GenericAdapter-API

Die durch das Framework **GenericAdapter** verfügbar gemachte API lautet wie folgt:

```
<UCMDB_Server>\lib\push-interfaces.jar
<UCMDB_Server>\lib\integrationFramework\GenericAdapter\generic-adapter-api-factory.jar
```

Die Entwicklung Ihrer **GenericAdapter**-Instanz erfordert möglicherweise auch die Federation-API:

```
<UCMDB_Server>\lib\federation-api.jar
```

Resource Locator-APIs

Die Ressource Locator-APIs können beim Bearbeiten von Jobs mit generischen Adaptern verwendet werden. Implementieren Sie die allgemeinen und Auffüllungs-Resource-Locator-APIs, um die Adapter-Ressourcen zu finden, die mit der TQL-Abfrage eines ausgewählten Jobs verknüpft sind.

Die folgende Abbildung zeigt eine allgemeine Resource Locator-API in der Java-Schnittstelle GenericConnector:

```
/**
 * This methods reports general resources used by the adapter.<br>
 * This allows editing these resources directly from the Integration Studio.
 *
 *
 * @param env the Adapter's environment
 * @param input  the requested information
 * @param output the returned resources
 * @see com.hp.ucmdb.federationspi.adapter.resource.GeneralResourcesLocator
 */
void locateGeneralResources(DataAdapterEnvironment env, LocateGeneralResourcesInput input, LocateGeneralResourcesOutput output);
```

Die folgende Abbildung zeigt eine Auffüllungs-Resource-Locator-API in der Java-Schnittstelle PopulationAdapterConnector:

```
/**
 * This methods reports population queries resources used by the adapter.<br>
 * This allows editing these resources directly from the Integration Studio.
 *
 * @param input  the requested information
 * @param output the returned resources
 * @see com.hp.ucmdb.federationspi.adapter.resource.PushQueriesResourceLocator
 */
void locatePopulationQueriesResources(DataAdapterEnvironment env, LocatePopulationQueriesResourcesInput input,
                                      LocatePopulationQueriesResourcesOutput output);
```

Um die zugehörigen Ressourcen für die TQL-Abfrage eines Jobs anzuzeigen, gehen Sie wie folgt vor:

1. Wählen Sie einen Integrationspunkt im Integration Studio aus.
2. Wählen Sie im Ausschnitt **Integrationsjobs** einen Job aus, und klicken Sie auf die Schaltfläche **Abfrageressourcen bearbeiten**  .

Erstellen eines Package für den generischen Adapter

Ein Package für den generischen Adapter ist mit einem Package für einen erweiterten generischen Push-Adapter vergleichbar. Um das erste ZIP-Archiv-Gerüst zu erstellen, empfiehlt es sich, ein vorhandenes Packages eines generischen Adapters zu kopieren, und dieses Ihren Anforderungen entsprechend anzupassen. Weitere Informationen zum Adapter-Package finden Sie unter "Umsetzen des Datenpush unter Verwendung des allgemeinen Adapters" auf Seite 276.

Folgende Unterschiede bestehen zwischen einem vorhandenen erweiterten Package für einen generischen Push-Adapter und einem Package für einen generischen Adapter:

- Adapter-XML-Unterschiede
  - Der Adapter-Klasse wird von **PushAdapter** zu **GenericAdapter** geändert:

    ```
    <className>com.hp.ucmdb.adapters.push.PushAdapter</className>
    ```

    ```
    <className>com.hp.ucmdb.adapters.GenericAdapter</className>
    ```

  - Der Adapter-Funktionen beinhalten die Auffüllungsfunktion

    ```
    <support-replicatioin-data>
        <source>
            <push-back-ids/>
            <instance-based-data/>
            <population-queries-resources-locator/>
        </source>
    ```

  - Mit der Definition des Auffüllungs-Connectors durch die Adaptereinstellung:

    ```
    <adapter-setting name="PopulationConnector.class.name">com.hp.ucmdb.connector.dummy.DummyPopulationConnector</adapter-setting>
    ```

    Der generische Adapter (unter Verwendung der Auffüllungsfunktion) erfordert auch die Definition der Push-Connector-Klasse:

    ```
    <adapter-setting name="PushConnector.class.name">com.hp.ucmdb.connector.dummy.DummyPushConnector</adapter-setting>
    ```

- Ordner der Zuordnungsdatei

  Im Gegensatz zu dem erweiterten generischen Push-Adapter (dessen Zuordnungsdateien im Ordner **<adapter_package_zip>/adapterCode/<*Adaptername*>/mappings** Ordner gespeichert sein müssen) müssen die Zuordnungen des generischen Adapters in drei separaten Ordner gespeichert sein (jeweils eine für Push, Auffüllung und Föderation). Die erforderlichen Ordner sind:

  <adapter_package_zip>/adapterCode/<*Adaptername*>/mappings/push
  <adapter_package_zip>/adapterCode/<*Adaptername*>/mappings/population
  <adapter_package_zip>/adapterCode/<*Adaptername*>/mappings/federation

  wobei sich **<adapter_package_zip>** auf das zip-Archiv bezieht, das Sie für das Package des allgemeinen Adapters erstellen werden.

Hinweis: Obwohl der generische Adapter alle drei Typen der Datensynchronisierung (Push, Auffüllung und Föderation) unterstützt, kann ein spezifischer generischer Adapter wahlweise auch nur eine Untergruppe dieser Typen unterstützen.

Was bei der Erstellung eines neuen Adapters aus einem vorhandenen Adapter zu bachten ist

- **TestAdapter\discoveryPatterns\TestAdapter.xml**
- Passen Sie die Datei **TestAdapter.xml** an:
  - `<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="TestAdapter" xsi:noNamespaceSchemaLocation="../../Patterns.xsd" description="…" schemaVersion="9.0" displayName="…">`
  - `<adapter-id>TestAdapter</adapter-id>`

- Die ZIP-Datei mit dem neuen Adapter sollte den gleichen Namen haben wie der Adapter selbst – TestAdapter.

Erstellen eines Adapter-Package

Stellen Sie sicher, dass das Adapter Package die folgenden Ordner enthält:

- **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-Abfrage wird jedoch automatisch erstellt, wenn das Package bereitgestellt wird.

Aktivieren/Deaktivieren der Attribut- und Link-Validierung auf Adapterebene

Sie können die Attribut- und Link-Validierung für generische Adapter auf Adapterebene aktivieren oder deaktivieren, indem Sie die folgende Einstellung hinzufügen:

```
<adapter-settings>

        <adapter-setting
name="enable.attributes.links.validation">true</adapter-setting>

</adapter-settings>
```

Um die Adapterebenen-Validierung von Attributen und Links zu aktivieren, ändern Sie die Adaptereinstellung **enable.attributes.links.validation** in **true**.

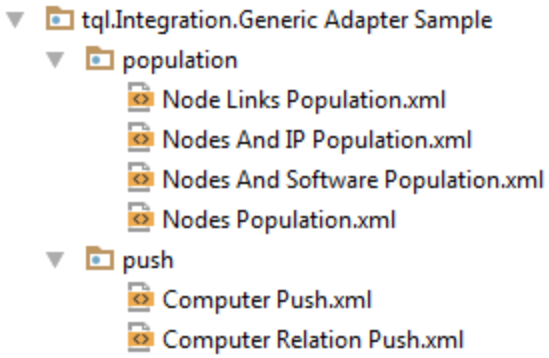
Um die Adapterebenen-Validierung von Attributen und Links zu deaktivieren, ändern Sie die Adaptereinstellung **enable.attributes.links.validation** in **false**.

Hinweis: Wenn die Einstellung nicht vorhanden ist, ist der Standardwert **true**, was bedeutet, dass die Attribut- und Link-Valdierung standardmäßig aktiviert ist.

TQL-Abfrage für das Auffüllen

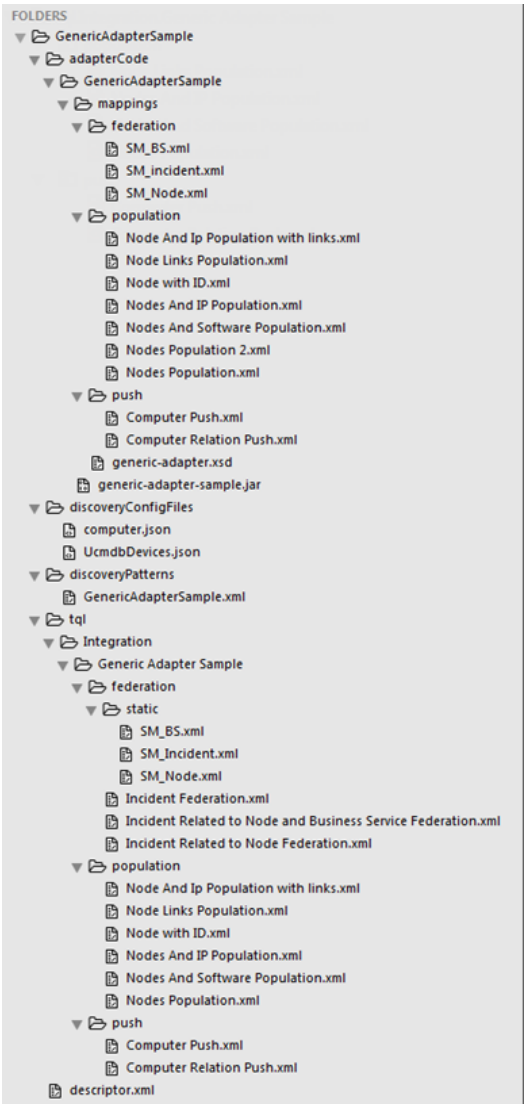
Die TQL-Abfragen, die für die Auffüllungsjobs verwendet werden, müssen im ZIP-Archiv des generischen Adapters enthalten und mit dem Adapter in UCMDB bereitgestellt werden. Die angegebe TQL-Abfrage muss in UCMDB vorhanden sein, wenn eine Auffüllungsnforderung während des Auffüllungsflusses erfolgt.

Diese TQL-Abfragen müssen im Ordner **<zip>/tql/<Ordner_1>/../<Ordner_n>** enthalten sein. Nachfolgend sehen Sie ein Beispiel für die Ordnerstruktur:



Obwohl Auffüllungs-TQL-Abfragen im oben angegebenen Ordner gespeichert werden, muss der Auffüllungs-Connector auch die unterstützten Auffüllungs-TQL-Abfragen in der entsprechenden Methode der Java-Benutzeroberfläche bestätigen. Weitere Informationen finden Sie unter "Auffüllungs-Connector" auf Seite 291.

Beispiel-Package



Unterschiede zwischen der Zuordnung für den Datenpush und der -auffüllung

Obwohl sowohl die Push- als auch die Auffüllungs-Zuordnungsdateien über das gleiche zugrundeliegende XML-Schema verfügen, werden die Dateien leicht unterschiedlich interpretiert. Weitere Informationen finden Sie unter "XML-Schemareferenz des allgemeinen Adapters" auf Seite 320.

Die folgende Push-Zuordnungs-Beispieldatei wird wie folgt interpretiert: Nehmen Sie die Ergebnisse der "Computer Push"-TQL-Abfrage (in UCMDB ausgeführt), die auch in der Root-Struktur vorhanden sind, und erstellen Sie die Entität **amComputer**, die später an AM gesendet wird.

```
<target_entities>
    <!--The query name must match the one selected in the UI-->
    <source_instance query-name="Computer Push" root-element-name="Root">
        <target_entity name="amComputer">
            <target_mapping name="TcpIpHostName" datatype="STRING" value="Root['name']"/>
            <target_mapping name="ComputerDesc" datatype="STRING" value="Root['os_description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Die folgende Auffüllungszuordnungs-Beispieldatei wird wie folgt interpretiert: Nehmen Sie die Ergebnisse der "Nodes Population"-TQL-Abfrage (im externen System ausgeführt), die auch in der PC-Struktur vorhanden sind, und erstellen Sie die UCMDB-Root-Entität (des Typs **Node**, wie in der TQL-Abfrage angegeben), die später in UCMDB hinzugefügt wird.

```
<target_entities>
    <!--The query name must match the one selected in the UI-->
    <source_instance query-name="Nodes Population" root-element-name="PC">
        <!-- need to match case in UCMDB class model-->
        <target_entity name="Root">
            <target_mapping name="name" datatype="STRING" value="PC['name']"/>
            <target_mapping name="description" datatype="STRING" value="PC['description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Protokolldateien des allgemeinen Adapters

Gehen Sie zur Fehlerbehebung und zum Debuggen folgendermaßen vor:

- Passen Sie die Protokollebenen in diesen Dateien an (legen Sie die Variable *loglevel* für die detailliertesten Ergebnisse auf TRACE fest):
  - **<UCMDB_DataFlowProbe>\conf\log\fcmdb.push.properties**

    <UCMDB_DataFlowProbe> ist das UCMDB Data Flow Probe-Installationsverzeichnis.
  - **<UCMDB_Server>\conf\log\reconciliation.properties**

    <UCMDB_Server> ist das Installationsverzeichnis des UCMDB-Servers.
- Analysieren Sie die folgenden Protokolldateien des allgemeinen Adapters:
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.all.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.configuration.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.connector.all.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.connector.configuration.log**

- **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.mapping.log**
- **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.all.log**

- Analysieren Sie die folgenden allgemeinen Protokolldateien:
  - **<UCMDB_DataFlowProbe>\runtime\log\probe-error.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\WrapperProbeGw.log**
  - **<UCMDB_Server>\runtime\log\error.log**
  - **<UCMDB_Server>\runtime\log\cmdb.reconciliation.log**

Adapter, die das Framework für allgemeine Adapter verwenden

Als Referenz für die Entwicklung Ihres benutzerdefinierten allgemeinen Adapters finden Sie die folgenden im Lieferumfang von UCMDB als Implementierungsrichtlinien enthaltenen Adapter, mit denen sich die Entwicklung Ihres Adapters beschleunigen lassen sollte:

- Asset Manager-Adapter
- Service Manager-Adapter

XML-Schemareferenz des allgemeinen Adapters

Das XML-Schema des allgemeinen Adapters befindet sich im Verzeichnis **schema** in der Datei **cmdb.jar**. Beim Schreiben der Zuordnungsdateien für den allgemeinen Adapter in externen Editoren sollte auf die Schema-Datei verwiesen werden. Der vollständige Pfad für die XSD-Datei lautet wie folgt:

<UCMDB_Server_Install_dir>/lib/cmdb.jar/schema/generic-adapter.xsd

Teil II: Verwenden von APIs

Kapitel 9: 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 323.
- **UCMDB-Webservice-API.** Ermöglicht das Schreiben von CI-Definitionen und topologischen Beziehungen in die UCMDB sowie das Abfragen der Informationen mittels TQL- und Ad-hoc-Abfragen. Weitere Informationen finden Sie unter "HP Universal CMDB-Webservice-API" auf Seite 331.
- **Java API zur Datenflussverwaltung.** Ermöglicht das Verwalten von Proben, Jobs, Triggern und Anmeldeinformationen für die Datenflussverwaltung. Weitere Informationen finden Sie unter "Java API zur Datenflussverwaltung" auf Seite 365.
- **API des Webservice "Datenflussverwaltung".** Ermöglicht das Verwalten von Proben, Jobs, Triggern und Anmeldeinformationen für die Datenflussverwaltung. Weitere Informationen finden Sie unter "Datenflussverwaltungswebservice-API" auf Seite 367.

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 10: 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\deploymdb-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" unten
- "Berechtigungen" unten

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 "UCMDB-API-Anwendungsfälle" auf Seite 329.

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.

Der Benutzer muss zusätzlich die Berechtigung für die allgemeine Aktion **Zugriff auf SDK** besitzen, um sich anmelden zu können.

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 327.

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**
getAuthorizationModelService	Durchführen von Autorisierungsoperationen (Erstellen von Benutzern und Benutzergruppen, Zuweisen von Rollen zu Benutzern und Gruppen usw.).
getClassModelService	Informationen über CI-Typen und Beziehungen.
getConfigurationService	Verwaltung der Infrastruktureinstellungen für die Serverkonfiguration
getDataStoreMgmtService	Abfragen von Datenspeicherinformationen, z. B. welche CIs und Attribute föderiert werden.
getDDMConfigurationService	Konfigurieren der Datenflussverwaltung
getDDMManagementService	Analysieren und Anzeigen des Fortschritts, der Ergebnisse und der Fehler der Datenflussverwaltung
getDDMZoneService	Importieren und Exportieren von Verwaltungszonen (mit ihren Aktivitäten).
getHistoryService	Informationen zur Historie der überwachten CIs (Änderungen, Löschungen usw.)
getImpactAnalysisService	Ausführen des Auswirkungsanalyseszenarios (auch bekannt als **Korrelation**).
getLicensingService	Abfragen von Informationen zu den auf dem System installierten Lizenzen.
getMultipleCMDBService	Konvertieren von globalen IDs und UCMDB-IDs.
getMultiTenancyService	Erstellen, Lesen, Aktualisieren und Löschen von Mandanten.
getPersistencyService	Beibehalten binärer Daten in Schlüssel-Wert-Paaren.
getQueryManagementService	Verwalten des Zugriffs auf Anfragen - Speichern, Löschen, Auflisten vorhandener Elemente. Ermöglicht außerdem die

Servicemethoden	Verwendung
	Prüfung von Abfragen und die Discovery von Abfrageabhängigkeiten.
getReconciliationService	Stellt Funktionen zum Identifizieren und Zusammenführen bereit.
getResourceBundleManagementService	Ressourcen-Tagging (Bundling-Services). Ermöglicht die explizite Erstellung neuer Tags und die Entfernung von Tags von allen getaggten Ressourcen.
getResourceManagementService	Bereitstellen von Ressourcen-Packages (der TQL-Abfragen, Ansichten, Benutzer usw.) für das System.
getSecurityService	Überprüfen der Gültigkeit der Anmeldeinformationen.
getServerService	Abfragen allgemeiner Informationen über das System.
getSnapshotService	Bereitstellen von Services für die Verwaltung von Baselines (Abrufen, Speichern, Vergleichen usw.)
getSoftwareSignatureService	Definieren der vom Datenflussverwaltungssystem zu ermittelnden Softwareelemente.
getStateService	Bereitstellen von Services für die Verwaltung von Status (Auflisten, Hinzufügen, Entfernen usw.)
getSystemHealthService	Bereitstellen von System Health-Services (Indikatoren zur grundlegenden Systemleistung, Kapazitäts- und Verfügbarkeitskennzahlen)
getTopologyQueryService	Abrufen von Informationen über IT Universe.
getTopologyUpdateService	Ändern von Informationen in IT Universe.
getUcmdbVersion	Abfragen der UCMDB- und Content Pack-Versionen sowie der Build-Informationen.
getViewArchiveService	Anzeigen der Ergebnisse der Archivierungsservices. Ermöglicht das Speichern des aktuellen Ansichtsergebnisses und das Abrufen zuvor gespeicherter Ergebnisse.
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.

Der Client kommuniziert mit dem Server über HTTP(S).

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: Es ist auch möglich, eine Verbindung zu einem regulären UCMDB-Benutzer (z. B. admin) herzustellen. Diese Option wird jedoch 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 Serveradresse wie folgt ein:

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

   Eventuell müssen Sie sich mit einem Benutzernamen und einem Kennwort anmelden.
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.

     Die Standard-Kennwortrichtlinie erfordert, dass das UCMDB-Kennwort alle vier der folgenden Zeichentypen enthält:
     - Großbuchstaben
     - Kleinbuchstaben

- Ziffern
- Symbole ,\:/. _?&%=+-[]()|

Außerdem muss das Kennwort eine bestimmte Mindestlänge haben, die in der Einstellung **Mindestlänge des Kennworts** festgelegt wird.

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 **Einstellungen** 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 **systemUser**-Methoden (**removeUser**, **resetPassword**, **UserAuthenticate** usw).

Es gibt zwei vordefinierte Systembenutzer. Es empfiehlt sich, deren Kennwörter nach der Installation mithilfe der Methode **resetPassword** zu ändern.

- **sysadmin/sysadmin**
- **UISysadmin/UISysadmin** (Dieser Benutzer ist auch der **SuperIntegrationUser**)

  Wenn Sie das Kennwort **UISysadmin** mit der Methode **resetPassword** ändern, müssen Sie wie folgt vorgehen:

  i. Suchen Sie in der JMX-Konsole den Service **UCMDB-UI:name=UCMDB Integration**.

  ii. Führen Sie **setCMDBSuperIntegrationUser** mit dem Benutzernamen und dem neuen Kennwort des Integrationsbenutzers aus.

UCMDB-API-Anwendungsfälle

Die in diesem Abschnitt aufgeführten 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/von 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 11: 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" unten
- "Berechtigungen" auf der nächsten Seite

Verwendung der API

Die UCMDB-Webservice-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.

Hinweis: Die Axis2-Verwaltungskonsole ist nicht zugänglich.

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**
calculateImpact getImpactPath getImpactRulesByGroupName getImpactRulesByNamePrefix	Allgemeine Aktion: **Auswirkungsanalyse ausführen**
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Keine

Hinweis: Wenn der Stammkontext in UCMDB geändert wurde, gehen Sie folgendermaßen vor, um den Zugriff auf die Webservice-API zu aktivieren:

1. Öffnen Sie die Konfigurationsdatei **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\web.xml** und suchen Sie den folgenden Abschnitt:

   ```
   <servlet-class>
   org.apache.axis2.transport.http.AxisServlet
   </servlet-class>
   ```

   Fügen Sie am Ende des Abschnitts die folgenden Zeilen hinzu:

   ```
   <init-param>
   <param-name>axis2.find.context</param-name>
   <param-value>false</param-value>
   </init-param>
   ```

2. Öffnen Sie die Konfigurationsdatei **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\conf\axis2.xml** und suchen Sie die folgende Zeile:

   ```
   <parameter name="enableSwA" locked="false">false</parameter>
   ```

   Fügen Sie am Ende der Zeile die folgende Zeile hinzu:

   ```
   <parameter name="contextRoot" locked="false">test1/setup1/axis2
   </parameter>
   ```

   Dabei gilt: **test1/setup1** ist Ihr Stammkontext.

   (Um den Stammkontext zu entfernen, entfernen Sie Text, den Sie zum Pfad hinzugefügt haben.)

3. Starten Sie den UCMDB-Server neu.

Starten des HP Universal CMDB Web Service

Verwenden Sie standardmäßige SOAP-Programmierverfahren in der HP Universal CMDB-Webservice-API, um serverseitige Methoden aufzurufen. 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 Port-Nummer 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 der UCMDB-Webservice-API" auf Seite 362.

Abfragen der CMDB

Die CMDB wird mithilfe der unter "UCMDB-Abfragemethoden" auf Seite 345 beschriebenen APIs abgefragt. Die Abfragen und die zurückgegebenen CMDB-Elemente enthalten grundsätzlich reale UCMDB -IDs. Beispiele für die Verwendung von Abfragemethoden finden Sie unter Query Example.

Dieser Abschnitt umfasst die folgenden Themen:

Just In Time-Antwortberechnung

Für alle Abfragemethoden berechnet der UCMDB-Server die von der Abfragemethode angeforderten Werte, wenn die Anforderung eingeht, und gibt die Ergebnisse auf Basis der neuesten 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 354 und im Abschnitt "releaseChunks" auf Seite 356.

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 CMDB

Sie aktualisieren die CMDB mit den APIs für die Aktualisierung. Weitere Informationen zu den API-Methoden finden Sie unter "UCMDB-Aktualisierungsmethoden" auf Seite 356.

Diese Aufgabe umfasst folgende Schritte:

- "Parameter für die UCMDB-Aktualisierung" unten
- "Verwendung von ID-Typen mit Aktualisierungsmethoden" unten

Parameter für die UCMDB-Aktualisierung

Dieses Thema beschreibt die Parameter, die nur von den Aktualisierungsmethoden des Service verwendet werden.

- **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 in die 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 `clientID` FEHLT. Weitere Informationen finden Sie unter "addCIsAndRelations" auf Seite 356 und "UCMDB-Abfragemethoden" auf Seite 345.

- **Einfügen neuer Beziehungen in die 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*.

Dieser Abschnitt enthält Informationen über die folgenden Methoden, die Informationen über CITs und Beziehungen zurückgeben:

- "getClassAncestors" unten
- "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 Seite 341.
className	Der Typname. Weitere Informationen finden Sie unter "Typname" auf Seite 343.

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" auf der nächsten Seite.
className	Der Typname. Weitere Informationen finden Sie unter "Allgemeine UCMDB-Parameter" auf der nächsten Seite.

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 Impact Analysis Example.

Allgemeine UCMDB-Parameter

In diesem Abschnitt werden die gebräuchlichsten Parameter der Servicemethoden beschrieben.

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 340.

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, so als ob 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 zugewiesene temporäre ID enthalten. Wenn eine temporäre ID verwendet wird, muss das Kennzeichen `temp` auf **true** gesetzt werden. Beim Löschen eines Elements kann `ID` leer sein. "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 341.

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 ID des `Relation`-Elements eine reale CMDB-ID oder eine temporäre ID sein. Beim Löschen eines Elements kann die `ID` leer sein. "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 339.

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 online 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 `Relation`-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 335.

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 341.
queryName	Der Name der parametrisierten TQL in der CMDB, für die die Karte abgerufen werden soll.
parameterizedNodeList	Die Bedingungen, die jeder Knoten erfüllen muss, um in die Abfrageergebnisse einbezogen zu werden.
queryTypedProperties	Eine Sammlung von Eigenschaftssätzen, die für die Elemente eines bestimmten CI-Typs abgerufen werden sollen.

Ausgabe

Parameter	Kommentar
topologyMap	Weitere Informationen finden Sie unter "TopologyMap" auf Seite 344.
chunkInfo	Weitere Informationen finden Sie unter "ChunkInfo" auf der vorherigen Seite und "Verarbeiten großer Antworten" auf Seite 335.

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 `parameterizedNodeList` 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.
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 341.
queryXML	Eine XML-Zeichenkette, die eine TQL ohne Ressourcen-Tags darstellt.
parameterizedNodeList	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 344.
chunkInfo	Weitere Informationen finden Sie unter "ChunkInfo" auf Seite 345 und "Verarbeiten großer Antworten" auf Seite 335.

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 341.
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
getChangedCIsResponseList	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 341.
ID	Die ID des CI, dessen Nachbarn abgerufen werden sollen. Es muss sich hierbei um eine reale CMDB- oder eine globale 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 343.
CIProperties	Die zu jedem Konfigurationselement zurückzugebenden Daten (das sogenannte Abfragelayout in der Benutzeroberfläche). Weitere Informationen finden Sie unter "TypedProperties" auf Seite 338.
relationProperties	Die zu jeder Beziehung zurückzugebenden Daten (das sogenannte Abfragelayout in der Benutzeroberfläche). Weitere Informationen finden Sie unter "TypedProperties" auf Seite 338.

Ausgabe

Parameter	Kommentar
Topologie	Weitere Informationen finden Sie unter "Topology" auf Seite 344.
comments	Nur für den internen Gebrauch.

getCIsByID

Die `getCIsByID`-Methode ruft Konfigurationselemente nach ihren CMDB- oder globalen IDs ab.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
CIsTypedProperties	Eine Sammlung von typisierten Eigenschaften. Weitere Informationen finden Sie unter "Andere Elemente für die Spezifikation von Eigenschaften" auf Seite 337.

Parameter	Kommentar
IDs	In dieser Sammlung sind nur reale CMDB- oder globale IDs gültig.

Ausgabe

Parameter	Kommentar
CIs	Eine Sammlung von CI-Elementen.
chunkInfo	Weitere Informationen finden Sie unter "ChunkInfo" auf Seite 345 und "Verarbeiten großer Antworten" auf Seite 335.

getCIsByType

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 341.
type	Der Klassenname. Weitere Informationen finden Sie unter "Typname" auf Seite 343.
properties	Die zu jedem Konfigurationselement zurückzugebenden Daten. Weitere Informationen finden Sie unter "CustomProperties" auf Seite 337.

Ausgabe

Parameter	Kommentar
CIs	Eine Sammlung von CI-Elementen.
chunkInfo	Weitere Informationen finden Sie unter: "ChunkInfo" auf Seite 345 und "Verarbeiten großer Antworten" auf Seite 335.

getFilteredCIsByType

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-/Kleinschreibung)	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.

Operator	Bedingungstyp / Anmerkungen
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 `Bismark` und `Bay of Biscay`, aber nicht mit `biscuit` überein.
Wie (ohne Groß-/Kleinschreibung)	Zeichenkette Der Operator `Wie (ohne Groß-/Kleinschreibung)` 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(11f);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.FloatOperator.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.setPropertyNames(Arrays.asList("name","size"));
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties);
```

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
type	Der Klassenname. Weitere Informationen finden Sie unter "Typname" auf Seite 343. 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 337.
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 345 und "Verarbeiten großer Antworten" auf Seite 335.

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 341.
viewName	Der Name einer Ansicht, die eine Teilmenge des Klassenmodells in der CMDB ist.

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 341.
queryName	Der Name einer TQL.
queryTypedProperties	Eine Sammlung von Eigenschaftssätzen, die für die Elemente eines bestimmten CI-Typs abgerufen werden sollen.

Ausgabe

Parameter	Kommentar
topologyMap	Weitere Informationen finden Sie unter "TopologyMap" auf Seite 344.
chunkInfo	Weitere Informationen finden Sie unter "ChunkInfo" auf Seite 345 und "Verarbeiten großer Antworten" auf Seite 335.

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 341.
queryName	Der Name einer TQL.
countInvisible	True: Die Ausgabe berücksichtigt auch CIs, die in der Abfrage als unsichtbar definiert wurden.

Ausgabe

Parameter	Kommentar
getTopologyQueryResultCountByNameResponse	Die mit der Abfrage übereinstimmende Instanzenanzahl.

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 345 und "Abfragen der CMDB" auf Seite 335.

Die Client-Applikation muss in der Lage sein, die Teilkarten zu verarbeiten.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
ChunkRequest	Die Anzahl der abzurufenden Chunks und der von der Abfragemethode zurückgegebene Wert des Parameters `ChunkInfo`.
queryTypedProperties	Eine Sammlung von Eigenschaftssätzen, die für die Elemente eines bestimmten CI-Typs abgerufen werden sollen.

Ausgabe

Parameter	Kommentar
topologyMap	Weitere Informationen finden Sie unter "TopologyMap" auf Seite 344.

Parameter	Kommentar
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().getCINodes().size();
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINodes().get(m).getCIs();
            for(int i=0 ; i < cis.getCIs().size(); i++) {
                // your code to process the CIs
            }
```

```
        }
}
```

releaseChunks

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 341.
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 341.
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 vorhanden ist.
CIsAndRelationsUpdates	Die zu aktualisierenden oder zu erstellenden Elemente. Weitere Informationen finden Sie unter "CIsAndRelationsUpdates" auf Seite 338.
ignoreValidation	True: Vor der Aktualisierung der CMDB wird keine Prüfung durchgeführt.
dataStore	Informationen über den Änderer.

Ausgabe

Parameter	Kommentar
createdIDsMapList	Die Liste 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 341.
CIsAndRelationsUpdates	Die zu löschenden Elemente. Weitere Informationen finden Sie unter "CIsAndRelationsUpdates" auf Seite 338
dataStore	Informationen über den Änderer.

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 341.
CIsAndRelationsUpdates	Die zu aktualisierenden Elemente. Weitere Informationen finden Sie unter "CIsAndRelationsUpdates" auf Seite 338.
ignoreValidation	True: Vor der Aktualisierung der CMDB wird keine Prüfung durchgeführt.
dataStore	Informationen über den Änderer.

Ausgabe

Parameter	Kommentar
createdIDsMapList	Die Liste der Client-IDs und CMDB-IDs. Weitere Informationen finden Sie unter "addCIsAndRelations" auf Seite 356.

UCMDB-Methoden zur Auswirkungsanalyse

Dieser Abschnitt enthält Informationen zu den folgenden Methoden:

- "calculateImpact" unten
- "getImpactPath" auf der nächsten Seite
- "getImpactRulesByNamePrefix" auf der nächsten Seite

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" auf der nächsten Seite verwendet.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
impactCategory	Der Typ des Ereignisses, das die simulierte Regel auslösen würde.
IDs	Eine Sammlung von CMDB- oder globalen 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 344.
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 Methode "calculateImpact" auf der vorherigen Seite wird als `identifier`-Eingabeargument der `getImpactPath`-Methode verwendet.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
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 341.
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 der UCMDB-Webservice-API

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

Siehe folgende Codebeispiele:

- The Example Base Class
- Query Example
- Update Example

- Class Model Example
- Impact Analysis Example

Diese Dateien befinden sich im folgenden Verzeichnis:

\\<UCMDB-Stammverzeichnis>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\WebServiceAPI_Samples

Kapitel 12: Java API zur Datenflussverwaltung

Dieses Kapitel umfasst folgende Themen:

Verwendung der Java-API zur Datenflussverwaltung

Hinweis: Verwenden Sie dieses Kapitel zusammen mit der DFM API-Java-Dokumentation, die in der Online-Dokumentationsbibliothek zur Verfügung steht.

In diesem Kapitel wird beschrieben, wie Sie mit benutzerdefinierten Werkzeugen oder Werkzeugen von Drittanbietern die Java-API von HP Datenflussverwaltung zur Datenflussverwaltung nutzen können. Die API stellt Methoden für folgende Aufgaben bereit:

- **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.

Die folgenden Services sind im Discovery Services-Package verfügbar:

- **DDMConfigurationService.** Services für die Konfiguration von Data Flow Probes, Clustern, IP-Bereichen und Anmeldeinformationen. Der Universal Discovery-Server kann mit einer XML-Datei oder über die Data Flow Probe konfiguriert werden.
- **DDMManagementService.** Services für die Analyse und Anzeige des Fortschritts, der Ergebnisse und der Fehler bei der Ausführung von Universal Discovery.
- **DDMSoftwareSignatureService.** Services für die Definition der Softwareelemente, die über eine Discovery von den Data Flow Probe-Komponenten ermittelt werden sollen. Die Definitionen gelten systemweit. Wenn mehrere Data Flow Probe-Komponenten definiert sind, gelten die Definitionen für alle Komponenten.
- **DDMZoneService.** Services für die Verwaltung der zonenbasierten Discovery.

Zusätzlich zu diesen Services gibt es Client-APIs zur Datenflussverwaltung, die bei der Erstellung von Jython-Adaptern verwendet werden. Weitere Informationen finden Sie unter "Entwickeln von Jython-Adaptern" auf Seite 37.

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.

Der Benutzer muss zusätzlich die Berechtigung für die allgemeine Aktion **Zugriff auf SDK** besitzen, um sich anmelden zu können.

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 327.

Kapitel 13: Datenflussverwaltungswebservice-API

Dieses Kapitel umfasst folgende Themen:

Datenflussverwaltungswebservice-API – Übersicht

In diesem Kapitel wird beschrieben, wie Sie mit benutzerdefinierten Werkzeugen oder Werkzeugen von Drittanbietern die Webservice-API der HP Datenflussverwaltung für die Verwaltung von Datenflüssen nutzen können.

Die Webservice-API der HP Datenflussverwaltung wird für die Integration von Applikationen mit HP Universal CMDB verwendet. 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

Hinweis:

- Ein Benutzer muss die Berechtigung für die allgemeine Aktion **Legacy-API ausführen** besitzen, um sich anmelden zu können.
- Für den Zugriff auf die Methoden muss der angemeldete Benutzer die Berechtigung für die generelle Aktion **Discovery und Integrationen ausführen** besitzen.

Die vollständige Dokumentation zu den verfügbaren Operationen finden Sie in der *Schemareferenz zu HP Universal Discovery*. 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.

Starten des HP Data Flow Management Web Service

Die Webservice-API der HP Datenflussverwaltung 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 und Datenstrukturen der Datenflussverwaltung

In diesem Abschnitt werden die Methoden und Datenstrukturen der Webservice-API der Datenflussverwaltung beschrieben. Darüber hinaus enthält der Abschnitt eine Übersicht über ihre Verwendungsbereiche. Die vollständige Dokumentation zu den Anforderungen und Antworten für jede Operation finden Sie in der *Schemareferenz zu HP Universal Discovery*.

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 Universal Discovery*.

IPList

Eine Liste der `IP`-Elemente, von denen jedes eine IPv4- oder eine IPv6-Adresse enthält.

Weitere Informationen finden Sie unter IPList in der *Schemareferenz zu HP Universal Discovery*.

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- oder IPv6-Adresse.

Weitere Informationen finden Sie unter IPRange in der *Schemareferenz zu HP Universal Discovery*.

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 Universal Discovery*.

Verwalten von Discovery-Job-Methoden

activateJob

Aktiviert den angegebenen Job.

Siehe "Codebeispiel" auf Seite 379.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
JobName	Der Name des Jobs.

deactivateJob

Deaktiviert den angegebenen Job.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
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 341.
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 341.

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 341.
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 341.
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 341.
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 341.
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 341.
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 341.
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 341.
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 341.
domainName	Der Name der Domäne.

Ausgabe

Parameter	Kommentar
domainType	Der Domänentyp.

getDomainsNames

Gibt die Namen der aktuellen Domänen zurück.

Siehe "Codebeispiel" auf Seite 379.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.

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 341.
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 379.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
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 341.
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 379.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
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 341.
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 379.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
domainName	Die zu aktualisierende Domäne.
protocolName	Der Name des Protokolls.
credentialsEntryParameters	Die Sammlung "CIProperties" 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 341.
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 341.
domainName	Die Domäne, für die die Anmeldeinformationen abgerufen werden.
protocolName	Der Name eines Protokolls, das in dieser Domäne verwendet wird.
credentialsEntryID	Die abzurufende ID der Anmeldeinformationen.

Ausgabe

Parameter	Kommentar
credentialsEntryParameters	Die Sammlung "CIProperties" der Anmeldeinformationen.

removeCredentialsEntry

Entfernt die angegebenen Anmeldeinformation aus dem Protokoll.

Eingabe

Parameter	Kommentar
cmdbContext	Weitere Informationen finden Sie unter "CmdbContext" auf Seite 341.
domainName	Die Domäne.
protocolName	Der Name eines Protokolls, das in der Domäne verwendet wird.
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 341.
domainName	Die Domäne, in der Anmeldeinformationen aktualisiert werden sollen.
protocolName	Der Name eines Protokolls, das in der Domäne verwendet wird.
credentialsEntryID	Die ID der zu aktualisierenden Anmeldeinformationen.
credentialsEntryParameters	Die Sammlung "CIProperties", 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 341.
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 341.
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 341.
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 341.
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 credentials entry for ntcmd protocol
        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;
    }
}
```

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 credentials entry for ntcmd protocol
        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
```

Senden von Feedback zur Dokumentation

Wenn Sie Anmerkungen zu diesem Dokument haben, können Sie sich per E-Mail an das Dokumentationsteam wenden. Wenn ein E-Mail-Client auf diesem System konfiguriert ist, klicken Sie auf den Link weiter oben und es wird ein E-Mail-Fenster mit folgendem Betreff geöffnet:

Feedback zu Entwicklerreferenzhandbuch (Universal CMDB10.20)

Schreiben Sie einfach Ihr Feedback in die E-Mail und klicken Sie auf die Option zum Senden.

Wenn kein E-Mail-Client verfügbar ist, kopieren Sie die oben genannten Informationen in einen Web-Mail-Client und senden Sie Ihr Feedback an cms-doc@hp.com.

Wir freuen uns auf Ihr Feedback!