HP Universal CMDB

Voor de besturingssystemen Windows en Red Hat Enterprise Linux

Softwareversie: 10.00

Referentiehandleiding voor ontwikkelaars

Release-datum van document: juni 2012

Release-datum van software: juni 2012

Wettelijke kennisgevingen

Garantie

De enige garanties voor HP-producten en -services worden uiteengezet in de uitdrukkelijke garantieverklaringen die worden geleverd bij de betreffende producten en services. De inhoud van dit document kan op geen enkele wijze worden aangemerkt als een aanvullende garantie. HP is niet aansprakelijk voor technische of redactionele fouten in dit document.

De informatie in dit document kan zonder voorafgaande kennisgeving worden gewijzigd.

Legenda voor beperkte rechten

Vertrouwelijke computersoftware. Geldige licentie van HP vereist voor bezit, gebruik of kopieën. In overeenstemming met FAR 12.211 en 12.212 worden commerciële computersoftware, documentatie voor computersoftware en technische gegevens voor commerciële items in licentie gegeven aan de regering van de VS onder de commerciële standaardlicentie van de verkoper.

Copyright-kennisgeving

Handelsmerk-kennisgevingen

Adobe™ is een handelsmerk van Adobe Systems Incorporated.

Microsoft® en Windows® zijn gedeponeerde handelsmerken van Microsoft Corporation in de Verenigde Staten.

UNIX® is een geregistreerd handelsmerk van The Open Group.

Documentatie-updates

De titelpagina van dit document bevat de volgende identificatiegegevens:

- Versienummer van software, waarmee de softwareversie wordt aangegeven.
- Release-datum van document, die na elke update van het document wordt gewijzigd.
- Release-datum van software, waarmee de release-datum van deze versie van de software wordt aangegeven.

Als u wilt controleren of er recente updates beschikbaar zijn of wilt controleren of u de meest recente versie van een document gebruikt, gaat u naar:

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

Als u toegang wilt tot deze site, moet u zich aanmelden voor een HP Passport en zich aanmelden. Als u zich wilt aanmelden voor een HP Passport-ID, gaat u naar:

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

U kunt eventueel ook klikken op de koppeling **New users - please register** (Nieuwe gebruikers - Aanmelden) op de aanmeldingspagina voor HP Passport.

U ontvangt ook bijgewerkte of nieuwe versies als u zich abonneert op de ondersteuningsservice voor het desbetreffende product. Neem contact op met uw HP-vertegenwoordiger voor meer informatie.

Ondersteuning

Ga naar de website van HP Software Support Online op:

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

Op deze website vindt u contactinformatie en details over de producten, services en ondersteuning die HP Software biedt.

In de online ondersteuning van HP Software vindt u methoden waarmee klanten zelf problemen kunnen oplossen. Hiermee krijgt u snel en efficiënt toegang tot interactieve tools voor technische ondersteuning die u nodig hebt om uw bedrijf te kunnen beheren. Als gewaardeerde ondersteuningsklant kunt op de ondersteuningssite profiteren van de volgende mogelijkheden:

- Interessante kennisdocumenten zoeken
- Ondersteuningscases en verbeteringsaanvragen indienen en volgen
- Softwarepatches downloaden
- Ondersteuningscontracten beheren
- Contactpersonen van HP opzoeken voor ondersteuning
- Informatie over beschikbare services bekijken
- Discussies voeren met andere softwareklanten
- Softwaretrainingen bekijken en u hiervoor aanmelden

Voor de meeste ondersteuningssecties moet u zich registreren als HP Passport-gebruiker en u vervolgens aanmelden. Voor verschillende secties moet u verder beschikken over een ondersteuningscontract. Om u te registreren voor een HP Passport-ID, gaat u naar:

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

Als u meer informatie wilt over toegangsniveaus, gaat u naar:

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

Vrijwaring voor PDF-versie van Online Help

Dit document is een PDF-versie van de Online Help. Dit PDF-bestand wordt meegeleverd zodat u eenvoudig meerdere onderwerpen uit de Help-informatie kunt afdrukken en zodat u de Online Help in PDF-formaat kunt lezen.

Opmerking: bepaalde onderwerpen kunnen niet goed naar PDF-formaat worden geconverteerd, waardoor er opmaakproblemen ontstaan. Sommige onderdelen van de Online Help ontbreken volledig in de PDF-versie. Deze onderwerpen kunnen wel worden afgedrukt vanuit de Online Help.

Inhoud

Discovery- en integratieadapters aanmaken

Hoofdstuk 1

Adapters ontwikkelen en schrijven

In dit hoofdstuk vindt u de volgende informatie:

Overzicht Adapters ontwikkelen en schrijven

Alvorens met de werkelijke planning voor de ontwikkeling van nieuwe adapters te beginnen, is het van belang dat u inzicht hebt in de processen en interacties die doorgaans aan deze ontwikkeling zijn gekoppeld.

In de volgende gedeelten leert u wat u moet weten en doen om met succes een discovery-ontwikkelingsproject te beheren en uit te voeren.

In dit gedeelte vindt u:

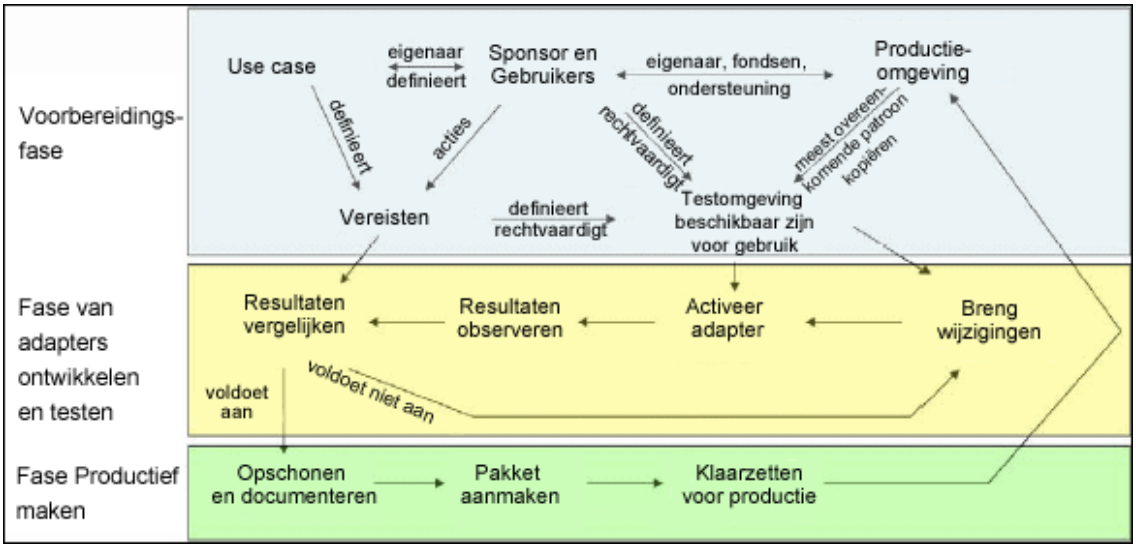
- Wordt ervan uitgegaan dat u beschikt over praktische kennis van HP Universal CMDB en enigszins vertrouwd bent met de basiselementen van het systeem. Het is bedoeld als hulpmiddel in het leerproces, maar vormt geen volledige handleiding.
- Worden de verschillende fasen voor planning, onderzoek en implementatie van nieuwe discovery-inhoud voor HP Universal CMDB behandeld. Daarnaast vindt u in dit hoofdstuk richtlijnen en overwegingen waarmee u rekening dient te houden.
- Vindt u informatie over de belangrijke API's van het Data Flow-beheer Framework. Raadpleeg de *HP Universal CMDB - API-referentie Data Flow-beheer* voor volledige documentatie over de beschikbare API's. (Er bestaan andere niet-officiële API's, maar deze kunnen worden gewijzigd zelfs als ze voor meegeleverde adapters worden gebruikt.)

Inhoud aanmaken

In dit gedeelte worden de volgende onderwerpen behandeld:

Adapterontwikkelingscyclus

In de volgende afbeelding ziet u een stroomdiagram voor het schrijven van adapters. De meeste tijd wordt besteed in het middelste gedeelte. Dit is de iteratieve lus voor ontwikkeling en tests.

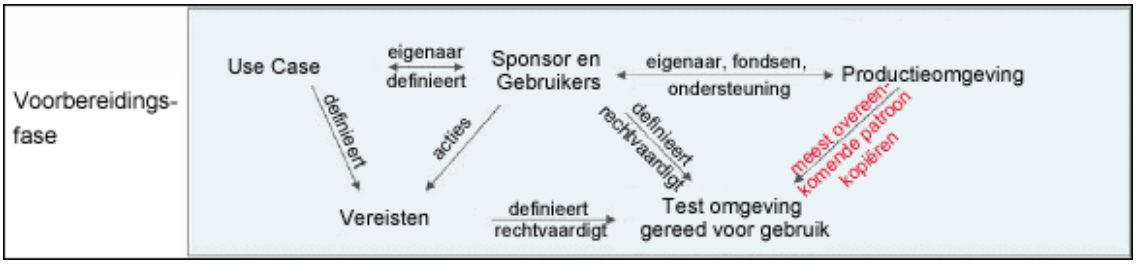


Elke fase van adapterontwikkeling gaat verder op basis van de vorige fase.

Wanneer u tevreden bent met de weergave en de werking van de adapter, kunt u de adapter in pakketten plaatsen. Maak een ZIP-pakketbestand met UCMDB Pakketbeheer of door de componenten handmatig te exporteren. U kunt dit pakket het beste op een ander UCMDB-systeem uitrollen en testen alvorens het vrij te geven voor productie. Zo verzekert u zich ervan dat alle componenten met succes in pakketten worden geplaatst. Raadpleeg "Pakketbeheer" in de *HP Universal CMDB – Handleiding Beheer* voor meer informatie over pakketten.

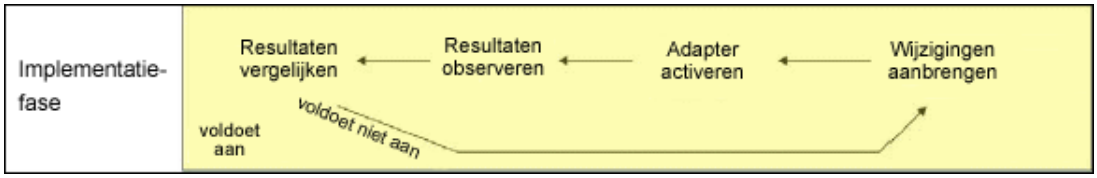
In de volgende gedeelten wordt elke fase gedetailleerder besproken en worden de belangrijkste stappen en best practices getoond:

Fase van onderzoek en voorbereiding



De **fase van onderzoek en voorbereiding** omvat de belangrijkste bedrijfsbehoeften en use cases. In deze fase wordt tevens gezorgd voor de noodzakelijke uitrusting om de adapter te ontwikkelen en te testen.

1. Wanneer u van plan bent een bestaande adapter te wijzigen, moet u als eerste technische stap een back-up maken van de betreffende adapter en controleren of u de adapter in de oorspronkelijke staat kunt herstellen. Als u een nieuwe adapter wilt aanmaken, kopieert u de adapter die er het meest op lijkt en slaat u deze onder een geschikte naam op. Zie "Het deelvenster Bronnen" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.
2. Onderzoek hoe de adapter gegevens moet verzamelen:
   - Externe tools/protocollen gebruiken om de gegevens te verkrijgen
   - Ontwikkelen hoe de adapter CI's moet aanmaken op basis van de gegevens
   - U weet nu hoe een vergelijkbare adapter eruit moet zien
3. Bepaal de meest gelijkende adapter op basis van het volgende:
   - Dezelfde aangemaakte CI's
   - Dezelfde gebruikte protocollen (SNMP)
   - Hetzelfde soort doelen (per type besturingssysteem, versies, enzovoort)
4. Kopieer het gehele pakket.
5. Pak het pakket in de werkruimte uit en wijzig de naam van de adapter- (XML) en Jython-bestanden (.PY).



Adapters ontwikkelen en testen

De **fase voor het ontwikkelen en testen van adapters** is een uitermate iteratief proces. Wanneer de adapter vorm begint aan te nemen, voert u eerst een test voor de uiteindelijke use cases uit, brengt u wijzigingen aan en voert u nogmaals een test uit. U herhaalt dit proces totdat de adapter voldoet aan de vereisten.

Kopie opstarten en voorbereiden

- Wijzig XML-onderdelen van de adapter: naam (id) op regel 1, aangemaakte CI-typen en naam aangeroepen Jython-script.
- Zorg ervoor dat de kopie wordt uitgevoerd met dezelfde resultaten als de oorspronkelijke adapter.
- Maak commentaarregels van het grootste gedeelte van de code, met name van de essentiële resultaatproducerende code.

Ontwikkelen en testen

- Gebruik andere voorbeeldcode om wijzigingen te ontwikkelen.
- Test de adapter door deze uit te voeren.
- Gebruik een speciale weergave om complexe resultaten te valideren. Voer een zoekopdracht uit om eenvoudige resultaten te valideren.

Adapters in pakketten plaatsen en productief maken

De **fase waarin adapters in pakketten worden geplaatst en productief worden gemaakt** is de laatste ontwikkelingsfase. Het is verstandig een laatste controle uit te voeren om foutopsporingsresten, documenten en opmerkingen op te schonen, beveiligingsoverwegingen te bekijken en dergelijke, alvorens verder te gaan en de adapters in pakketten te plaatsen. U moet in ieder geval altijd een Leesmij-document maken om de werking van de adapter uit te leggen. Als iemand (misschien uzelf wel) in de toekomst eens naar deze adapter moet kijken, zal deze persoon zeer geholpen zijn met zelfs de geringste documentatie.

Opschonen en documenteren

- Verwijder foutopsporing.
- Voorzie alle functies van commentaar en voeg enkele opmerkingen aan het begin van het hoofdgedeelte toe.
- Maak een voorbeeld-TQL aan en geef deze weer zodat de gebruiker deze kan testen.

Pakket aanmaken

- Exporteer adapters, TQL, enzovoort met Pakketbeheer. Zie "Pakketbeheer" in de *HP Universal CMDB – Handleiding Beheer* voor meer informatie over dit onderwerp.
- Controleer alle afhankelijkheden die uw pakket met andere pakketten heeft, bijvoorbeeld of de CI's die door deze pakketten zijn aangemaakt, invoer-CI's voor uw adapter zijn.
- Maak een pakketzipbestand met Pakketbeheer. Zie "Pakketbeheer" in de *HP Universal CMDB – Handleiding Beheer* voor meer informatie over dit onderwerp.
- Test ontwikkeling door delen van de nieuwe inhoud te verwijderen en opnieuw uit te rollen of een ander testsysteem uit te rollen.

Data Flow-beheer en -integratie

DFM-adapters kunnen met andere producten worden geïntegreerd. Bekijk de volgende definities:

- In DFM wordt specifieke inhoud van vele doelen verzameld.
- Met integratie worden meerdere typen inhoud van één systeem verzameld.

Houd er rekening mee dat bij deze definities geen onderscheid wordt gemaakt tussen de verzamelmethoden. Dat gebeurt in DFM ook niet. Het ontwikkelproces voor een nieuwe adapter is hetzelfde als het ontwikkelproces voor nieuwe integratie. U doet hetzelfde onderzoek, maakt dezelfde keuzen voor nieuwe dan wel bestaande adapters, schrijft de adapters op dezelfde manier, enzovoort. Er zijn maar een paar dingen anders:

- De planning van de definitieve adapter. Integratie-adapters worden mogelijk vaker uitgevoerd dan discovery-adapters, maar dat is afhankelijk van de use cases.
- Invoer-CI's:
  - Integratie: niet-getriggerde CI's die moeten worden uitgevoerd zonder invoer: een bestandsnaam of bron die via de adapterparameter wordt doorgegeven.
  - Discovery: gebruikt gewone, CMDB-CI's voor invoer.

Voor integratieprojecten moet u altijd een bestaande adapter hergebruiken. De richting van de integratie (van HP Universal CMDB naar een ander product of van een ander product naar HP Universal CMDB) kan van invloed zijn op de manier waarop u ontwikkeling benadert. U kunt gebruikmaken van veldpakketten die u voor eigen gebruik kunt kopiëren met behulp van beproefde methoden.

Van HP Universal CMDB naar een ander project:

- Maak een TQL aan waarmee de te exporteren CI's en relaties worden geproduceerd.
- Gebruik een generieke wrapperadapter om de TQL uit te voeren en schrijf de resultaten naar een XML-bestand voor het externe product dat moet worden gelezen.

> **Opmerking:** neem voor voorbeelden van veldpakketten contact op met HP Software Support.

Om een ander product te integreren in HP Universal CMDB is de werking van de integratie-adapter afhankelijk van de wijze waarop gegevens door het andere product worden vrijgegeven:

Integratietype	Opnieuw te gebruiken referentievoorbeeld
Rechtstreeks toegang krijgen tot de database van het product	HP ED
Een door een export geproduceerd CSV- of XML-bestand inlezen	HP ServiceCenter
Toegang krijgen tot de API van een product	BMC Atrium/Remedy

Bedrijfswaarde koppelen aan discovery-ontwikkeling

De use case voor het ontwikkelen van nieuwe discovery-inhoud moet worden aangestuurd door een business-case en -plan om bedrijfswaarde te produceren. Dat wil zeggen: het doel van het toewijzen van systeemcomponenten aan CI's en het toevoegen ervan aan de CMDB is verschaffen van bedrijfswaarde.

De inhoud kan niet altijd worden gebruikt voor toewijzing van applicaties, hoewel dit een vaak voorkomende tussentijdse stap is voor vele use cases. Ongeacht het eindgebruik van de inhoud moet uw plan antwoord geven op de volgende vragen van deze benadering:

- Wie is de gebruiker? Hoe moet de gebruiker reageren op de informatie die door de CI's wordt verschaft (en de relaties ertussen)? Wat is de zakelijke context waarin de CI's en relaties moeten worden bekeken? Is de gebruiker van deze CI's een persoon, een product of beide?
- Als ik de perfecte combinatie van CI's en relaties in de CMDB heb, hoe plan ik dan het gebruik ervan om bedrijfswaarde te produceren?
- Hoe dient de perfecte toewijzing eruit te zien?
  - Met welke term wordt het beste de betekenis weergegeven van de relaties tussen elk CI?
  - Welke CI-typen zijn het belangrijkst om op te nemen?
  - Wat is het eindgebruik en wie is de eindgebruiker van de kaart?
- Wat is de perfecte rapportindeling?

Na de zakelijke acceptatie is de volgende stap de bedrijfswaarde in een document op te nemen. Dit houdt in de perfecte kaart af te beelden met een tekentool en inzicht te verkrijgen in de impact en afhankelijkheden tussen CI's, rapporten, hoe wijzigingen worden bijgehouden, welke wijziging van belang is, controle, compliance en aanvullende bedrijfswaarde zoals vereist voor de use cases.

Naar deze tekening (of dit model) wordt verwezen als de **blauwdruk**.

Als het bijvoorbeeld voor de applicatie van belang is te weten wanneer een configuratiebestand is veranderd, moet het bestand worden toegewezen en gekoppeld aan het desbetreffende CI (waaraan het is gerelateerd) in de getekende kaart.

Maak gebruik van een SME (Subject Matter Expert) op het betreffende gebied, die de eindgebruiker is van de ontwikkelde inhoud. Deze expert moet de essentiële entiteiten (CI's met attributen en relaties) naar voren brengen die de CMDB moet bevatten om bedrijfswaarde op te leveren.

Eén methode kan eruit bestaan een vragenlijst aan de eigenaar van de applicatie voor te leggen (tevens de SME in dit geval). De eigenaar moet de bovenstaande doelstellingen en blauwdruk kunnen specificeren. In ieder geval moet de eigenaar een actuele architectuur van de applicatie kunnen verschaffen.

U moet alleen belangrijke gegevens en geen onnodige gegevens toewijzen: u kunt de adapter later altijd nog uitbreiden. Met de doelstelling moet een beperkte discovery worden ingesteld die werkt en waarde verschaft. Toewijzing van grote hoeveelheden gegevens levert kaarten op die meer indruk wekken, maar de ontwikkeling ervan kan verwarrend en tijdrovend zijn.

Zodra het model en de bedrijfswaarde duidelijk zijn, gaat u verder met de volgende fase. Deze fase kan worden herhaald aangezien concretere informatie uit de volgende fasen wordt verschaft.

Integratievereisten onderzoeken

De voorwaarde voor deze fase is een **blauwdruk** van de CI's en relaties die door DFM moeten worden gedetecteerd. Deze blauwdruk moet de te detecteren attributen omvatten. Zie "Overzicht Adapters ontwikkelen en schrijven" op pagina 14 voor meer informatie over dit onderwerp.

In dit gedeelte vindt u de volgende onderwerpen:

Een bestaande adapter wijzigen

U wijzigt een bestaande adapter wanneer een meegeleverde adapter of een veldadapter aanwezig is, maar houd rekening met het volgende:

- Specifieke benodigde attributen worden niet gedetecteerd.
- Een specifiek type doel (besturingssysteem) wordt niet gedetecteerd of wordt onjuist gedetecteerd.
- Er wordt geen specifieke relatie gedetecteerd of aangemaakt.

Als een bestaande adapter slechts een deel van het werk verricht, moet u in de eerste plaats de bestaande adapters evalueren en nagaan of een van deze adapters bijna doet wat nodig is. Als dat het geval is, kunt u de bestaande adapter wijzigen.

U dient ook te evalueren of er een bestaande veldadapter beschikbaar is. Veldadapters zijn discovery-adapters die beschikbaar zijn maar niet zijn meegeleverd. Neem contact op met HP Software Support om de actuele lijst met veldadapters te ontvangen.

Een nieuwe adapter schrijven

In de volgende gevallen moet een nieuwe adapter worden ontwikkeld:

- Als het sneller gaat een adapter te schrijven dan de informatie handmatig in de CMDB in te voeren (over het algemeen vanaf ongeveer 50 tot 100 CI's en relaties) of als het geen eenmalige actie betreft.
- Als de noodzaak de inspanning rechtvaardigt.
- Als er geen meegeleverde of veldadapters beschikbaar zijn.
- Als de resultaten kunnen worden hergebruikt.
- Als de doelomgeving of de bijbehorende gegevens beschikbaar zijn (u kunt niet detecteren wat u niet kunt zien).

Modelonderzoek

- Blader in het UCMDB-klassemodel (CI-typebeheer) en vergelijk de entiteiten en relaties van uw **blauwdruk** met bestaande CIT's. Het wordt ten zeerste aanbevolen bij het huidige model te blijven om mogelijke complicaties tijdens versie-upgrade te voorkomen. Als u het model moet uitbreiden, moet u nieuwe CIT's aanmaken aangezien meegeleverde CIT's kunnen worden overschreven bij een upgrade.
- Als bepaalde entiteiten, relaties of attributen in het huidige model ontbreken, moet u ze aanmaken. Het verdient de voorkeur een pakket aan te maken met deze CIT's (die later tevens alle discovery, weergaven en andere aan dit pakket gerelateerde artefacten omvat) aangezien u deze CIT's bij elke installatie van HP Universal CMDB moet kunnen uitrollen.

Technologieonderzoek

Nadat u hebt gecontroleerd of de CMDB de relevante CI's bevat, bestaat de volgende fase eruit te bepalen hoe deze gegevens uit de relevante systemen moeten worden opgehaald.

Het ophalen van gegevens betreft gewoonlijk het met een protocol toegang krijgen tot een beheerdeel van de applicatie, werkelijke gegevens van de applicatie of configuratiebestanden of databases die aan de applicatie zijn gerelateerd. Elke gegevensbron die informatie over een systeem kan verschaffen is waardevol. Technologieonderzoek vereist naast uitgebreide kennis van het desbetreffende systeem soms ook creativiteit.

Voor zelfgebouwde applicaties kan het handig zijn een vragenformulier aan de eigenaar van de applicatie te verstrekken. In dit formulier moet de eigenaar alle gebieden in de applicatie vermelden die voor de blauwdruk en bedrijfswaarden benodigde informatie kunnen verschaffen. Deze informatie kan onder andere beheerdatabases, configuratiebestanden, logbestanden, beheerinterfaces, administratieprogramma's, webservices, verzonden berichten of events omvatten.

Voor standaardproducten moet u zich richten op documentatie, forums, of ondersteuning van het product. Zoek naar beheerdershandleidingen, invoegtoepassingen, integratiehandleidingen, beheerhandleidingen, enzovoort. Als er nog steeds gegevens ontbreken in de beheerinterfaces, bekijkt u de configuratiebestanden van de applicatie, registeritems, logbestanden, NT-eventlogboeken en alle artefacten van de applicatie die zorgen voor correcte werking.

Richtlijnen om de juiste methode van toegang tot gegevens te kiezen

Relevantie: selecteer bronnen of een combinatie van bronnen die de meeste gegevens verschaffen. Als één bron de meeste informatie verschaft terwijl de rest van de informatie verspreid is of moeilijk te benaderen, probeert u te beoordelen wat de waarde van de resterende informatie is tegenover de inspanning of het risico om deze informatie te verkrijgen. Soms kunt u besluiten de blauwdruk te beperken als de waarde of de kosten niet in verhouding staan tot de geïnvesteerde moeite.

Hergebruik: als HP Universal CMDB al een specifieke verbindingsprotocolondersteuning omvat, is dat een goede reden om die te gebruiken. Het houdt in dat het DFM Framework een gereed gemaakte client en configuratie voor de verbinding moet verschaffen. Anders moet u wellicht in infrastructuurontwikkeling investeren. U kunt de momenteel ondersteunde HP Universal CMDB-verbindingsprotocollen bekijken: **Data Flow-beheer > Instellingen Data Flow-probe> deelvenster Domeinen en probes**. Zie "Deelvenster Domeinen en probes" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

U kunt nieuwe protocollen toevoegen door nieuwe CI's aan het model toe te voegen. Neem contact op met HP Software Support voor meer informatie.

Opmerking: om toegang te krijgen tot Windows-registergegevens kunt u WMI of NTCmd gebruiken.

Beveiliging: voor toegang tot informatie zijn gewoonlijk referenties (gebruikersnaam, wachtwoord) vereist, die in de CMDB worden ingevoerd en in het hele product beveiligd zijn. Kies indien mogelijk en als beveiliging niet in strijd is met andere principes die u hebt ingesteld, de minst gevoelige referenties of protocollen die wel voldoen aan toegangsbehoeften. Als informatie bijvoorbeeld zowel via JMX (standaardbeheerinterface, beperkt) als Telnet beschikbaar is, verdient het de voorkeur JMX te gebruiken aangezien hiermee beperkte toegang wordt overgenomen en (meestal) geen toegang tot het onderliggende platform.

Gemak: sommige beheerinterfaces kunnen geavanceerdere functies omvatten. Het kan bijvoorbeeld gemakkelijker zijn query's uit te geven (SQL, WMI) dan te navigeren door boomstructuren met informatie of reguliere expressies op te bouwen voor parseren.

Ontwikkelaarsdoelgroep: de personen die uiteindelijk adapters ontwikkelen, kunnen neigen naar een bepaalde technologie. Hiermee moet ook rekening worden gehouden als twee technologieën vrijwel dezelfde informatie verschaffen tegen gelijke kosten op andere gebieden.

Samenvatting

Het resultaat van deze fase is een document waarin de toegangsmethoden worden beschreven en de relevante informatie die van elke methode kan worden afgeleid. Het document moet ook een toewijzing vanuit elke bron naar alle relevante blauwdrukgegevens bevatten.

Elke toegangsmethode moet volgens de bovenstaande instructies worden gemarkeerd. Ten slotte moet u nu beschikken over een plan van de bronnen die moeten worden gedetecteerd en welke informatie vanuit elke bron moet worden geëxtraheerd in het blauwdrukmodel (dat nu moet zijn toegewezen aan het corresponderende UCMDB-model).

Integratie-inhoud ontwikkelen

Voordat u een nieuwe integratie aanmaakt, moet u weten wat de vereisten van de integratie zijn:

- Moeten gegevens naar de CMDB worden gekopieerd met de integratie? Moeten de gegevens per geschiedenis worden bijgehouden? Is de bron onbetrouwbaar?

  Vulling is vereist.

- Moeten gegevens zonder onderbreking worden gefedereerd voor weergaven en TQL-query's met de integratie? Is nauwkeurigheid van wijzigingen in gegevens essentieel? Is de hoeveelheid gegevens te groot om naar de CMDB te kopiëren, maar is de aangevraagde hoeveelheid gegevens meestal klein?

  Federation is vereist.

- Moeten gegevens naar externe gegevensbronnen worden gepusht met de integratie?

  Datapush is vereist.

> **Opmerking:** voor optimale flexibiliteit kunnen Federation- en Vulling-stromen voor dezelfde integratie worden geconfigureerd.

Raadpleeg "Integration Studio" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over de verschillende integratietypen.

Er zijn vier verschillende opties beschikbaar om integratie-adapters aan te maken:

1. Jython-adapter
   - Het klassieke discovery-patroon
   - Geschreven in Jython
   - Gebruikt voor vulling

   Zie "Jython-adapters ontwikkelen" op pagina 40 voor meer informatie over dit onderwerp.
2. Java-adapter
   - Een adapter waarmee een van de adapterinterfaces in het Federation SDK Framework wordt geïmplementeerd.
   - Kan voor een of meer Federation-, Vulling- of Datapush-items worden gebruikt (afhankelijk van de vereiste implementatie).
   - Vanaf het begin geschreven in Java, zodat code kan worden geschreven waarmee verbinding kan worden gemaakt met alle mogelijke bronnen of doelen.
   - Geschikt voor taken die allemaal één gegevensbron of -doel verbinden.

   Zie "Java-adapters ontwikkelen" op pagina 156 voor meer informatie over dit onderwerp.
3. Algemene DB-adapter
   - Een abstracte adapter gebaseerd op de Java-adapter; gebruikt het Federation SDK Framework.
   - Hiermee kunnen adapters worden aangemaakt die verbinding maken met externe opslagplaatsen voor gegevens.
   - Ondersteunt zowel Federation als Vulling (met een Java-invoegtoepassing die voor ondersteuning van wijzigingen is geïmplementeerd).
   - Relatief eenvoudig te definiëren, aangezien deze hoofdzakelijk is gebaseerd op XML- en eigenschapsconfiguratiebestanden.
   - Hoofdconfiguratie is gebaseerd op een **Orm.xml**-bestand dat UCMDB-klassen en databasekolommen koppelt.
   - Geschikt voor taken die allemaal één gegevensbron verbinden.

   Zie "Algemene database-adapters ontwikkelen" op pagina 84 voor meer informatie over dit onderwerp.
4. Algemene push-adapter
   - Een abstracte adapter die is gebaseerd op de Java-adapter (het Federation SDK Framework) en de Jython-adapter.

- Hiermee kunnen adapters worden aangemaakt die gegevens naar externe doelen pushen.
- Relatief eenvoudig te definiëren, aangezien u alleen de toewijzing tussen UCMDB-klassen en XML hoeft te definiëren en een Jython-script waarmee de gegevens naar het doel worden gepusht.
- Geschikt voor taken die elk met één gegevensdoel verbinden.
- Gebruikt voor datapush.

Zie "Push-adapters ontwikkelen" op pagina 185 voor meer informatie over dit onderwerp.

In de volgende tabel worden de mogelijkheden van elke adapter weergegeven:

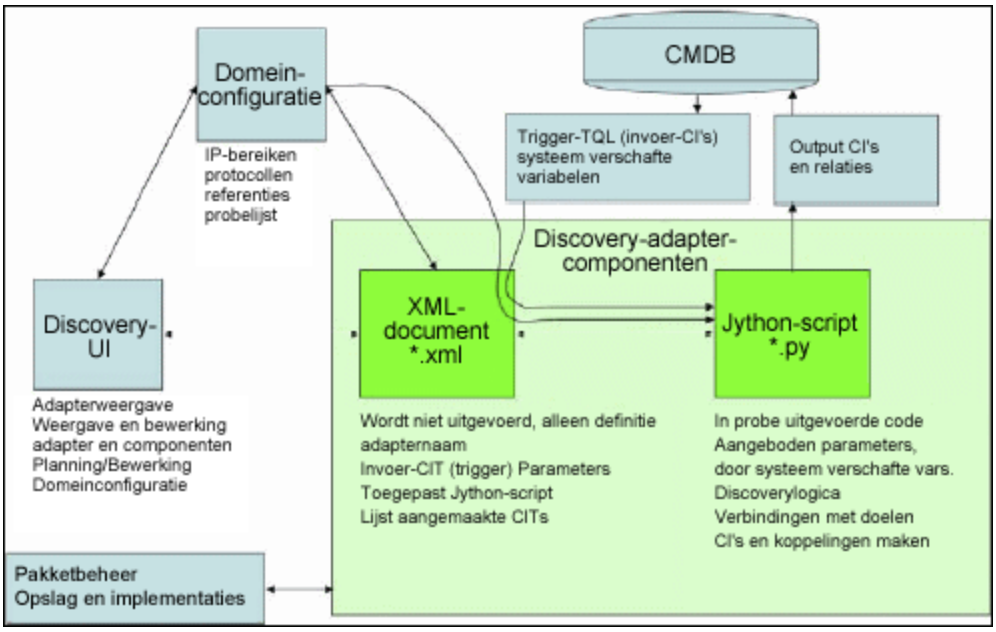
Flow/adapter	Jython-adapter	Java-adapter	GDB-adapter	Push-adapter
Vulling	X	X	X	
Federation		X	X	
Datapush		X		X

Discovery-inhoud ontwikkelen

In dit gedeelte worden de volgende onderwerpen behandeld:

Discovery-adapters en gerelateerde componenten

In het onderstaande diagram worden de componenten van een adapter weergegeven en de componenten waarmee interactie plaatsvindt om discovery uit te voeren. De componenten in groen zijn de werkelijke adapters en de componenten in blauw zijn componenten waarmee interactie met adapters plaatsvindt.



Houd er rekening mee dat de minimale invulling van een adapter twee bestanden is: een XML-document en een Jython-script. Het Discovery Framework, zoals invoer-CI's, referenties en door gebruikers verschafte bibliotheken, wordt tijdens de uitvoering aangeboden aan de adapter. Beide discovery-adaptercomponenten worden via Data Flow-beheer beheerd. Ze worden in de praktijk in de CMDB zelf opgeslagen. Hoewel het externe pakket blijft bestaan, wordt er niet naar verwezen voor bewerking. Pakketbeheer maakt behoud van de nieuwe mogelijkheid van discovery- en integratie-inhoud mogelijk.

Invoer-CI's voor de adapter worden door een TQL verschaft en worden aan het adapterscript aangeboden in door het systeem verschafte variabelen. Adapterparameters worden ook opgegeven als bestemmingsgegevens. U kunt de werking van de adapter dus configureren in overeenstemming met de specifieke functie van een adapter.

De DFM-applicatie wordt gebruikt om nieuwe adapters aan te maken en te testen. U gebruikt de pagina's Bedieningspaneel Discovery, Adapterbeheer en Instellingen Data Flow-probe tijdens het schrijven van adapters.

Adapters worden als pakketten opgeslagen en getransporteerd. Met de applicatie Pakketbeheer en de JMX-console worden pakketten aangemaakt op basis van nieuwe adapters en worden adapters in nieuwe systemen uitgerold.

Adapters scheiden

Technisch gezien kan een gehele discovery in één adapter worden gedefinieerd. In een goed ontwerp wordt een complex systeem echter verdeeld in eenvoudigere, beter te beheren componenten.

Hierna vindt u richtlijnen en best practices voor het verdelen van het adapterproces:

- Discovery dient plaats te vinden in fasen. Elke fase moet worden vertegenwoordigd door een adapter waarmee een gebied of laag van het systeem wordt toegewezen. Om verder te gaan met het detecteren van het systeem moeten adapters worden gebaseerd op de vorige fase of laag die moet worden gedetecteerd. Adapter A wordt bijvoorbeeld geactiveerd door een TQL-

resultaat van een applicatieserver en de laag van de applicatieserver wordt met deze adapter toegewezen. Als onderdeel van deze toewijzing wordt een JDBC-verbindingscomponent toegewezen. Met adapter B wordt een JDBC-verbindingscomponent geregistreerd als een trigger-TQL en worden de resultaten van adapter A gebruikt om toegang te krijgen tot de databaselaag (bijvoorbeeld via het URL-attribuut van JDBC) en wordt de databaselaag toegewezen.

- **Het tweefasenverbindingsparadigma**: voor toegang tot gegevens op de meeste systemen zijn referenties vereist. Dit houdt in dat een combinatie van gebruikersnaam/wachtwoord voor deze systemen moet worden geprobeerd. De DFM-beheerder verstrekt op veilige wijze referentiegegevens aan het systeem en kan verscheidene, geprioritiseerde aanmeldreferenties opgeven. Hiernaar wordt verwezen als de **Protocollenlijst**. Als het systeem om wat voor reden dan ook niet toegankelijk is, heeft het geen zin verdere discovery uit te voeren. Als de verbinding is gelukt, moet er voor toekomstige discovery-toegang een manier zijn om aan te geven welke referentieset met succes is gebruikt.

  Deze twee fasen leiden in de volgende gevallen tot een scheiding van de twee adapters:

  - **Verbindingsadapter**. Dit is een adapter waarmee een initiële trigger wordt geaccepteerd en waarmee wordt gezocht naar de aanwezigheid van een externe agent op die trigger. Dit gebeurt door in de protocollenlijst alle items te proberen die overeenkomen met het type van deze agent. Als het lukt, verschaft deze adapter als resultaat het CI van een remote agent (SNMP, WMI, enzovoort). Hiermee wordt ook verwezen naar het juiste item in de protocollenlijst voor toekomstige verbindingen. Dit agent-CI is vervolgens deel van een trigger voor de inhoudsadapter.
  - **Inhoudsadapter**. Voor deze adapter is vereist dat de verbinding van de vorige adapter is gelukt (vereisten die zijn opgegeven door de TQL's). Voor deze adaptertypen hoeft niet meer de hele protocollenlijst te worden doorzocht aangezien de juiste referenties kunnen worden verkregen van het CI van de externe agent, waarmee kan worden aangemeld bij het gedetecteerde systeem.
- Verschillende planningsoverwegingen kunnen ook van invloed zijn op discovery-verdeling. Op een systeem mogen bijvoorbeeld alleen tijdens rustige uren query's worden uitgevoerd. Dus zelfs al is het zinvol om de adapter samen te voegen met dezelfde adapter waarmee een ander systeem wordt gedetecteerd, moet u als gevolg van de verschillende planningen twee adapters aanmaken.
- Discovery van verschillende beheerinterfaces of -technologieën om hetzelfde systeem te detecteren moet in afzonderlijke adapters worden geplaatst. Hierdoor kunt u de toegangsmethode activeren die voor elk systeem of elke organisatie geschikt is. Zo hebben sommige organisaties WMI-toegang tot computers maar zijn er geen SNMP-agents op geïnstalleerd.

Discovery-adapters implementeren

Een DFM-taak heeft als doel toegang te krijgen tot externe (of lokale) systemen, uitgepakte gegevens te modelleren als CI's en de CI's in de CMDB op te slaan. De taak bestaat uit de volgende stappen:

1. **Een adapter aanmaken.**

   U configureert een adapterbestand waarin de context, parameters en resultaattypen worden bewaard, door de scripts te selecteren die deel van de adapter moeten zijn. Zie "Stap 1: een adapter aanmaken" op pagina 29 voor meer informatie over dit onderwerp.

2. **Een nieuwe Discovery-taak aanmaken**

   U configureert een taak met planningsgegevens en een trigger-query. Zie "Stap 2: een taak toewijzen aan de adapter" op pagina 37 voor meer informatie over dit onderwerp.

3. **Discovery-code bewerken.**

   U kunt de Jython- of Java-code bewerken die aanwezig is in de adapterbestanden en waarmee naar het DFM Framework wordt verwezen. Zie "Stap 3: Jython-code aanmaken" op pagina 38 voor meer informatie over dit onderwerp.

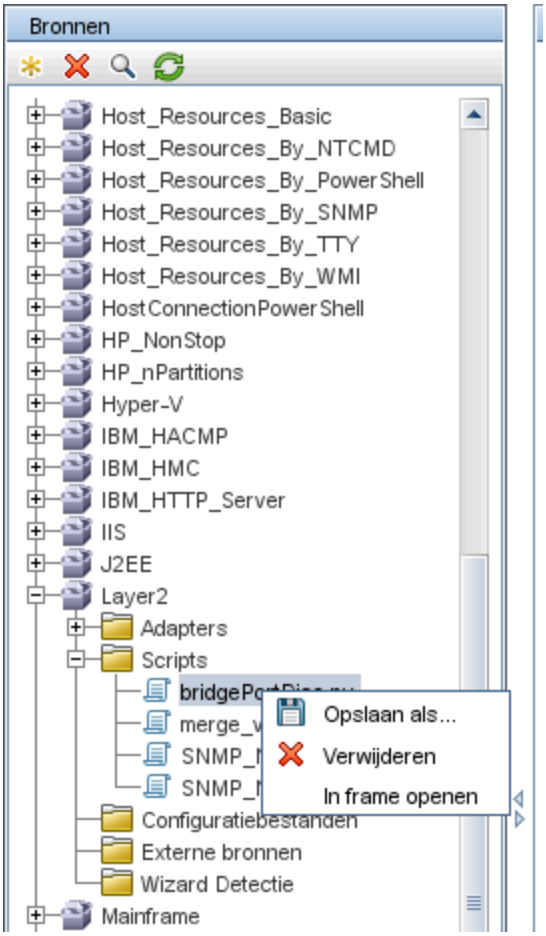
Als u nieuwe adapters wilt schrijven, maakt u alle bovenstaande componenten aan. Daarvan wordt elke component automatisch gekoppeld aan de component in de vorige stap. Zodra u bijvoorbeeld een taak aanmaakt en de relevante adapter selecteert, wordt het adapterbestand aan de taak gekoppeld.

Adaptercode

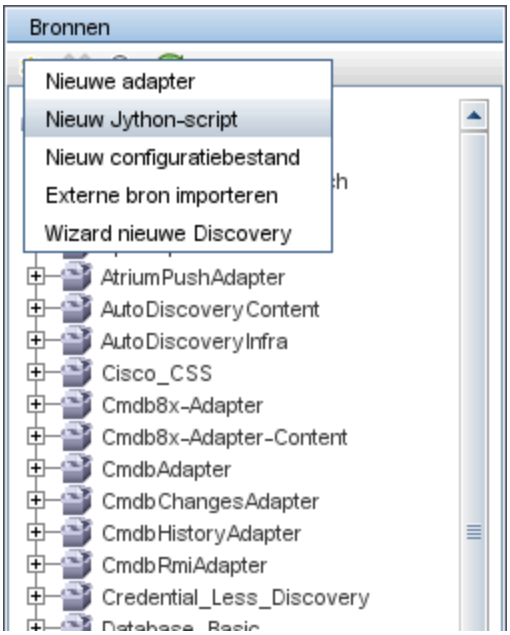
De werkelijke implementatie voor het maken van verbinding met het externe systeem, waarbij een query wordt uitgevoerd op de gegevens van het systeem en deze als CMDB-gegevens worden toegewezen, wordt uitgevoerd door de Jython-code. De code bevat bijvoorbeeld de logica om verbinding te maken met een database en gegevens van die database uit te pakken. In dit geval verwacht de code een JDBC-URL, een gebruikersnaam, een wachtwoord, een poort en dergelijke te ontvangen. Deze parameters zijn specifiek voor elk exemplaar van de database waarin de TQL-query wordt uitgevoerd. U definieert deze variabelen in de adapter (in de Trigger-CI-gegevens) en wanneer de taak wordt uitgevoerd, worden deze specifieke details voor uitvoering doorgegeven aan de code.

De adapter kan naar deze code verwijzen door een Java-klassenaam of een Jython-scriptnaam. In dit gedeelte wordt het schrijven van DFM-code als Jython-scripts besproken.

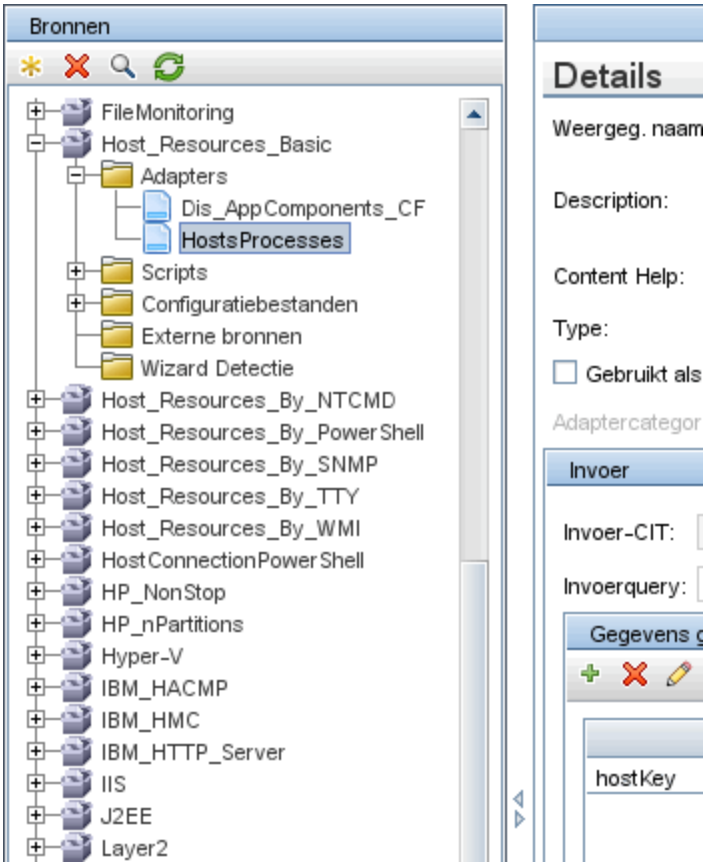
Een adapter kan een lijst met scripts bevatten die moeten worden gebruikt bij uitvoering van discovery. Wanneer een nieuwe adapter wordt aangemaakt, maakt u meestal een nieuw script aan en wijst u het aan de adapter toe. Een nieuw script bevat basissjablonen, maar u kunt een van de andere scripts als een sjabloon gebruiken door met de rechtermuisknop te klikken en **Opslaan als** te kiezen:



Zie "Stap 3: Jython-code aanmaken" op pagina 38 voor meer informatie over het schrijven van nieuwe Jython-scripts. U voegt scripts via het venster Bronnen toe:



De scripts op de lijst worden achter elkaar uitgevoerd in de volgorde waarin ze in de adapter zijn gedefinieerd:



Opmerking: een script moet worden opgegeven zelfs als het uitsluitend als een bibliotheek door een ander script wordt gebruikt. In dit geval moet het bibliotheekscript worden gedefinieerd voordat het door het script wordt gebruikt. In dit voorbeeld is het script `processdbutils.py` een bibliotheek die door het laatste `host_processes.py`-script wordt gebruikt. Bibliotheken onderscheiden zich van gewone uitvoerbare scripts doordat ze niet over de functie `DiscoveryMain()` beschikken.

Stap 1: een adapter aanmaken

Een adapter kan als de definitie van een functie worden beschouwd. Met deze functie wordt een invoerdefinitie gedefinieerd, wordt logica op de invoer uitgevoerd, wordt de uitvoer gedefinieerd en wordt een resultaat verschaft.

Met elke adapter worden invoer en uitvoer opgegeven: zowel invoer als uitvoer zijn Trigger-CI's die specifiek in de adapter zijn gedefinieerd. Met de adapter worden gegevens uit het Trigger-CI voor invoer uitgepakt en deze gegevens worden als parameters aan de code doorgegeven. (Gegevens van gerelateerde CI's worden soms ook aan de code doorgegeven. Raadpleeg "Venster Gerelateerde CI's" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie.) De code van een adapter is generiek, met uitzondering van deze specifieke Trigger-CI-invoerparameters die aan de code worden doorgegeven.

Raadpleeg "Universal Discovery-concepten" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over invoercomponenten.

In dit gedeelte vindt u de volgende onderwerpen:

1. Adapterinvoer (Trigger-CIT en Invoerquery) definiëren

Met de componenten Trigger-CIT en Invoerquery definieert u specifieke CI's als adapterinvoer:

- Met Trigger-CIT wordt gedefinieerd welk CIT als de invoer voor de adapter wordt gebruikt. Voor een adapter die bijvoorbeeld IP's gaat detecteren, is Network het invoer-CIT.
- De invoerquery is een gewone, bewerkbare query waarmee de query voor de CMDB wordt gedefinieerd. Met de invoerquery worden aanvullende beperkingen in het CIT gedefinieerd (als de taak bijvoorbeeld een `hostID`- of `application_ip`-attribuut vereist) en kunt u, indien door de adapter vereist, meer CI-gegevens definiëren.

  Als voor de adapter aanvullende gegevens van de CI's zijn vereist die zijn gerelateerd aan het trigger-CI, kunt u aanvullende knooppunten toevoegen aan de invoer-TQL. Zie "Voorbeeld van definitie van invoerquery" op volgende pagina hieronder en "Query-knooppunten en relaties toevoegen aan de TQL-query" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie.
- De gegevens van het trigger-CI bevatten naast alle vereiste gegevens in het trigger-CI ook gegevens (indien gedefinieerd) van de andere knooppunten in de invoer-TQL. In DFM worden variabelen gebruikt om gegevens van de CI's op te halen. Wanneer de taak naar de probe wordt gedownload, worden de gegevensvariabelen van het trigger-CI vervangen door werkelijke waarden die aanwezig zijn in de attributen voor werkelijke CI-exemplaren.

Voorbeeld van definitie van trigger-CIT:

In dit voorbeeld wordt met een trigger-CIT gedefinieerd dat IP-CI's in de adapter zijn toegestaan.

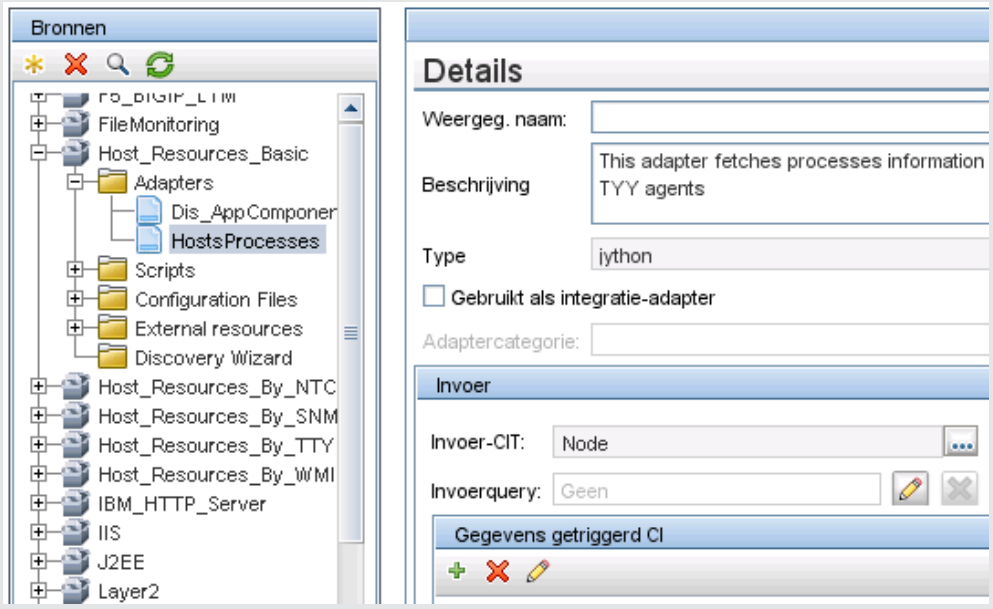
a. Open **Data Flow-beheer > Adapterbeheer**. Selecteer de adapter **HostProcesses** (**Pakketten> Host_Resources_Basic > Adapters > HostProcesses**).

b. Zoek het vak Invoer-CIT. Zie "Tabblad Adapterdefinitie" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

c. Klik op de knop CI-type selecteren om het dialoogvenster Gedetecteerde klasse selecteren te openen. Zie "Dialoogvenster Gedetecteerde klasse selecteren" in de *HP Universal CMDB – Handleiding Data Flow Management*

d. Selecteer het CIT.

In dit voorbeeld is het IP-CI (host) toegestaan in de adapter:



Voorbeeld van definitie van invoerquery

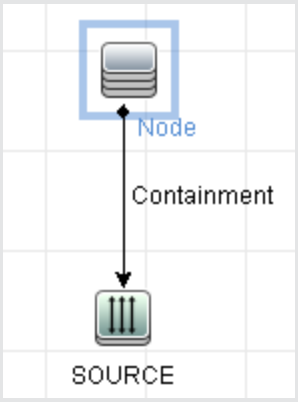
In dit voorbeeld wordt met de TQL-invoerquery gedefinieerd dat het `IpAddress`-CI (geconfigureerd in het vorige voorbeeld als het trigger-CIT) moet worden verbonden met een `Node`-CI.

a. Open **Data Flow-beheer > Adapterbeheer**. Zoek het vak Invoerquery. Klik op de knop **Bewerken** om de Editor invoer-query te openen. Zie "Venster Editor invoer-query" in de *HP Universal CMDB – Handleiding Data Flow Management*

b. Geef in de Editor invoer-query het trigger-CI-knooppunt de naam **BRON**: klik met de rechtermuisknop op het knooppunt en kies **Eigenschappen query-knooppunt**. Verander de naam in het vak **Elementnaam** in **BRON**.

c. Voeg een `Node`-CI en een `Containment`-relatie toe aan de `IpAddress`-CI. Raadpleeg "Venster Editor invoer-query" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over het werken met de Editor invoer-

query.



Het **IpAddress**-CI is verbonden met een **Node**-CI. De invoer-TQL bestaat uit twee knooppunten, **Node** en **IpAddress**, met een koppeling ertussen. Het **IpAddress**-CI heet **BRON**.

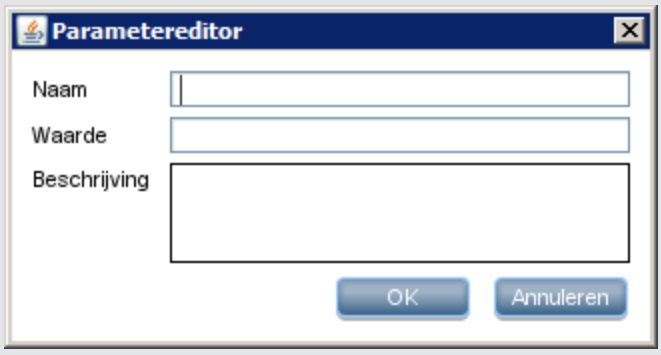
Voorbeeld van het toevoegen van variabelen aan de TQL-invoerquery:

In dit voorbeeld voegt u de variabelen DIRECTORY en CONFIGURATION_FILE aan de TQL-invoerquery toe die in het vorige voorbeeld zijn aangemaakt. Met deze variabelen kunt u gemakkelijker definiëren wat u moet detecteren, in dit geval om de configuratiebestanden te vinden die zich op de hosts bevinden die zijn gekoppeld aan de IP's die u moet detecteren.

a. Geef de invoer-TQL weer die in het vorige voorbeeld is aangemaakt.

   Open **Data Flow-beheer > Adapterbeheer**. Zoek het venster Gegevens getriggerd CI. Zie "Tabblad Adapterdefinitie" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

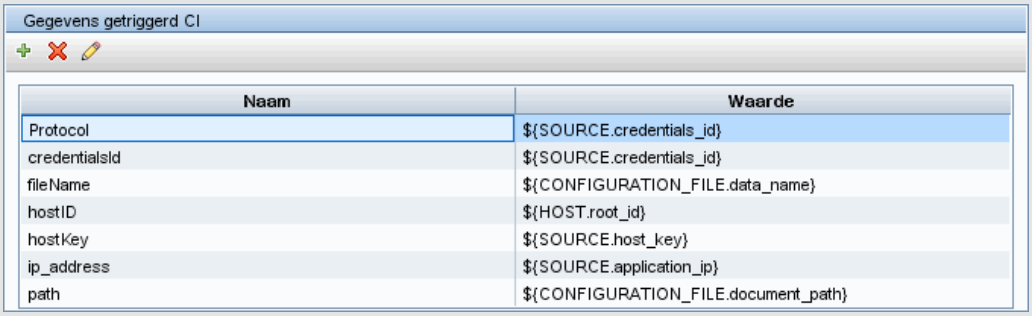
b. Voeg variabelen aan de invoer-TQL toe. Open **Data Flow-beheer > Adapterbeheer** voor meer informatie. Zoek het venster Gegevens getriggerd CI. Zie "Tabblad Adapterdefinitie" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.



Voorbeeld van het vervangen van variabelen met werkelijke gegevens:

In dit voorbeeld worden de **IpAddress**-CI-gegevens met variabelen vervangen door feitelijke gegevens die in uw systeem aanwezig zijn in echte **IpAddress**-CI-exemplaren.

Gegevens getriggerd CI voor het **IpAddress**-CI bevat een `fileName`-variabele. Met deze variabele kan het knooppunt **CONFIGURATION_FILE** in de invoer-TQL worden vervangen door de werkelijke waarden van het configuratiebestand dat zich op een host bevindt:



Naam	Waarde
Protocol	${SOURCE.credentials_id}
credentialsId	${SOURCE.credentials_id}
fileName	${CONFIGURATION_FILE.data_name}
hostID	${HOST.root_id}
hostKey	${SOURCE.host_key}
ip_address	${SOURCE.application_ip}
path	${CONFIGURATION_FILE.document_path}

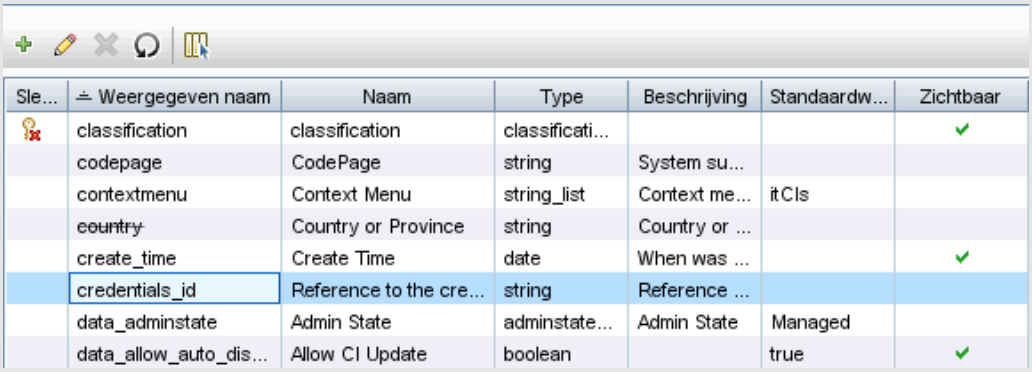
De gegevens getriggerd CI worden naar de probe geüpload waarbij alle variabelen door werkelijke waarden zijn vervangen. Het adapterscript bevat een opdracht om met het DFM Framework de werkelijke waarden van de gedefinieerde variabelen op te halen:

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

Voor de variabelen `fileName` en `path` worden de attributen `data_name` en `document_path` uit het knooppunt **CONFIGURATION_DOCUMENT** gebruikt (gedefinieerd in de editor Invoer-query; zie vorig voorbeeld).

Klik hier voor een illustratie.

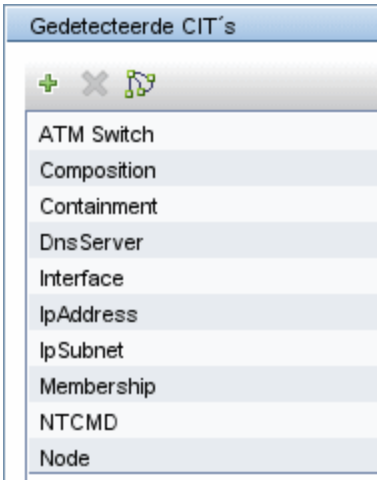
De variabelen `Protocol`, `credentialsId` en `ip_address` gebruiken de attributen `root_class`, `credentials_id` en `application_ip`:



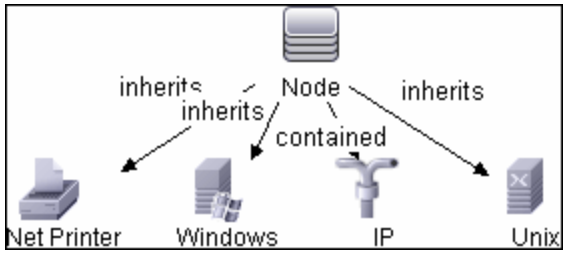
Sle...	Weergegeven naam	Naam	Type	Beschrijving	Standaardw...	Zichtbaar
	classification	classification	classificati...			✔
	codepage	CodePage	string	System su...		
	contextmenu	Context Menu	string_list	Context me...	itCIs	
	~~country~~	Country or Province	string	Country or ...		
	create_time	Create Time	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. Adapteruitvoer definiëren

De uitvoer van de adapter is een lijst met gedetecteerde CI's (**Data Flow-beheer > Adapterbeheer > tabblad Adapterdefinitie > Gedetecteerde CIT's**) en de koppelingen ertussen:



U kunt de CIT's ook als een topologiekaart weergeven, dat wil zeggen: de componenten en de manier waarop ze aan elkaar worden gekoppeld (klik op de knop **Gedetecteerde CIT's weergeven als kaart**):



De gedetecteerde CI's worden door de DFM-code geretourneerd (dat wil zeggen: het Jython-script) in de indeling `ObjectStateHolderVector` van UCMDB. Zie "Resultaten genereren met het Jython-script" op pagina 45 voor meer informatie over dit onderwerp.

Voorbeeld van adapteruitvoer:

In dit voorbeeld definieert u welke CIT's deel moeten uitmaken van de IP-CI-uitvoer.

a. Open **Data Flow-beheer > Adapterbeheer**.

b. Selecteer **Network > Adapters > NSLOOKUP_on_Probe** in het venster Bronnen.

c. Zoek op het tabblad Adapterdefinitie het venster Gedetecteerde CIT's.

d. De CIT's die deel moeten uitmaken van de adapteruitvoer, worden weergegeven. Voeg CIT's toe aan of verwijder ze van de lijst. Zie "Tabblad Adapterdefinitie" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

3. ## Adapterparameters negeren

Als u een adapter voor meerdere taken wilt configureren, kunt u adapterparameters negeren. De adapter `SQL_NET_Dis_Connection` wordt bijvoorbeeld door zowel de taak `MSSQL Connection by SQL` als de taak `Oracle Connection by SQL` gebruikt.

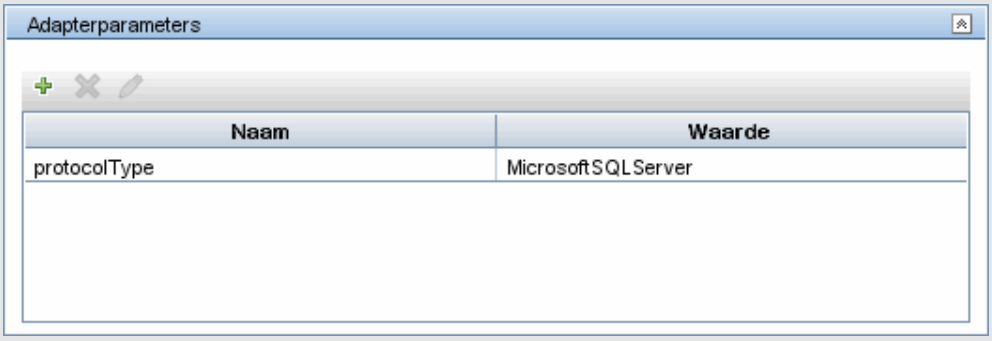
Voorbeeld van het negeren van een adapterparameter:

Dit voorbeeld laat een situatie zien waarbij een adapterparameter wordt genegeerd zodat één adapter kan worden gebruikt om zowel Microsoft SQL Server- als Oracle-databases te detecteren.

a. Open **Data Flow-beheer > Adapterbeheer**.

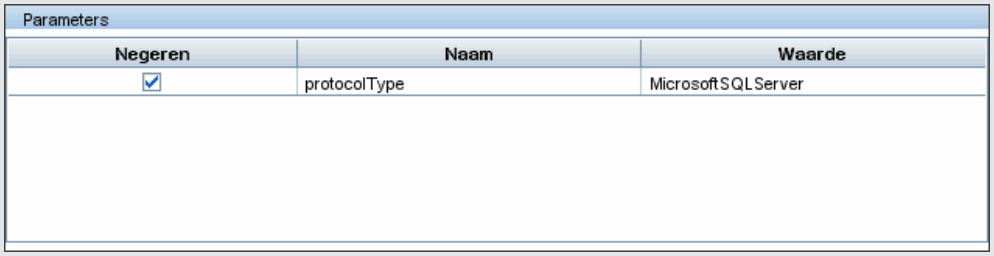
b. Selecteer **Database_Basic > Adapters > SQL_NET_Dis_Connection** in het venster Bronnen.

c. Zoek op het tabblad Adapterdefinitie het venster **Adapterparameters**. De parameter `protocolType` heeft de waarde **all**:



d. Klik met de rechtermuisknop op de adapter **SQL_NET_Dis_Connection_MsSql** en kies **Naar Discovery-taak gaan> MSSQL Connection by SQL**.

e. Open het tabblad Eigenschappen. Zoek het venster Parameters:



De waarde `all` wordt overschreven door de waarde `MicrosoftSQLServer`.

Opmerking: de taak **Oracle Connection by SQL** bevat dezelfde parameter maar de waarde wordt overschreven door een Oracle-waarde.

Zie "Tabblad Adapterdefinitie" in the *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over het toevoegen, verwijderen en bewerken van parameters.

Op basis van deze parameter wordt in DFM begonnen met zoeken naar Microsoft SQL Server-exemplaren.

4. Probe-selectie negeren - optioneel

De UCMDB-server bevat een dispatcher die de door de UCMDB zijn ontvangen trigger-CI's herkent en automatisch bepaalt welke probe de taak voor elk trigger-CI moet uitvoeren op

basis van een van de volgende opties.

- **Voor het CI-type IP-adres:** neem de probe die voor dit IP is gedefinieerd.
- **Voor het CI-type actieve software:**gebruik de attributen **application_ip** en **application_ip_domain** en kies de probe die voor het IP in het relevante domein is gedefinieerd.
- **Voor andere CI-typen:** gebruik het IP van het knooppunt op basis van het gerelateerde knooppunt van het CI (als dat bestaat).

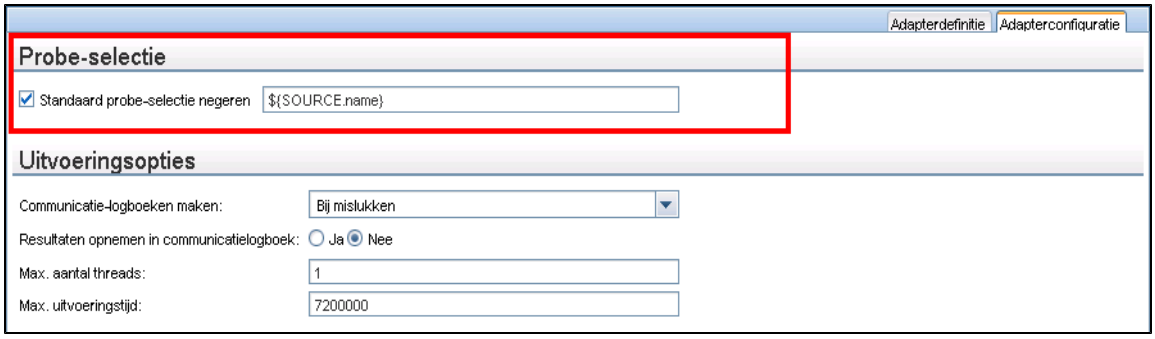
De probe wordt automatisch geselecteerd op basis van het gerelateerde knooppunt van het CI. Na het ophalen van het aan het CI gerelateerde knooppunt kiest de dispatcher een van de IP's van het knooppunt en kiest het de probe in overeenstemming met de definities van het netwerkbereik van de probe.

In de volgende gevallen moet u de probe handmatig opgeven en kunt u geen gebruikmaken van het automatische dispatcher-mechanisme:

- U weet al welke probe voor de adpater moet worden uitgevoerd en de automatische dispatcher is niet nodig om de probe te selecteren (bijvoorbeeld als het trigger-CI de probe-gateway is).
- Het automatisch selecteren van de probe kan mislukken. Dit kan gebeuren in de volgende situaties:
  - Een trigger-CI heeft geen gerelateerd knooppunt (zoals het netwerk-CIT).
  - Het knooppunt van een trigger-CI heeft meerdere IP's, die elk bij een andere probe behoren.

Om deze problemen op te lossen kunt u als volgt opgeven welke probe door de adapter moet worden gebruikt:

a. in het gedeelte Probe-selectie selecteert u **Standaard probe-selectie negeren** zoals hieronder weergegeven.



b. in het vak Probe geeft u de probe op die voor de taak moet worden gebruikt.

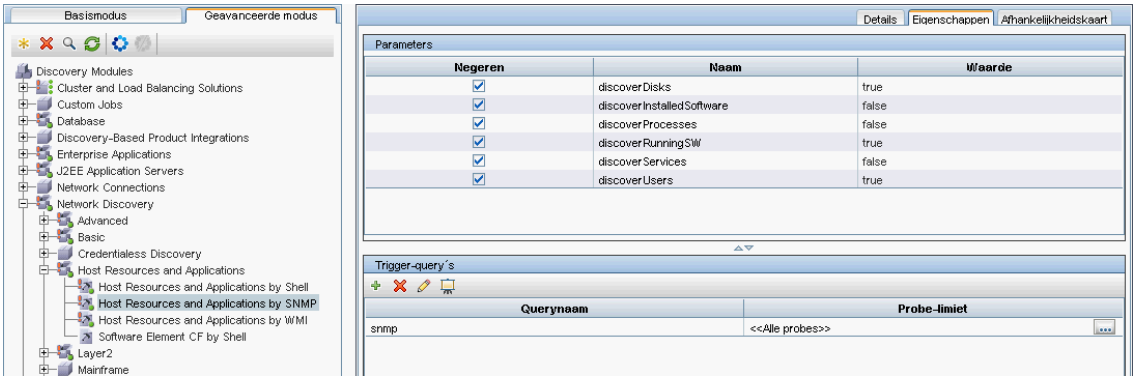
5. ## Een klassepad configureren voor een extern proces - optioneel

Zie "Uitvoering van externe processen configureren" op pagina 38 voor meer informatie over dit onderwerp.

Stap 2: een taak toewijzen aan de adapter

Aan elke adapter zijn een of meer taken gekoppeld waarmee het uitvoeringsbeleid wordt gedefinieerd. Met taken kan dezelfde adapter verschillend worden ingepland voor verschillende sets getriggerde CI's en kunnen bovendien verschillende parameters voor elke set worden verschaft.

De taken worden in de boomstructuur Discovery-modules weergegeven en dit is de entiteit die de gebruiker activeert. Dat wordt in de afbeelding hieronder weergegeven.



Een trigger TQL kiezen

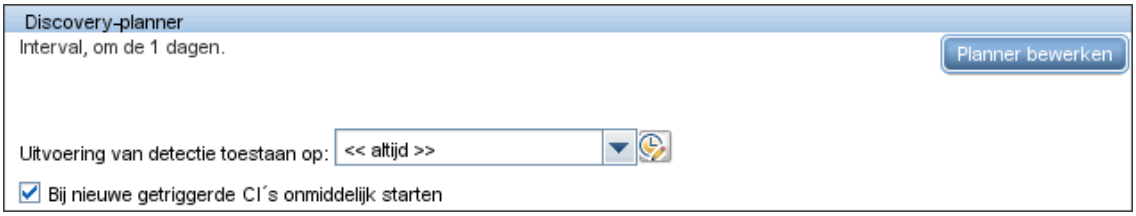
Elke taak wordt gekoppeld aan trigger-TQL's. Met deze trigger-TQL's worden resultaten gepubliceerd die als invoertrigger-CI's voor de adapter van deze taak worden gebruikt.

Met een trigger-TQL kunnen beperkingen aan een invoer-TQL worden toegevoegd. Als de resultaten van een invoer-TQL bijvoorbeeld IP's zijn die zijn verbonden met SNMP, kunnen de resultaten van een trigger-TQL IP's zijn die met SNMP zijn verbonden binnen het bereik 195.0.0.0-195.0.0.10.

Opmerking: een trigger-TQL moet naar de objecten verwijzen waarnaar de invoer-TQL verwijst. Als een invoer-TQL een query uitvoert voor IP's waarop SNMP draait, kunt u geen trigger-TQL (voor dezelfde taak) definiëren om een query uit te voeren voor IP's die met een host zijn verbonden, omdat bepaalde IP's mogelijk niet verbonden zijn met een SNMP-object, zoals is vereist door de invoer-TQL.

Planningsinformatie instellen

Met de planningsinformatie voor de probe wordt opgegeven wanneer de code op trigger-CI's moet worden uitgevoerd. Als het selectievakje **Bij nieuwe getriggerde CI's onmiddellijk starten** is ingeschakeld, wordt de code ook eenmaal uitgevoerd op elk trigger-CI wanneer de probe wordt bereikt, ongeacht toekomstige planningsinstellingen.



Voor elke geplande vermelding van elke taak wordt met de probe de code uitgevoerd voor alle trigger-CI's die voor die taak zijn samengevoegd. Zie "Dialoogvenster Planner" op pagina 1 in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

Adapterparameters negeren

Bij het configureren van een taak kunt u de adapterparameters negeren. Zie "Adapterparameters negeren" op pagina 34 voor meer informatie over dit onderwerp.

Stap 3: Jython-code aanmaken

HP Universal CMDB gebruikt Jython-scripts voor het schrijven van adapters. Het script `SNMP_Connection.py` wordt bijvoorbeeld gebruikt door de adapter `SNMP_NET_Dis_Connection` om te proberen computers te verbinden met SNMP. Jython is een op Python gebaseerde taal, uitgevoerd door Java.

Voor meer informatie over hoe u in Jython werkt, kunt u de volgende websites raadplegen:

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

Zie "Jython-code aanmaken" op pagina 40 voor meer informatie over dit onderwerp.

Uitvoering van externe processen configureren

U kunt Discovery uitvoeren voor een Discovery-taak in een proces dat los staat van het proces van de Data Flow-probe.

U kunt de taak bijvoorbeeld in een afzonderlijk extern proces uitvoeren als de taak gebruikmaakt van JAR-bibliotheken die een andere versie hebben dan de bibliotheken van de probe of die daarmee incompatibel zijn.

U kunt de taak ook in een afzonderlijk extern proces uitvoeren als de taak in potentie veel geheugen gebruikt (vanwege het grote aantal gegevens) en u de probe wilt behoeden voor mogelijke geheugenproblemen.

Als u een taak als een extern proces wilt configureren, definieert u de volgende parameters in het configuratiebestand van de bijbehorende adapter:

Parameter	Beschrijving
remoteJVMArgs	JVM-parameters voor het externe Java-proces.
runInSeparateProcess	Wanneer deze parameter is ingesteld op **true**, wordt de Discovery-taak in een afzonderlijk proces uitgevoerd.
remoteJVMClasspath	(Optioneel) Hiermee kan het klassepad van het externe proces worden aangepast, waardoor het standaard probe-klassepad overschreven. Dat is handig als er sprake is van incompatibele versies tussen de jars van de probe en aangepaste jars die nodig zijn voor de door de klant gedefinieerde discovery.

Parameter	Beschrijving

	Als de **remoteJVMClasspath**-parameter niet gedefinieerd, of leeg, is wordt het standaard probe-klassepad gebruikt.
	
	Als u een nieuwe Discovery-taak opzet en u wilt er zeker van zijn dat de JAR-bibliotheek van de probe niet conflicteert met de JAR-bibliotheken van de taak, moet u ten minste het minimale klassepad gebruiken die nodig is om de basis-discovery te kunnen uitvoeren. Het minimale klassepad is vastgelegd in het bestand **DataFlowProbe.properties** met de parameter **basic_discovery_ minimal_classpath**.
	
	Voorbeelden van het aanpassen van **remoteJVMClasspath**:
	
	• U kunt aangepaste jars vóór of achter het standaard probe-klassepad toevoegen door de parameter **remoteJVMClasspath** als volgt aan te passen:
	
	`aangepast1.jar;%classpath%;aangepast2.jar -`
	
	In dit geval wordt **aangepast1.jar** vóór het standaard probe-klassepad geplaatst en **aangepast2.jar** daarachter.
	
	• U kunt het minimale klassepad gebruiken door de parameter **remoteJVMClasspath** als volgt aan te passen:
	
	`aangepast1.jar;%minimal_ classpath%;aangepast2.jar`

Hoofdstuk 2

Jython-adapters ontwikkelen

In dit hoofdstuk vindt u de volgende informatie:

HP API-referentie Data Flow-beheer

Raadpleeg de *HP Universal CMDB - API-referentie Data Flow-beheer* voor volledige documentatie over de beschikbare API's. De bestanden bevinden zich in de volgende map:

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

Jython-code aanmaken

HP Universal CMDB gebruikt Jython-scripts voor het schrijven van adapters. Het script `SNMP_ Connection.py` wordt bijvoorbeeld gebruikt door de adapter `SNMP_NET_Dis_Connection` om te proberen computers te verbinden met SNMP. Jython is een op Python gebaseerde taal, uitgevoerd door Java.

Voor meer informatie over hoe u in Jython werkt, kunt u de volgende websites raadplegen:

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

In het volgende gedeelte wordt beschreven hoe het werkelijke schrijven van Jython-code in het DFM Framework in zijn werk gaat. In dit gedeelte wordt specifiek aandacht besteed aan de contactpunten tussen het Jython-script en het Framework dat wordt aangeroepen. Ook worden de Jython-bibliotheken en -hulpprogramma's beschreven die waar mogelijk moeten worden gebruikt.

Opmerking:

- Scripts die worden geschreven voor DFM, moeten compatibel zijn met Jython versie 2.1.
- Raadpleeg de *HP Universal CMDB - API-referentie Data Flow-beheer* voor volledige

documentatie over de beschikbare API's.

In dit gedeelte vindt u de volgende onderwerpen:

Externe JAR-bestanden van Java gebruiken in Jython

Wanneer nieuwe Jython-scripts worden ontwikkeld, zijn er soms externe Java-bibliotheken (JAR-bestanden) of uitvoerbare bestanden van derden nodig als Java-hulpprogramma-archieven, verbindingsarchieven zoals JAR-bestanden voor het JDBC-stuurprogramma of als uitvoerbare bestanden (zo wordt **nmap.exe** gebruikt voor discovery zonder referenties).

Deze bronnen moeten in het pakket worden gebundeld onder de map **Externe bronnen**. Elke bron die in deze map is geplaatst, wordt automatisch verzonden naar een probe die wordt verbonden met uw HP Universal CMDB-server.

Daarnaast wordt elke JAR-bestandsbron bij het starten van discovery geladen in het klassepad van Jython. Hierbij worden alle klassen beschikbaar gemaakt voor import en gebruik.

Uitvoering van de code

Nadat een taak is geactiveerd, wordt een taak met alle vereiste informatie naar de probe gedownload.

De probe wordt gestart terwijl de DFM-code wordt uitgevoerd met de informatie die is opgegeven in de taak.

De Jython-codestroom wordt gestart vanuit een hoofdvermelding in het script, code wordt uitgevoerd om CI's te detecteren en resultaten van een vector van gedetecteerde CI's worden verschaft.

Meegeleverde scripts wijzigen

Wanneer u wijzigingen aanbrengt in meegeleverde scripts, moet u daar alleen minimale wijzigingen in aanbrengen en alle noodzakelijke methoden in een extern script plaatsen. U kunt wijzigingen dan efficiënter bijhouden en uw code wordt niet overschreven wanneer u overstapt op een nieuwere HP Universal CMDB-versie.

Met de volgende code bestaande uit één regel in een meegeleverd script wordt bijvoorbeeld een methode aangeroepen waarmee een webservernaam op een applicatiespecifieke manier wordt berekend:

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

De complexere logica waarmee wordt bepaald hoe deze naam wordt berekend, is opgenomen in een extern script:

```
# 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
```

Sla het externe script in de map Externe bronnen op. Zie "Het deelvenster Bronnen" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp. Als u dit script aan een pakket toevoegt, kunt u dit script ook voor andere taken gebruiken. Zie "Pakketbeheer" in de *HP Universal CMDB – Handleiding Beheer* voor meer informatie over werken met Pakketbeheer.

Tijdens de upgrade wordt de wijziging die u in de code van één regel aanbrengt, overschreven door de nieuwe versie van het meegeleverde script. U moet de regel dus vervangen. Het externe script wordt echter niet overschreven.

Structuur van het Jython-bestand

Het Jython-bestand bestaat uit drie delen in een bepaalde volgorde:

1. Importbewerkingen
2. Hoofdfunctie - DiscoveryMain
3. Functiedefinities (optioneel)

Het volgende is een voorbeeld van een Jython-script:

```
#   imports section
from appilog.common.system.types import ObjectStateHolder
from appilog.common.system.types.vectors import
```

```
ObjectStateHolderVector
# Function definition
def foo:
    # do something
#   Main Function
def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()
    ## Write implementation to return new result CIs here...
    return OSHVResult
```

Importbewerkingen

Jython-klassen worden verspreid over hiërarchische naamruimten. In versie 7.0 of hoger zijn er, in tegenstelling tot vorige versies, geen impliciete importbewerkingen. Elke klasse die u gebruikt moet dus expliciet worden geïmporteerd. (Deze wijziging is aangebracht om prestatieredenen en om het Jython-script begrijpelijker te maken door noodzakelijke details niet te verbergen.)

- Zo importeert u een Jython-script:

  ```
  import logger
  ```

- Zo importeert u een Java-klasse:

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

Hoofdfunctie – DiscoveryMain

Elk uitvoerbaar Jython-scriptbestand bevat een hoofdfunctie: DiscoveryMain.

De functie `DiscoveryMain` is de hoofdvermelding in het script. Het is de eerste functie die wordt uitgevoerd. Met de hoofdfunctie kunnen andere functies worden aangeroepen die in de scripts worden gedefinieerd:

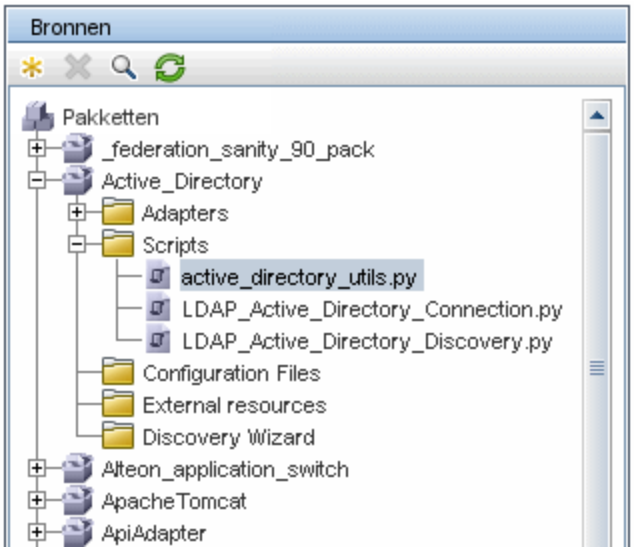
```
def DiscoveryMain(Framework):
```

Het Framework-argument moet worden opgegeven in de definitie van de hoofdfunctie. Dit argument wordt gebruikt door de hoofdfunctie voor het ophalen van informatie die is vereist om de scripts uit te voeren (zoals informatie in het trigger-CI en parameters) en kan ook worden gebruikt om fouten te rapporteren die tijdens de uitvoering van het script optreden.

U kunt een Jython-script aanmaken zonder enige hoofdmethode. Dergelijke scripts worden als bibliotheekscripts gebruikt die vanuit andere scripts worden aangeroepen.

Functiedefinitie

Elk script kan extra functies bevatten die via de hoofdcode worden aangeroepen. Met een dergelijke functie kan een andere functie worden aangeroepen, die in het huidige script of in een ander script aanwezig is (met de instructie `import`). Als u een ander script wilt gebruiken, moet u het toevoegen aan de sectie Scripts van het pakket:



Voorbeeld van een functie waarmee een andere functie wordt aangeroepen:

In het volgende voorbeeld wordt met de hoofdcode de methode `doQueryOSUsers(..)` aangeroepen waarmee weer de interne methode `doOSUserOSH(..)` wordt aangeroepen:

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

    sw_obj.setAttribute('data_name', name)
    # return the object
    return sw_obj
def doQueryOSUsers(client, OSHVResult):
    _hostObj = modeling.createHostOSH(client.getIpAddress())
    data_name_mib = '1.3.6.1.4.1.77.1.2.25.1.1,
1.3.6.1.4.1.77.1.2.25.1.2,string'
    resultSet = client.executeQuery(data_name_mib)
    while resultSet.next():
        UserName = resultSet.getString(2)
        ########## send object ##############
        OSUserOSH = doOSUserOSH(UserName)
        OSUserOSH.setContainer(_hostObj)
        OSHVResult.add(OSUserOSH)
def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()
    try:
        client = Framework.getClientFactory(ClientsConsts.SNMP_
PROTOCOL_NAME).createClient()
    except:
        Framework.reportError('Connection failed')
    else:
        doQueryOSUsers(client, OSHVResult)
        client.close()
    return OSHVResult
```

Als dit script een globale bibliotheek is die betrekking heeft op een groot aantal adapters, kunt u het toevoegen aan de lijst met scripts in het configuratiebestand `jythonGlobalLibs.xml` in plaats van het aan elke adapter toe te voegen (**Adapterbeheer > Deelvenster Bronnen > AutoDiscoveryContent > Configuratiebestanden**).

Resultaten genereren met het Jython-script

Elk Jython-script wordt op een specifiek trigger-CI uitgevoerd en wordt beëindigd met resultaten die worden geretourneerd door de retourwaarde van de functie `DiscoveryMain`.

Het scriptresultaat is eigenlijk een groep CI's en koppelingen die moeten worden ingevoegd of bijgewerkt in de CMDB. Met het script wordt deze groep CI's en koppelingen geretourneerd in de indeling `ObjectStateHolderVector`.

De klasse `ObjectStateHolder` is een manier om objecten of koppelingen voor te stellen die zijn gedefinieerd in de CMDB. Het object `ObjectStateHolder` bevat de CIT-naam en een lijst met attributen en de bijbehorende waarden. `ObjectStateHolderVector` is een vector van `ObjectStateHolder`-exemplaren.

ObjectStateHolder-syntaxis

In dit gedeelte wordt uitgelegd hoe de DFM-resultaten in een UCMDB-model moeten worden samengesteld.

Voorbeeld van het instellen van attributen in de CI's:

Met de klasse ObjectStateHolder wordt de DFM-grafiek met resultaten beschreven. Elk CI en elke koppeling (relatie) wordt in een exemplaar van de klasse `ObjectStateHolder` geplaatst, zoals in het volgende Jython-codevoorbeeld:
siebel application server 1 appServerOSH = ObjectStateHolder('siebelappserver') 2 appServerOSH.setStringAttribute('data_name', sblsvrName) 3 appServerOSH.setStringAttribute ('application_ip', ip) 4 appServerOSH.setContainer (appServerHostOSH)

- Met regel 1 wordt een CI van het type **siebelappserver** aangemaakt.
- Met regel 2 wordt het attribuut **data_name** met de waarde **sblsvrName** aangemaakt. Dit is een Jython-variabele ingesteld met de waarde die voor de servernaam is gedetecteerd.
- Met regel 3 wordt een niet-sleutelattribuut ingesteld dat in de CMDB wordt bijgewerkt.
- Regel 4 betreft het samenstellen van de containment (het resultaat is een grafiek). Hiermee wordt opgegeven dat deze applicatieserver zich in een host bevindt (een andere `ObjectStateHolder`-klasse in het bereik).

Opmerking: elk CI dat door het Jython-script wordt gerapporteerd, moet waarden voor alle sleutelattributen van het CT-type van het CI bevatten.

Voorbeeld van relaties (koppelingen):

Met het volgende koppelingsvoorbeeld wordt uitgelegd hoe de grafiek wordt voorgesteld:
1 linkOSH = ObjectStateHolder('route') 2 linkOSH.setAttribute('link_end1', gatewayOSH) 3

linkOSH.setAttribute('link_end2', appServerOSH)

- Met regel 1 wordt de koppeling aangemaakt (die ook van de klasse `ObjectStateHolder` is. Het enige verschil is dat `route` een koppelings-CI-type is).
- Met regel 2 en 3 worden de knooppunten aan het uiteinde van elke koppeling opgegeven. Dit gebeurt met de attributen **end1** en **end2** van de koppeling die moet worden opgegeven (omdat dat de minimale sleutelattributen van elke koppeling zijn). De attribuutwaarden zijn `ObjectStateHolder`-exemplaren. Zie "Koppeling" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over End 1 and End 2.

Let op: een koppeling is directioneel. U moet controleren of de knooppunten End 1 en End 2 overeenkomen met geldige CIT's aan elk uiteinde. Als de knooppunten niet geldig zijn, mislukt de validatie van het resulterende object en wordt het niet correct gerapporteerd. Zie "CIT-relaties" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.

Voorbeeld van vector (CI's verzamelen):

Als objecten met attributen en koppelingen met objecten aan de uiteinden zijn aangemaakt, moet u ze groeperen. Dit doet u door ze als volgt toe te voegen aan een `ObjectStateHolderVector`-exemplaar:

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

Zie de methode sendObjects voor meer informatie over het rapporteren van dit samengestelde resultaat aan het Framework zodat het kan worden verzonden naar de CMDB-server.

Nadat de resultaatgrafiek is gemaakt in een `ObjectStateHolderVector`-exemplaar, moet deze worden teruggestuurd naar het DFM Framework zodat de grafiek kan worden ingevoegd in de CMDB. Dit gebeurt door het exemplaar `ObjectStateHolderVector` te retourneren als het resultaat van de functie `DiscoveryMain()`.

Opmerking: zie "modeling.py" op pagina 77 in "Jython-bibliotheken en -hulpprogramma's" op pagina 75 voor meer informatie over het aanmaken van **OSH** voor algemene CIT's.

Framework-exemplaar

Het Framework-exemplaar is het enige argument dat in de hoofdfunctie wordt opgegeven in het Jython-script. Dit is een interface die kan worden gebruikt om informatie op te halen die nodig is om het script uit te voeren (bijvoorbeeld informatie over trigger-CI's en adapterparameters) en wordt ook gebruikt om fouten te rapporteren die tijdens de uitvoering van het script optreden. Zie "HP API-referentie Data Flow-beheer" op pagina 40 voor meer informatie over dit onderwerp.

Het juiste gebruik van het Framework-exemplaar is om dit als argument door te geven aan elke methode die er gebruik van maakt.

Voorbeeld:

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

In dit gedeelte worden de belangrijkste toepassingen van Framework beschreven:

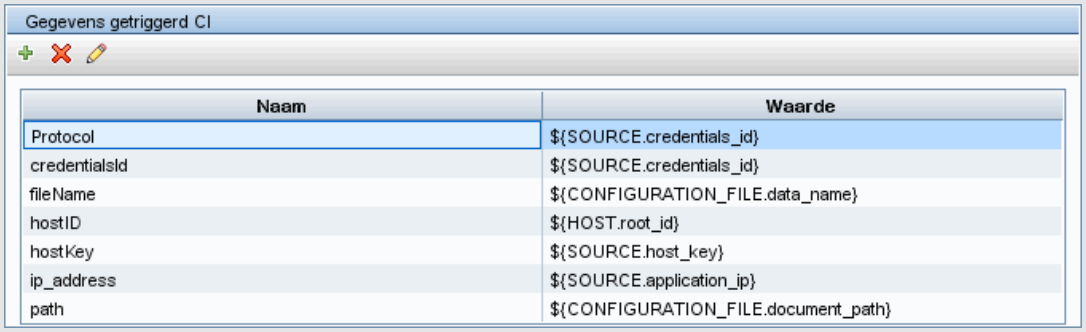
- "Framework.getTriggerCIData(String attributeName)" beneden
- "Framework.createClient(credentialsId, props)" beneden
- "Framework.getParameter (String parameterName)" op pagina 49
- "Framework.reportError(String message) en Framework.reportWarning(String message)" op pagina 49

Framework.getTriggerCIData(String attributeName)

Deze API verschaft de tussentijdse stap tussen de trigger-CI-gegevens die zijn gedefinieerd in de adapter en het script.

Voorbeeld van het ophalen van referentie-informatie:

U vraagt de volgende trigger-CI-gegevens aan:



Als u de referentie-informatie van de taak wilt ophalen, gebruikt u de volgende API:

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

Framework.createClient(credentialsId, props)

U maakt een verbinding met een externe computer door een clientobject aan te maken en opdrachten op die client uit te voeren. Als u een client wilt aanmaken, haalt u de klasse `ClientFactory` op. Met de methode getClientFactory() wordt het type van het aangevraagde clientprotocol ontvangen. De protocolconstanten worden gedefinieerd in de klasse ClientsConsts. Zie *HP Universal CMDB Discovery and Integration Content Guide* voor meer informatie over referenties en ondersteunde protocollen.

Voorbeeld van het aanmaken van een clientexemplaar voor de referentie-ID:

Zo maakt u een `Client`-exemplaar voor de referentie-ID aan:

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

U kunt nu het exemplaar `Client` gebruiken om verbinding te maken met de relevante computer of applicatie.

Voorbeeld van het aanmaken van een WMI-client en het uitvoeren van een WMI-query:

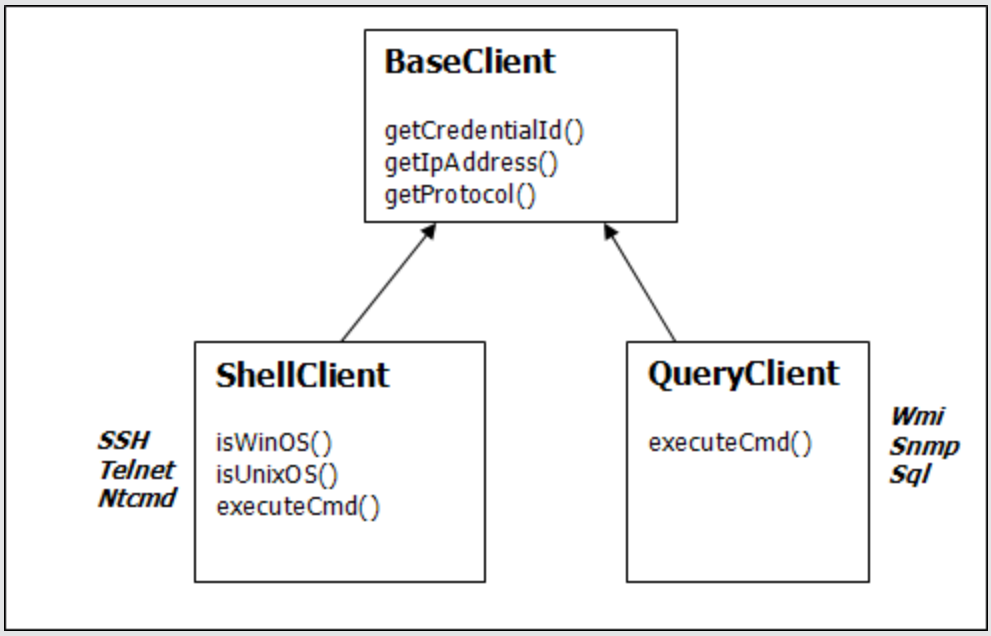
Zo maakt u een WMI-client aan en voert u een WMI-query met de client uit:

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

FROM    Win32_LogicalMemoryConfiguration")
```

Opmerking: om de `createClient()`-API aan het werk te krijgen voegt u de volgende parameter aan de gegevensparameters van het trigger-CI toe: **credentialsId = ${SOURCE.credentials_id}** in het deelvenster Gegevens getriggerd CI. U kunt de referentie-ID ook handmatig toevoegen bij het aanroepen van de functie:
wmiClient = clientFactory().createClient(credentials_id).

In het volgende diagram wordt de hiërarchie van de clients met de bijbehorende algemeen ondersteunde API's getoond:

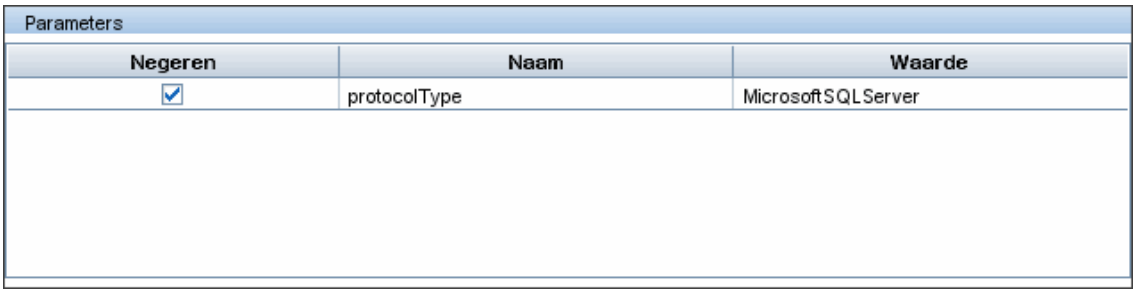


Zie BaseClient, ShellClient en QueryClient in de *HP Discovery and Dependency Mapping Schema Reference* voor meer informatie over de clients en de daardoor ondersteunde API's. De bestanden bevinden zich in de volgende map:

<UCMDB-hoofdmap>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

Framework.getParameter (String parameterName)

U moet naast informatie over het trigger-CI ook vaak een adapterparameterwaarde ophalen. Bijvoorbeeld:



Voorbeeld van het ophalen van de waarde van de parameter protocolType:

Als u de waarde van de parameter `protocolType` wilt ophalen van het Jython-script, gebruikt u de volgende API:

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

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

Sommige fouten (zoals verbindingsfouten, hardwareproblemen, time-outs) kunnen tijdens de uitvoering van een script optreden. Wanneer dergelijke fouten worden gedetecteerd, kan in Framework over het probleem worden gerapporteerd. Het bericht dat wordt gerapporteerd, bereikt de server en wordt voor de gebruiker weergegeven.

Voorbeeld van het rapporteren van een fout en een foutbericht:

In het volgende voorbeeld wordt het gebruik van de API `reportError(<Error Msg>)` getoond:

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

U kunt een van de twee API's, `Framework.reportError(String message)` en `Framework.reportWarning(String message)`, gebruiken om het probleem te rapporteren. Het verschil tussen de twee API's is dat wanneer een fout wordt gerapporteerd, op de probe een communicatielogbestand wordt opgeslagen met de parameters van de gehele sessie voor het bestandssysteem. Op deze manier kunt u de sessie volgen en de fout beter begrijpen.

Zie "Foutberichten" op pagina 79 voor meer informatie over foutberichten.

Juiste referenties vinden (voor verbindingsadapters)

Een adapter die verbinding probeert te maken met een extern systeem, moet alle mogelijke referenties proberen. Een van de parameters die nodig is voor het aanmaken van een client (via ClientFactory) is de referentie-ID. Met het verbindingsscript wordt toegang verkregen tot mogelijke referentiesets en deze sets worden een voor een geprobeerd met de methode `clientFactory.getAvailableProtocols()`. Wanneer één referentieset lukt, wordt met de adapter een CI-verbindingsobject op de host van dit trigger-CI gerapporteerd (met de referentie-ID die overeenkomt met het IP) aan de CMDB. Volgende adapters kunnen dit verbindingsobject-CI rechtstreeks gebruiken om verbinding te maken met de referentieset (dat wil zeggen: de adapters hoeven niet alle mogelijke referenties opnieuw te proberen).

In het volgende voorbeeld wordt getoond hoe alle vermeldingen van het SNMP-protocol worden verkregen. Hier wordt het IP verkregen van de trigger-CI-gegevens (`# Get the Trigger CI data values`).

Met het verbindingsscript worden alle mogelijke protocolreferenties aangevraagd (`# Go over all the protocol credentials`) en geprobeerd in een lus totdat er één lukt (`resultVector`). Zie het item **tweefasenverbindingsparadigma** in "Adapters scheiden" op pagina 25.

```
import logger
from appilog.collectors.clients import ClientsConsts
from appilog.common.system.types.vectors import
ObjectStateHolderVector
        def mainFunction(Framework):
resultVector = ObjectStateHolderVector()
        # Get the Trigger CI data values
        ip_address = Framework.getDestinationAttribute('ip_address')
        ip_domain  = Framework.getDestinationAttribute('ip_domain')
        # Create the client factory for SNMP
        clientFactory = framework.getClientFactory(ClientsConsts.SNMP_
PROTOCOL_NAME)
        protocols = clientFactory.getAvailableProtocols(ip_address,
ip_domain)

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

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

Uitzonderingen van Java afhandelen

Sommige Java-klassen leveren een foutcode op bij een fout. Het wordt aanbevolen de foutcode op te vangen en af te handelen. Anders wordt de adapter onverwacht beëindigd.

Wanneer een bekende foutcode wordt opgevangen, moet u meestal de stacktracering ervan naar het logboek afdrukken en een geschikt bericht doorgeven aan de gebruikersinterface, zoals:

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

Als de uitzondering niet fataal is en het script kan worden voortgezet, moet u de aanroep voor de methode `reportError()` weglaten zodat met het script kan worden doorgegaan.

Lokalisatie in Jython-adapters ondersteunen

Met de functie voor meertalige landinstellingen kunnen verschillende besturingssysteemtalen worden gebruikt in DFM en zijn er geschikte aanpassingen tijdens uitvoering mogelijk.

Voorheen vóór Content Pack 3.00 werd in DFM statisch opgegeven codering gebruikt om uitvoer van alle netwerkdoelen te behandelen. Deze benadering is echter niet geschikt voor een meertalig IT-netwerk: om hosts met verschillende besturingssysteemtalen te detecteren moesten probe-beheerders DFM-taken steeds met andere taakparameters verschillende keren handmatig opnieuw uitvoeren. Door deze procedure ontstond niet alleen een grote extra belasting van het netwerk, maar werd ook niet geprofiteerd van meerdere belangrijke functies van DFM, zoals direct aanroepen van taken in een trigger-CI of automatische vernieuwing van gegevens in UCMDB door Planningsbeheer.

De volgende landinstellingstalen worden standaard ondersteund: Japans, Russisch en Duits. De standaardlandinstelling is Engels.

In dit gedeelte worden de volgende onderwerpen behandeld:

Ondersteuning voor een nieuwe taal toevoegen

Hier wordt beschreven hoe ondersteuning voor een nieuwe taal wordt toegevoegd.

Deze taak omvat de onderstaande stappen:

- "Bronbundel (*.properties-bestanden) toevoegen" beneden
- "Taalobjecten declareren en registreren" beneden

1. ## Bronbundel (*.properties-bestanden) toevoegen

   Voeg een bronbundel toe op basis van de taak die moet worden uitgevoerd. De volgende tabel bevat de DFM-taken en de bronbundel die door elke taak wordt gebruikt:

Taak	Basisnaam van bronbundel
Bestandscontrole per shell	langFileMonitoring
Hostbronnen en applicaties per shell	langHost_Resources_By_TTY, langTCP
Hosts per shell met NSLOOKUP in DNS-server	langNetwork
Hostverbinding per shell	langNetwork
Netwerkgegevens verzamelen per shell of SNMP	langTCP
Hostbronnen en applicaties per SNMP	langTCP
Microsoft Exchange-verbinding per NTCMD, Microsoft Exchange-topologie per NTCMD	msExchange
MS Cluster per NTCMD	langMsCluster

   Zie "Werken met bronbundels" op pagina 55 voor meer informatie over bundels.

2. ## Taalobjecten declareren en registreren

   Als u een nieuwe taal wilt definiëren, voegt u de volgende twee coderegels aan het script **shellutils.py** toe, dat momenteel de lijst met alle ondersteunde talen bevat. Het script wordt meegeleverd in het pakket `AutoDiscoveryContent`. Als u het script wilt bekijken, opent u het venster Adapterbeheer. Zie "Venster Adapterbeheer" in de HP Universal CMDB –

Handleiding Data Flow Management voor meer informatie over dit onderwerp.

a. Declareer de taal als volgt:

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

Zie "API-referentie" op pagina 56 voor meer informatie over de klassetaal. Zie http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html voor meer informatie over het deelvenster Klasse Landinstelling. U kunt een bestaande landinstelling gebruiken of een nieuwe landinstelling definiëren.

b. Registreer de taal door deze aan de volgende verzameling toe te voegen:

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

Standaardtaal wijzigen

Als de taal van het besturingssysteem niet kan worden bepaald, wordt de standaardtaal gebruikt. De standaardtaal wordt in het bestand **shellutils.py** opgegeven.

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

Als u de standaardtaal wilt wijzigen, initialiseert u de variabele DEFAULT_LANGUAGE met een andere taal. Zie "Ondersteuning voor een nieuwe taal toevoegen" op vorige pagina voor meer informatie over dit onderwerp.

Tekenset voor codering bepalen

De geschikte tekenset voor het decoderen van opdrachtuitvoer wordt tijdens de uitvoering bepaald. De meertalige oplossing wordt gebaseerd op de volgende feiten en veronderstellingen:

1. Het is mogelijk de besturingssysteemtaal onafhankelijk van de landinstelling te bepalen, bijvoorbeeld door de opdracht **chcp** in Windows uit te voeren of de opdracht **locale** in Linux.
2. Relation Language-Encoding is alom bekend en kan statisch worden gedefinieerd. De Russische taal heeft bijvoorbeeld twee zeer populaire coderingen: `Cp866` en `Windows-1251`.
3. Eén tekenset voor elke taal verdient de voorkeur. Zo is de voorkeurstekenset voor het Russisch bijvoorbeeld `Cp866`. Dit betekent dat de meeste opdrachten uitvoer in deze codering produceren.
4. Codering waarin de volgende opdrachtuitvoer wordt verschaft is onvoorspelbaar, maar is een van de mogelijke coderingen voor een bepaalde taal. Wanneer bijvoorbeeld een Windows-computer wordt gebruikt met een Russische landinstelling, wordt de opdrachtuitvoer **ver** in Cp866 verschaft, maar de opdracht **ipconfig** in Windows-1251.
5. Een bekende opdracht produceert sleutelwoorden in de uitvoer. De opdracht **ipconfig** bevat bijvoorbeeld de vertaalde vorm van de string **IP-Address**. De opdrachtuitvoer **ipconfig** bevat dus **IP-Address** voor het Engelse besturingssysteem, IP-Адрес voor het Russische besturingssysteem, **IP-Adresse** voor het Duitse besturingssysteem, enzovoort.

Nadat is gedetecteerd in welke taal de opdrachtuitvoer wordt geproduceerd (# 1), worden mogelijke tekensets beperkt tot een of twee (# 2). Bovendien is het bekend welke sleutelwoorden deze uitvoer bevat (# 5).

De oplossing bestaat er daarom uit de opdrachtuitvoer te decoderen met een van de mogelijke coderingen door naar een sleutelwoord in het resultaat te zoeken. Als het sleutelwoord wordt gevonden, wordt de huidige tekenset als de juiste beschouwd.

Nieuwe taken definiëren voor gebruik met gelokaliseerde gegevens

Hier wordt beschreven hoe u een nieuwe taak kunt schrijven die samen met gelokaliseerde gegevens kan worden gebruikt.

Met Jython-scripts worden gewoonlijk opdrachten uitgevoerd en wordt de bijbehorende uitvoer geparseerd. Om deze opdrachtuitvoer op een correct gedecodeerde manier te ontvangen gebruikt u de API voor de klasse **ShellUtils**. Zie "HP Universal CMDB Web Service API - overzicht" op pagina 231 voor meer informatie over dit onderwerp.

Deze code heeft meestal de volgende vorm:

```
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. Maak een client aan:

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

2. Maak een exemplaar van de klasse **ShellUtils** aan en voeg de besturingssysteemtaal eraan toe. Als de taal niet wordt toegevoegd, wordt de standaardtaal gebruikt (doorgaans Engels):

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

   Tijdens objectinitialisatie wordt in DFM de computertaal automatisch gedetecteerd en wordt voorkeurscodering ingesteld op basis van het vooraf gedefinieerde object `Language`. De voorkeurscodering is het eerste exemplaar dat in de coderingslijst verschijnt.

3. Haal de geschikte bronbundel van **shellclient** op met de methode **getLanguageBundle**:

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

4. Haal een sleutelwoord uit de bronbundel op, dat geschikt is voor een bepaalde opdracht:

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

5. Roep de methode **executeCommandAndDecode** aan en geef het sleutelwoord door aan de

methode in het object **ShellUtils**:

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

Het object ShellUtils object is ook nodig om een gebruiker aan de API-referentie te koppelen (waar deze methode gedetailleerd wordt beschreven).

6. Parseer de uitvoer zoals gewoonlijk.

Opdrachten decoderen zonder sleutelwoord

Bij de huidige lokalisatiebenadering wordt een sleutelwoord gebruikt om alle opdrachtuitvoer te decoderen. Zie stap "Haal een sleutelwoord uit de bronbundel op, dat geschikt is voor een bepaalde opdracht:" in "Nieuwe taken definiëren voor gebruik met gelokaliseerde gegevens" op vorige pagina voor meer informatie.

Bij een andere benadering wordt een sleutelwoord gebruikt om alleen de eerste opdrachtuitvoer te decoderen en vervolgens worden meer opdrachten gedecodeerd met de tekenset waarmee de eerste opdracht is gedecodeerd. Hiervoor gebruikt u de methoden **getCharsetName** en **useCharset** van het object **ShellUtils**.

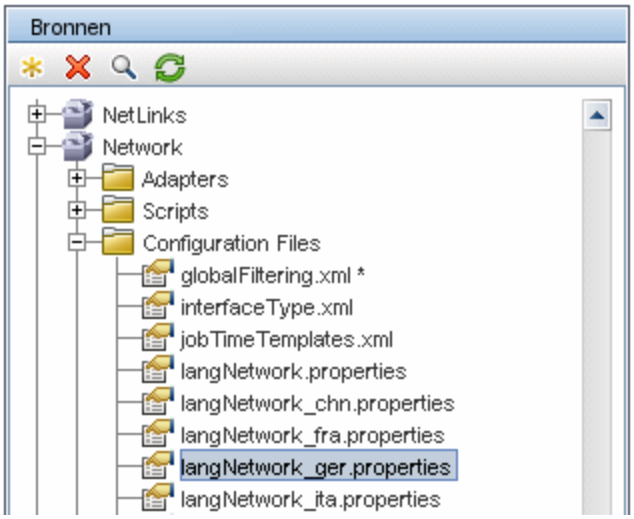
De gewone use case werkt als volgt:

1. Roep de methode **executeCommandAndDecode** eenmaal aan.
2. Haal de naam van de recentst gebruikte tekenset op via de methode **getCharsetName**.
3. Laat **shellUtils** deze tekenset standaard gebruiken door de methode **useCharset** aan te roepen in het object **ShellUtils**.
4. Roep de methode **execCmd** van **ShellUtils** een keer of meerdere keren aan. De uitvoer wordt geretourneerd met de tekenset die is opgegeven in de vorige stap. Er treden geen extra decoderingsbewerkingen op.

Werken met bronbundels

Een bronbundel is een bestand dat de extensie properties heeft (***.properties**). Een eigenschappenbestand kan worden beschouwd als een woordenlijst waarin gegevens worden opgeslagen in de indeling van `sleutel = waarde`. Elke rij in een eigenschappenbestand bevat één koppeling van `sleutel = waarde`. De hoofdfunctionaliteit van een bronbundel bestaat eruit een waarde te retourneren op basis van de bijbehorende sleutel.

Bronbundels bevinden zich op de probe-computer: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles**. Deze worden net als elk ander configuratiebestand van de UCMDB-server gedownload. Ze kunnen worden bewerkt, toegevoegd of verwijderd in het venster Bronnen. Zie "Deelvenster Configuratiebestand" in de HP Universal CMDB – Handleiding Data Flow Management voor meer informatie over dit onderwerp.



Wanneer een bestemming wordt gedetecteerd, moet in DFM doorgaans tekst worden geparseerd van opdrachtuitvoer of bestandsinhoud. Dit parseren wordt vaak gebaseerd op een reguliere expressie. Voor andere talen zijn andere reguliere expressies vereist om voor parseren te worden gebruikt. Voor code die eenmaal voor alle talen moet worden geschreven, moeten alle taalspecifieke gegevens worden geëxtraheerd naar bronbundels. Er is voor elke taal een bronbundel. (Hoewel een bronbundel gegevens voor verschillende talen kan bevatten, bevat één bronbundel in DFM altijd gegevens voor één taal.)

Het Jython-script zelf bevat geen hardcoded, taalspecifieke gegevens (bijvoorbeeld taalspecifieke reguliere expressies). Met het script wordt de taal van het externe systeem bepaald, wordt de juiste bronbundel geladen en worden alle taalspecifieke gegevens op basis van een specifieke sleutel opgehaald.

In DFM hebben bronbundels een specifieke naamindeling: `<base_name>_<language_identifier>.properties`, bijvoorbeeld `langNetwork_spa.properties`. (De standaardbronbundel heeft de volgende indeling: `<base_name>.properties`, bijvoorbeeld `langNetwork.properties`.)

Met de indeling `base_name` wordt het beoogde doel van deze bundel aangegeven. Met bijvoorbeeld **langMsCluster** wordt bedoeld dat de bronbundel taalspecifieke bronnen bevat die door de MS Cluster-taken worden gebruikt.

De indeling `language_identifier` is een acroniem van drie letters waarmee de taal wordt aangegeven. Zo staat `rus` voor de Russische taal en `ger` voor de Duitse taal. Deze taal-ID is opgenomen in de declaratie van het object `Language`.

API-referentie

In dit gedeelte worden de volgende onderwerpen behandeld:

- "Methode getLanguageBundle" op volgende pagina
- "Veld osLanguage" op volgende pagina

Taalklasse

Deze klasse bevat informatie over de taal, zoals postfix van bronbundel, mogelijke codering, enzovoort.

Velden

Naam	Beschrijving
locale	Java-object dat de landinstelling vertegenwoordigt.
bundlePostfix	Postfix van bronbundel. Deze postfix wordt in bestandsnamen van bronbundels gebruikt om de taal te identificeren. Zo bevat de bundel **langNetwork_ger.properties** een **ger**-bundelpostfix.
charsets	Tekensets waarmee deze taal wordt gecodeerd. Elke taal kan meerdere tekensets hebben. Zo wordt de Russische taal doorgaans gecodeerd met de codering `Cp866` en `Windows-1251`.
wmiCodes	De lijst met WMI-codes die door het Microsoft Windows-besturingssysteem worden gebruikt om de taal te identificeren. Alle mogelijke codes worden weergegeven op http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (de sectie OSLanguage). Een van de methoden om de taal van het besturingssysteem te identificeren is een query uit te voeren op het WMI-klassebesturingssysteem voor de eigenschap `OSLanguage`.
codepage	Codetabel die met een specifieke taal wordt gebruikt. Zo wordt `866` gebruikt voor Russische computers en `437` voor Engelse computers. Een van de methoden om de besturingssysteemtaal te identificeren is de bijbehorende standaardcodetabel op te halen (bijvoorbeeld met de opdracht `chcp`).

Methode executeCommandAndDecode

Deze methode is bedoeld voor Jython-scripts voor business logic. Deze methode bevat de decoderingsbewerking en retourneert een gedecodeerde opdrachtuitvoer.

Argumenten

Naam	Beschrijving
cmd	De werkelijke opdracht die moet worden uitgevoerd.
keyword	Het sleutelwoord dat voor de decoderingsbewerking moet worden gebruikt.
framework	Het Framework-object dat aan elk uitvoerbaar Jython-script in DFM is doorgegeven.
timeout	De time-out van de opdracht.

Naam	Beschrijving
waitForTimeout	Hiermee wordt opgegeven of de client moet wachten wanneer de time-out is overschreden.
useSudo	Hiermee wordt opgegeven of `sudo` moet worden gebruikt (alleen relevant voor UNIX-computerclients).
language	Hiermee kan de taal rechtstreeks worden opgegeven in plaats van een taal automatisch te detecteren.

Methode getCharsetName

Met deze methode wordt de naam van de laatst gebruikte tekenset geretourneerd.

Methode useCharset

Met deze methode wordt de tekenset ingesteld in het exemplaar `ShellUtils`, waarin deze tekenset voor initiële gegevensdecodering wordt gebruikt.

Argumenten

Naam	Beschrijving
charsetName	De naam van de tekenset, bijvoorbeeld `windows-1251` of `UTF-8`.

Zie ook "Methode getCharsetName" boven.

Methode getLanguageBundle

Deze methode moet worden gebruikt om de juiste bronbundel op te halen. Hiermee wordt de volgende API vervangen:

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

Argumenten

Naam	Beschrijving
baseName	De naam van de bundel zonder het taalachtervoegsel, bijvoorbeeld `langNetwork`.
language	Het taalobject. `ShellUtils.osLanguage` moet hier worden doorgegeven.
framework	Het Framework, algemeen object dat aan elk uitvoerbaar Jython-script in DFM wordt doorgegeven.

Veld osLanguage

Dit veld bevat een object dat de taal vertegenwoordigt.

Werken met Discovery Analyzer

De tool Discovery Analyzer is bedoeld voor foutopsporing bij het ontwikkelen van pakketten, scripts of andere inhoud. Met de tool wordt een taak uitgevoerd voor een externe bestemming en worden logboeken geretourneerd met informatie, waarschuwings- en foutdetails en resultaten van gedetecteerde CI's.

Resultaten worden niet altijd aan de UI gerapporteerd. Dit komt doordat de resultaten op twee manieren worden gerapporteerd en slechts één manier wordt ondersteund. Bovendien wordt het communicatielogboek niet ondersteund in Eclipse.

Bij het uitvoeren van de tool in Eclipse moet in het bestand **DataFlowProbe.properties** (**C:\hp\UCMDB\DataFlowProbe\conf\DataFlowProbe.properties**) de volgende parameter zijn ingesteld op **true**:

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

Zie "Discovery Analyzer uitvoeren via Eclipse" op pagina 65 voor meer informatie over dit onderwerp.

In alle andere gevallen (wanneer de tool wordt uitgevoerd via het bestand **cmd** of terwijl de probe wordt uitgevoerd), moet deze vlag worden ingesteld op **false**:

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

Taken en records

Een taakbestand bevat gegevens over een taak die moet worden uitgevoerd. De taak bevat informatie zoals de naam van de taak en vereiste parameters waarmee het trigger-CI wordt gedefinieerd, zoals het externe bestemmingsadres.

Een recordbestand bevat naast taakinformatie de resultaten van een specifieke uitvoering, dat wil zeggen: de gedetailleerde communicatie (inclusief een antwoord) tussen de probe of Discovery Analyzer (afhankelijk van de module waarmee de taak is uitgevoerd) en de externe bestemming.

Een taak die wordt gedefinieerd door een taakbestand, kan voor een externe bestemming worden uitgevoerd. Een taak echter die door een recordbestand wordt gedefinieerd (dat extra gegevens over een specifieke uitvoering bevat), kan worden uitgevoerd en kan ook worden afgespeeld (dat wil zeggen: dezelfde uitvoering die in het recordbestand is gedocumenteerd kan worden gereproduceerd).

Logboeken

Logboeken bevatten als volgt informatie over de laatste uitvoering:

- **Algemeen logboek**. Dit logboek bevat alle informatie, fouten en waarschuwingen die tijdens de uitvoering kunnen optreden.
- **Communicatielogboek**. Dit logboek bevat de gedetailleerde communicatie tussen Discovery Analyzer en de externe bestemming (inclusief het antwoord). Na de uitvoering kan het logboek als een recordbestand worden opgeslagen.

- **Resultatenlogboek**. Bevat een lijst met gedetecteerde CI's. De weergave van elk CI is afhankelijk van het ontwerp van de adapters en scripts.

U kunt alle logboeken samen of elk logboek apart opslaan. Wanneer u alle logboeken opslaat, worden ze samen onder één naam opgeslagen.

Als u een recordbestand opnieuw afspeelt, worden dezelfde gegevens in het communicatielogboek weergegeven. Het enige verschil is de tijd van uitvoering.

Beperking: de communicatie- en resultatenlogboeken zijn niet beschikbaar wanneer Discovery Analyzer via Eclipse wordt uitgevoerd.

In dit gedeelte worden de volgende stappen beschreven:

1. ## Vereisten

   - De probe moet worden geïnstalleerd. (Discovery Analyzer wordt geïnstalleerd als onderdeel van het probe-installatieproces en deelt bronnen met de probe.)
   - De probe hoeft niet te worden uitgevoerd terwijl u met Discovery Analyzer werkt.

     Als de probe echter al is uitgevoerd voor een UCMDB-server, zijn alle vereiste bronnen al naar het bestandssysteem gedownload. Als de probe niet is uitgevoerd, kunt u via het menu Instellingen bronnen uploaden die voor Discovery Analyzer nodig zijn. Zie "Import Settings" op pagina 64 voor meer informatie over dit onderwerp.
   - De CMDB-server hoeft niet te worden geïnstalleerd.

2. ## Toegang krijgen tot Discovery Analyzer

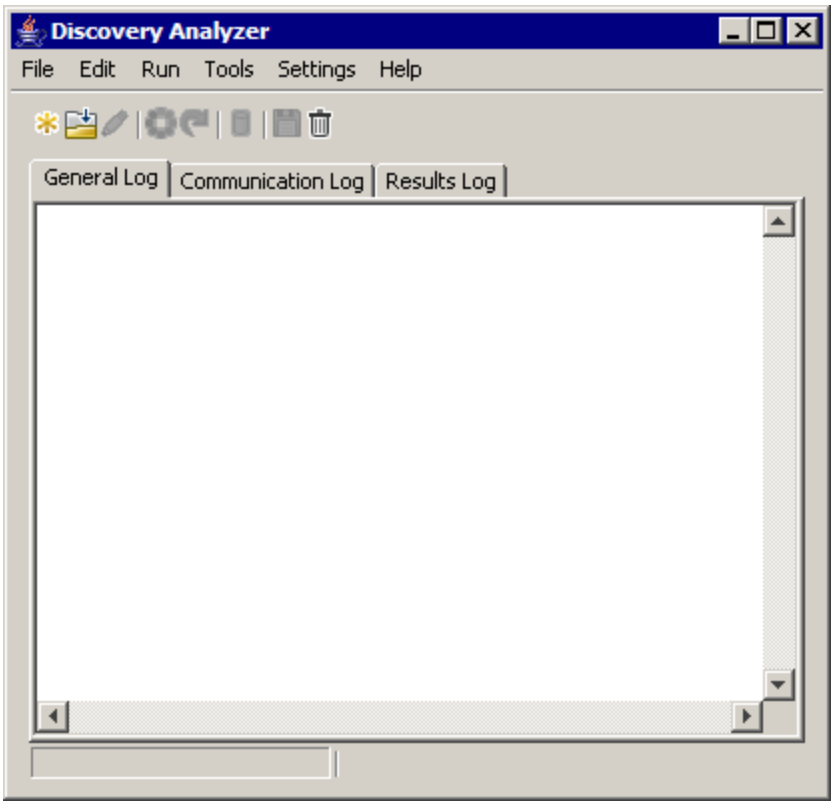
   U kunt op een van de volgende manieren toegang krijgen tot Discovery Analyzer:

- Wanneer u werkt met Eclipse.

  De probe-installatie wordt met een Eclipse-standaardwerkruimte geleverd. Deze ruimte is te vinden op **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**. Deze werkruimte bevat naast een Jython-script om Discovery Analyzer te starten (**startDiscoveryAnalyzerScript.py**) een koppeling naar alle DFM-scripts. Als u de tool op deze manier start, kunt u onderbrekingspunten zoeken in de Jython-scripts voor foutopsporingsdoeleinden.

- Rechtstreeks door op het bestand in de volgende map te dubbelklikken: **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzer.cmd**. Zie het volgende gedeelte voor meer informatie.

Het venster Discovery Analyzer wordt geopend:



3. ## Taken definiëren

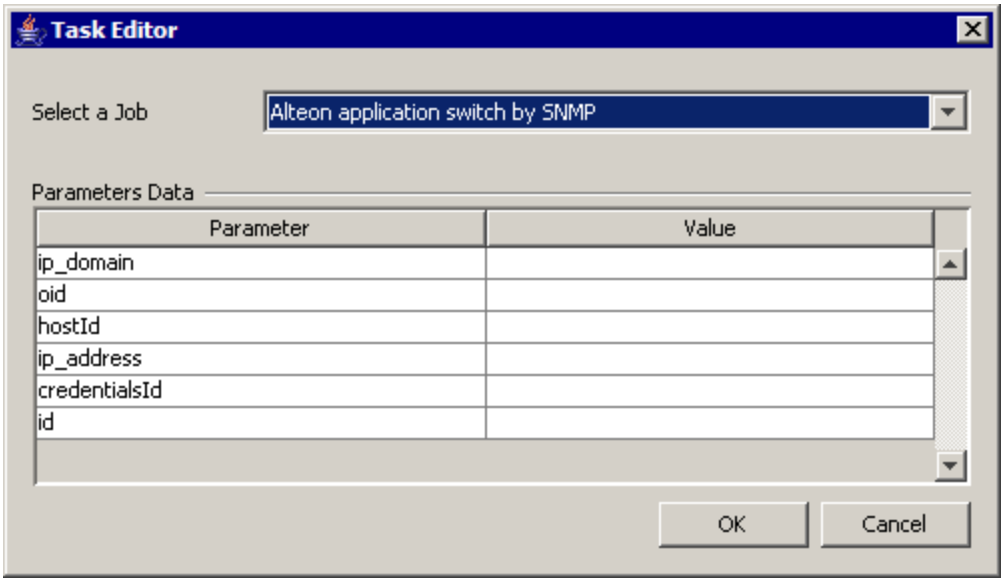
U definieert een taak met een van de volgende methoden:

- Door een nieuwe taak te definiëren. Zie "Nieuwe taken definiëren" op volgende pagina voor meer informatie over dit onderwerp.
- Door een taak vanuit het recordbestand te importeren. Zie "Records ophalen" op volgende pagina voor meer informatie over dit onderwerp.
- Door een opgeslagen taak vanuit een taakbestand te importeren. Zie "Taakbestanden openen" op pagina 63 voor meer informatie over dit onderwerp.
- Door een taak vanuit de interne database van de probe op te halen. Zie "Taken uit de database importeren" op pagina 63 voor meer informatie over dit onderwerp.

4. ## Nieuwe taken definiëren

   a. Geef de taakeditor weer: klik op de knop **New Task** .

   Met de taakeditor wordt een lijst met taken weergegeven die momenteel in het bestandssysteem aanwezig zijn. Deze lijst wordt steeds bijgewerkt wanneer de probe taken ontvangt van de server of wanneer pakketten handmatig in het menu Settings worden uitgerold.



   b. Selecteer een taak.

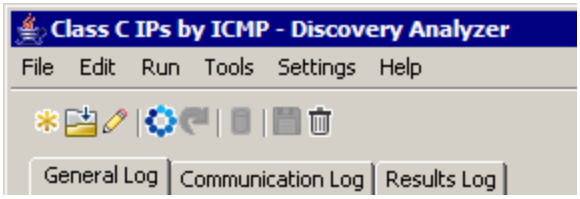
   c. Voer waarden voor alle parameters in.

   De hier weergegeven parameters zijn DFM-adapterparameters. Deze kunnen in het deelvenster Discovery Pattern Parameters op het tabblad Pattern Signature worden weergegeven. Zie "Tabblad Adapterdefinitie" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

   Alle velden zijn verplicht (tenzij het script van een taak vereist dat het veld leeg is).

   Voor parameters die een invoerwaarde van een ID of referentie-ID vereisen, kunt u willekeurig aangemaakte ID's gebruiken: klik met de rechtermuisknop op het waardevak en selecteer **Generate random CMDB ID** of **Credential Chooser**.

   De taak is nu actief en de naam van de geopende taak wordt op de titelbalk weergegeven:



   d. Ga verder met de procedure voor het definiëren van een taak. Zie "Taak en logboeken opslaan" op volgende pagina voor meer informatie over dit onderwerp.

5. ## Records ophalen

U kunt een taak definiëren door een recordbestand te openen met gegevens over een specifieke uitvoering. Als een taak op deze manier wordt gedefinieerd, kunt u de specifieke uitvoering reproduceren door de afspeeloptie te selecteren. (Als een taak opnieuw wordt afgespeeld, worden antwoorden ontvangen van de gegevens die zijn opgeslagen in het recordbestand en niet van de externe bestemming.)

Selecteer **File > Open Record**. Blader naar de map waar u de record hebt opgeslagen. De record is nu actief en de naam van de taak wordt op de titelbalk weergegeven.

Zie "DFM-code opnemen" op pagina 74 voor meer informatie over het verkrijgen van een recordbestand.

6. ## Taakbestanden openen

   U kunt een taak op basis van een taakbestand definiëren: selecteer **File > Open Task**.

7. ## Taken uit de database importeren

   U kunt een taak uit de probe-database ophalen op voorwaarde dat de probe al is uitgevoerd en actieve taken in de interne database heeft. U kunt de parameterwaarden gebruiken om de taak te definiëren.

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

   b. In het dialoogvenster dat wordt geopend, selecteert u de taak die moet worden uitgevoerd en klikt u op **OK**.

   c. Ga verder met de procedure voor het definiëren van een taak. Zie "Taak en logboeken opslaan" beneden voor meer informatie over dit onderwerp.

8. ## Taken bewerken

   Nadat een taak is gedefinieerd, wordt de naam van de taak (of het bestand) op de titelbalk weergegeven. Nu kan het bestand worden bewerkt.

   a. Selecteer **Edit > Edit Task**.

   b. Breng wijzigingen in de taak aan en klik op **OK**.

9. ## Taak en logboeken opslaan

   U kunt taakparameters opslaan: selecteer **File > Save Task**.

   De volgende opties zijn alleen beschikbaar nadat een taak is uitgevoerd.

   - Sla een record van de taak op. U kunt de taakparameters en de resultaten van de taakuitvoering opslaan: Selecteer **File > Save Record**.
   - Sla een logboek van de taak op. Selecteer **File > Save General Log**.
   - Sla resultaten op: selecteer **File > Save Results**.

10. ## Taken uitvoeren

    De volgende stap in de procedure bestaat eruit de taak uit te voeren die u hebt aangemaakt.

    a. Importeer het configuratiebestand voor referenties/bereiken. Zie "Import Settings" op volgende pagina voor meer informatie over dit onderwerp.

    b. Als u de taak alleen voor een externe bestemming wilt uitvoeren, klikt u op de knop **Run**

Task.

De taak wordt in Discovery Analyzer uitgevoerd en er wordt informatie weergegeven in de drie logboekbestanden: **General**, **Communication** en **Results**.

c. U kunt de logboekbestanden samen of afzonderlijk opslaan: selecteer **File > Save General Log**, **Save Record**, **Save Results** of **Save All Logs**. Zie "Logboeken" op pagina 59 voor meer informatie over de logboekbestanden.

d. Als een taak uit een recordbestand wordt opgehaald, kan de in dit bestand gedocumenteerde uitvoering worden gereproduceerd door te klikken op de knop **Playback**. Hetzelfde communicatielogboek wordt weergegeven, maar de uitvoeringstijd wordt bijgewerkt.

11. Taakresultaten naar de server verzenden

Als de uitvoering van een taak wordt beëindigd met resultaten (dat wil zeggen: op het tabblad Results Log wordt een lijst weergegeven met gedetecteerde CI's), kunt u de resultaten naar de UCMDB-server verzenden. Dit is bijvoorbeeld handig als u eerder een script aan het testen was terwijl de server inactief was.

Opmerking: u kunt resultaten alleen naar een UCMDB-server verzenden die taken ontvangt van de probe die op dezelfde computer als Discovery Analyzer wordt geïnstalleerd.

12. Import Settings

Als u taken of het recordbestand voor afspelen wilt uitvoeren, moet u het bestand **domainScopeDocument.bin** importeren. Tijdens het importeren voert u een wachtwoord in.

a. Start een webbrowser en voer de volgende URL in: **http://localhost:8080/jmx-console**. Wellicht zult u zich moeten aanmelden met een gebruikersnaam en wachtwoord.

b. Klik op **UCMDB:service=DiscoveryManager** om de weergavepagina van JMX MBEAN te openen.

c. Zoek naar de bewerking **exportCredentialsAndRangesInformation**. Doe het volgende:

- Voer de klant-ID in (de standaard is **1**).
- Voer een naam in voor het geëxporteerde bestand.
- Voer het wachtwoord in.
- Stel **isEncrypted** op **False** in.

d. Klik op **Invoke** om het bestand **domainScopeDocument.bin** te exporteren.

Wanneer het exportproces succesvol wordt voltooid, wordt het bestand opgeslagen op de volgende locatie: **C:\hp\UCMDB\UCMDBServer\conf\discovery\<customer_dir>**.

e. Kopieer het bestand **domainScopeDocument.bin** naar het bestandssysteem Data Flow-probe en importeer het door het volgende te selecteren: **Settings > Import domainScopeDocument**.

Opmerking: tijdens het importeren van het bestand **domainScopeDocument** wordt

u gevraagd een wachtwoord op te geven. Deze aanvraag wordt ook steeds weergegeven als Discovery Analyzer opnieuw wordt gestart en voordat de eerste taak of record wordt uitgevoerd.

13. **Onderbrekingspunten**

    Als u Discovery Analyzer via het Python-script uitvoert, kunt u onderbrekingspunten aan uw script toevoegen.

14. **Eclipse configureren**

    Zie "Discovery Analyzer uitvoeren via Eclipse" beneden voor meer informatie over het uitvoeren van uw Jython-scripts in de foutopsporingsmodus.

Discovery Analyzer uitvoeren via Eclipse

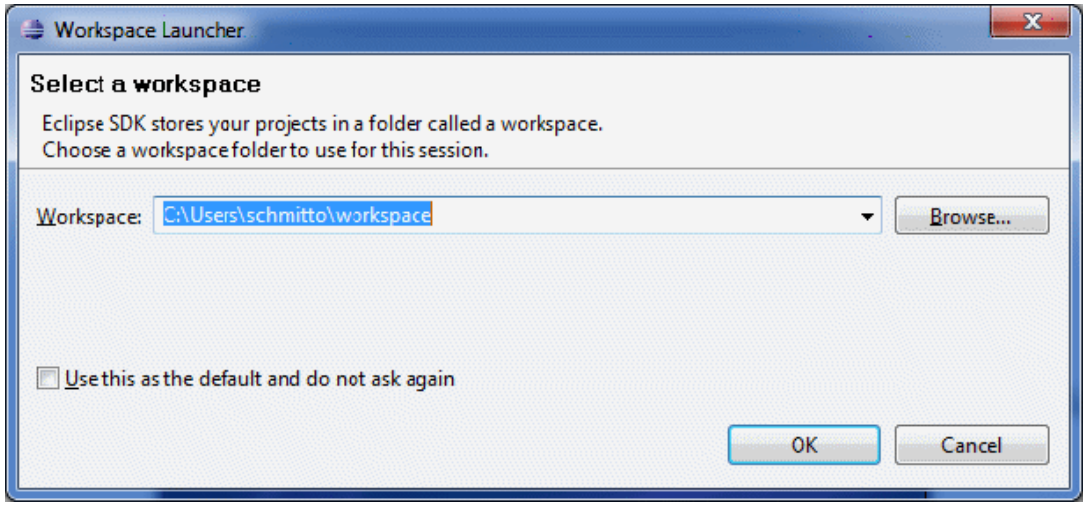
In deze taak wordt uitgelegd hoe Eclipse moet worden geconfigureerd zodat u uw Jython-scripts in de foutopsporingsmodus kunt uitvoeren om een betere zichtbaarheid te verkrijgen voor taak-threads, trigger-CI's en resultaten.

In dit gedeelte worden de volgende stappen beschreven:

- "Vereisten" beneden
- "Eclipse uitpakken en het programma starten" beneden
- "Standaardwerkruimte configureren" beneden
- "PyDev Extensions configureren" op volgende pagina
- "Discovery Analyzer-werkruimte configureren" op pagina 67
- "Klassepad en interpreter configureren" op pagina 71
- "Discovery Analyzer uitvoeren" op pagina 73

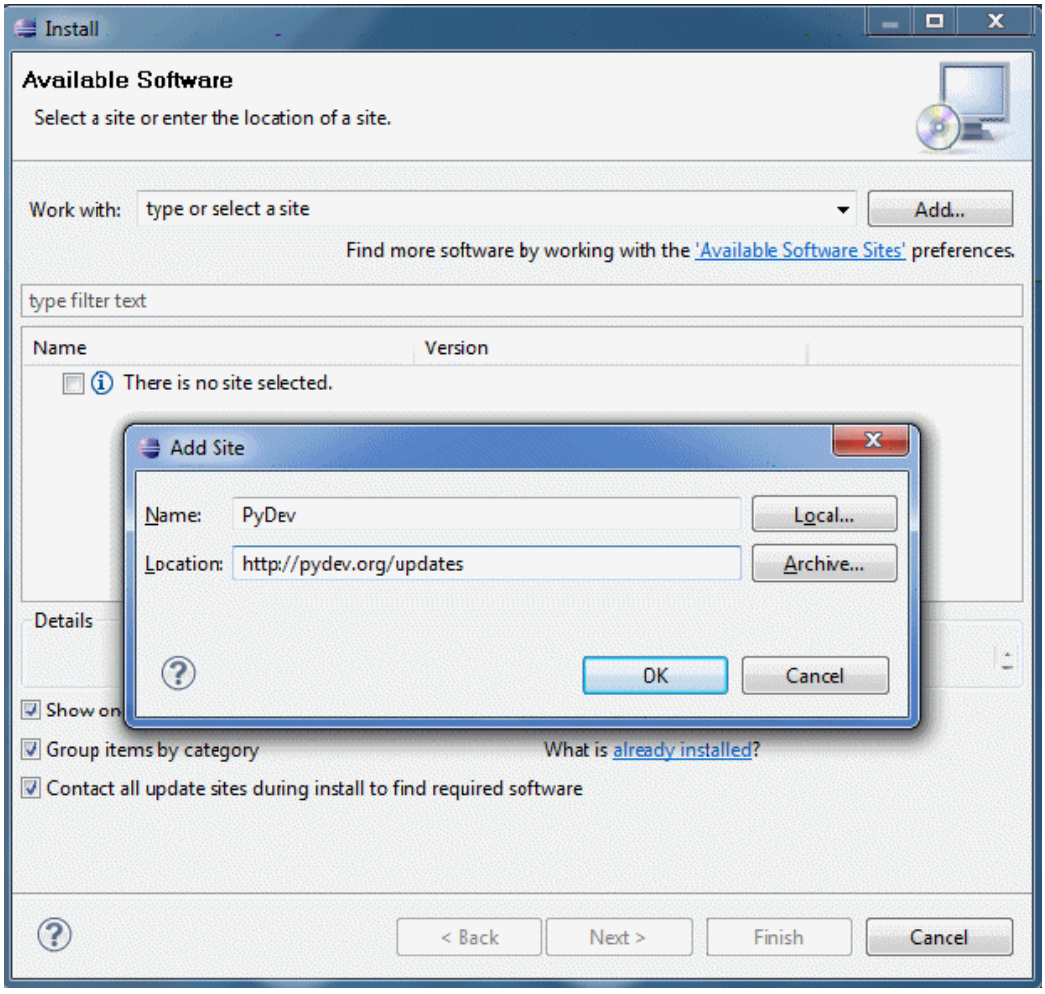
1. **Vereisten**
   - Installeer de laatste Eclipse-versie op uw computer. De applicatie is beschikbaar op www.eclipse.org.
   - Controleer of de Data Flow-probe op dezelfde computer wordt geïnstalleerd.
   - Controleer of de parameter **appilog.agent.local.discoveryAnalyzerFromEclipse** in het bestand **DataFlowProbe.properties** wordt ingesteld op **true**.
2. Eclipse uitpakken en het programma starten
3. **Standaardwerkruimte configureren**

   Configureer de standaardwerkruimte waar alle projecten en gerelateerde gegevens in Eclipse worden opgeslagen en bewaard.



4. ## PyDev Extensions configureren

   a. Open **Help > Install New Software**, klik op **Add**, typ een naam voor de PyDev-invoegtoepassing en voeg in het veld Location de URL van de website toe waar **pydev** kan worden gedownload: **http://pydev.org/updates**. Klik op **OK**.

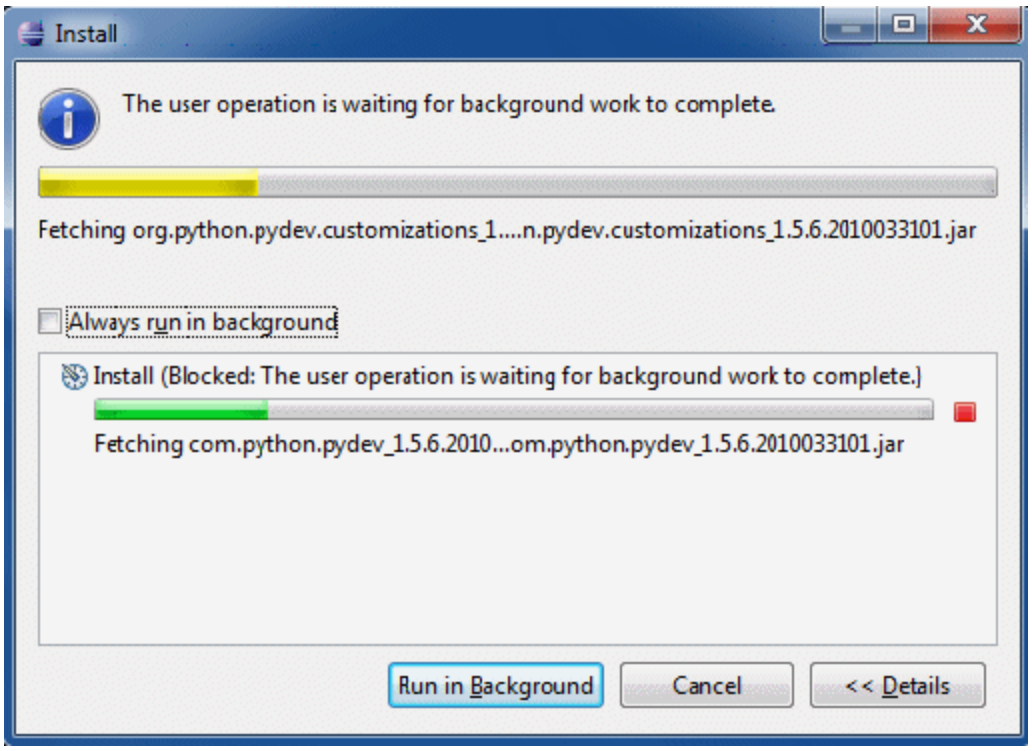


> **Opmerking:** PyDev en PyDev Extensions zijn nu in één invoegtoepassing samengevoegd aangezien PyDev Extensions nu open-source zijn. Ga naar http://pydev.org voor extra informatie.

b. Selecteer **Pydev** in het venster dat wordt geopend. De tweede invoegtoepassing is een invoegtoepassing voor taakgerichte UI. Klik op **Next**, controleer de installatiegegevens en klik nogmaals op **Next**.

c. Accepteer de licentieovereenkomst en klik op **Next**.

d. Pydev wordt geïnstalleerd. Als u wordt gevraagd niet-ondertekende inhoud te installeren, bevestigt u door te klikken op **OK**.

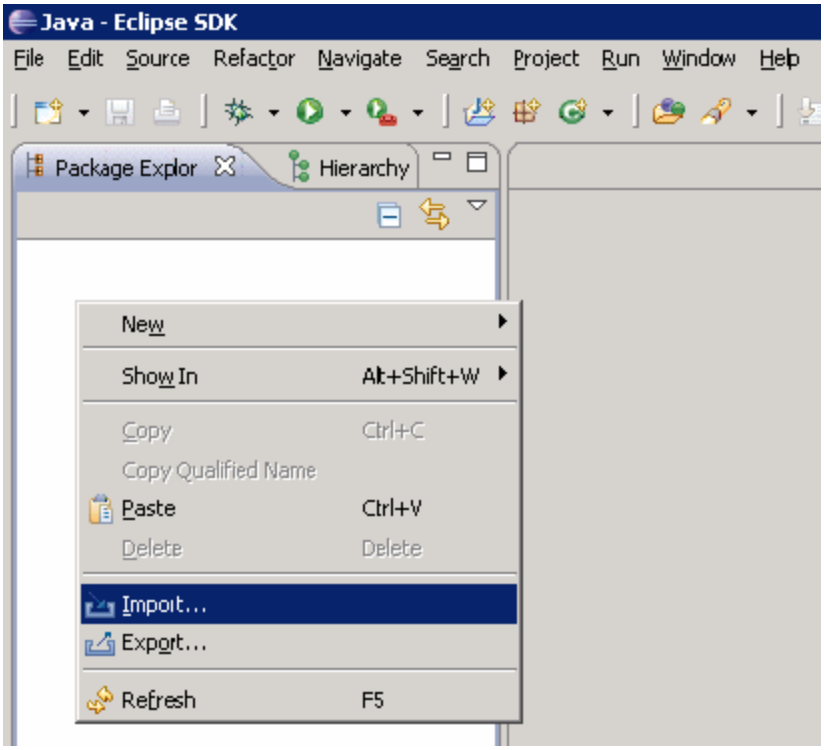


e. Start Eclipse opnieuw.

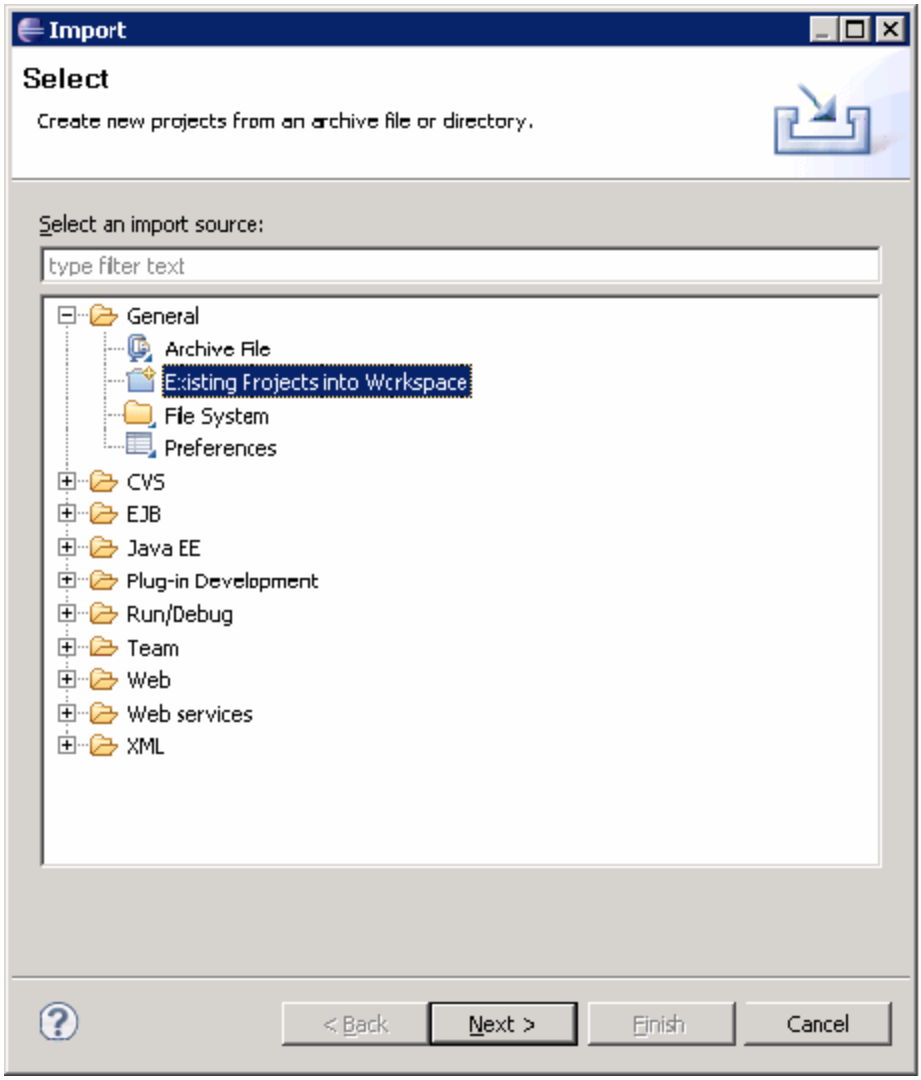
PyDev wordt nu in uw Eclipse-IDE geïnstalleerd. U hebt nieuwe perspectieven in Eclipse en IDE kan Python-scripts interpreteren (tekst markeren, extra configuratieopties, enzovoort).

5. Discovery Analyzer-werkruimte configureren

a. Importeer noodzakelijke bestanden: klik met de rechtermuisknop in het witte gebied in Package Explorer en klik op **Import** om de vooraf geconfigureerde, bij de probe-installatie meegeleverde **discoveryAnalyzerWorkspace** te importeren.



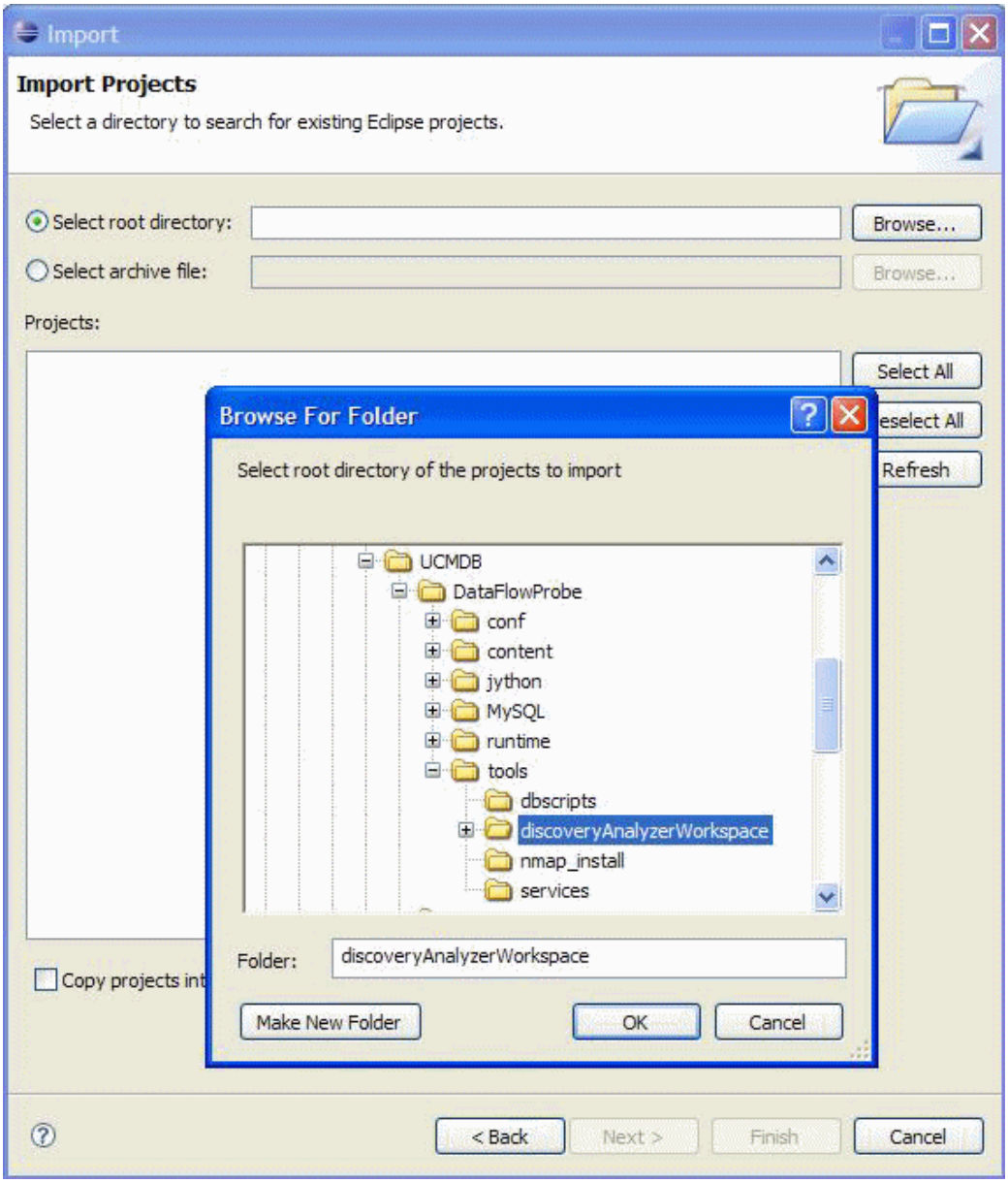
b. Selecteer onder **General** de optie **Existing projects into Workspace** om het project in de Eclipse-werkruimte te importeren.



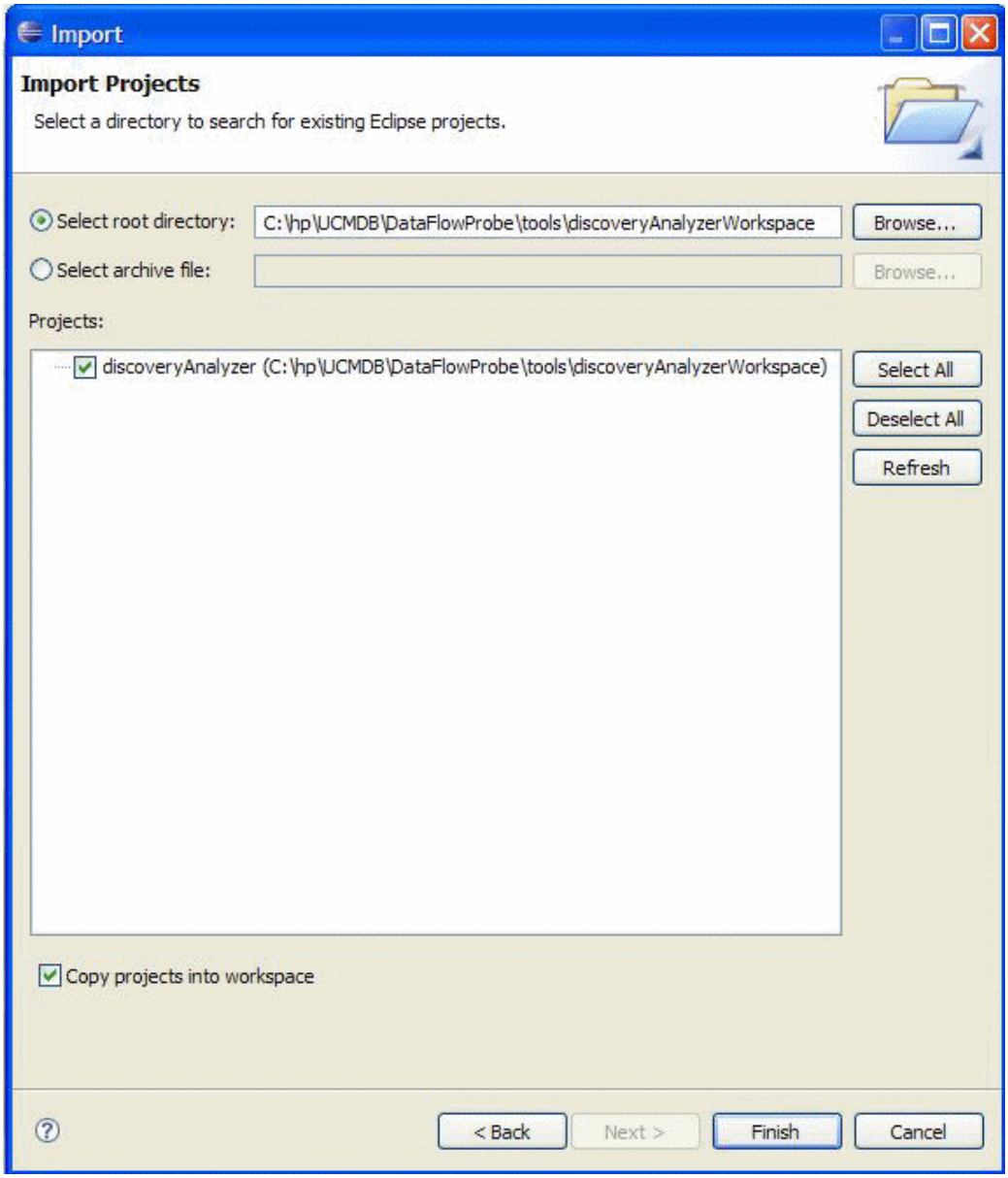
c. Selecteer onder **Select root directory**, de Analyzer-werkruimte die zich meestal bevindt onder :

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

d. Selecteer **Copy projects into workspace** om een echte kopie van de bestaande werkruimte aan te maken. Dit is een belangrijke stap: Als het niet lukt, kunt u de oorspronkelijke **discoveryAnalyserWorkspace** opnieuw importeren.



e. Klik op **Finish** om het importeren te starten.



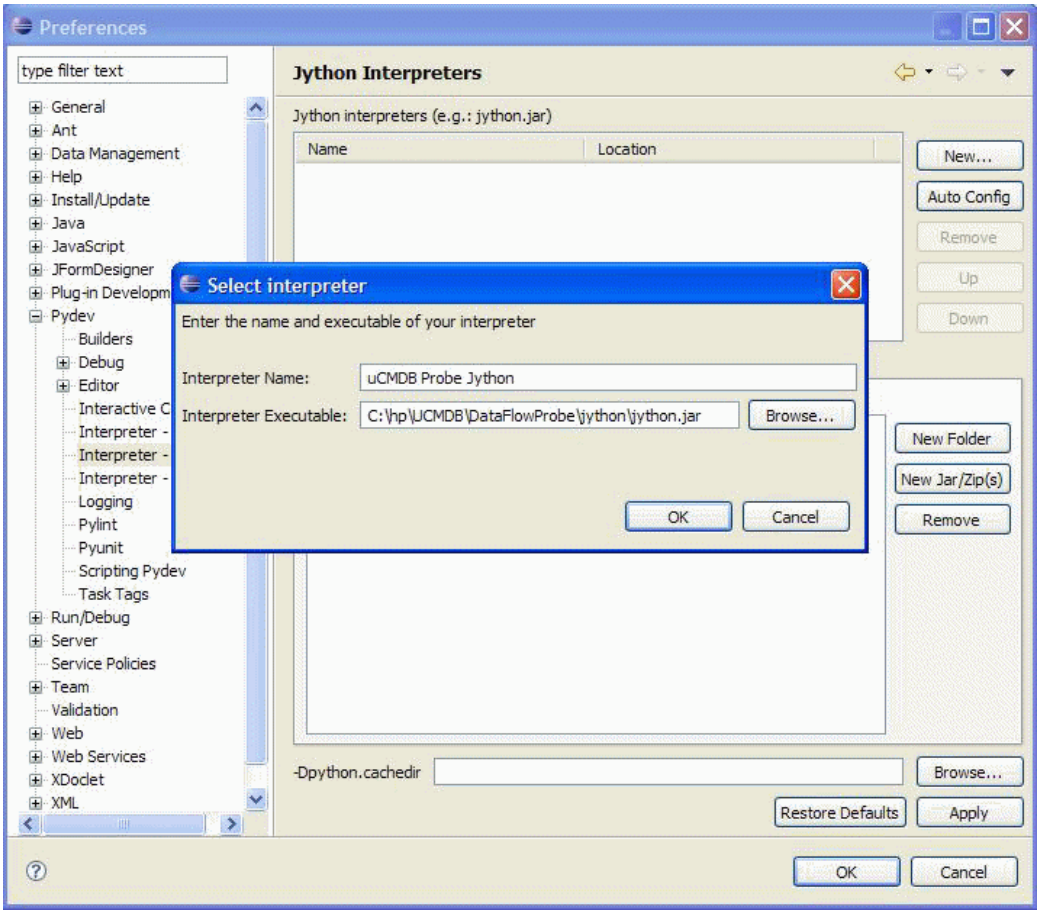
6. ## Klassepad en interpreter configureren

   a. Klik met de rechtermuisknop op **discoveryAnalyzerWorkspace** en selecteer **Properties** om de projectspecifieke instellingen weer te geven.

   b. Ga naar **Pydev > Interpreter/Grammar** en klik op **Please configure an interpreter in the related preferences before proceeding**.

   Met deze stap wordt dezelfde Jython-interpreter geconfigureerd als de probe gebruikt om ervoor te zorgen dat scripts niet door een andere Jython-versie worden geïnterpreteerd.

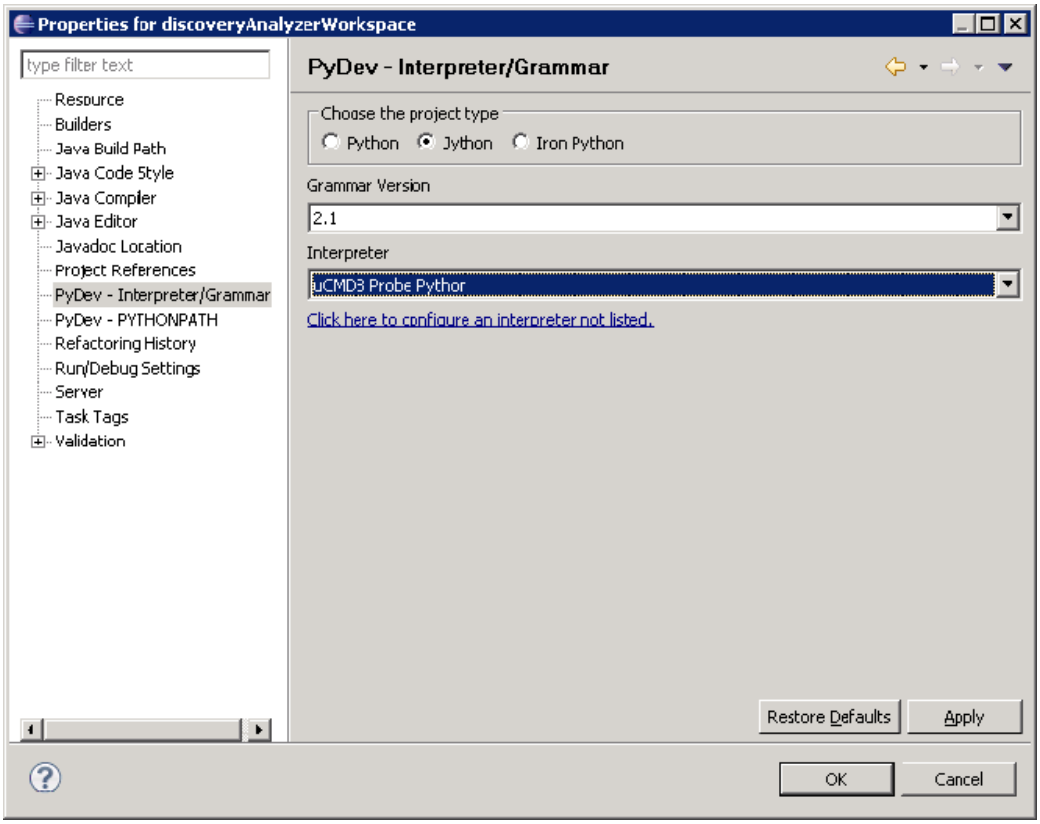
   c. Klik op **New**, typ de naam voor de interpreter en selecteer het bestand in de volgende map: **C:\hp\UCMDB\DataFlowProbe\jython\jython.jar**.



d. Klik op **OK**. Als een venster wordt weergegeven waarin u wordt gevraagd de mappen te selecteren die in uw Python-systeempad moeten worden geïmporteerd, wijzigt u niets (moeten **C:\hp\UCMDB\DataFlowProbe\jython** en **C:\hp\UCMDB\DataFlowProbe\jython\lib** zijn) en klikt u op **OK**.

e. Klik op **Apply** en vervolgens op **OK**.

f. Klik op **Interpreter** en selecteer de zojuist aangemaakte interpreter.



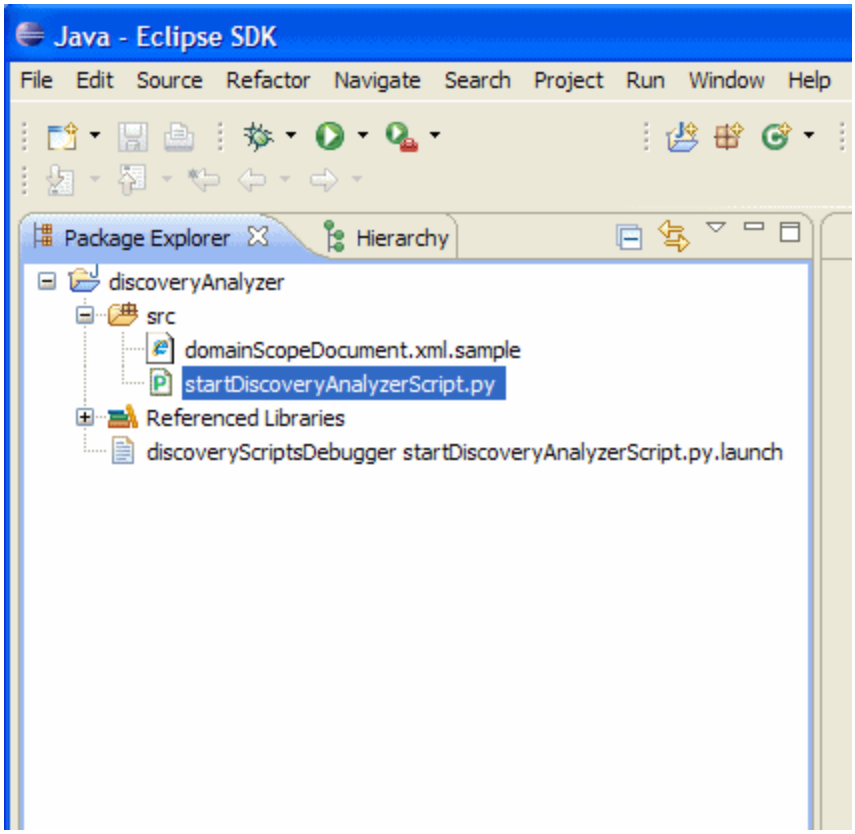
g. Klik op **Apply** en vervolgens op **OK**.

De Jython-interpreter is nu dezelfde die ook door de probe wordt gebruikt.

7. Discovery Analyzer uitvoeren

a. Voeg een onderbrekingspunt in het Jython-script toe waarin fouten moeten worden opgespoord.

b. Als u Discovery Analyzer wilt starten, selecteert u **startDiscoveryAnalyzerScript.py** in het project **discoveryAnalyzerWorkspace\src**. Klik met de rechtermuisknop op het bestand en kies **Debug as > Jython run**.

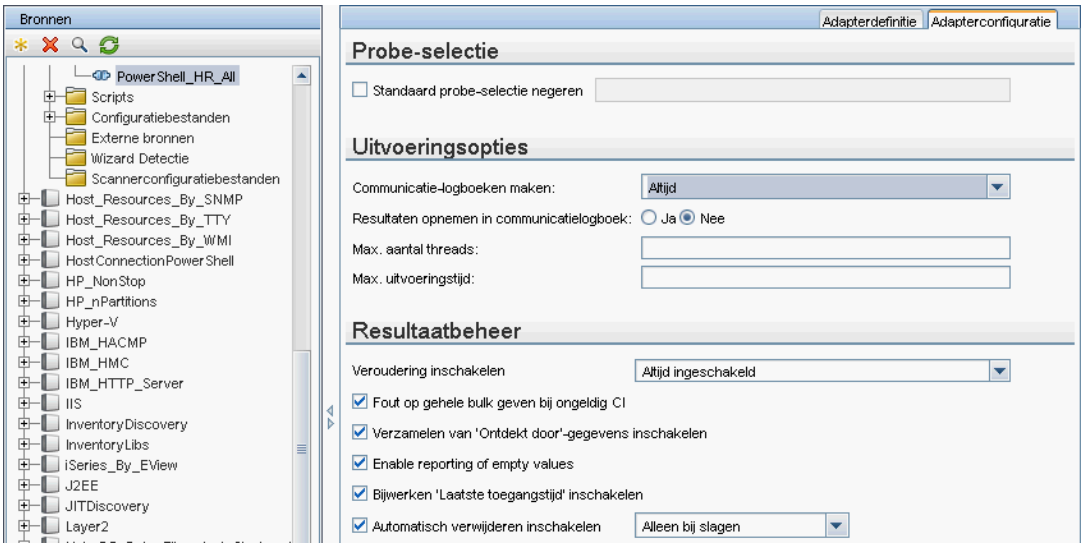


DFM-code opnemen

Het kan erg handig zijn een gehele uitvoering, inclusief alle parameters, op te nemen, zoals bij het opsporen van fouten in code en het testen van code. Met deze taak wordt beschreven hoe een gehele uitvoering met alle relevante variabelen wordt opgenomen. Bovendien kunt u zelfs op foutopsporingsniveau extra foutopsporingsinformatie bekijken die doorgaans niet naar logbestanden wordt afgedrukt.

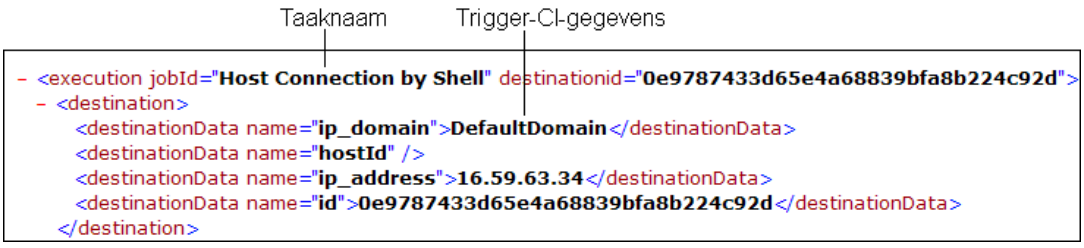
Zo kunt u DFM-code opnemen:

1. Open **Data Flow-beheer > Bedieningspaneel Discovery**. Klik met de rechtermuisknop op de taak waarvan de uitvoering moet worden vastgelegd en selecteer **Naar adapter gaan** om de module Adapterbeheer te openen.
2. Ga naar het deelvenster **Uitvoeringsopties** op het tabblad **Adapterconfiguratie**, zoals hieronder wordt weergegeven:

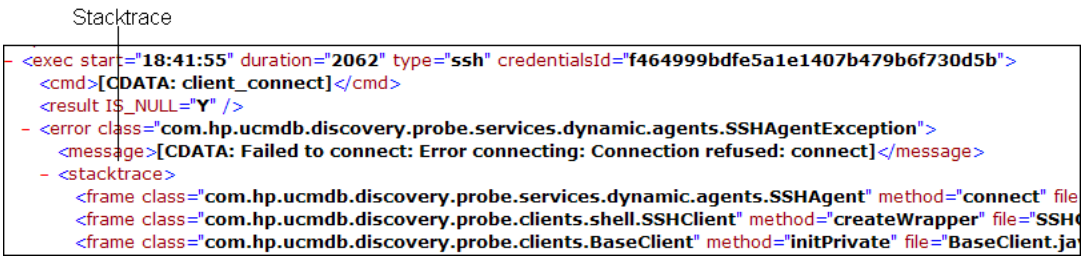


3. Wijzig het vak **Communicatielogboeken maken** in **Altijd**. Zie "Het deelvenster Uitvoeringsopties" in de HP Universal CMDB – Handleiding Data Flow Management voor meer informatie over het instellen van opties voor logboeken.

   Het volgende voorbeeld is het XML-logboekbestand dat wordt aangemaakt wanneer de taak `Hostverbinding per shell` wordt uitgevoerd en het vak **Communicatielogboeken aanmaken** is ingesteld op **Altijd** of **Bij mislukken**:



   In het volgende voorbeeld worden het bericht en de stacktrace-parameters weergegeven:



Jython-bibliotheken en -hulpprogramma's

Verscheidene hulpprogrammascripts worden veel gebruikt in adapters. Deze scripts maken deel uit van het pakket `AutoDiscovery` en bevinden zich onder: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** met de andere scripts die naar de probe worden gedownload.

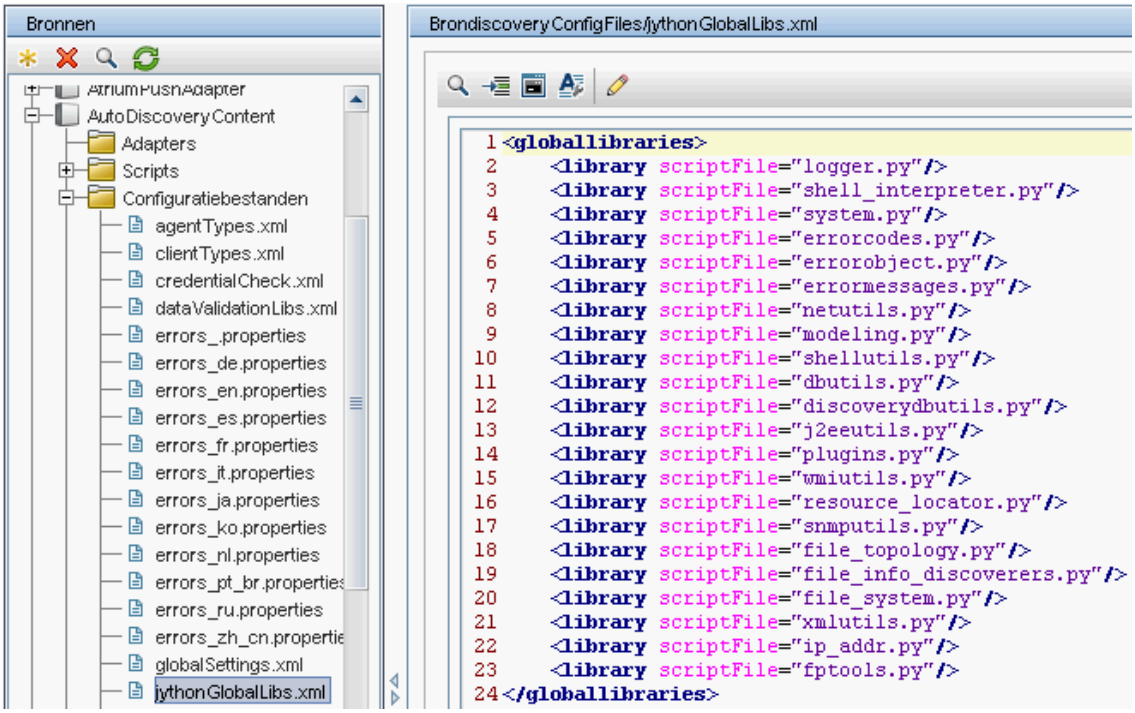
Opmerking: de map `discoveryScript` wordt dynamisch aangemaakt wanneer de probe

begint te werken.

Als u een van de hulpprogrammascripts wilt gebruiken, voegt u de volgende import-regel aan de import-sectie van het script toe:

```
import <scriptnaam>
```

De Python-bibliotheek van AutoDiscovery bevat Jython-hulpprogrammascripts. Deze bibliotheekscripts worden beschouwd als externe bibliotheek van DFM. Ze worden gedefinieerd in het bestand `jythonGlobalLibs.xml` (dat zich bevindt in de map **Configuratiebestanden**).



Elk script dat wordt weergegeven in het bestand `jythonGlobalLibs.xml` wordt standaard geladen bij het opstarten van de probe. U hoeft ze dus niet expliciet in de adapterdefinitie te gebruiken.

In dit gedeelte vindt u de volgende onderwerpen:

logger.py

Het script **logger.py** bevat logboekhulpprogramma's en helper-functies voor rapportage van fouten. U kunt de bijbehorende API's voor foutopsporing, informatie en fouten aanroepen om naar de logboekbestanden te schrijven. Logboekberichten worden geregistreerd in **C:\hp\UCMDB\DataFlowProbe\runtime\log**.

Berichten worden in het logboekbestand ingevoerd in overeenstemming met het foutopsporingsniveau dat is gedefinieerd voor de appender PATTERNS_DEBUG in het bestand **C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties**. (Standaard is het niveau DEBUG.) Zie "Ernstniveaus van fouten" op pagina 82 voor meer informatie over dit onderwerp.

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

De informatie- en foutberichten worden ook weergegeven in de opdrachtpromptconsole.

Er zijn twee sets met API's:

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

Met de eerste set wordt de samenvoeging van alle bijbehorende stringargumenten op het juiste logboekniveau geleverd en met de tweede set wordt de samenvoeging tegelijk met de stacktracering van de meest recente foutcode geleverd om meer informatie te verschaffen, bijvoorbeeld:

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

modeling.py

Het script **modeling.py** bevat API's voor het aanmaken van hosts, IP's, proces-CI's, enzovoort. Met deze API's kunnen algemene objecten worden aangemaakt en wordt de code leesbaarder. Bijvoorbeeld:

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

netutils.py

De bibliotheek **netutils.py** wordt gebruikt voor het ophalen van netwerk- en TCP-informatie, zoals het ophalen van besturingssysteemnamen, het controleren of een MAC-adres geldig is, het controleren of een IP-adres geldig is, enzovoort. Bijvoorbeeld:

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

shellutils.py

Met de bibliotheek **shellutils.py** wordt een API verschaft voor het uitvoeren van shellopdrachten en het ophalen van de eindstatus van een uitgevoerde opdracht, en kunnen meerdere opdrachten op basis van die eindstatus worden uitgevoerd. De bibliotheek wordt geïnitialiseerd met een shellclient en met de client worden opdrachten uitgevoerd en resultaten opgehaald. Bijvoorbeeld:

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

Hoofdstuk 3

Foutberichten

In dit hoofdstuk vindt u de volgende informatie:

Foutberichten - overzicht

Tijdens de discovery kunnen veel fouten worden ontdekt, zoals verbindingsfouten, hardwareproblemen, uitzonderingen, time-outs, enzovoort. In DFM worden deze fouten telkens wanneer de gewone discovery-stroom mislukt, weergegeven in het Bedieningspaneel Discovery, zowel in de Basismodus als de Geavanceerde modus. U kunt details weergeven van het trigger-CI dat het probleem heeft veroorzaakt om het foutbericht zelf te bekijken.

In DFM wordt onderscheid gemaakt tussen fouten die soms kunnen worden genegeerd (een onbereikbare host bijvoorbeeld) en fouten die moeten worden aangepakt (problemen met referenties bijvoorbeeld of ontbrekende configuratie- of DLL-bestanden). Bovendien worden fouten in DFM eenmaal gerapporteerd, zelfs als dezelfde fout in opeenvolgende uitvoeringen optreedt. Ook als een fout slechts eenmaal optreedt, wordt deze gerapporteerd.

Wanneer een pakket wordt aangemaakt, kunt u bijbehorende berichten als bronnen aan het pakket toevoegen. Tijdens de uitrol van een pakket worden de berichten ook op de juiste locatie uitgerold. Berichten moeten voldoen aan conventies, zoals wordt beschreven in "Conventies voor het schrijven van foutberichten" beneden.

In DFM worden meertalige foutberichten ondersteund. U kunt de berichten die u schrijft lokaliseren, zodat deze in de lokale taal worden weergegeven.

Zie "Deelvenster Detectiestatus" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over het zoeken naar fouten.

Zie "Het deelvenster Uitvoeringsopties" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over het instellen van communicatielogboeken.

Conventies voor het schrijven van foutberichten

- Elke fout wordt geïdentificeerd door een foutberichtcode en een matrix van argumenten (**int, String[]**). Met een combinatie van een berichtcode en een matrix van argumenten wordt een specifieke fout gedefinieerd. De matrix van argumenten kan null zijn.
- Elke foutcode wordt toegewezen aan een **kort bericht** dat bestaat uit een vaste string en een **gedetailleerd bericht** dat een sjabloonstring is met nul of meer argumenten. Er wordt van uitgegaan dat het aantal argumenten in de sjabloon en het werkelijke aantal parameters

overeenkomen.

Voorbeeld van foutberichtcode:

10234 kan een fout zijn met het korte bericht:

```
Verbindingsfout
```

en het gedetailleerde bericht:

```
Kan niet verbinden via protocol {0} vanwege time-out van {1} msec
```

waarbij

{0} = het eerste argument: een protocolnaam

{1} = het tweede argument: de duur van de time-out in msec

In dit gedeelte vindt u ook de volgende onderwerpen:

- "Inhoud van eigenschappenbestand" beneden
- "Eigenschappenbestand voor foutberichten" beneden
- "Naamgevingsconventies voor landinstellingen" beneden
- "Foutberichtcodes" op volgende pagina
- "Niet-geclassificeerde inhoudsfouten" op volgende pagina
- "Wijzigingen in framework" op pagina 82

Inhoud van eigenschappenbestand

Een eigenschappenbestand moet twee sleutels voor elke foutberichtcode bevatten. Bijvoorbeeld voor fout 45:

- **DDM_ERROR_MESSAGE_SHORT_45**. Korte foutbeschrijving.
- **DDM_ERROR_MESSAGE_LONG_45**. Lange foutbeschrijving (kan parameters bevatten, bijvoorbeeld {**0**},{**1**}).

Eigenschappenbestand voor foutberichten

Een eigenschappenbestand bevat een koppeling tussen een foutberichtcode en twee berichten (kort en gedetailleerd).

Nadat een eigenschappenbestand is uitgerold, worden de bijbehorende gegevens samengevoegd met bestaande gegevens. Dat wil zeggen dat nieuwe berichtcodes worden toegevoegd terwijl oude berichtcodes worden overschreven.

Eigenschappenbestanden voor infrastructuur maken deel uit van het pakket **AutoDiscoveryInfra**.

Naamgevingsconventies voor landinstellingen

- Voor de standaardlandinstelling: **<bestandsnaam>.properties.errors**
- Voor een specifieke landinstelling: **<bestandsnaam>_xx.properties.errors**

waarbij **xx** de landinstelling is (bijvoorbeeld **infraerr_fr.properties.errors** of **infraerr_en_us.properties.errors**).

Foutberichtcodes

De volgende foutcodes zijn standaard opgenomen in HP Universal CMDB. U kunt uw eigen foutberichten aan deze lijst toevoegen.

Naam fout	Foutcode	Beschrijving
Intern	100-199	Meestal afgeleid van uitzonderingen die tijdens Jython-scriptuitvoeringen zijn opgetreden
Verbinding	200-299	Verbinding mislukt, geen agent op doelcomputer, bestemming onbereikbaar, enzovoort
Gerelateerd aan referenties	300-399	Toegang geweigerd, verbindingspoging geblokkeerd doordat referenties ontbreken
Time-out	400-499	Time-out opgetreden tijdens verbinding/opdracht
Onverwachte fout of ongeldig gedrag	500-599	Ontbrekende configuratiebestanden, onverwachte onderbrekingen, enzovoort
Ophalen van informatie	600-699	Ontbrekende informatie op doelcomputers, fout bij uitvoeren van query op agent voor informatie, enzovoort
Gerelateerd aan bronnen	700-799	Fouten die te maken hebben met onvoldoende geheugen of clients die niet juist zijn vrijgegeven
Parseren	800-899	Fout bij parseren van tekst
Codering	900	Fout in invoer, niet-ondersteunde codering
Gerelateerd aan SQL	901-903, 924	Fouten die van SQL-bewerkingen zijn ontvangen
Gerelateerd aan HTTP	904-909	Fouten gegenereerd tijdens HTTP-verbindingen, geparseerd uit HTTP-foutcodes.
Specifieke applicatie	910-923	Fout gerapporteerd vanwege applicatiespecifieke problemen, bijvoorbeeld verkeerde LSOF-versie, geen wachtrijbeheerders gevonden, enzovoort

Niet-geclassificeerde inhoudsfouten

Om oude inhoud te ondersteunen zonder een regressie te veroorzaken worden fouten met berichtcode 100 (dat wil zeggen: niet-geclassificeerde scriptfout) met de voor applicaties en SDK relevante methoden anders afgehandeld.

Deze fouten worden niet gegroepeerd op basis van de bijbehorende berichtcode (ze worden dus niet beschouwd als fouten van hetzelfde type), maar ze worden gegroepeerd op basis van de inhoud van het bericht. Dat wil zeggen: als een fout door een script met de oude, achterhaalde methoden wordt gerapporteerd (met een berichtstring en zonder foutcode), krijgen alle berichten dezelfde

foutcode. In de voor applicaties of SDK relevante methoden worden verschillende berichten echter als verschillende fouten weergegeven.

Wijzigingen in framework

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

De volgende methoden worden aan de interface toegevoegd:

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

De volgende oude methoden worden nog steeds ondersteund voor achterwaartse compatibiliteitsdoeleinden, maar worden als verouderd gemarkeerd:

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

Ernstniveaus van fouten

Wanneer een adapter is uitgevoerd voor een trigger-CI, wordt een status geretourneerd. Als geen fout of waarschuwing wordt gerapporteerd, is de status **Geslaagd**.

Ernstniveaus worden hier van het meest beperkte tot het breedste bereik weergegeven:

Fatale fouten

Op dit niveau worden ernstige fouten gerapporteerd, zoals een probleem met de infrastructuur, ontbrekende DLL-bestanden of uitzonderingen:

- Kan de taak niet genereren (probe is niet gevonden, variabelen zijn niet gevonden, enzovoort).
- Het script kan niet worden uitgevoerd.
- De resultaten kunnen niet op de server worden verwerkt en de gegevens worden niet naar de CMDB geschreven.

Fouten

Op dit niveau worden problemen gerapporteerd die ervoor zorgen dat geen gegevens kunnen worden opgehaald in DFM. Bekijk deze fouten aangezien er meestal bepaalde actie voor moet worden ondernomen (bijvoorbeeld om een time-outwaarde te verhogen, een bereik te wijzigen, een parameter te wijzigen, nog een gebruikersreferentie toe te voegen, enzovoort).

- In gevallen waar gebruikersinterventie kan helpen, wordt een fout gerapporteerd, die een probleem met referenties of het netwerk betreft, waarvoor mogelijk nader onderzoek vereist is. (Dit zijn geen fouten in discovery maar in configuratie.)
- Interne fout, gewoonlijk vanwege onverwacht gedrag met betrekking tot de gedetecteerde computer of applicatie, zoals ontbrekende configuratiebestanden, enzovoort.

Waarschuwingen

Wanneer een uitvoering lukt maar er wel sprake is van minder ernstige problemen waarmee u

rekening moet houden, wordt in DFM de ernstgraad gemarkeerd als **Waarschuwing**. U moet deze CI's bekijken om te zien of er gegevens ontbreken alvorens met een gedetailleerdere foutopsporingssessie te beginnen. **Waarschuwing** kan berichten bevatten over het ontbreken van een geïnstalleerde agent op een externe host of berichten dat een attribuut niet juist is berekend door ongeldige gegevens.

- Ontbrekende verbindingsagent (SNMP, WMI)
- Discovery lukt, maar niet alle beschikbare informatie is gedetecteerd

Hoofdstuk 4

Algemene database-adapters ontwikkelen

In dit hoofdstuk vindt u de volgende informatie:

Algemene database-adapter - overzicht

Het doel van het platform voor de algemene database-adapter bestaat eruit adapters aan te maken die kunnen worden geïntegreerd met relationele databasebeheersystemen (RDBMS), en TQL-query's en vullingstaken voor de database uit te voeren. De RDBMS-systemen die door de algemene database-adapter worden ondersteund, zijn Oracle, Microsoft SQL Server en MySQL.

Deze versie van de database-adapterimplementatie is gebaseerd op de JPA-standaard (Java Persistence API) met de Hibernate ORM-bibliotheek als de persistentieprovider.

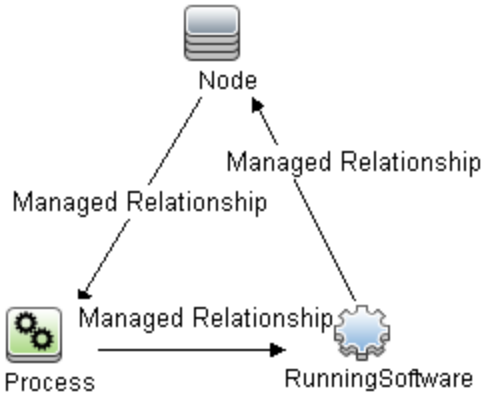
TQL-query's voor de algemene database-adapter

Voor vullingstaken moet elke vereiste indeling van een CI worden gecontroleerd in het dialoogvenster Indelingsinstellingen in Modeling Studio. Zie "Dialoogvenster Eigenschappen query-knooppunt/Relatie-eigenschappen" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp. Het is belangrijk te weten dat er voor een CI wellicht een attribuut moet worden geïdentificeerd. Zonder een dergelijk attribuut kan het CI niet worden toegevoegd aan UCMDB.

De volgende beperkingen gelden alleen voor de TQL-query's die door de algemene database-adapter worden berekend:

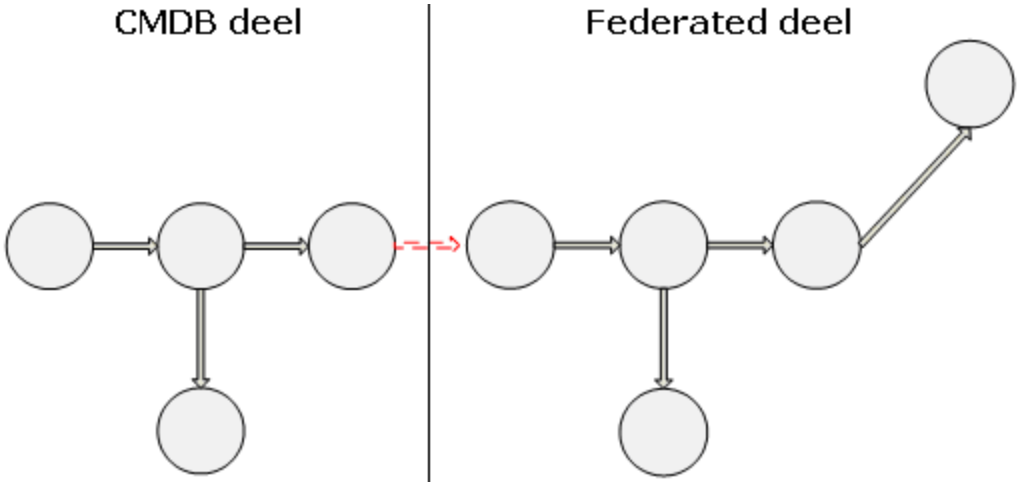
- Subgrafieken worden niet ondersteund
- Compound-relaties worden niet ondersteund
- Cyclussen of cyclusdelen worden niet ondersteund

  De volgende TQL-query is een voorbeeld van een cyclus:



- Functie-indeling wordt niet ondersteund.
- 0..0-kardinaliteit wordt niet ondersteund.
- De `Join`-relatie wordt niet ondersteund.
- Kwalificatorvoorwaarden worden niet ondersteund.
- Voor verbinding tussen twee CI's moet er een relatie in de vorm van een tabel of externe sleutel

aanwezig zijn in de externe databasebron.



Afstemming

Afstemming wordt als onderdeel van de TQL-berekening uitgevoerd op de adapter. Om de afstemming te laten plaatsvinden wordt de CMDB-zijde toegewezen aan een federated entiteit die afstemmings-CIT wordt genoemd.

Toewijzing Elk attribuut in de CMDB wordt aan een kolom in de gegevensbron toegewezen.

Hoewel toewijzing rechtstreeks plaatsvindt, worden transformatiefuncties voor de toewijzingsgegevens ook ondersteund. U kunt nieuwe functies toevoegen via de Java-code (bijvoorbeeld kleine letters, hoofdletters). Het doel van deze functies is waardeconversies in te schakelen (waarden die in de CMDB in de ene indeling worden opgeslagen en in de federated database in een andere indeling).

Opmerking:

- Om de CMDB en externe databasebron te verbinden moet er een geschikte koppeling in de database aanwezig zijn. Zie "Vereisten" op pagina 89 voor meer informatie over dit onderwerp.
- Afstemming met de CMDB-ID wordt eveneens ondersteund
- Afstemming met de algemene ID wordt eveneens ondersteund.

Hibernate als JPA-provider

Hibernate is een OR-toewijzingstool (Object-Relational), waarmee Java-klassen kunnen worden toegewezen aan tabellen in verschillende typen relationele databases (zoals Oracle en Microsoft SQL Server). Zie "Functionele beperkingen" op pagina 154 voor meer informatie over dit onderwerp.

In een elementaire toewijzing wordt elke Java-klasse aan één tabel toegewezen. Met geavanceerdere toewijzing is overervingstoewijzing mogelijk (zoals kan gebeuren in de CMDB-database).

Andere ondersteunde functies omvatten toewijzing van een klasse aan verschillende tabellen, ondersteuning voor verzamelingen en koppelingen van het type een-op-een, een-op-veel en veel-op-een. Zie "Koppelingen" op volgende pagina hieronder voor meer informatie.

Voor onze doelen hoeven er geen Java-klassen aangemaakt te worden. De toewijzing wordt gedefinieerd op basis van de klassemodel-CIT's van de CMDB aan de databasetabellen.

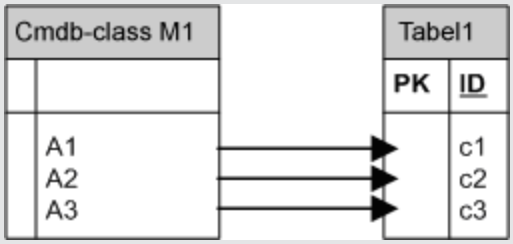
In dit gedeelte vindt u ook de volgende onderwerpen:

Voorbeeld van OR-toewijzing

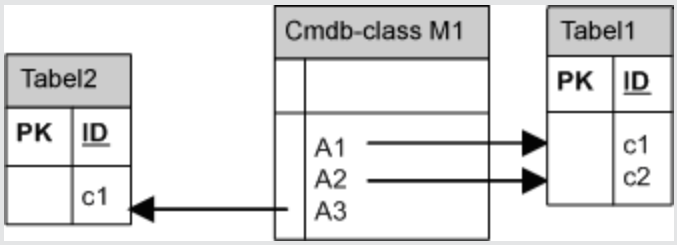
In de volgende voorbeelden wordt OR-toewijzing (Object-Relational) beschreven:

Voorbeeld van 1 CMDB -klasse toegewezen aan 1 databasetabel:

Klasse `M1`, met attributen `A1`, `A2` en `A3`, wordt toegewezen aan kolommen `c1`, `c2` en `c3` van tabel `1`. Dit betekent dat een `M1`-exemplaar een overeenkomende rij in tabel `1` heeft.

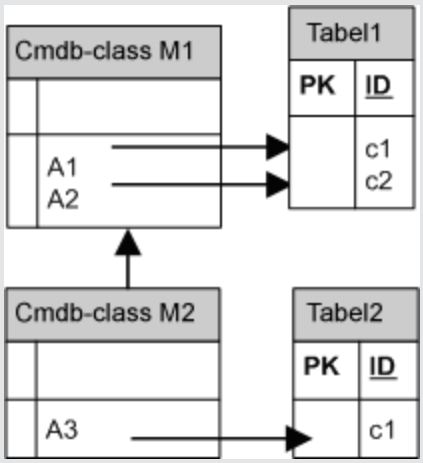


Voorbeeld van 1 CMDB -klasse toegewezen aan 2 databasetabellen:



Voorbeeld van overerving:

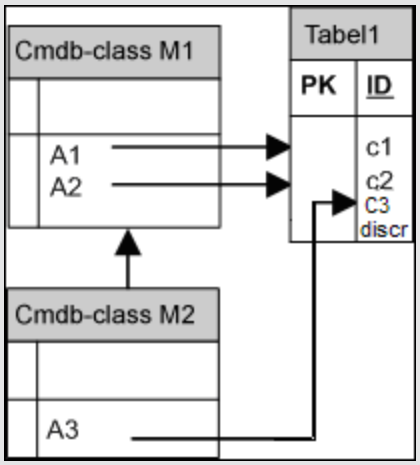
Dit geval wordt gebruikt in de CMDB, waarin elke klasse een eigen databasetabel heeft.



Voorbeeld van overerving van één tabel met Discriminator:

Een gehele hiërarchie van klassen wordt toegewezen aan één databasetabel, waarvan de kolommen een superset met alle attributen van de toegewezen klassen bevatten. De tabel bevat ook een aanvullende kolom (`Discriminator`), waarvan de waarde aangeeft welke specifieke klasse aan dit item moet worden toegewezen.

Wanneer u gebruikmaakt van Discriminator-mogelijkheden, kunt u geen klasse overslaan in de hiërarchie. Dat wil zeggen: aangezien C3 overerft van C2 en C2 overerft van C1, kunt u niet alleen C1 en C3 definiëren, maar moet u alle drie de klassen definiëren.



Koppelingen

Er zijn drie typen koppelingen: een-op-veel, veel-op-een en veel-op-veel. Voor verbinding tussen de verschillende databaseobjecten moet een van deze koppelingen worden gedefinieerd met behulp van een externe-sleutelkolom (voor het geval van een-op-veel) of een toewijzingstabel (voor het geval van veel-op-veel).

Bruikbaarheid

Aangezien het JPA-schema zeer uitgebreid is, wordt een gestroomlijnd XML-bestand verschaft om definities te vereenvoudigen.

De use case voor het gebruik van dit XML-bestand is als volgt: federated gegevens worden in één federated klasse gemodelleerd. Deze klasse heeft veel-op-een relaties met een niet-federated CMDB-klasse. Bovendien is er slechts één mogelijk relatietype tussen de federated klasse en de niet-federated klasse.

Voorbereidingen treffen voor het aanmaken van adapters

Met deze taak worden de voorbereidingen beschreven die nodig zijn voor het aanmaken van een adapter.

Opmerking: U kunt voorbeelden bekijken van de algemene database-adapter in de UCMDB API. Het DDMi Adapter-voorbeeld bevat naast een gecompliceerd **orm.xml**-bestand ook de implementaties voor bepaalde invoegtoepassingsinterfaces.

Deze taak omvat de onderstaande stappen:

- "Vereisten" beneden
- "Een CI-type aanmaken" op pagina 91
- "Een relatie aanmaken" op pagina 91

1. ## Vereisten

   Als u wilt valideren of u de database-adapter met uw database kunt gebruiken, controleert u het volgende:

   - Of de afstemmingsklassen en de bijbehorende attributen (ook bekend als meervoudige knooppunten) in de database aanwezig zijn. Als de afstemming bijvoorbeeld wordt uitgevoerd op basis van knooppuntnaam, controleert u of er een tabel is die een kolom met knooppuntnamen bevat. Als de afstemming wordt uitgevoerd volgens knooppunt `cmdb_id`, controleert u of er een kolom is met CMDB-ID's die overeenkomen met de CMDB-ID's van de knooppunten in de CMDB. Zie "Afstemming " op pagina 86 voor meer informatie over afstemming.

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

- Als u twee CIT's met een relatie wilt correleren, moeten er correlatiegegevens tussen de CIT-tabellen bestaan. De correlatie kan plaatsvinden door een externe-sleutelkolom of een toewijzingstabel. Om bijvoorbeeld te correleren tussen knooppunt en ticket moet er een kolom aanwezig zijn in de tickettabel die de knooppunt-ID bevat, een kolom in de knooppunttabel met de ticket-ID die ermee is verbonden, of een toewijzingstabel waarvan `end1` de knooppunt-ID en `end2` de ticket-ID is. Zie "Hibernate als JPA-provider" op pagina 86 voor meer informatie over correlatiegegevens.

  In de volgende tabel wordt de externe-sleutelkolom `NODE_ID` weergegeven:

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	Seriële buscontroller	Intel ® 82801EB USB Universal Host Controller
3581	2	Systeem	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3581	3	Weergave	ATI ES1000
3581	4	Randapparatuur basissysteem	HP ProLiant iLO 2 Legacy Support Function

- Elk CIT kan aan een of meer tabellen worden toegewezen. Als u één CIT aan meerdere tabellen wilt toewijzen, controleert u of er een primaire tabel is waarvan de primaire sleutel aanwezig is in de andere tabellen, en die een unieke waardekolom is.

  Een ticket wordt bijvoorbeeld aan twee tabellen toegewezen: `ticket1` en `ticket2`. De eerste tabel heeft de kolommen `c1` en `c2` en de tweede tabel heeft de kolommen `c3` en `c4`. Om mogelijk te maken dat ze als één tabel worden beschouwd, moeten beide dezelfde primaire sleutel hebben. Het is ook mogelijk dat de primaire sleutel van de eerste tabel een kolom is in de tweede tabel.

  In het volgende voorbeeld delen de tabellen dezelfde primaire sleutel met de naam `CARD_ID`:

CARD_ID	CARD_TYPE	CARD_NAME
1	Seriële buscontroller	Intel ® 82801EB USB Universal Host Controller
2	Systeem	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3	Weergave	ATI ES1000
4	Randapparatuur basissysteem	HP ProLiant iLO 2 Legacy Support Function

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(Standaard-USB-hostcontroller)
3	Hewlett-Packard Company
4	(Standaardsysteemapparaten)
5	Hewlett-Packard Company

2. ## Een CI-type aanmaken

   In deze stap kunt u een CIT aanmaken dat de gegevens in het RDBMS (de externe gegevensbron) vertegenwoordigt.

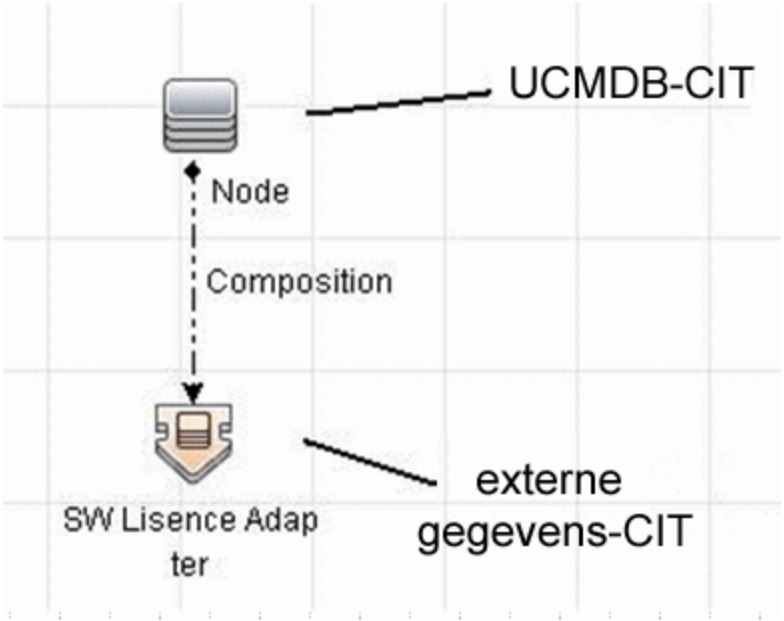
   a. Open in UCMDB het CI-typebeheer en maak een nieuw CI-type aan. Zie "Een CI-type aanmaken" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.

   b. Voeg de noodzakelijke attributen aan het CIT toe, zoals de laatste toegangstijd, leverancier, enzovoort. Dit zijn de attributen die met de adapter van de externe gegevensbron worden opgehaald en in CMDB-weergaven worden geplaatst.

3. ## Een relatie aanmaken

   In deze stap voegt u een relatie tussen het UCMDB-CIT en het nieuwe CIT toe waarmee de gegevens worden vertegenwoordigd waarop in de externe gegevensbron federation moet worden toegepast.

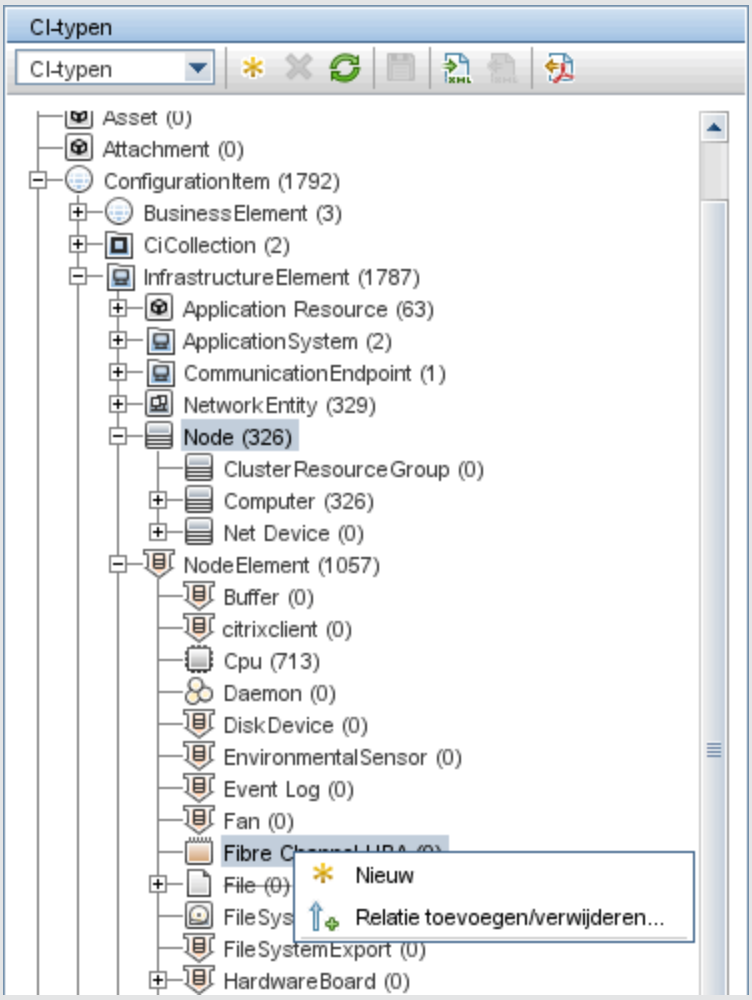
   Voeg de juiste, geldige relaties aan het nieuwe CIT toe. Zie "Het dialoogvenster Relatie toevoegen/verwijderen" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.



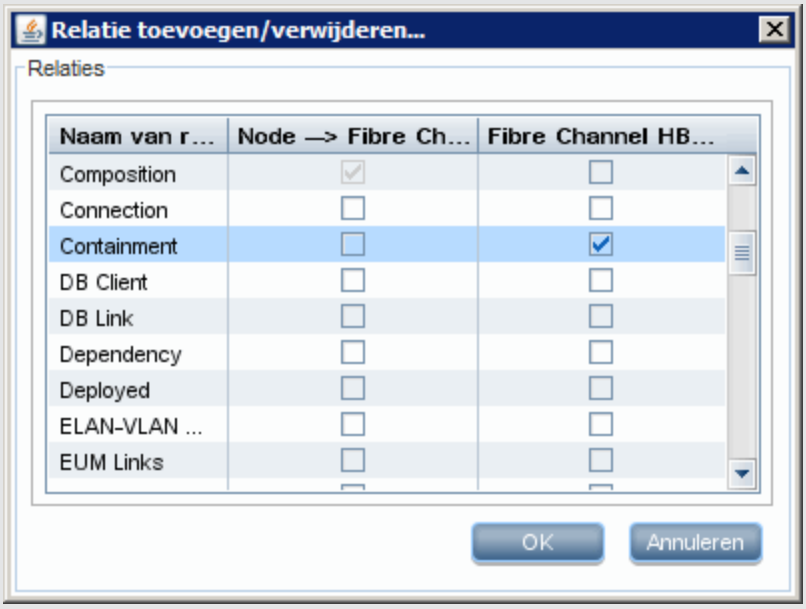
Opmerking: In deze fase kunt u de federated gegevens nog niet zien of de externe gegevens invullen, aangezien u de methode nog niet hebt gedefinieerd waarmee u de gegevens wilt invoeren.

Voorbeeld van het aanmaken van een Containment-relatie:

a. In het CIT-beheer selecteert u de twee CITs:



b. Maak een **Containment**-relatie tussen de twee CIT's aan:



Het adapterpakket voorbereiden

In deze stap zoekt en configureert u het pakket Algemene DB-adapter.

1. Zoek het pakket **db-adapter.zip** in de map **C:\hp\UCMDB\UCMDBServer\content\adapters**
   .
2. Pak het pakket in een lokale tijdelijke map uit.
3. Bewerk het XML-bestand van de adapter:
   - Open het bestand **discoveryPatterns\db_adapter.xml** in een teksteditor.
   - Zoek het attribuut **adapter id** en vervang de naam:

     ```
     <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">
     ```

     Als de adapter het vullen met gegevens ondersteunt, moet de volgende mogelijkheid aan het element **<adapter-capabilities>** worden toegevoegd:

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

Het weergegeven label of de ID wordt in de lijst met adapters weergegeven in het deelvenster Integratiepunt in HP Universal CMDB.

Wanneer een algemene DB-adapter wordt aangemaakt, is het niet nodig de tag **changes-source** in de tag **support-replicatioin-data** te bewerken. Als de invoegtoepassing **FcmdbPluginForSyncGetChangesTopology** wordt geïmplementeerd, wordt de gewijzigde topologie van de laatste uitvoering geretourneerd. Als de invoegtoepassing niet is geïmplementeerd, wordt de volledige topologie geretourneerd en wordt automatische verwijdering uitgevoerd op basis van de geretourneerde CI's.

Informatie over het vullen van de CMDB met gegevens kunt u vinden in "De pagina Integration Studio " in de *HP Universal CMDB – Handleiding Data Flow Management*.

- Als de toewijzingsengine van versie 8.x op de adapter wordt gebruikt (wat inhoudt dat niet de nieuwe toewijzingsengine voor afstemming wordt gebruikt), vervangt u het volgende element:

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

  met:

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

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

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

- Zoek de definitie **category**:

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

  Wijzig de categorienaam **Generic** in de categorie van uw keuze.

  > **Opmerking:** adapters waarvan de categorieën worden opgegeven als **Generic**, worden niet in Integration Studio weergegeven wanneer u een nieuw integratiepunt aanmaakt.

- De verbinding met de database kan worden beschreven met een gebruikersnaam (schema), wachtwoord, databasetype, database-hostnaam en databasenaam of -SID.

  Voor dit type verbinding hebben parameters de volgende elementen in het gedeelte **parameter** van het XML-bestand van de adapter:

  ```
  <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>
```

Opmerking: Dit is de standaardconfiguratie. Daarom bevat het bestand **db_ adapter.xml** deze definitie al.

Er zijn situaties waarin de verbinding met de database niet op deze manier kan worden geconfigureerd. Dat geldt bijvoorbeeld voor het verbinding maken met Oracle RAC verbinding maken via een ander databasestuurprogramma dan het stuurprogramma dat wordt geleverd bij de CMDB.

Voor dergelijke situaties kunt u de verbinding beschrijven middels een gebruikersnaam (schema), wachtwoord en een verbindings-URL-string.

Bewerk het gedeelte met XML-parameters in de adapter als volgt om dit te definiëren:

```
<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>
```

Een voorbeeld van een URL die verbinding maakt met een Oracle RAC met behulp van het meegeleverde Data Direct-stuurprogramma is:
jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true.

4. Open in de tijdelijke map de map **adapterCode** en wijzig de naam van **GenericDBAdapter** in de waarde van **adapter id** die in de vorige stap werd gebruikt.

   Deze map bevat de jar-bestanden waarmee de federation-logica wordt uitgevoerd, zoals de adapternaam, de query's en klassen in de CMDB en de velden in het RDBMS dat de adapter ondersteunt.

5. Configureer de adapter indien nodig. Zie "De adapter configureren – minimale methode" beneden voor meer informatie over dit onderwerp.

6. Maak een `*.zip`-bestand aan met dezelfde naam als u aan het attribuut **adapter id** hebt gegeven, zoals beschreven in stap"Bewerk het XML-bestand van de adapter:" op pagina 93.

   Opmerking: het bestand **descriptor.xml** is een standaardbestand dat in elk pakket aanwezig is.

7. Sla het nieuwe pakket op dat u in de vorige stap hebt aangemaakt. De standaardmap voor adapters is: **C:\hp\UCMDB\UCMDBServer\content\adapters**.

De adapter configureren – minimale methode

Met de volgende procedure wordt een methode beschreven waarmee het klassemodel in de CMDB aan een RDBMS wordt toegewezen.

Deze configuratiebestanden bevinden zich in het pakket **db-adapter.zip** in de map **C:\hp\UCMDB\UCMDBServer\content\adapters**, dat u hebt uitgepakt in stap "Pak het pakket in een lokale tijdelijke map uit." op pagina 93 van "Het adapterpakket voorbereiden" op pagina 93.

Opmerking: Het bestand **orm.xml** dat automatisch wordt gegenereerd als resultaat van de uitvoering van deze methode, is een goed voorbeeld dat u kunt gebruiken wanneer u met de geavanceerde methode werkt.

U gebruikt deze minimale methode wanneer u het volgende moet doen:

- Zorgen voor federation/vullen van één knooppunt zoals een knooppuntattribuut.
- De mogelijkheden van de algemene DB-adapter tonen.

Deze methode:

- ondersteunt alleen federation/vullen van één knooppunt
- ondersteunt slechts veel-op-een virtuele relaties

Deze taak omvat de onderstaande stappen:

- "Bestand adapter.conf configureren" beneden
- "Bestand simplifiedConfiguration.xml configureren" op volgende pagina

Bestand adapter.conf configureren

In deze stap wijzigt u de instellingen in het bestand `adapter.conf` zodat de adapter gebruikmaakt van de vereenvoudigde configuratiemethode.

1. Open het bestand **adapter.conf** in een teksteditor.
2. Zoek de volgende regel: **use.simplified.xml.config=<true/false>**.
3. Wijzig de regel in **use.simplified.xml.config=true**.

Bestand simplifiedConfiguration.xml configureren

In deze stap configureert u het bestand **simplifiedConfiguration.xml** door het CIT in de CMDB toe te wijzen aan de velden in de RDBMS-tabel.

1. Open het bestand **simplifiedConfiguration.xml** in een teksteditor.

   Dit bestand bevat een sjabloon die u gebruikt voor elke entiteit die moet worden toegewezen.

   Opmerking: bewerk het bestand **simplifiedConfiguration.xml** niet in een versie van Kladblok van Microsoft Corporation. Gebruik Notepad++, UltraEdit of een andere teksteditor van derden.

2. Breng wijzigingen in de volgende attributen aan:

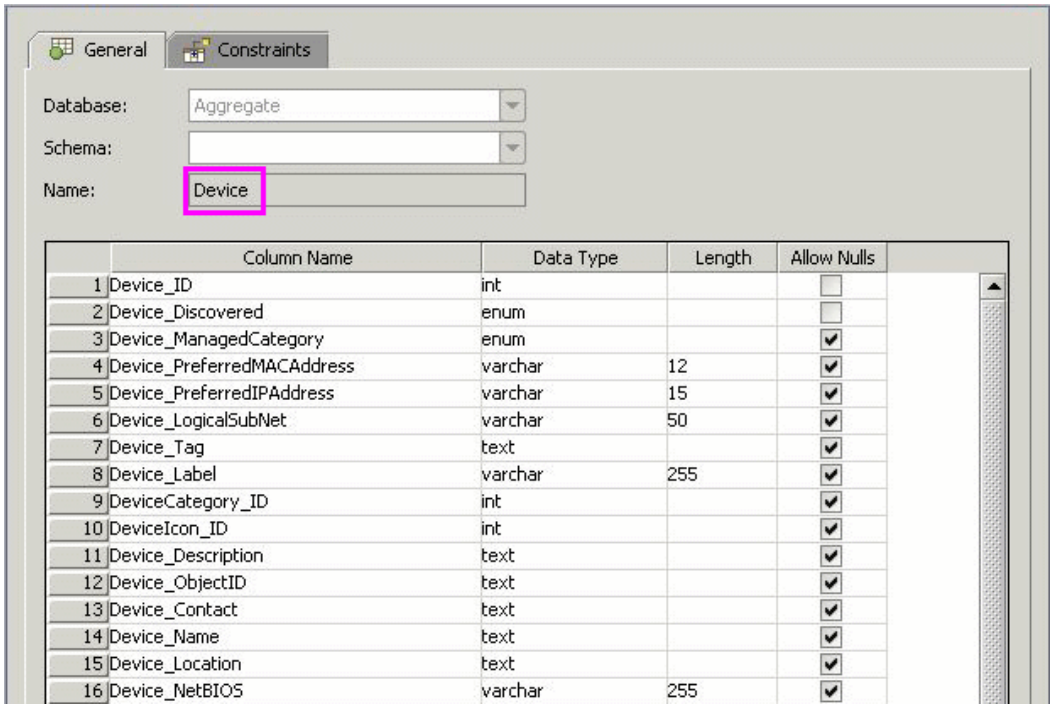
   - De CIT-naam in UCMDB (cmdb-class-name) en de bijbehorende tabelnaam in het RDBMS (default-table-name):

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

     Het attribuut `cmdb-class-name` wordt van het knooppunt-CIT overgenomen:

     Het attribuut `default-table-name` wordt van de tabel Device overgenomen:

- De unieke ID in het RDBMS:

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

  Opmerking: De primaire sleutel is identiek met entiteit-ID in het bestand orm.xml.

- De afstemmingsregel (reconciliation-by-two-nodes):

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

- Het afstemmingsattribuut in UCMDB (`cmdb-attribute-name`) en in het RDBMS (`column-name`):

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

- De naam van het CIT (`cmdb-class-name`) en de naam van de bijbehorende tabel in het RDBMS (`default-table-name`). Tevens de CMDB-relatie (`connected-cmdb-class-name`) en de CIT-relatie (`link-class-name`):

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

- De primaire sleutel en de externe sleutel:

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

- De unieke ID in het RDBMS:

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

- De toewijzing tussen het attribuut CMDB (`cmdb-attribute-name`) en de kolomnaam in het RDBMS (`column-name`):

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

3. Sla het bestand op.

Voorbeeld: Een knooppunt en IP-adres vullen met de vereenvoudigde methode

In dit voorbeeld wordt getoond hoe een **Node** gekoppeld door een containment-koppeling aan **IP Address** in de UCMDB wordt gevuld. RDBMS heeft een tabel met de naam **simpleNode** die gegevens over de computernaam, het computerknooppunt en het IP-adres van de computer bevat.

De inhoud van de tabel **simpleNode** wordt hierna weergegeven:

	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

Vullen wordt als volgt in drie fasen uitgevoerd:

- "simplifiedConfiguration.xml aanmaken" beneden
- " De TQL aanmaken" op volgende pagina
- "Een integratiepunt maken" op pagina 101

simplifiedConfiguration.xml aanmaken

Maak **simplifiedConfiguration.xml** als volgt:

1. Maak als volgt een **cmdb-class**-entiteit:

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

   Het CI-type is **node** en de RDBMS-tabelnaam is **simpleNode**.

2. Stel de primary-key van de tabel als volgt in:

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

   De primaire sleutel is identiek aan entiteit-ID in het bestand **orm.xml**.

3. Stel de regel **reconciliation-by-two-nodes** als volgt in:

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

   Met deze tag wordt de relatie gedefinieerd tussen de CI-typen **Node** en **IpAddress**. Het relatietype is Containment-koppeling. De afstemming vindt plaats door de twee verbonden CI-typen. De attribuuttoewijzing van het verbonden knooppunt (in dit geval IpAddress) wordt gedefinieerd in het attribuut **connected-node**.

4. Voeg de voorwaarde **or** tussen de afstemmingsattributen als volgt toe:

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

   Met deze tag wordt een OR-relatie gedefinieerd tussen de afstemmingsattributen. Dit houdt in dat met het eerste afstemmingsattribuut dat **true** is, de hele afstemmingsregel wordt ingesteld op **true**.

5. Voeg de volgende attributen toe:

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

   Met deze tag wordt een toewijzing ingesteld tussen de **node.name** in de UCMDB aan de kolom **host_name** in de tabel **simpleNode**.

   Doe hetzelfde met het attribuut **data_note**:

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

Voeg het verbonden knooppuntattribuut toe:

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

Met deze tag wordt een toewijzing ingesteld tussen de **ip_address.name** aan de kolom **ip_ address** in de tabel **simpleNode**.

6. Sluit de geopende tag in de volgende volgorde:

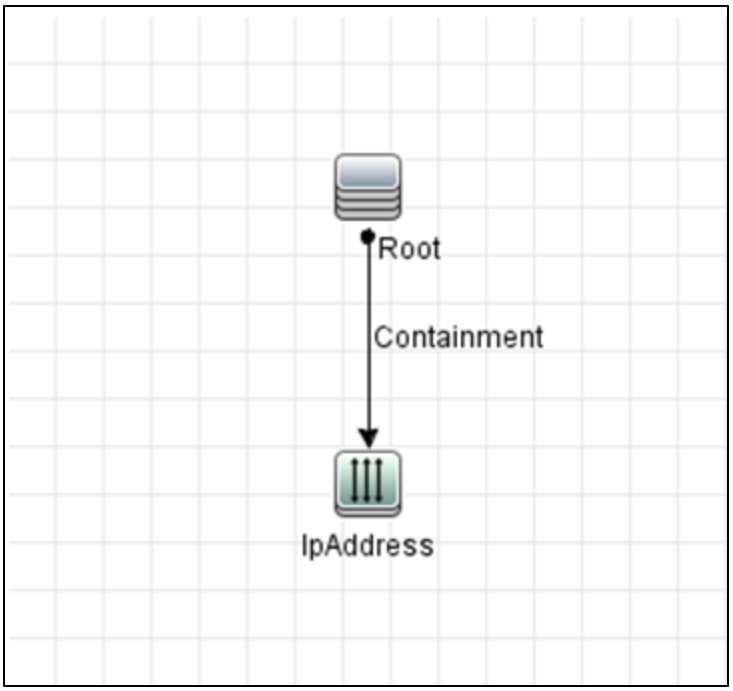
```
</or>

</reconciliation-by-two-nodes>

</cmdb-class>
```

De TQL aanmaken

De TQL is een **node** verbonden door een containment-koppeling met **ip_address**. Het knooppunt moet worden gemarkeerd als **root**, zoals hieronder wordt getoond.



U kunt de TQL als volgt aanmaken:

1. Ga naar Modeling > Modeling Studio.
2. Klik op de knop Nieuw en maak een nieuwe query aan.
3. Ga naar het tabblad CI-typen en sleep het CI-type Node en het CI-type IpAddress naar het TQL-scherm.
4. Verbind **Node** en **IpAddress** met Containment-koppeling.
5. Klik met de rechtermuisknop op het element **Node** en kies Eigenschappen query-knooppunt.

6. Wijzig **Element name** in **Root**.
7. Ga naar het tabblad **Elementindeling**. Selecteer bij **Attribuutvoorwaarde** de optie **Specifieke attributen**. Kies **Naam** en **Opmerking** in het venster Beschikbare attributen en verplaats deze naar het venster Specifieke attributen.
8. Klik met de rechtermuisknop op het element **IpAddress** en kies Eigenschappen query-knooppunt.
9. Ga naar het tabblad **Elementindeling**. Selecteer bij **Attribuutvoorwaarde** de optie **Specifieke attributen**. Kies **Naam** in het venster Beschikbare attributen en verplaats deze naar het venster Specifieke attributen.
10. Sla de TQL op.

Een integratiepunt maken

Maak het integratiepunt als volgt aan:

1. Ga naar Data Flow-beheer > Integration Studio en klik op de knop **Nieuw integratiepunt**.
2. Voeg de details van het integratiepunt in en klik op OK.
3. Selecteer op het tabblad Vullen de knop **Nieuwe integratietaak** en voeg de eerder gemaakt TQL toe.
4. Sla het integratiepunt op en klik op de knop Volledige synchronisatie uitvoeren.

De adapter configureren – Geavanceerde methode

Deze configuratiebestanden bevinden zich in het pakket **db-adapter.zip** in de map **C:\hp\UCMDB\UCMDBServer\content\adapters**, dat u hebt uitgepakt in stap "Pak het pakket in een lokale tijdelijke map uit." op pagina 93 van "Het adapterpakket voorbereiden" op pagina 93.

Deze taak omvat de onderstaande stappen:

- "Bestand orm.xml configureren" beneden
- "Bestand reconciliation_types.txt configureren" op pagina 105
- "Bestand reconciliation_rules.txt configureren " op pagina 105

Bestand orm.xml configureren

In deze stap wijst u de CIT's en relaties in de CMDB aan de tabellen in het RDBMS toe.

1. Open het bestand **orm.xml** in een teksteditor.

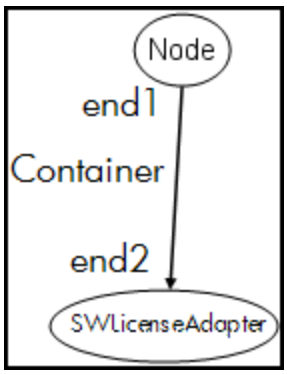
   Dit bestand bevat standaard een sjabloon waarmee u zoveel CIT's en relaties kunt toewijzen als u nodig hebt.

   > **Opmerking:** Bewerk het bestand **orm.xml** niet in een versie van Kladblok van Microsoft Corporation. Gebruik Notepad++, UltraEdit of een andere teksteditor van derden.

2. Breng wijzigingen in het bestand aan volgens de gegevensentiteiten die moeten worden toegewezen. Zie de volgende voorbeelden voor meer informatie.

   De volgende relatietypen kunnen in het bestand **orm.xml** worden toegewezen:

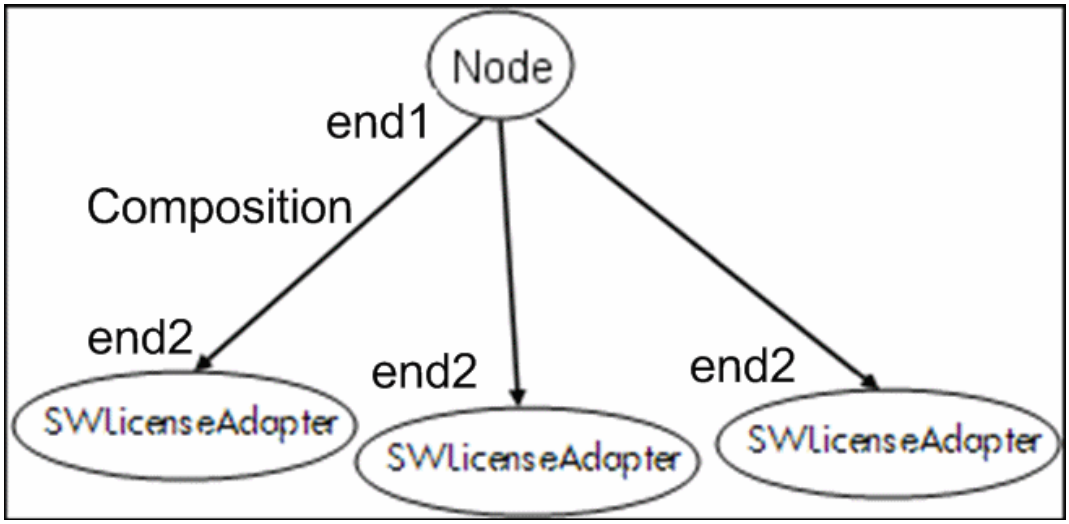
- Een-op-een:



De code voor dit relatietype is:

```
<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>
```

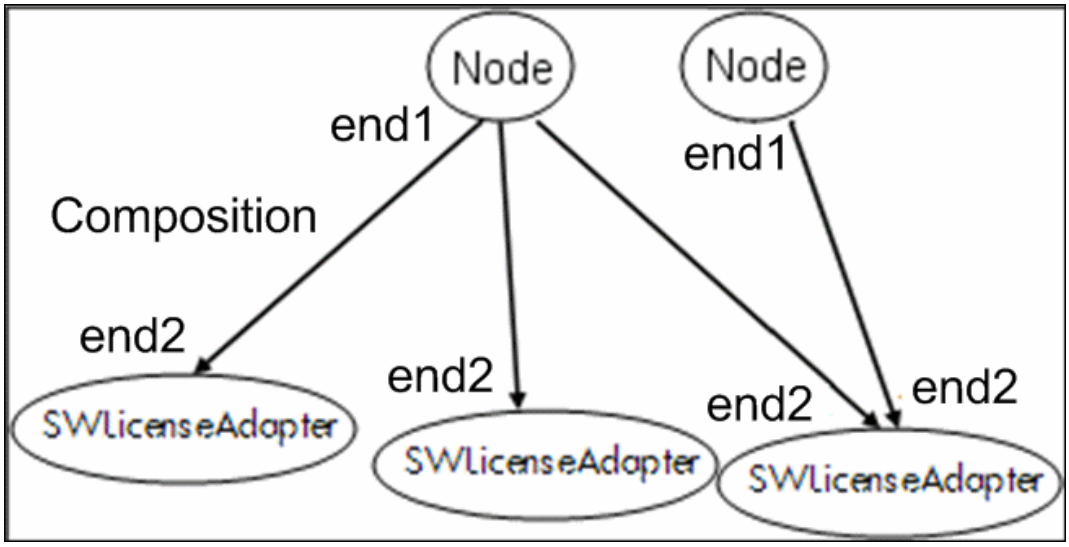
- Veel-op-een:



De code voor dit relatietype is:

```
<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>
```

- Veel-op-veel:



De code voor dit relatietype is:

```
<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>
```

Zie "Naamgevingsconventies" op pagina 125 voor meer informatie over naamgevingsconventies.

Voorbeeld van entiteitstoewijzing tussen het datamodel en het RDBMS:

Opmerking: attributen die niet hoeven te worden geconfigureerd, worden uit de volgende voorbeelden weggelaten.

- De klasse van het CMDB-CIT:

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

- De naam van de tabel in het RDBMS:

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

- De kolomnaam van de unieke ID in de tabel RDBMS:

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

- De naam van het attribuut in het CMDB-CIT:

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

- De naam van het tabelveld in de externe gegevensbron:

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

- De naam van het nieuwe CIT dat u hebt aangemaakt in "Een CI-type aanmaken" op pagina 91:

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

- De naam van de bijbehorende tabel in het RDBMS:

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

- De unieke ID in het RDBMS:

- De attribuutnaam in het CMDB-CIT en de naam van het bijbehorende attribuut in het RDBMS:

Voorbeeld van relatietoewijzing tussen het datamodel en het RDBMS:

- De klasse van de CMDB-relatie:

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

- De naam van de RDBMS-tabel waarin de relatie wordt uitgevoerd:

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

- De unieke ID in het 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>
```

- Het relatietype en het CMDB-CIT:

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

- De velden voor primaire sleutel en externe sleutel in het RDBMS:

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

Bestand reconciliation_types.txt configureren

Open het bestand **reconciliation_types.txt** in een teksteditor.

Zie "Bestand reconciliation_types.txt" op pagina 134 voor meer informatie over dit onderwerp.

Bestand reconciliation_rules.txt configureren

In deze stap definieert u de regels op basis waarvan de adapter de CMDB en het RDBMS afstemt (alleen als de toewijzingsengine wordt gebruikt, voor achterwaartse compatibiliteit met versie 8.x):

1. Open **META-INF\reconciliation_rules.txt** in een teksteditor.
2. Breng wijzigingen in het bestand aan volgens het CIT dat u toewijst. Als u bijvoorbeeld een knooppunt-CIT wilt toewijzen, gebruikt u de volgende expressie:

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

   Opmerking:

   - als de gegevens in de database hoofdlettergevoelig zijn, moet u het besturingsteken (**^**) niet verwijderen.
   - controleer of elke vierkante haak openen een bijbehorende vierkante haak sluiten heeft.

Zie "Bestand reconciliation_rules.txt (voor achterwaartse compatibiliteit)" op pagina 134 voor meer informatie over dit onderwerp.

Invoegtoepassingen implementeren

Met deze taak wordt beschreven hoe u een algemene DB-adapter kunt implementeren met invoegtoepassingen.

Opmerking: Alvorens een invoegtoepassing voor een adapter te schrijven, moet u alle noodzakelijke stappen in "Het adapterpakket voorbereiden" op pagina 93 hebben uitgevoerd.

1. Kopieer de volgende jar-bestanden vanuit de installatiemap van de UCMDB-server naar uw ontwikkelingsklassepad:
   - Kopieer het bestand **db-interfaces.jar** en het bestand **db-interfaces-javadoc.jar** vanuit de map **tools\adapter-dev-kit\db-adapter-framework**.
   - Kopieer het bestand **federation-api.jar** en het bestand **federation-api-javadoc.jar** vanuit de map **\tools\adapter-dev-kit\SampleAdapters\production-lib**.

     Opmerking: meer informatie over het ontwikkelen van een invoegtoepassing vindt u in de bestanden **db-interfaces-javadoc.jar** en **federation-api-javadoc.jar** en in de onlinedocumentatie op:

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

2. Schrijf een Java-klasse waarmee de Java-interface van de invoegtoepassing wordt geïmplementeerd. De interfaces worden in het bestand **db-interfaces.jar** gedefinieerd. In de onderstaande tabel wordt de interface opgegeven die voor elke invoegtoepassing moet worden geïmplementeerd:

Type invoegtoepassing	Interfacenaam	Methode
Volledige topologie synchroniseren	FcmdbPluginForSyncGetFullTopology	getFullTopology
Wijzigingen synchroniseren	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Indeling synchroniseren	FcmdbPluginForSyncGetLayout	getLayout
Ondersteunde query's ophalen	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Definitie en resultaten van TQL-query's wijzigen	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Indelingsverzoek voor CI's wijzigen	FcmdbPluginGetCIsLayout	getCisLayout
Indelingsverzoek voor koppelingen wijzigen	FcmdbPluginGetRelationsLayout	getRelationsLayout
Push Back-ID's	FcmdbPluginPushBackIds	getPushBackIdsSQL

De klasse van de invoegtoepassing moet een openbare standaardconstructor hebben. Alle interfaces stellen een methode beschikbaar met de naam initPlugin. Deze methode wordt gegarandeerd aangeroepen vóór enige andere methode en wordt gebruikt om de adapter te initialiseren met het omgevingsobject van de relevante adapter.

Als **FcmdbPluginForSyncGetChangesTopology** is geïmplementeerd, zijn er twee verschillende manieren om de wijzigingen te melden:

- **Vermeld altijd de volledige topologie van knooppunten.** Op basis van deze topologie zoekt de functie auto-delete uit welke CI's moeten worden verwijderd. In dit geval moet de functie auto-delete worden ingeschakeld met behulp van het volgende:

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

- **Vermeld elk CI-exemplaar dat is verwijderd/bijgewerkt.** n dit geval moet de functie auto-

delete worden uitgeschakeld met behulp van het volgende:

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

3. Zorg ervoor dat u de Federation SDK JAR en de JAR's van de algemene DB-adapter in uw klassepad hebt voordat u uw Java-code compileert. De Federation SDK is het bestand **federation_api.jar**, dat zich bevindt in de map **C:\hp\UCMDB\UCMDBServer\lib**.

4. Pak uw klasse in een jar-bestand in en plaats het onder de map adapterCode\<naam van uw adapter> in het adapterpakket alvorens het uit te rollen.

   De invoegtoepassingen worden geconfigureerd met het bestand **plugins.txt**, dat zich bevindt in de map **\META-INF** van de adapter.

   Hierna vindt u een voorbeeld van het bestand van de DDMi-adapter:

```
# 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]
```

   Legenda:

   # - Een commentaarregel.

   [<Adapter Type>] – begin van de definitiesectie voor een specifiek adaptertype.

   Er kan zich een lege regel bevinden onder elk [<Adapter Type>], wat inhoudt dat er geen invoegtoepassingsklasse is gekoppeld, of de volledig gekwalificeerde naam van uw invoegtoepassingsklasse kan worden weergegeven.

5. Pak uw adapter met het nieuwe jar-bestand en het bijgewerkte bestand **plugins.xml** in. De rest van de bestanden in het pakket moet hetzelfde zijn als in elke adapter die is gebaseerd op de algemene DB-adapter.

De adapter implementeren

1. Ga in UCMDB naar Pakketbeheer. Zie "Pagina Pakketbeheer" in de *HP Universal CMDB – Handleiding Beheer* voor meer informatie over dit onderwerp.

2. Klik op het pictogram **Pakketten uitrollen naar server (vanaf plaatselijk station)** en blader naar uw adapterpakket. Selecteer het pakket en klik op **Openen** en vervolgens op **Uitrollen** om het pakket in Pakketbeheer weer te geven.

3. Selecteer uw pakket in de lijst en klik op het pictogram **Pakketbronnen weergeven** om te controleren of de pakketinhoud wordt herkend door Pakketbeheer.

De adapter bewerken

Nadat u de adapter hebt aangemaakt en uitgerold, kunt u deze vervolgens bewerken in UCMDB. Zie "Adapter Management" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

Een integratiepunt maken

In deze stap controleert u of de federation werkt. Dit betekent dat zowel de verbinding als het XML-bestand geldig zijn. Met deze controle wordt echter niet geverifieerd of de XML aan de juiste velden in het RDBMS toewijst.

1. Open vanuit UCMDB Integration Studio (**Data Flow-beheer > Integration Studio**).
2. Maak een integratiepunt aan. Zie "Dialoogvenster Nieuw integratiepunt/Integratiepunt bewerken" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

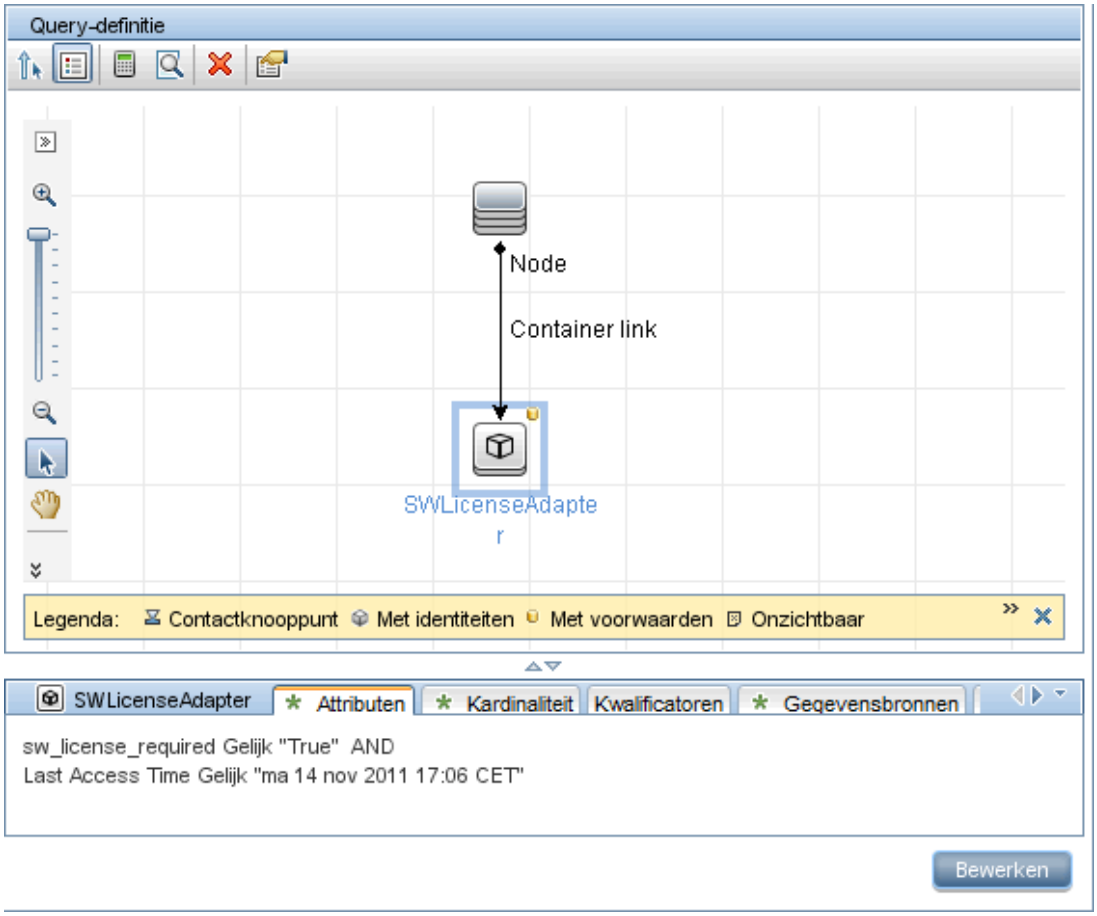
   Op het tabblad Federation worden alle CIT's weergegeven waarop federation kan worden uitgevoerd met dit integratiepunt. Zie "Het tabblad Federation" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

Een weergave aanmaken

In deze stap maakt u een weergave aan waarmee u exemplaren van het CIT kunt bekijken.

1. Open vanuit UCMDB Modeling Studio (**Modeling > Modeling Studio**).
2. Maak een weergave aan. Zie "Een patroonweergave maken" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.
3. U kunt voorwaarden aan de TQL toevoegen, bijvoorbeeld: de laatste toegangstijd is groter dan

zes maanden:



Resultaten berekenen

In deze stap controleert u de resultaten.

1. Open vanuit UCMDB Modeling Studio (**Modellering > Modeling Studio**).
2. Open een weergave.
3. Bereken resultaten door te klikken op de knop **Aantal query-resultaten berekenen** .
4. Klik op de knop **Voorbeeld** om de CI's in de weergave te bekijken.

Resultaten bekijken

In deze stap bekijkt u de resultaten en foutopsporingsproblemen in de procedure. Als bijvoorbeeld niets in de weergave wordt getoond, controleert u de definities in het bestand **orm.xml**, verwijdert u de relatie-attributen en laad u de adapter opnieuw.

1. Open vanuit UCMDB IT Universe Manager (**Modeling > IT Universe Manager**).
2. Selecteer een CI. Op het tabblad Eigenschappen worden de resultaten van de federation weergegeven.

Rapporten weergeven

In deze stap bekijkt u topologierapporten. Zie "Overzicht topologierapporten" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.

Logboekbestanden inschakelen

U kunt de logboekbestanden raadplegen om te begrijpen wat berekeningsstromen en de adapterlevenscyclus inhouden en om foutopsporingsgegevens te bekijken. Zie "Adapterlogboekbestanden" op pagina 152 voor meer informatie over dit onderwerp.

Toewijzing tot stand brengen tussen CIT-attributen en databasetabellen met Eclipse

Let op: Deze procedure is bedoeld voor gebruikers met een gevorderde kennis van inhoudsontwikkeling. Neem contact op met HP Software Support voor eventuele vragen.

Met deze taak wordt beschreven hoe u de JPA-invoegtoepassing die bij de J2EE-versie van Eclipse wordt geleverd, installeert en gebruikt om het volgende te doen:

- Grafische toewijzing tussen CMDB-klasseattributen en databasetabelkolommen inschakelen.
- Handmatige bewerking van het toewijzingsbestand (`orm.xml`) met correctheidscontrole inschakelen. De correctheidscontrole omvat naast een syntaxiscontrole ook verificatie of de klasseattributen en toegewezen databasetabelkolommen juist zijn vermeld.
- Uitrol van het toewijzingsbestand inschakelen voor de CMDB-server en het weergeven van de fouten, als een nadere correctheidscontrole.
- Een voorbeeldquery op de CMDB-server definiëren en deze rechtstreeks vanuit Eclipse uitvoeren om het toewijzingsbestand te testen.

Versie 1.1 van de invoegtoepassing is compatibel met UCMDB versie 9.01 of hoger en Eclipse IDE for Java EE Developers, versie 1.2.2.20100217-2310 of hoger.

Deze taak omvat de onderstaande stappen:

- "Vereisten" op volgende pagina
- "Installatie" op volgende pagina
- "Werkomgeving voorbereiden" op volgende pagina
- "Een adapter aanmaken" op pagina 112
- "De invoegtoepassing CMDB configureren" op pagina 112
- "Het UCMDB-klassemodel importeren" op pagina 112
- "Het ORM-bestand samenstellen – UCMDB-klassen aan databasetabellen toewijzen" op pagina 113

1. ## Vereisten

   Installeer de laatste update van **Java Runtime Environment (JRE) 6** op de computer waar u Eclipse vanaf de volgende website uitvoert: **http://java.sun.com/javase/downloads/index.jsp**.

2. ## Installatie

   a. Download en pak **Eclipse IDE for Java EE Developers** uit vanuit **http://www.eclipse.org/downloads**in een lokale map, bijvoorbeeld **C:\Program Files\eclipse**.

   b. Kopieer **com.hp.plugin.import_cmdb_model_1.0.jar** vanuit **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\bin** naar **C:\Program Files\Eclipse\plugins**.

   c. Start **C:\Program Files\Eclipse\eclipse.exe**. Als een bericht verschijnt dat de virtuele Java-computer niet is gevonden, start u **eclipse.exe** met de volgende opdrachtregel:

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

3. ## Werkomgeving voorbereiden

   In deze stap stelt u de eigenschappen voor werkruimte, database, verbindingen en stuurprogramma in.

   a. Pak het bestand **workspaces_gdb.zip** uit vanuit **C:\hp\UCMDB\ UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** naar **C:\Documents and Settings\All Users.**

   > **Opmerking:** u moet het exacte mappad gebruiken. Als u het bestand in het verkeerde pad uitpakt of het bestand ingepakt laat, werkt de procedure niet.

b. Kies in Eclipse **File > Switch Workspace > Other**:

Als u werkt met:

- SQL Server, selecteert u de volgende map: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.
- MySQL, selecteert u de volgende map: **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
- Oracle, selecteert u de volgende map: **C:\Documents and Settings\All Users\workspace_gdb_oracle**.

c. Klik op **OK**.

d. Geef in Eclipse de Project Explorer-weergave weer en selecteer **<Active project> > JPA Content > persistence.xml > <active projectnaam> > orm.xml**.

e. Klik in de weergave Data Source Explorer (het deelvenster linksonder) met de rechtermuisknop op de databaseverbinding en selecteer het menu **Properties**.

f. Selecteer in het dialoogvenster **Properties for <Connection name>Common** en schakel het selectievakje **Connect every time the workbench is started** in. Selecteer **Driver Properties** en vul de verbindingseigenschappen in. Klik op **Test Connection** en controleer of deze verbinding werkt. Klik op **OK**.

g. Klik in de weergave Data Source Explorer met de rechtermuisknop op de databaseverbinding en klik op **Connect**. Er wordt een boomstructuur weergegeven met de databaseschema's en -tabellen onder het databaseverbindingspictogram.

4. Een adapter aanmaken

Maak een adapter aan met behulp van de richtlijnen in "Stap 1: een adapter aanmaken" op pagina 29.

5. De invoegtoepassing CMDB configureren

a. Klik in Eclipse op **UCMDB > Settings** om het dialoogvenster **CMDB Settings** te openen.

b. Selecteer indien nodig het nieuwe JPA-project als het actieve project.

c. Voer de CMDB-hostnaam in, bijvoorbeeld **localhost** of **labm1.itdep1**. U hoeft het poortnummer of `http://`-voorvoegsel niet in het adres op te nemen.

d. Vul de gebruikersnaam en het wachtwoord in waarmee u toegang kunt krijgen tot de CMDB-API, doorgaans **admin**/**admin**.

e. Controleer of de map **C:\hp** op de CMDB-server als een netwerkstation wordt toegewezen.

f. Selecteer de basismap van de relevante adapter onder **C:\hp**. De basismap is de map die het bestand **dbAdapter.jar** en de submap **META-INF** bevat. Het bijbehorende pad moet **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<adapternaam>** zijn. Controleer dat er geen backslash (\) aan het einde staat.

6. Het UCMDB-klassemodel importeren

In deze stap selecteert u de CIT's die als JPA-entiteiten moeten worden toegewezen.

a. Klik op **UCMDB > Import CMDB Class Model** om het dialoogvenster **CI Type Selection** te openen.

b. Selecteer de CI-typen die u als JPA-entiteiten wilt toewijzen. Klik op **OK**. De CI-typen worden als Java-klassen geïmporteerd. Controleer of deze onder de map **src** van het actieve project worden weergegeven.

7. Het ORM-bestand samenstellen – UCMDB-klassen aan databasetabellen toewijzen

In deze stap wijst u de Java-klassen (die u in de vorige stap hebt geïmporteerd) aan de databasetabellen toe.

a. Controleer of de databaseverbinding is gelukt. Klik met de rechtermuisknop op het actieve project (dat standaard myProject heet) in Project Explorer. Selecteer de JPA-weergave, schakel het selectievakje **Override default schema from connection** in en selecteer het relevante databaseschema. Klik op **OK**.

b. Wijs een CIT toe: klik in de weergave JPA Structure met de rechtermuisknop op de vertakking **Entity Mappings** en selecteer **Add Class**. Het dialoogvenster **Add Persistent Class** wordt geopend. Wijzig het veld **Map as** niet (**Entity**).

c. Klik op **Browse** en selecteer de klasse UCMDB die moet worden toegewezen (alle UCMDB-klassen behoren tot het pakket **generic_db_adapter**).

d. Klik in beide dialoogvensters op **OK**. De geselecteerde klasse wordt weergegeven onder de vertakking **Entity Mappings** in de weergave JPA Structure.

> **Opmerking:** als de entiteit zonder een attributenstructuur wordt weergegeven, klikt u met de rechtermuisknop op het actieve project in de weergave Project Explorer. Kies **Close** en vervolgens **Open**.

e. Selecteer in de weergave JPA Details de primaire databasetabel waaraan de UCMDB-klasse moet worden toegewezen. Laat alle andere velden ongewijzigd.

8. ID's toewijzen

In overeenstemming met JPA-standaarden moet elke persistente klasse minimaal één ID-attribuut hebben. Voor UCMDB-klassen kunt u maximaal drie attributen als ID's toewijzen. Potentiële ID-attributen worden **id1**, **id2** en **id3** genoemd.

Zo wijst u een ID-attribuut toe:

a. Vouw de bijbehorende klasse onder de vertakking **Entity Mappings** in de weergave JPA Structure uit, klik met de rechtermuisknop op het relevante attribuut (bijvoorbeeld **id1**) en selecteer **Add Attribute to XML and Map…**.

b. Het dialoogvenster **Add Persistent Attribute** wordt geopend. Selecteer **Id** in het veld **Map as** en klik op **OK**.

c. Selecteer in de weergave JPA Details de databasetabelkolom waaraan het ID-veld moet worden toegewezen.

9. Attributen toewijzen

In deze stap wijst u attributen aan de databasekolommen toe.

a. Vouw de bijbehorende klasse onder de vertakking **Entity Mappings** in de weergave JPA Structure uit, klik met de rechtermuisknop op het relevante attribuut (bijvoorbeeld **host_hostname**) en selecteer **Add Attribute to XML and Map…**.

b. Het dialoogvenster **Add Persistent Attribute** wordt geopend. Selecteer **Basic** in het veld **Map as** en klik op **OK**.

c. Selecteer in de weergave JPA Details de databasetabelkolom waaraan het attribuutveld moet worden toegewezen.

10. ## Een geldige koppeling toewijzen

Voer de stappen uit die worden beschreven in stap"Het ORM-bestand samenstellen – UCMDB-klassen aan databasetabellen toewijzen" op vorige pagina voor het toewijzen van een UCMDB-klasse waarmee een geldige koppeling wordt aangegeven. De naam van zo'n klasse heeft de volgende structuur: **<end1 entity name>_<link name>_<end 2 entity name>**. Een **Bevat**-koppeling tussen een host en een locatie wordt aangegeven door een Java-klasse waarvan de naam **generic_db_adapter.host_contains_location** is. Zie "Bestand reconciliation_rules.txt (voor achterwaartse compatibiliteit)" op pagina 134 voor meer informatie over dit onderwerp.

a. Wijs de ID-attributen van de koppeling toe, zoals wordt beschreven in "ID's toewijzen" op vorige pagina. Vouw voor elk ID-attribuut de groep met **Details**-selectievakjes in de weergave JPA Details uit en schakel de selectievakjes **Insertable** en **Updateable** uit.

b. Wijs de attributen **end1** en **end2** van de koppelingsklasse als volgt toe: Voor elk van de attributen **end1** en **end2** van de koppelingsklasse:

- Vouw de bijbehorende klasse onder de vertakking **Entity Mappings** in de weergave JPA Structure uit, klik met de rechtermuisknop op het relevante attribuut (bijvoorbeeld **end1**) en selecteer **Add Attribute to XML and Map…**.
- Selecteer in het dialoogvenster **Add Persistent AttributeMany to One** of **One to One** in het veld **Map as**.
- Selecteer **Many to One** als het opgegeven **end1**- of **end2**-CI meerdere koppelingen van dit type kan hebben. Selecteer anders **One to One**. Bijvoorbeeld: voor een **host_contains_ip**-koppeling moet het **host**-eind worden toegewezen als **Many to One**, aangezien één host meerdere IP's kan hebben en het **ip**-eind moet worden toegewezen als **One to One**, aangezien één IP slechts één host kan hebben.
- Selecteer in de weergave JPA Details **Target entity**, bijvoorbeeld **generic_db_adapter.host**.
- Schakel in de sectie **Join Columns** van de weergave JPA Details de optie **Override Default** in. Klik op **Edit**. Selecteer in het dialoogvenster **Edit Join Column** de externe-sleutelkolom van de koppelingsdatabasetabel die naar een item in de tabel van de doelentiteit **end1**/**end2** verwijst. Als de kolomnaam, waarnaar wordt verwezen, in de tabel van de doelentiteit **end1**/**end2** wordt toegewezen aan het bijbehorende ID-attribuut, laat u **Referenced Column Name** ongewijzigd. Anders moet u de naam van de kolom selecteren waarnaar de externe-sleutelkolom verwijst. Schakel de selectievakjes **Insertable** en **Updatable** uit en klik op **OK**.
- Als de doelentiteit **end1**/**end2** meer dan één ID heeft, klikt u op de knop **Add** om

aanvullende join-kolommen toe te voegen en wijst u ze toe op de manier die wordt beschreven in de vorige stap.

11. Het ORM-bestand samenstellen – secundaire tabellen gebruiken

Met JPA kan een Java-klasse worden toegewezen aan meer dan één databasetabel. Zo kan **Host** worden toegewezen aan de tabel **Device** om persistentie van de meeste attributen mogelijk te maken en aan de tabel **NetworkNames** om persistentie van **host_hostName** mogelijk te maken. In dit geval is **Device** de primaire tabel en **NetworkNames** de secundaire tabel. Er kan een willekeurig aantal secundaire tabellen worden gedefinieerd. De enige voorwaarde is dat er een een-op-een relatie is tussen de items van de primaire en secundaire tabellen.

12. Een secundaire tabel definiëren

Selecteer de juiste klasse in de weergave JPA Structure. Open in de weergave **JPA Details** de sectie **Secondary Tables** en klik op **Add**. Selecteer in het dialoogvenster **Add Secondary Table** de juiste secundaire tabel. Laat de andere velden ongewijzigd.

Als de primaire en de secundaire tabel niet dezelfde primaire sleutels hebben, configureert u de join-kolommen in de sectie **Primary Key Join Columns** van de weergave **JPA Details**.

13. Een attribuut aan een secundaire tabel toewijzen

U wijst als volgt een klasseattribuut aan een veld van een secundaire tabel toe:

a. Wijs het attribuut toe, zoals wordt hierboven beschreven in "Attributen toewijzen" op pagina 113.

b. Selecteer in de sectie **Column** van de weergave JPA Details de naam van de secundaire tabel in het veld **Table** om de standaardwaarde te vervangen.

14. Een bestaand ORM-bestand als een basis gebruiken

Als u een bestaand **orm.xml**-bestand als een basis wilt gebruiken voor het bestand dat u ontwikkelt, voert u de volgende stappen uit:

a. Controleer of alle CIT's die in het bestaande **orm.xml**-bestand zijn toegewezen, in het actieve Eclipse-project worden geïmporteerd.

b. Selecteer en kopieer de entiteitstoewijzingen geheel of gedeeltelijk in het bestaande bestand.

c. Selecteer het tabblad **Source** van het bestand **orm.xml** in het Eclipse JPA-perspectief.

d. Plak alle gekopieerde entiteitstoewijzingen onder de tag **<entity-mappings>** van het bewerkte **orm.xml**-bestand, onder de tag **<schema>**. Controleer of de schematag wordt geconfigureerd, zoals hierboven beschreven in stap "Het ORM-bestand samenstellen – UCMDB-klassen aan databasetabellen toewijzen" op pagina 113. Alle geplakte entiteiten worden nu weergegeven in de weergave JPA Structure. Toewijzingen kunnen voortaan zowel grafisch als handmatig worden bewerkt via de xml-code van het bestand **orm.xml**.

e. Klik op **Opslaan**.

15. Een bestaand ORM-bestand importeren van een adapter

Als er al een adapter aanwezig is, kan de Eclipse-invoegtoepassing worden gebruikt om het bijbehorende ORM-bestand grafisch te bewerken. Importeer het **orm.xml**-bestand in Eclipse,

bewerk het met de invoegtoepassing en rol het vervolgens weer uit op de UCMDB-computer. Als u het ORM-bestand wilt importeren, drukt u op de knop op de Eclipse-werkbalk. Er wordt een bevestigingsvenster weergegeven. Klik op **OK**. Het ORM-bestand wordt van de UCMDB-computer gekopieerd naar het actieve Eclipse-project en alle relevante klassen worden vanuit het UCMDB-klassemodel geïmporteerd.

Als de relevante klassen niet worden weergegeven in de weergave JPA Structure, klikt u met de rechtermuisknop op het actieve project in de weergave Project Explorer, kiest u **Close** en vervolgens **Open**.

Het ORM-bestand kan voortaan grafisch worden bewerkt met Eclipse en vervolgens weer op de UCMDB-computer worden uitgerold, zoals hieronder wordt beschreven in "Het ORM-bestand in de CMDB uitrollen" beneden.

16. ## De correctheid van het orm.xlm-bestand controleren – ingebouwde correctheidscontrole

Met de Eclipse JPA-invoegtoepassing wordt gecontroleerd of er fouten aanwezig zijn. Deze fouten worden gemarkeerd in het bestand **orm.xml**. Zowel syntaxis- (zoals verkeerde tagnaam, niet-gesloten tag, ontbrekende ID) en toewijzingsfouten (zoals verkeerde attribuutnaam of databasetabelveldnaam) worden gecontroleerd. Als er sprake is van fouten, verschijnt er een beschrijving van de fouten in de weergave **Problems**.

17. ## Een nieuw integratiepunt aanmaken

Als er geen integratiepunt aanwezig is in de CMDB voor deze adapter, kunt u het in Integration Studio aanmaken. Zie "Integration Studio" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

Vul de naam van het integratiepunt in het dialoogvenster dat wordt geopend in. Het bestand **orm.xml** wordt naar de adaptermap gekopieerd. Er wordt een integratiepunt aangemaakt met alle geïmporteerde CI-typen als bijbehorende ondersteunde klassen, met uitzondering van CIT's met meervoudige knooppunten, als deze in het bestand **reconciliation_rules.txt** worden geconfigureerd. Zie "Bestand reconciliation_rules.txt (voor achterwaartse compatibiliteit)" op pagina 134 voor meer informatie over dit onderwerp.

18. ## Het ORM-bestand in de CMDB uitrollen

Sla het bestand **orm.xml** op en rol het uit op de UCMDB-server door te klikken op **UCMDB > Deploy ORM**. Het bestand **orm.xml** wordt naar de adaptermap gekopieerd en de adapter wordt opnieuw geladen. Het bewerkingsresultaat wordt weergegeven in een **Operation Result**-dialoogvenster. Als er tijdens het opnieuw laden een fout optreedt, wordt de stacktracering van Java-uitzonderingen weergegeven in het dialoogvenster. Als er nog geen integratiepunt met de adapter is gedefinieerd, worden er geen toewijzingsfouten gedetecteerd bij de uitrol.

19. ## Een TQL-voorbeeldquery aanmaken

a. Definieer een query (geen weergave) in Modeling Studio. Zie "Modeling Studio" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

b. Maak een integratiepunt aan met behulp van de adapter die u hebt aangemaakt in stap "Een nieuw integratiepunt aanmaken" boven. Zie "Dialoogvenster Nieuw integratiepunt/Integratiepunt bewerken" in de *HP Universal CMDB – Handleiding Data*

Flow Management voor meer informatie over dit onderwerp.

c. Tijdens het aanmaken van de adapter controleert u of de CI-typen die in de query moeten voorkomen, door dit integratiepunt worden ondersteund.

d. Bij de configuratie van de CMDB-invoegtoepassing, gebruikt u deze voorbeeldquery-naam in het dialoogvenster Settings. Zie de stap hierboven, "De invoegtoepassing CMDB configureren" op pagina 112, voor meer informatie.

e. Klik op de knop **Run TQL** om een voorbeeld-TQL uit te voeren en controleer of de vereiste resultaten worden geretourneerd met het nieuwe bestand **orm.xml**.

Adapterconfiguratiebestanden

De bestanden die in dit gedeelte worden beschreven, bevinden zich in het pakket **db-adapter.zip** in de map **C:\hp\UCMDB\UCMDBServer\content\adapters**.

In dit gedeelte worden de volgende configuratiebestanden beschreven:

Algemene configuratie

- **adapter.conf.** Het adapterconfiguratiebestand. Zie "Bestand adapter.conf " op volgende pagina voor meer informatie over dit onderwerp.

Eenvoudige configuratie

- **simplifiedConfiguration.xml**. Configuratiebestand dat **orm.xml**, **transformations.txt** en **reconciliation_rules.txt** met minder mogelijkheden vervangt. Zie "Bestand simplifiedConfiguration.xml" op pagina 119 voor meer informatie over dit onderwerp.

Geavanceerde configuratie

- **orm.xml**. Het OR-toewijzingsbestand (Object-Relational) waarin u een toewijzing tot stand brengt tussen CMDB-CIT's en databasetabellen. Zie "Bestand orm.xml" op pagina 121 voor meer informatie over dit onderwerp.
- **reconciliation_types.txt**. Bevat de regels op basis waarvan de afstemmingstypen worden geconfigureerd. Zie "Bestand reconciliation_types.txt" op pagina 134 voor meer informatie over dit onderwerp.

- **reconciliation_rules.txt**. Bevat de afstemmingsregels. Zie "Bestand reconciliation_rules.txt (voor achterwaartse compatibiliteit)" op pagina 134 voor meer informatie over dit onderwerp.
- **transformations.txt.** Bestand met transformaties waarin u de converters opgeeft die moeten worden toegepast bij de conversie van de CMDB-waarde naar de databasewaarde, en omgekeerd. Zie "Bestand transformations.txt" op pagina 136 voor meer informatie over dit onderwerp.
- **Discriminator.properties.** Met dit bestand wordt elk ondersteund CI-type toegewezen aan een door komma's gescheiden lijst met mogelijke gerelateerde waarden. Zie "Bestand discriminator.properties" op pagina 136 voor meer informatie over dit onderwerp.
- **Replication_config.txt.**Dit bestand bevat een door komma's gescheiden lijst met CI- en relatietypen waarvan de eigenschapsvoorwaarden worden ondersteund door de replicatie-invoegtoepassing. Zie "Bestand replication_config.txt" op pagina 138 voor meer informatie over dit onderwerp.
- **Fixed_values.txt.**Met dit bestand kunt u vaste waarden configureren voor specifieke attributen van bepaalde CIT's. Zie "Bestand fixed_values.txt" op pagina 138 voor meer informatie over dit onderwerp.

Hibernate-configuratie

- **persistence.xml.** Gebruikt om meegeleverde Hibernate-configuraties te overschrijven. Zie "Bestand persistence.xml" op pagina 138 voor meer informatie over dit onderwerp.

Bestand adapter.conf

Dit bestand bevat de volgende instellingen:

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

  Opmerking: Gebruik van dit bestand houdt in dat `orm.xml`, `transformations.txt` en `reconciliation_rules.txt` worden vervangen door minder mogelijkheden.

- **dal.ids.chunk.size=300.** Wijzig deze waarde niet.
- **dal.use.persistence.xml=false**. **true**: de adapter leest de Hibernate-configuratie uit `persistence.xml`.

  Opmerking: het wordt afgeraden de Hibernate-configuratie te overschrijven.

- **performance.memory.id.filtering=true**. Wanneer de GDBA TQLS uitvoert, kan in sommige gevallen een groot aantal ID's worden opgehaald en naar de database worden teruggestuurd met behulp van SQL. Om al dit werk te vermijden en de prestaties te verbeteren, probeert de GDBA de gehele weergave/tabel te lezen en worden de resultaten in het geheugen gefilterd.
- **id.reconciliation.cmdb.id.type=string/bytes**. Bij het toewijzen van de algemene DB-adapter.met behulp van ID-afstemming (zie de stap "Het bestand reconciliation_types.txt configureren (voor de UCMDB 9.0x-toewijzingsengine)" in "De toewijzingsengine implementeren" op pagina 180 voor meer informatie) kunt u de **cmdb_id** toewijzen aan een **string**- of **bytes/raw**-kolomtype door de eigenschap **META-INF/ adapter.conf** te wijzigen.

- **performance.memory.id.filtering=true**. Deze parameter is optioneel. Als de parameter niet in het bestand voorkomt, is de standaardwaarde **true**. Als de waarde **true** is, probeert de algemene DB-adapter één SQL-instructie te genereren voor elke query die wordt uitgevoerd (voor vullingstaken of voor een federated query). Gebruik van één enkele SQL-instructie verbetert de prestaties en het geheugengebruik van de algemene DB-adapter. Is de waarde **false**, dan genereert de algemene DB-adapter meerdere SQL-instructies, hetgeen langer kan duren en meer geheugen gebruikt dan bij één enkele SQL-instructie. Zelfs wanneer dit attribuut wordt ingesteld op **true**, zal de adapter geen afzonderlijke SQL-instructie genereren wanneer de volgende scenario's van toepassing zijn:
  - De database waarmee de adapter verbinding maakt is geen Oracle- of SQL-server.
  - De uitgevoerde TQL bevat een kardinaliteit anders dan 0..* en 1..* (bijvoorbeeld als er een kardinaliteit is zoals 2..* of 0..2).
- **in.expression.size.limit=950** (standaard). Met deze parameter wordt de expressie 'IN' van de uitgevoerde SQL gesplitst wanneer de omvangslimiet van de lijst met argumenten wordt bereikt.

Bestand simplifiedConfiguration.xml

Dit bestand wordt gebruikt voor eenvoudige toewijzing van UCMDB-klassen aan databasetabellen. Als u de sjabloon wilt openen om het bestand te bewerken, navigeert u naar **Adapterbeheer > db-adapter > Configuratiebestanden**.

In dit gedeelte vindt u de volgende onderwerpen:

- "Bestandssjabloon simplifiedConfiguration.xml" beneden
- "Beperkingen" op pagina 121

Bestandssjabloon simplifiedConfiguration.xml

De eigenschap **CMDB-class-name** is het type meervoudig knooppunt (het knooppunt waarmee federated CIT's worden verbonden in de TQL):

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

reconciliation-by-two-nodes. Afstemming kan plaatsvinden met één knooppunt of twee knooppunten. In dit voorbeeld worden bij de afstemming twee knooppunten gebruikt.

connected-node-CMDB-class-name. Het type van de tweede klasse, dat nodig is in de afstemmings-TQL.

CMDB-link-type. Het relatietype dat nodig is in de afstemmings-TQL.

link-direction. De richting van de relatie in de afstemmings-TQL (van `node` naar `ip_address` of van `ip_address` naar `node`):

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

De afstemmingsexpressie in de vorm van OR's; elke OR bevat AND's.

is-ordered. Bepaalt of afstemming plaatsvindt in volgorde of door een reguliere `OR`-vergelijking.

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

Als de afstemmingseigenschap wordt opgehaald uit de hoofdklasse (het meervoudige knooppunt), gebruikt u de tag **attribute** en anders gebruikt u de tag **connected-node-attribute**.

ignore-case.true: wanneer gegevens in het UCMDB-klassemodel worden vergeleken met gegevens in het RDBMS, wordt geen onderscheid gemaakt tussen hoofdletters en kleine letters:

```
                <attribute CMDB-attribute-name="name" column-name="
[column_name]" ignore-case="true"/>
```

De kolomnaam is de naam van de externe-sleutelkolom (de kolom met waarden die verwijzen naar de primaire-sleutelkolom met meerdere knooppunten).

Als de primaire-sleutelkolom met meerdere knooppunten is samengesteld uit verschillende kolommen, moeten er verschillende externe-sleutelkolommen zijn, één voor elke primaire-sleutelkolom.

```
        <foreign-primary-key column-name="[column_name]" CMDB-class-
primary-key-column="[column_name]"/>
```

Als er weinig primaire-sleutelkolommen zijn, dupliceert u deze kolom.

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

De eigenschappen **from-CMDB-converter** en **to-CMDB-converter** zijn Java-klassen waarmee de volgende interfaces worden geïmplementeerd:

- com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB
- com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB

Gebruik deze converters als de waarden in de CMDB en in de database niet hetzelfde zijn.

In dit voorbeeld wordt `GenericEnumTransformer` gebruikt om de enumerator te converteren in overeenstemming met het XML-bestand dat tussen de haakjes is geschreven (**generic-enum-transformer-example.xml**):

```
    <attribute CMDB-attribute-name="[CMDB_attribute_name]" column-
name="[column_name]" from-CMDB-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.im-
pl.
GenericEnumTransformer(generic-enum-transformer-example.xml)" to-CMDB-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.im-
pl.
GenericEnumTransformer(generic-enum-transformer-example.xml)" />
    <attribute CMDB-attribute-name="[CMDB_attribute_name]" column-
```

```
name="[column_name]" />
    <attribute CMDB-attribute-name="[CMDB_attribute_name]" column-
name="[column_name]" />
  </class>
</generic-DB-adapter-config>
```

Beperkingen

- Kan worden gebruikt om alleen TQL-query's met één knooppunt (in de databasebron) toe te wijzen. U kunt bijvoorbeeld een `node >-ticket` en een `ticket`-TQL-query uitvoeren. Als u de hiërarchie van knooppunten uit de database wilt overhalen, moet u het geavanceerde bestand **orm.xml** gebruiken.
- Alleen een-op-veel relaties worden ondersteund. U kunt bijvoorbeeld een of meer tickets op elk knooppunt overhalen. U kunt geen tickets overhalen die tot meer dan één knooppunt behoren.
- U kunt niet dezelfde klasse verbinden met verschillende typen CMDB-CIT's. Als u bijvoorbeeld definieert dat `ticket` is verbonden met `node`, kan deze niet tevens met `application` worden verbonden.

Bestand orm.xml

Dit bestand wordt gebruikt voor toewijzing van CMDB-CIT's aan databasetabellen.

Een sjabloon waarmee u een nieuw bestand kunt aanmaken, bevindt zich in de map **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF**.

Als u de sjabloon wilt openen om het bestand te bewerken, navigeert u naar **Adapterbeheer > db-adapter > Configuratiebestanden**.

In dit gedeelte vindt u de volgende onderwerpen:

Bestandssjabloon 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>
```

Wijzig de pakketnaam niet.

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

entity. De naam van het CMDB-CIT. Dit is de entiteit met meerdere knooppunten.

Controleer of **class** een voorvoegsel in de vorm van **generic_db_adapter.** bevat.

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

Gebruik een secundaire tabel als de entiteit aan meer dan één tabel wordt toegewezen.

```
        <secondary-table name="" />
        <attributes>
```

Gebruik voor een overerving van één tabel met Discriminator de volgende code:

```
        <inheritance strategy="SINGLE_TABLE" />
        <discriminator-value>node</discriminator-value>
        <discriminator-column name="[column_name]" />
```

Attributen met tag **id** zijn de primaire-sleutelkolommen. Controleer dat de naamgevingsconventies voor deze primaire-sleutelkolommen **idX** zijn (id1, id2, enzovoort) waarin **X** de kolomindex in de primaire sleutel is.

```
            <id name="id1">
```

Wijzig alleen de kolomnaam van de primaire sleutel.

```
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
```

basic. Gebruikt om de CMDB-attributen te declareren. Bewerk alleen de eigenschappen **name** en **column_name**.

```
            <basic name="name">
                <column updatable="false" insertable="false" name="
[column_name]" />
            </basic>
```

Wijs voor een overerving van één tabel met Discriminator de extra klassen als volgt toe:

```
    <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>
```

In het volgende voorbeeld wordt een CMDB-attribuutnaam zonder voorvoegsel weergegeven:

```
            <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>
```

Dit is een relatie-entiteit. De naamgevingsconventie is **end1Type_linkType_end2Type**. In dit voorbeeld is **end1Typenode** en is **linkTypecomposition**.

```
    <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>
```

De doelentiteit is de entiteit waarnaar deze eigenschap verwijst. In dit voorbeeld wordt **end1** toegewezen aan entiteit **node**.

many-to-one. Veel relaties kunnen met één knooppunt worden verbonden.

join-column. De kolom die **end1**-ID's bevat (de doelentiteit-ID's).

referenced-column-name. De kolomnaam in de doelentiteit (**node**) die de ID's bevat die in de join-kolom worden gebruikt.

```
            <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. Eén relatie kan worden verbonden met één **[CMDB_class_name]**.

```
            <one-to-one target-entity="[CMDB_class_name]"
name="end2">
                <join-column updatable="false" insertable="false"
referenced-column-name="" name="[column_name]" />
            </one-to-one>
        </attributes>
    </entity>
</entity-mappings>
```

node attribute. Dit is een voorbeeld van het toevoegen van een knooppuntattribuut.

```
<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>
```

Meerdere ORM-bestanden

Meerdere toewijzingsbestanden worden ondersteund. Elke toewijzingsbestandsnaam moet eindigen met **orm.xml**. Alle toewijzingsbestanden moeten onder de map META-INF van de adapter worden geplaatst.

Naamgevingsconventies

- In elke entiteit moet de klasse-eigenschap overeenkomen met het voorvoegsel van `generic_db_adapter`.
- Primaire-sleutelkolommen moeten namen gebruiken als **idX** waarin **X = 1**, **2**, ..., in overeenstemming met het aantal primaire sleutels in de tabel.
- Attribuutnamen moeten overeenkomen met namen van klasseattributen, ook in hoofdletters en kleine letters.
- De relatienaam heeft de vorm `end1Type_linkType_end2Type`.
- CMDB-CIT's, die ook gereserveerde woorden in Java zijn, moeten een voorvoegsel krijgen in de vorm van **gdba_**. Voor het CMDB-CIT **goto** moet de ORM-entiteit bijvoorbeeld de naam **gdba_goto** krijgen.

Inline SQL-instructies gebruiken in plaats van tabelnamen

U kunt entiteiten aan inline `Select`-clausules toewijzen in plaats van aan databasetabellen. Dit is vergelijkbaar met het definiëren van een weergave in de database en het toewijzen van een entiteit aan deze weergave. Bijvoorbeeld:

```
    <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 dit voorbeeld moeten de knooppuntattributen worden toegewezen aan columns id1, name en host_os in plaats van aan id, name en os.

De volgende beperkingen zijn van toepassing:

- De inline SQL-instructie is alleen beschikbaar wanneer Hibernate als de JPA-provider wordt gebruikt.
- Ronde haken rondom de inline Select-clausule van SQL zijn verplicht.
- Het element **<schema>** mag niet aanwezig zijn in het bestand **orm.xml**. In het geval van

Microsoft SQL Server 2005 houdt dit in dat alle tabelnamen het voorvoegsel `dbo.` moeten hebben in plaats van ze globaal te definiëren met `<schema>dbo</schema>`.

Schema orm.xml

In de volgende tabel worden de algemene elementen van het bestand **orm.xml** uitgelegd. Het volledige schema is te vinden op http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. De lijst is niet volledig. Met deze lijst wordt voornamelijk het specifieke gedrag van de JPA-standaard (Java Persistence API) voor de algemene database-adapter verklaard.

Naam en pad van element	Beschrijving	Attributen
entity-mappings	Het hoofdelement voor het entiteitstoewijzingsdocument. Dit element moet exact hetzelfde zijn als het element in de GDBA-voorbeeldbestanden.	
description (entity-mappings)	Een vrije-tekstbeschrijving van het entiteitstoewijzingsdocument. (Optioneel)	
package (entity-mappings)	De naam van het Java-pakket dat de toewijzingsklassen bevat. Moet altijd de tekst `generic_db_adapter` bevatten.	1. **Naam:** name **Beschrijving:** De naam van het UCMDB CI-type waaraan deze entiteit wordt toegewezen. Als deze entiteit wordt toegewezen aan een koppeling in de CMDB, moet de naam van de entiteit de syntaxis `<end_1>_<link_name>_<end_2>` hebben. Met `node_composition_cpu` wordt bijvoorbeeld een entiteit toegewezen aan de samenstellingskoppeling tussen een knooppunt en een CPU. Als de naam van het CI-type hetzelfde is als de naam van de Java-klasse zonder het voorvoegsel package, kan dit veld worden weggelaten. **Is verplicht?:** Optioneel **Type:** String 2. **Naam:** class **Beschrijving:** De volledig gekwalificeerde naam van de

Naam en pad van element	Beschrijving	Attributen
		Java-klasse die voor deze DB-entiteit wordt aangemaakt. De naam van het pakket van de Java-klasse moet hetzelfde zijn als de naam die in het element `package` is gegeven. U kunt als de klassenaam geen gereserveerde Java-woorden gebruiken, zoals interface of switch. Voeg in plaats daarvan het voorvoegsel `gdba_` aan de naam toe (interface wordt dan `generic_db_adapter.gdba_interface`. **Is verplicht?:** Vereist **Type:** String
table (entity-mappings>entity)	Met dit element wordt de primaire tabel van de DB-entiteit gedefinieerd. Kan slechts eenmaal verschijnen. Vereist.	**Naam:** name **Beschrijving:** De naam van de primaire tabel. Als de naam van de tabel het schema niet bevat waartoe de tabel behoort, wordt alleen naar de tabel gezocht in het schema van de gebruiker waarmee het integratiepunt is aangemaakt. Dit kan ook een geldige SELECT-instructie zijn. Als dit een SELECT-instructie is, moet deze tussen haakjes worden geplaatst. **Is verplicht?:** Vereist **Type:** String
secondary-table (entity-mappings>entity)	Dit element kan ook worden gebruikt om een secundaire tabel voor de DB-entiteit te definiëren. Deze tabel moet met de primaire tabel worden verbonden met een 1-op-1 relatie. U kunt meer dan één secundaire tabel definiëren. Optioneel.	**Naam:** name **Beschrijving:** De naam van de secundaire tabel. Als de naam van de tabel het schema niet bevat waartoe de tabel behoort, wordt alleen naar de tabel gezocht in het schema van de gebruiker waarmee het integratiepunt is aangemaakt. Dit kan ook een geldige SELECT-instructie zijn. Als dit een SELECT-instructie is, moet deze tussen haakjes worden geplaatst. **Is verplicht?:** Vereist **Type:** String

Naam en pad van element	Beschrijving	Attributen
primary-key-join-column (entity-mappings > entity > secondary-table)	Als de secundaire tabel en de primaire tabel niet worden verbonden met velden met dezelfde naam, wordt met dit element de naam van het primaire-sleutelveld gedefinieerd dat moet worden verbonden met het primaire-sleutelveld van de primaire tabel.	**Naam:** name **Beschrijving:** De naam van het primaire-sleutelveld in de secundaire tabel. Als dit element niet bestaat, wordt ervan uitgegaan dat het primaire-sleutelveld dezelfde naam heeft als het primaire-sleutelveld van de primaire tabel. **Is verplicht?:** Optioneel **Type:** String
inheritance (entity-mappings>entity)	Als de huidige entiteit de bovenliggende entiteit voor een familie van DB-entiteiten is, gebruikt u dit element om het als zodanig te markeren. Optioneel.	**Naam:** strategy **Beschrijving:** Definieert de manier waarop de overerving in uw database wordt geïmplementeerd. **Is verplicht?:** Vereist **Type:** Een van de volgende waarden: • SINGLE_TABLE: Deze entiteit en alle onderliggende entiteiten zijn in dezelfde tabel aanwezig. • JOINED: De onderliggende entiteiten bevinden zich in samengevoegde tabellen. • TABLE_PER_CLASS: Elke entiteit wordt volledig gedefinieerd door een afzonderlijke tabel.
discriminator-column (entity-mappings>entity)	Als de overerving van het type SINGLE_TABLE is, wordt dit element gebruikt om de naam van het veld te definiëren waarmee het type entiteit voor elke rij wordt bepaald.	**Naam:** name **Beschrijving:** De naam van de Discriminator-kolom. **Is verplicht?:** Vereist **Type:** String
discriminator-value (entity-mappings>entity)	Met dit element wordt het type van de specifieke entiteit in de overervingsstructuur gedefinieerd. Deze naam moet hetzelfde zijn als de naam die is gedefinieerd in het bestand **discriminator.properties** voor de waardegroep van dit specifieke entiteittype.	
attributes	Het hoofdelement van alle	

Naam en pad van element	Beschrijving	Attributen
(entity-mappings>entity)	attribuuttoewijzingen voor een entiteit.	
id (entity-mappings > entity attributes)	Met dit element wordt het sleutelveld voor de entiteit gedefinieerd. Er moet minimaal één id-veld zijn gedefinieerd. Als er meer dan één id-element bestaat, wordt met het bijbehorende veld een compound-sleutel voor de entiteit aangemaakt. U moet zoveel mogelijk compound-sleutels voor CI-entiteiten vermijden (niet voor koppelingen).	**Naam:** name **Beschrijving:** Een string van het type idX, waarin X een getal is tussen 1 en 9. De eerste id moet worden gemarkeerd als id1, de tweede als id2, enzovoort. Dit is NIET de naam van het sleutelattribuut in UCMDB. **Is verplicht?:** Vereist **Type:** String
basic (entity-mappings > entity attributes)	Met dit element wordt een toewijzing gedefinieerd tussen een veld in de tabel, dat geen deel uitmaakt van de primaire sleutel van de tabel, en een UCMDB-attribuut.	**Naam:** name **Beschrijving:** De naam van het UCMDB-attribuut waaraan het veld wordt toegewezen. Dit attribuut moet aanwezig zijn in het UCMDB CI-type waaraan de huidige entiteit wordt toegewezen. **Is verplicht?:** Vereist **Type:** String
column (entity-mappings > entity > attributes >id -OF- (entity-mappings > entity > attributes > basic)	Hiermee wordt de naam van de kolom gedefinieerd in de tabel voor basic-toewijzing of een id-veld.	1. **Naam:** name **Beschrijving:** De naam van het veld. **Is verplicht?:** Vereist **Type:** String 2. **Naam:** table **Beschrijving:** De naam van de tabel waartoe het veld behoort. Dit moet de primaire tabel of een van de secundaire tabellen zijn die voor de entiteit zijn gedefinieerd. Als dit attribuut wordt weggelaten, wordt ervan uitgegaan dat het veld tot de primaire tabel behoort. **Is verplicht?:** Optioneel **Type:** String
one-to-one	Hiermee wordt een kolom	1. **Naam:** name

Naam en pad van element	Beschrijving	Attributen
(entity-mappings > entity > attributes)	gedefinieerd waarvan de waarde zich in een andere tabel bevindt, en de twee tabellen worden met een een-op-een relatie verbonden. Dit element wordt alleen ondersteund voor koppelingsentiteitstoewijzingen en niet voor andere CI-typen. Dit is de enige manier om een toewijzing tussen een tabel en een UCMDB-koppeling te definiëren.	**Beschrijving:** Welke van de twee einden met dit veld wordt vertegenwoordigd. **Is verplicht?:** Vereist **Type:**`end1` of `end2` 2. **Naam:** target-entity **Beschrijving:** De naam van de entiteit waarnaar het eind verwijst. **Is verplicht?:** Vereist **Type:** Een van de entiteitnamen die in het entiteitstoewijzingsdocument zijn gedefinieerd.
join-column (entity-mappings > entity attributes > one-to-one)	Hiermee wordt gedefinieerd hoe de doelentiteit die is gedefinieerd in het bovenliggende een-op-een element, en de huidige entiteit moeten worden samengevoegd.	1. **Naam:** name **Beschrijving:** De naam van het veld in de huidige tabel waarmee de een-op-een join wordt uitgevoerd. **Is verplicht?:** Vereist **Type:** String 2. **Naam:** name **Beschrijving:** De naam van een veld in de gekoppelde entiteit waarover de join moet worden uitgevoerd. Als dit attribuut wordt weggelaten, wordt ervan uitgegaan dat de gekoppelde tabel een kolom heeft met dezelfde naam als het veld dat in het name-attribuut is gedefinieerd. **Is verplicht?:** Optioneel **Type:** String

Voorbeeld van het aanmaken van het bestand orm.xml

In het hier gegeven voorbeeld ziet u hoe u het bestand **orm.xml** aanmaakt. In dit voorbeeld worden SQL-tabellen in een externe database toegewezen aan CI-typen in de UCMDB.

Uitgaande van tabellen met de volgende indeling in de externe database: vul de tabel **Hosts** met knooppunten, de tabel **IP_Addresses** met IP-adressen en maak als volgt koppelingen aan tussen de knooppunten en IP-adressen:

Tabel Hosts

host_name	host_id
Test1	1
Test2	2
Test3	3

Tabel IP_Addresses

ip_address	ip_id
10.1.1.1	1
10.2.2.2	2
10.3.3.2	3
10.4.4.4	4

Tabel Host_IP_Link (koppelingen tussen knooppunten en IP-adressen)

host_id	ip_id
1	1
2	2
2	3
3	4

De primaire sleutel voor de tabel **Hosts** is het veld **host_id** en de primaire sleutel in de tabel **IP_ Addresses Table** is het veld **ip_id**. In de tabel **Host_IP_Link** zijn **host_id** en **ip_id** externe sleutels in de tabel **Hosts** en **IP_Addresses Table**.

Op basis van de bovenstaande tabellen maakt u het bestand **orm.xml** aan volgens de onderstaande stappen: De in dit voorbeeld gebruikte entiteiten zijn **node**, **ip_address** en **node_ containment_ip_address**

1. Maak de entiteit **node** aan door **host_id** van de tabel **Hosts** als volgt toe te wijzen:

```
<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>
```

class van de entiteit moet een CI-type zijn dat al in de UCMDB bestaat. **name** van de tabel is de tabel binnen de database die zowel een ID als de Host-gegevens bevat. Het ID-attribuut is vereist voor het identificeren van specifieke hosts en wordt later in de toewijzing gebruikt. In dit voorbeeld wordt het attribuut **name** van deze entiteit gevuld met de kolom **host_name** in de tabel hosts.

2. Voor de volgende entiteit wijst u de IP-adressen uit de tabel Interfaces toe:

```
<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. Vervolgens moet de koppeling tussen het knooppunt en het IP-adres worden aangemaakt door middel van de toewijzingstabel, en verwijst u naar het veld **ip_id** (hoewel het desgewenst zowel naar het veld **host_id** als het veld **ip_id** kan verwijzen).

```
<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>
```

De entiteitnaam voor de container heeft de volgende indeling: [end1 CIT]_[link CIT]_[end2 CIT]. Aangezien het CI-type van de koppeling **containment** is, is voor dit voorbeeld de entiteitnaam voor de container: **node_containment_ip_address** en is de entiteitklasse **generic_db_adapter.node_containment_ip_address**. De ID is vereist in dit blok met code en hoewel in dit voorbeeld slechts één ID van de interface wordt gebruikt, zouden beide kolommen kunnen verwijzen naar id1 en id2. Daarvoor zou de code dan zijn:

```
<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>
```

De twee uiteinden van deze koppeling zijn ‘many-to-one’ en ‘one-to-one’, hetgeen betekent dat enerzijds elk IP-adres gekoppeld wordt aan 1 knooppunt en anderzijds dat een knooppunt aan

een groot aantal IP-adressen kan worden gekoppeld. De opgenomen kolommen komen van de tabel Koppelingen en verwijzen naar de tabellen Hosts en Interfaces.

Bestand reconciliation_types.txt

Met dit bestand worden de afstemmingstypen geconfigureerd.

Elke rij in het bestand vertegenwoordigt een CMDB-CIT dat wordt verbonden met het CIT van een federated database in de TQL-query.

Bestand reconciliation_rules.txt (voor achterwaartse compatibiliteit)

Dit bestand wordt gebruikt om de afstemmingsregels te configureren als u afstemming wilt uitvoeren wanneer DBMappingEngine in de adapter wordt geconfigureerd. Als u DBMappingEngine niet gebruikt, wordt het algemene UCMDB-afstemmingsmechanisme gebruikt en hoeft dit bestand niet geconfigureerd te worden.

Elke rij in het bestand staat voor een regel. Bijvoorbeeld:

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

Het meervoudige knooppunt wordt gevuld met de naam van het meervoudige knooppunt (het CMDB-CIT dat met het CIT van de federated database in de TQL wordt verbonden).

Deze expressie omvat de logica die bepaalt of twee meervoudige knooppunten gelijk zijn (één meervoudig knooppunt in de CMDB en het andere in de databasebron).

De expressie is samengesteld uit `OR's` of `AND's`.

De conventie met betrekking tot attribuutnamen in het expressiedeel is `[className].[attributeName]`. Bijvoorbeeld: `attributeName` in de klasse `ip_address` wordt als `ip_address.name` geschreven.

Voor een geordende vergelijking (als met de eerste `OR`-subexpressie een antwoord wordt geretourneerd dat de meervoudige knooppunten niet gelijk zijn, wordt de tweede OR-subexpressie niet vergeleken), moet u `ordered expression` in plaats van `expression` gebruiken.

Als u tijdens een vergelijking geen onderscheid wilt maken tussen hoofdletters en kleine letters, gebruikt u het besturingsteken (**^**).

De parameters `end1_type`, `end2_type` en `link_type` worden alleen gebruikt als de afstemmings-TQL twee knooppunten bevat en niet alleen een meervoudig knooppunt. In dit geval is de afstemmings-TQL `end1_type > (link_type) > end2_type`.

De relevante indeling hoeft niet te worden toegevoegd, aangezien deze van de expressie wordt overgenomen.

Typen afstemmingsregels

Afstemmingsregels nemen de vorm van OR- en AND-voorwaarden over. U kunt deze regels in verschillende knooppunten definiëren (zo wordt knooppunt geïdentificeerd door `name from`

`nodeAND/ORname from ip_address`).

Met de volgende opties wordt een overeenkomst gevonden:

- **Geordende vergelijking.** De afstemmingsexpressie wordt van links naar rechts gelezen. Twee `OR`-subexpressies worden als gelijk beschouwd als ze waarden hebben en deze gelijk zijn. Twee `OR`-subexpressies worden als ongelijk beschouwd als beide waarden hebben en deze niet gelijk zijn. Voor elk ander geval is er geen uitkomst en wordt de volgende `OR`-subexpressie getest op gelijkheid.

  name from node OR from ip_address. Als zowel de CMDB als de gegevensbron `name` bevatten en ze gelijk zijn, worden de knooppunten als gelijk beschouwd. Als beide `name` hebben maar niet gelijk zijn, worden de knooppunten als niet gelijk beschouwd zonder de `name` van `ip_address` te testen. Als in de CMDB of de gegevensbron `name of node` ontbreekt, wordt de `name of ip_address` gecontroleerd.

- **Reguliere vergelijking.** Als er een overeenkomst voorkomt in een van de `OR`-subexpressies, worden de CMDB en de gegevensbron als gelijk beschouwd.

  name from node OR from ip_address. Als er geen overeenkomst is in `name of node`, wordt `name of ip_address` op gelijkheid gecontroleerd.

Voor complexe afstemmingen, waarin de afstemmingsentiteit in het klassemodel wordt gemodelleerd als verschillende CIT's met relaties (zoals `node`), bevat de toewijzing van een supersetknooppunt alle relevante attributen van alle gemodelleerde CIT's.

Opmerking: hierdoor is er een beperking dat alle afstemmingsattributen in de gegevensbron zich moeten bevinden in tabellen die dezelfde primaire sleutel delen.

Een andere beperking geeft aan dat de afstemmings-TQL-query niet meer dan twee knooppunten mag hebben. Bijvoorbeeld: de TQL `node > ticket` heeft een knooppunt in de CMDB en een ticket in de gegevensbron.

Voor afstemming van de resultaten moet `name` worden opgehaald uit het knooppunt en/of `ip_address`.

Als de `name` in de CMDB de indeling `*.m.com` heeft, kan een converter vanuit CMDB naar de federated database en omgekeerd worden gebruikt om deze waarden te converteren.

De kolom `node_id` in de databasetickettabel wordt gebruikt om een verbinding te maken tussen de entiteiten (de gedefinieerde koppeling kan ook in een knooppunttabel worden gemaakt):

DB-knooppunt	
PK	node_id
	naam

DB IP_Address	
PK	ip_id
	naam

DB-ticket	
PK	ticket_id
	node_id

Opmerking: De drie tabellen moeten deel uitmaken van de federated RDBMS-bron en niet de CMDB-database.

Bestand transformations.txt

Dit bestand bevat alle converterdefinities.

De indeling is zodanig dat elke regel een nieuwe definitie bevat.

Bestandssjabloon 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. De entiteitsnaam zoals deze wordt weergegeven in het bestand `orm.xml`.

attribute. De attribuutnaam zoals deze wordt weergegeven in het bestand `orm.xml`.

to_DB_class. De volledig gekwalificeerde naam van een klasse waarmee de interface **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB** wordt geïmplementeerd. De elementen tussen haakjes worden aan deze klasseconstructor doorgegeven. Met deze converter kunt u CMDB-waarden converteren naar databasewaarden, bijvoorbeeld om het achtervoegsel **.com** aan elke knooppuntnaam toe te voegen.

from_DB_class. De volledig gekwalificeerde naam van een klasse waarmee de interface **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB** wordt geïmplementeerd. De elementen tussen haakjes worden aan deze klasseconstructor doorgegeven. Met deze converter kunt u CMDB-waarden converteren naar databasewaarden, bijvoorbeeld om het achtervoegsel **.com** aan elke knooppuntnaam toe te voegen.

Zie "Meegeleverde converters" op pagina 139 voor meer informatie over dit onderwerp.

Bestand discriminator.properties

Met dit bestand wordt elk ondersteund CI-type (dat ook wordt gebruikt als een discriminator-waarde in orm.xml) toegewezen aan een door komma's gescheiden lijst met mogelijke overeenkomende waarden van de discriminator-kolom.

Bij toepassing van een voorwaarde gebruikt u de volgende syntaxis: `like(condition)`, waarbij `condition` een string is die de volgende jokertekens kan bevatten:

- **%** (procentteken) - hiermee kunt u elke string van elke mogelijke lengte vergelijken (met inbegrip van strings met nul tekens)
- **_** (onderstrepingsteken) - hermee kunt u één afzonderlijk teken vergelijken

`like(%unix%)` komt bijvoorbeeld overeen met unix, linux, unix-aix, enzovoort. Lijkend op-voorwaarden mogen alleen op kolommen met strings worden toegepast.

U kunt ook één discriminator-waarde hebben die is toegewezen aan elke waarde die geen deel uitmaakt van een andere discriminator door `'all-other'` aan te geven.

Als voor de adapter die u aanmaakt, discriminator-mogelijkheden worden gebruikt, moet u alle discriminator-waarden in het bestand **discriminator.properties** definiëren.

Voorbeeld van Discriminator-toewijzing:

Zo ondersteunt de adapter de CI-typen `node`, `nt` en `unix`, en de database bevat één tabel, met de naam `t_nodes`, met daarin een kolom met de naam **type**. Is het type 10001, dan geeft de rij een knooppunt aan, is het type 10004, dan geeft het een unix-machine aan, enzovoort. Het bestand **discriminator.properties** kan er zo uitzien:

```
node=10001, 10005 nt=10002,10003 unix=2% mainframe=all-other
```

Het bestand **orm.xml** bevat de volgende 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>
```

Het attribuut discriminator_column wordt dan als volgt berekend:

- Als **type**`10002` of `10003` voor een bepaald item bevat, wordt het item toegewezen aan het **nt**-CIT.
- Als **type**`10001` of `10005` voor een bepaald item bevat, wordt het item toegewezen aan het **nt**-CIT.
- Als **type** begint met `2` voor een bepaald item bevat, wordt het item toegewezen aan het **unix**-CIT.
- Elke andere waarde in de kolom **type** wordt toegewezen aan het **mainframe**-CIT.

> **Opmerking:** Het **node**-CIT is ook het bovenliggend item van **nt** en **unix**.

Bestand replication_config.txt

Dit bestand bevat een door komma's gescheiden lijst met CI- en relatietypen waarvan de eigenschapsvoorwaarden worden ondersteund door de replicatie-invoegtoepassing. Zie "Invoegtoepassingen" op pagina 143 voor meer informatie over dit onderwerp.

Bestand fixed_values.txt

Met dit bestand kunt u vaste waarden configureren voor specifieke attributen van bepaalde CIT's. Op deze manier kan aan elk van deze attributen een vaste waarde worden toegewezen die niet in de database is opgeslagen.

Het bestand moet nul of meer items met de volgende indeling bevatten:

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

Bijvoorbeeld:

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

Bovendien ondersteunt het bestand een reeks constanten. Gebruik de volgende syntaxis om een lijst met constanten te definiëren:

```
entity[<NaamEntiteit>] attribute[<NaamAttribuut>] value[{<Waarde1>,
<Waarde2>, <Waarde3>, ... }]
```

Bestand persistence.xml

Dit bestand wordt gebruikt om de standaardinstellingen van Hibernate te overschrijven en ondersteuning voor databasetypen toe te voegen die niet meegeleverd zijn (OOB-databasetypen zijn Oracle Server, Microsoft SQL Server en MySQL).

Als u een nieuw databasetype moet ondersteunen, moet u ervoor zorgen dat een provider van een verbindingspool beschikbaar is (de standaardinstelling is `c3p0`) en een JDBC-stuurprogramma voor uw database (plaats de `*.jar`-bestanden in de adaptermap).

Als u alle beschikbare Hibernate-waarden wilt bekijken die kunnen worden gewijzigd, controleert u de klasse **org.hibernate.cfg.Environment** (meer informatie kunt u vinden in http://www.hibernate.org).

Voorbeeld van het bestand 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>
```

Meegeleverde converters

U kunt de volgende converters (transformers) gebruiken om federated query's en replicatietaken naar en van databasegegevens te converteren.

In dit gedeelte vindt u de volgende onderwerpen:

Converter enum-transformer

Voor deze converter wordt een XML-bestand gebruikt dat als een invoerparameter is opgegeven.

Met het XML-bestand wordt een toewijzing tot stand gebracht tussen hardcoded CMDB-waarden en databasewaarden (enums). Als een van de waarden niet bestaat, kunt u ervoor kiezen dezelfde waarde te retourneren, null te retourneren of een uitzondering te genereren.

De transformer voert een vergelijking uit tussen twee strings met behulp van een al dan niet hoofdlettergevoelige methode. De standaardinstelling is hoofdlettergevoelig. Gebruik de volgende syntaxis voor het definiëren van de niet-hoofdlettergevoelige instelling: `case-sensitive="false"` in het element enum-transformer.

Gebruik één XML-toewijzing voor elk entiteitattribuut.

Opmerking: deze converter kan worden gebruikt voor beide velden `to_DB_class` en `from_DB_class` in het bestand **transformations.txt**.

Invoerbestand-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>
```

Voorbeeld van het converteren van de waarde 'sys' naar de waarde 'System':

In dit voorbeeld wordt de waarde `sys` in de CMDB getransformeerd naar de waarde `System` in de federated database, en de waarde `System` in de federated database wordt naar de waarde `sys` in de CMDB getransformeerd.

Als de waarde niet voorkomt in het XML-bestand (bijvoorbeeld de string `demo`), retourneert de converter dezelfde invoerwaarde die wordt ontvangen.

```
<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>
```

Voorbeeld van de conversie van een externe of CMDB-waarde naar een null-waarde:

In dit voorbeeld wordt een waarde van **NNN** in de externe database getransformeerd naar een null-waarde in de CMDB-database.

```
<value cmdb-value="null" is-cmdb-value-null="true" external-db-
value="NNN"/>
```

In dit voorbeeld wordt de waarde **OOO** in de CMDB getransformeerd naar een null-waarde in de externe database.

```
<value cmdb-value="OOO" external-db-value="null" is-db-value-
null="true"/>
```

Converter SuffixTransformer

Met deze converter worden achtervoegsels toegevoegd aan of verwijderd uit de bronwaarde van de CMDB of de federated database.

Er zijn twee implementaties:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddSuffixTransformer**. Hiermee wordt het achtervoegsel (opgegeven als invoer) toegevoegd

bij het converteren van federated-databasewaarde naar CMDB-waarde en wordt het achtervoegsel verwijderd bij het converteren van CMDB-waarde naar federated-databasewaarde.

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemoveSuffixTransformer**. Hiermee wordt het achtervoegsel (opgegeven als invoer) verwijderd bij het converteren van federated-databasewaarde naar CMDB-waarde en wordt het achtervoegsel toegevoegd bij het converteren van CMDB-waarde naar federated-databasewaarde.

Converter PrefixTransformer

Met deze converter wordt een voorvoegsel toegevoegd aan of verwijderd uit de CMDB-waarde of de federated-databasewaarde.

Er zijn twee implementaties:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddPrefixTransformer**. Hiermee wordt het voorvoegsel (opgegeven als invoer) toegevoegd bij het converteren van federated-databasewaarde naar CMDB-waarde en wordt het voorvoegsel verwijderd bij het converteren van CMDB-waarde naar federated-databasewaarde.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemovePrefixTransformer**. Hiermee wordt het voorvoegsel (opgegeven als invoer) verwijderd bij het converteren van federated-databasewaarde naar CMDB-waarde en wordt het voorvoegsel toegevoegd bij het converteren van CMDB-waarde naar federated-databasewaarde.

Converter BytesToStringTransformer

Met deze converter worden bytematrices in de CMDB geconverteerd naar de bijbehorende stringweergave in de federated-databasebron.

De converter is:
com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytes ToStringTransformer.

De convertor StringDelimitedListTransformer

Met deze convertor wordt één enkele lijst met strings getransformeerd naar een integer/string-lijst in de CMDB.

De converter is: **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl. StringDelimitedListTransformer.**

Invoegtoepassingen

De algemene database-adapter ondersteunt de volgende invoegtoepassingen:

- Een optionele invoegtoepassing voor volledige topologiesynchronisatie.
- Een optionele invoegtoepassing voor het synchroniseren van wijzigingen in topologie. Als er geen invoegtoepassing voor het synchroniseren van wijzigingen wordt geïmplementeerd, is het mogelijk een andere synchronisatie uit te voeren, maar die synchronisatie is eigenlijk een volledige synchronisatie.

- Een optionele invoegtoepassing voor het synchroniseren van indeling.
- Een optionele invoegtoepassing om ondersteunde query's voor synchronisatie op te halen. Als deze invoegtoepassing niet wordt gedefinieerd, worden alle TQL-namen geretourneerd.
- Een interne, optionele invoegtoepassing om de TQL-definitie en het TQL-resultaat te wijzigen.
- Een interne, optionele invoegtoepassing om een indelingsverzoek en CI-resultaat te wijzigen.
- Een interne, optionele invoegtoepassing om een indelingsverzoek en relatieresultaat te wijzigen.
- Een interne, optionele invoegtoepassing om de actie van Push Back-ID's te wijzigen.

Zie "Invoegtoepassingen implementeren" op pagina 105 voor meer informatie over het uitrollen van invoegtoepassingen.

Configuratievoorbeelden

In dit gedeelte vindt u voorbeelden van configuraties.

In dit gedeelte vindt u de volgende onderwerpen:

Use case

Use case. Een TQL-query is:

node > (composition) > card

waarbij:

- **node** is de CMDB-entiteit
- **card** de bronentiteit van de federated database is
- **composition** de relatie is tussen beide

Het voorbeeld wordt uitgevoerd voor de `ED`-database. `ED nodes` worden opgeslagen in de tabel `Device` en `card` wordt opgeslagen in de tabel `hwCards`. In de volgende voorbeelden wordt `card` altijd op dezelfde manier toegewezen.

Afstemming van één knooppunt

In dit voorbeeld wordt de afstemming uitgevoerd voor de eigenschap `name`.

Vereenvoudigde definitie

De afstemming vindt plaats per `knooppunt` en wordt benadrukt door de speciale tag **CMDB-class**.

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-single-node>
            <or>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
            </or>
        </reconciliation-by-single-node>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="composition">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

Geavanceerde definitie

Bestand orm.xml

Let op de toevoeging van de relatietoewijzing. Zie de definitiesectie in "Bestand orm.xml" op pagina 121 voor meer informatie.

Voorbeeld van het bestand 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>
```

Bestand reconciliation_types.txt

Zie "Bestand reconciliation_types.txt" op pagina 134 voor meer informatie over dit onderwerp.

```
node
```

Bestand reconciliation_rules.txt

Zie "Bestand reconciliation_rules.txt (voor achterwaartse compatibiliteit)" op pagina 134 voor meer informatie over dit onderwerp.

```
multinode[node] expression[node.name]
```

Bestand transformation.txt

Dit bestand blijft leeg aangezien er geen waarden hoeven te worden geconverteerd in dit voorbeeld.

Afstemming van twee knooppunten

In dit voorbeeld wordt afstemming berekend in overeenstemming met de eigenschap `name` van `node` en van `ip_address` met verschillende variaties.

De afstemmings-TQL-query is **node > (containment) > ip_address**.

Vereenvoudigde definitie

De afstemming is per `name` van `node` OR van `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

De afstemming is per `name` van `node` AND van `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>
```

De afstemming is per `name` van `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
            <or>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
```

```
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

Geavanceerde definitie

Bestand orm.xml

Aangezien de afstemmingsexpressie niet wordt gedefinieerd in dit bestand, moet dezelfde versie voor elke afstemmingsexpressie worden gebruikt.

Bestand reconciliation_types.txt

Zie "Bestand reconciliation_types.txt" op pagina 134 voor meer informatie over dit onderwerp.

```
node
```

Bestand reconciliation_rules.txt

Zie "Bestand reconciliation_rules.txt (voor achterwaartse compatibiliteit)" op pagina 134 voor meer informatie over dit onderwerp.

```
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]
```

Bestand transformation.txt

Dit bestand blijft leeg aangezien er geen waarden hoeven te worden geconverteerd in dit voorbeeld.

Primaire sleutel gebruiken die meerdere kolommen bevat

Als de primaire sleutel wordt samengesteld uit meer dan één kolom, wordt de volgende code aan de XML-definities toegevoegd:

Vereenvoudigde definitie

Er is meer dan één primaire-sleuteltag en voor elke kolom is er een tag.

```
    <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>
```

Geavanceerde definitie

Bestand orm.xml

Er wordt een nieuwe `id`-entiteit toegevoegd waarmee aan de primaire-sleutelkolommen wordt toegewezen. Entiteiten die deze `id`-entiteit gebruiken, moeten een speciale tag toevoegen.

Als u een externe sleutel gebruikt (tag join-column) voor een dergelijke primaire sleutel, moet u een toewijzing tot stand brengen tussen elke kolom in de externe sleutel en een kolom in de primaire sleutel.

Zie "Bestand orm.xml" op pagina 121 voor meer informatie over dit onderwerp.

Voorbeeld van het bestand 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>
```

Transformaties gebruiken

In het volgende voorbeeld wordt de algemene **enum**-transformer geconverteerd van waarden `1`, `2`, `3` naar respectievelijk waarden `a`, `b`, `c` in de kolom `name`.

Het toewijzingsbestand is `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>
```

Vereenvoudigde definitie

```
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address"
          CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name"
                 from-CMDB-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
                 transform.impl.GenericEnumTransformer(generic-enum-
transformer-example.
```

```
                xml)" to-CMDB-
converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
                transform.impl.GenericEnumTransformer(generic-enum-
transformer-example.
                xml)" />
               <connected-node-attribute CMDB-attribute-name="name"
                column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
```

Geavanceerde definitie

Er is alleen een wijziging in het bestand **transformation.txt**.

Bestand transformation.txt

Zorg ervoor dat de attribuutnamen en entiteitnamen hetzelfde zijn als in het bestand `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)]
```

Adapterlogboekbestanden

U kunt de volgende logboekbestanden raadplegen om te begrijpen wat de berekeningsstromen en de adapterlevenscyclus inhouden en om foutopsporingsgegevens te bekijken.

In dit gedeelte vindt u de volgende onderwerpen:

- "Logboekniveaus" beneden
- "Logboeklocaties" op volgende pagina

Logboekniveaus

U kunt het logboekniveau configureren voor elk van de logboeken.

Open in een teksteditor het bestand **C:\hp\UCMDB\UCMDBServer\conf\log\ fcmdb.gdba.properties**
.

Het standaardlogboekniveau is **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL loglevel=ERROR
```

- Als u het logboekniveau voor alle logboekbestanden wilt verhogen, wijzigt u **loglevel=ERROR** in **loglevel=DEBUG** of **loglevel=INFO**.
- Als u het logboekniveau voor een specifiek bestand wilt wijzigen, wijzigt u de specifieke **log4j**-categorieregel dienovereenkomstig. Als u bijvoorbeeld het logboekniveau van fcmdb.gdba.dal.sql.log wilt wijzigen in **INFO**, wijzigt u:

```
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
```

Logboeklocaties

De logboekbestanden bevinden zich in de map **C:\hp\UCMDB\UCMDBServer\runtime\log**.

- **Fcmdb.gdba.log**

  Het logboek voor de adapterlevenscyclus. Geeft informatie over het moment wanneer de adapter wordt gestart of gestopt en welke CIT's door deze adapter worden ondersteund.

  Raadpleeg dit logboek voor initiatiefouten (adapter geladen/ontladen).

- **fcmdb.log**

  Raadpleeg dit logboek voor uitzonderingen.

- **cmdb.log**

  Raadpleeg dit logboek voor uitzonderingen.

- **Fcmdb.gdba.mapping.engine.log**

  Het logboek van de toewijzingsengine. Geeft informatie over de afstemmings-TQL-query die de toewijzingsengine gebruikt, en de afstemmingstopologieën die tijdens de verbindingsfase worden vergeleken.

  Raadpleeg dit logboek wanneer een TQL-query geen resultaten geeft, terwijl u weet dat de database relevante CI's bevat, of als de resultaten onverwacht zijn (controleer de afstemming).

- **Fcmdb.gdba.TQL.log**

  Het TQL-logboek. Geeft informatie over de TQL-query's en de resultaten ervan.

  Raadpleeg dit logboek wanneer een TQL-query geen resultaten retourneert en het logboek van de toewijzingsengine aangeeft dat de federated-databasebron geen resultaten bevat.

- **Fcmdb.gdba.dal.log**

  Het logboek van de DAL-levenscyclus. Geeft informatie over het genereren van CIT's en databaseverbindingsdetails.

  Raadpleeg dit logboek wanneer u geen verbinding kunt maken met de database of wanneer er CIT's of attributen zijn die niet door de query worden ondersteund.

- **Fcmdb.gdba.dal.command.log**

  Het logboek van DAL-bewerkingen. Geeft informatie over interne DAL-bewerkingen die worden aangeroepen. (Dit logboek is vergelijkbaar met `cmdb.dal.command.log`).

- **Fcmdb.gdba.dal.SQL.log**

  Het logboek van DAL SQL-query's. Geeft informatie over aangeroepen JPAQL's (objectgeoriënteerde SQL-query's) en de resultaten ervan.

  Raadpleeg dit logboek wanneer u geen verbinding kunt maken met de database of wanneer er CIT's of attributen zijn die niet door de query worden ondersteund.

- **Fcmdb.gdba.hibrnate.log**

  Het Hibernate-logboek. Geeft informatie over de SQL-query's die worden uitgevoerd, de parsering van elke JPAQL naar SQL, de resultaten van de query's, gegevens over Hibernate-caching, enzovoort. Zie "Hibernate als JPA-provider" op pagina 86 voor meer informatie over Hibernate.

Externe referenties

Zie http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html. voor meer informatie over de JavaBeans 3.0-specificatie.

Probleemoplossing en beperkingen

In dit gedeelte worden probleemoplossingen voor en beperkingen van de algemene database-adapter beschreven.

Algemene beperkingen

- SQL Server NTLM-verificatie wordt niet ondersteund.
- Gebruik Notepad++, UltraEdit of een andere teksteditor van derden voor het bewerken van de sjabloonbestanden bij het bijwerken van een adapterpakket, in plaats van Kladblok (elke versie) van Microsoft Corporation. Hiermee wordt het gebruik voorkomen van speciale symbolen die ertoe kunnen leiden dat de uitrol van het pakket mislukt.

JPA-beperkingen

- Alle tabellen moeten een primaire-sleutelkolom hebben.
- Attribuutnamen van de CMDB-klasse moeten de naamgevingsconventie van JavaBeans volgen (namen moeten bijvoorbeeld beginnen met een kleine letter).
- Twee CI's die met één relatie in het klassemodel worden verbonden, moeten een rechtstreekse koppeling in de database hebben (als `node` bijvoorbeeld is verbonden met `ticket`, moet er een externe sleutel of koppelingstabel zijn waarmee ze worden verbonden).
- Verschillende tabellen die aan hetzelfde CIT worden toegewezen, moeten dezelfde primaire-sleuteltabel hebben.

Functionele beperkingen

- U kunt een handmatige relatie aanmaken tussen de CMDB en federated CIT's. Om virtuele relaties te kunnen definiëren moet een speciale relatielogica worden gedefinieerd (deze kan worden gebaseerd op eigenschappen van de federated klasse).
- Federated CIT's kunnen geen trigger-CIT's in een impactregel zijn, maar ze kunnen in een TQL-query van een impactanalyse worden opgenomen.
- Een federated CIT kan deel uitmaken van een enrichment-TQL, maar kan niet worden gebruikt als het knooppunt waarop enrichment wordt uitgevoerd (u kunt het federated CIT niet toevoegen, bijwerken of verwijderen).
- Gebruik van een klassekwalificator in een voorwaarde wordt niet ondersteund.
- Subgrafieken worden niet ondersteund.

- Compound-relaties worden niet ondersteund
- De CMDB-`ID` van het externe CI wordt samengesteld uit de bijbehorende primaire sleutel, en niet uit de bijbehorende sleutelattributen.
- Een kolom van het type `bytes` kan niet als een primaire-sleutelkolom in Microsoft SQL Server worden gebruikt.
- Berekening van TQL-query's mislukt als de namen van attribuutvoorwaarden die voor een federated knooppunt zijn gedefinieerd, niet zijn toegewezen in het bestand **orm.xml**.
- De algemene DB-Adapter ondersteunt geen Windows-verificatie voor SQL Server.

Hoofdstuk 5

Java-adapters ontwikkelen

In dit hoofdstuk vindt u de volgende informatie:

Overzicht Federation Framework

Opmerking:

- De term **relatie** is gelijk aan de term **koppeling**.
- De term **CI** is gelijk aan de term **object**.
- Een grafiek is een verzameling knooppunten en koppelingen.

Het Federation Framework gebruikt een API voor het ophalen van informatie uit federated bronnen. Het Federation Framework maakt drie belangrijke functies mogelijk:

- **Federation** in real-time. Alle query's worden uitgevoerd in originele gegevensopslaglocaties en de resultaten worden in real-time ingevoerd in de CMDB.
- **Vulling.** Voorziet de CMDB van gegevens (topologische gegevens en CI-eigenschappen) vanuit een externe gegevensbron.
- **Datapush.** Verstuurt gegevens (topologische gegevens en CI-eigenschappen) vanuit de lokale CMDB naar een externe gegevensbron.

Alle actietypen vereisen een adapter voor elke gegevensopslaglocatie, die zorgt voor de specifieke mogelijkheden van de opslaglocatie en het ophalen en/of bijwerken van de vereiste gegevens. Elk verzoek aan de gegevensopslaglocatie wordt via de adapter verwerkt.

In dit gedeelte vindt u ook de volgende onderwerpen:

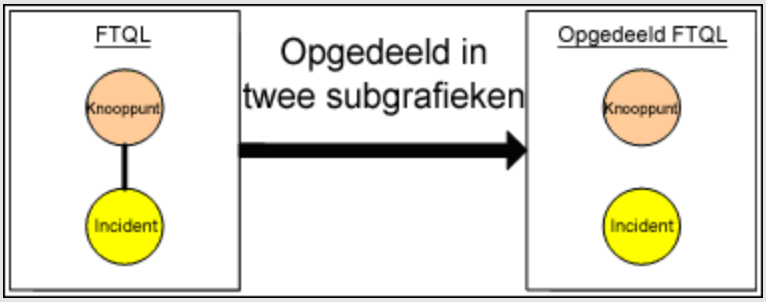
Federation in real-time

Met federated TQL-query's kunt u gegevens ophalen uit een externe gevensopslaglocatie zonder dat de gegevens worden gerepliceerd.

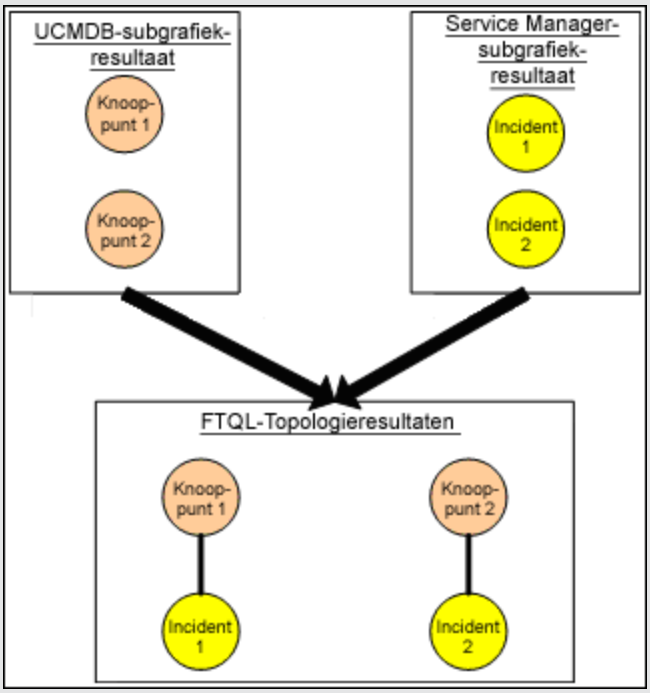
Een federated TQL-query gebruikt adapters die externe gevensopslaglocaties voorstellen. Zo kunt u de juiste externe verbanden leggen tussen CI's uit verschillende externe gevensopslaglocaties en de UCMDB-CI's.

Voorbeeld van federation in real-time:

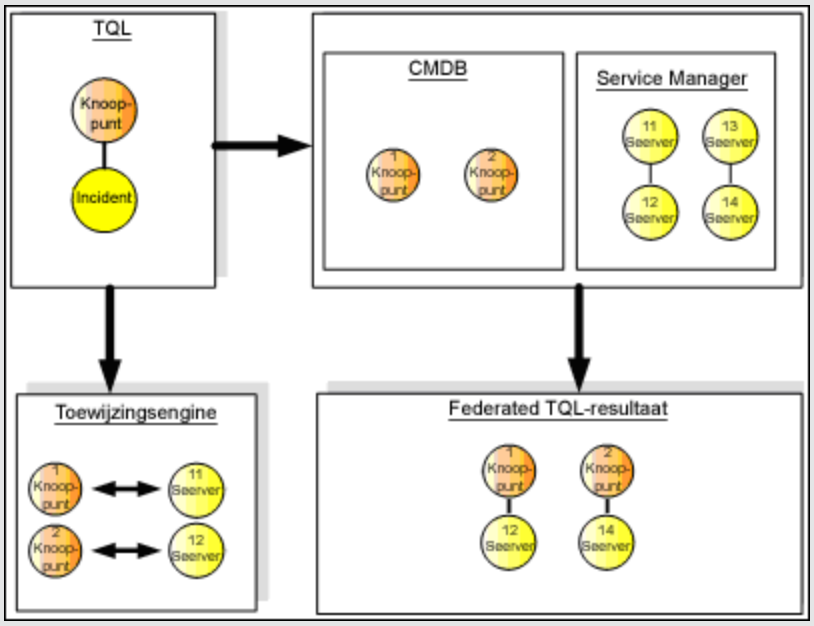
1. Het Federation Framework verdeelt een federated TQL-query in diverse subgrafieken, waarbij alle knooppunten in een subgrafiek verwijzen naar dezelfde gegevensopslaglocatie. Elke subgrafiek staat in verbinding met de andere subgrafieken via een virtuele relatie (maar zelf bevat het geen virtuele relaties).



2. Nadat de federated TQL-query is opgedeeld in subgrafieken, berekent het Federation Framework de topologie van elke subgrafiek en verbindt het twee geschikte subgrafieken door virtuele relaties te creëren tussen de betreffende knooppunten.



3. Nadat de federated TQL-topologie is berekend, haalt het Federation Framework een indeling op voor het topologieresultaat.



Datapush

U gebruikt de datapush-stroom om uw huidige lokale CMDB te synchroniseren met een service op afstand of een gegevensopslaglocatie.

Voor datapush worden gegevensopslaglocaties verdeeld in twee categorieën: bron (lokale CMDB) en doel. Gegevens worden opgehaald uit de bronopslaglocatie en overgebracht naar de

doelopslaglocatie. Het datapush-proces is gebaseerd op query-namen, hetgeen betekent dat de gegevens worden gesynchroniseerd tussen de bron (lokale CMDB) en de doelopslaglocaties, om vervolgens te worden opgehaald op basis van de naam van een TQL-query uit de lokale CMDB.

De datapush-processtroom bestaat uit de volgende stappen:

1. Ophalen van de topologieresultaten met handtekeningen uit de bronopslaglocatie.
2. Vergelijken van de nieuwe resultaten met de vorige resultaten.
3. Ophalen van een volledige indeling (dus alle CI-eigenschappen) van CI's en relaties, alleen voor de gewijzigde resultaten.
4. Bijwerken van de doelopslaglocatie met de ontvangen volledige indeling van CI's en relaties. Als er CI's of relaties zijn verwijderd in de bronopslaglocatie en de query is exclusief, verwijdert het replicatieproces ook de CI's of relaties in de doelopslaglocatie.

De CMDB heeft twee verborgen gegevensbronnen (**hiddenRMIDataSource** en **hiddenChangesDataSource**), die altijd de 'bron-gegevensbron' zijn in datapush-stromen. Om een nieuwe adapter voor datapush-stromen te implementeren, hoeft u alleen de 'doel'-adapter te implementeren.

Vulling

U gebruikt de vullingstroom om de CMDB te vullen met gegevens uit externe bronnen.

De stroom gebruikt altijd één 'bron-gegevensbron' om de gegevens op te halen en pusht de opgehaalde gegevens naar de probe in een proces dat vergelijkbaar is met de stroom van een discovery-taak.

Om een nieuwe adapter voor een vullingstroom te implementeren hoeft u alleen de bron-adapter te implementeren, omdat de Data Flow-probe als doel fungeert.

De adapter in de vullingstroom wordt uitgevoerd in de probe. Foutopsporing en vastlegging moet plaatsvinden in de probe en niet in de CMDB.

De vullingstroom is gebaseerd op query-namen, hetgeen wil zeggen dat gegevens worden gesynchroniseerd tussen de bronopslaglocatie en de Data Flow-probe en worden opgehaald op basis van een query-naam in de bronopslaglocatie. In CMDB bijvoorbeeld is de query-naam de naam van de TQL-query. In een andere gegevensopslaglocatie kan de query-naam echter ook een codenaam zijn die gegevens retourneert. De adapter is ontworpen voor een correcte behandeling van de query-naam.

Elke taak kan worden gedefinieerd als een exclusieve taak. Dit betekent dat de CI's en de relaties in de taakresultaten uniek zijn in de lokale CMDB en dat geen andere query naar het doel leidt. De adapter van de bronopslaglocatie ondersteunt specifieke query's en kan gegevens ophalen uit deze gegevensopslaglocatie. Met de adapter van de doelopslaglocatie kunnen de opgehaalde gegevens in deze gegevensopslaglocatie worden bijgewerkt.

SourceDataAdapter-stroom

- Haalt de topologieresultaten met handtekeningen op uit de bronopslaglocatie.
- Vergelijkt de nieuwe resultaten met de vorige resultaten.
- Haalt een volledige indeling op (dus alle CI-eigenschappen) van CI's en relaties, alleen voor de

gewijzigde resultaten.

- Werkt de doelopslaglocatie bij met de ontvangen volledige indeling van CI's en relaties. Als er CI's of relaties zijn verwijderd in de bronopslaglocatie en de query is exclusief, verwijdert het replicatieproces ook de CI's of relaties in de doelopslaglocatie.

SourceChangesDataAdapter-stroom

- Haalt de topologieresultaten op sinds de laatste opgegeven datum.
- Haalt een volledige indeling op (dus alle CI-eigenschappen) van CI's en relaties, alleen voor de gewijzigde resultaten.
- Werkt de doelopslaglocatie bij met de ontvangen volledige indeling van CI's en relaties. Als er CI's of relaties zijn verwijderd in de bronopslaglocatie en de query is exclusief, verwijdert het replicatieproces ook de CI's of relaties in de doelopslaglocatie.

PopulateDataAdapter-stroom

- Haalt de volledige topologie op met het aangevraagde indelingsresultaat.
- Gebruikt het segmentmechanisme van de topologie om gegevens in segmenten op te halen.
- De probe filtert alle gegevens eruit die al eerder waren opgehaald.
- Werkt de doelopslaglocatie bij met de ontvangen indeling van CI's en relaties. Als er CI's of relaties zijn verwijderd in de bronopslaglocatie en de query is exclusief, verwijdert het replicatieproces ook de CI's of relaties in de doelopslaglocatie.

PopulateChangesDataAdapter-stroom

- Haalt de topologie op met het aangevraagde indelingsresultaat dat is veranderd sinds de laatste uitvoering.
- Gebruikt het segmentmechanisme van de topologie om gegevens in segmenten op te halen.
- De probe filtert alle gegevens eruit die al eerder waren opgehaald (inclusief deze stroom).
- Werkt de doelopslaglocatie bij met de ontvangen indeling van CI's en relaties. Als er CI's of relaties zijn verwijderd in de bronopslaglocatie en de query is exclusief, verwijdert het replicatieproces ook de CI's of relaties in de doelopslaglocatie.

Interactie adapter en toewijzing met het Federation Framework

Een adapter is een entiteit in UCMDB die de externe gegevens vertegenwoordigt (gegevens die niet zijn opgeslagen in UCMDB). In federated stromen worden alle interacties met externe gegevensbronnen uitgevoerd via adapters. Voor replicaties en federated TQL-query's worden verschillende interactiestromen en de adapterinterfaces van het Federation Framework gebruikt.

In dit gedeelte vindt u ook de volgende onderwerpen:

Levenscyclus adapter

Voor elke externe gegevensopslaglocatie wordt er een aparte adapter aangemaakt. De levenscyclus van de adapter begint met de eerste actie die erop wordt toegepast (zoals `TQL berekenen` of `gegevens ophalen/bijwerken`). Wanneer de **start**methode wordt aangeroepen, ontvangt de adapter informatie over de omgeving, zoals de configuratie van de gegevensopslaglocatie, de logger, etc. De levenscyclus van de adapter eindigt als de gegevensopslaglocatie wordt verwijderd uit de configuratie en de **afsluit**methode wordt aangeroepen. Dit betekent dat de adapter stateful is en indien nodig de verbinding met de externe gegevensopslaglocatie kan bevatten.

Hulpmethoden adapter

De adapter heeft diverse `hulp`methoden voor het toevoegen van configuraties van externe gegevensopslaglocaties. Deze methoden zijn geen onderdeel van de levenscyclus van de adapter en maken bij elke aanroep een nieuwe adapter aan.

- De eerste methode test de verbinding met de externe gegevensopslaglocatie voor een bepaalde configuratie. testConnection kan op de UCMDB-server worden uitgevoerd of op de Data Flow-probe, afhankelijk van het type adapter.
- De tweede methode is alleen van belang voor de bronadapter en retourneert de ondersteunde query's voor replicatie (deze methode wordt alleen op de probe uitgevoerd).
- De derde methode is alleen van belang voor federation- en vullingstromen en retourneert ondersteunde externe klassen via de externe gegevensopslaglocatie (deze methode wordt alleen op de UCMDB-server uitgevoerd).

Al deze methoden worden gebruikt als u integratieconfiguraties aanmaakt of bekijkt.

Federation Framework voor federated TQL-query's

In dit gedeelte vindt u de volgende onderwerpen:

Zie "Interacties tussen Federation Framework, server, adapter en toewijzingsengine" op volgende pagina voor diagrammen die de interactie tussen Federation Framework, UCMDB, adapter en toewijzingsengine illustreren.

Definities en termen

Afstemmingsgegevens. De regel voor het koppelen van CI's van het opgegeven type die zijn ontvangen uit de CMDB en de externe gegevensopslaglocatie. Er zijn drie typen afstemmingsregels:

- **ID-afstemming.** Dit kan alleen worden gebruikt als de externe gegevensopslaglocatie de CMDB-ID bevat van afstemmingsobjecten.
- **Eigenschapafstemming.** Dit wordt gebruikt als het koppelen alleen kan plaatsvinden aan de

hand van eigenschappen van het afstemmings-CI-type.

- **Topologie-afstemming.** Dit wordt gebruikt als de eigenschappen van extra CIT's (dus niet alleen van het afstemmings-CIT) nodig zijn voor het uitvoeren van een koppeling op afstemmings-CI's. U kunt bijvoorbeeld een afstemming van het knooppunttype uitvoeren via de eigenschap `naam` die behoort bij het `ip_address`-CIT.

Afstemmingsobject. Dit object wordt aangemaakt door de adapter volgens de ontvangen afstemmingsgegevens. Dit object moet verwijzen naar een extern CI en wordt gebruikt door de toewijzingsengine om een verbinding tot stand te brengen tussen de externe CI's en de CMDB-CI's.

Afstemmings-CI-type. Het type CI dat de afstemmingsobjecten vertegenwoordigt. Deze CI's moeten zowel in de CMDB als in de externe gegevensopslaglocaties worden opgeslagen.

Toewijzingsengine. Een component voor de identificatie van relaties tussen CI's uit verschillende gegevensopslaglocaties die een virtuele relatie met elkaar hebben. De identificatie wordt uitgevoerd door het afstemmen van CMDB-afstemmingsobjecten en externe CI-afstemmingsobjecten.

Toewijzingsengine

Federation Framework gebruikt de toewijzingsengine om de federated TQL-query te berekenen. De toewijzingsengine verbindt de CI's die zijn ontvangen uit verschillende gegevensopslaglocaties en zijn verbonden via virtuele relaties. De toewijzingsengine levert ook afstemmingsgegevens voor de virtuele relatie. Het ene einde van de virtuele relatie moet verwijzen naar de CMDB. Dit einde is een `afstemmings`type. Voor de berekening van de twee subgrafieken kan een virtuele relatie aan elk eindknooppunt beginnen.

Federated adapter

De federated adapter haalt twee soorten gegevens uit externe gegevensopslaglocaties: externe CI-gegevens en afstemmingsobjecten die tot externe CI's behoren.

- **Externe CI-gegevens.** De externe gegevens die niet aanwezig zijn in de CMDB. Dit zijn de doelgegevens van de externe gegevensopslaglocatie.
- **Gegevens afstemmingsobjecten.** De hulpgegevens die worden gebruikt door het Federation Framework om verbinding te maken tussen CMDB-CI's en externe gegevens. Elk afstemmingsobject moet verwijzen naar een extern CI. Het type afstemming is het type (of subtype) van een van de virtuele relatie-einden waaruit de gegevens worden opgehaald. Afstemmingsobjecten moeten aansluiten bij de ontvangen adapter om gegevens af te stemmen. Er zijn drie typen afstemmingsobjecten: `IdReconciliationObject`, `PropertyReconciliationObject` en `TopologyReconciliationObject`.

In de DataAdapter-gebaseerde interfaces (DataAdapter, PopulateDataAdapter en PopulateChangesDataAdapter) is de afstemming vereist als onderdeel van de query-definitie.

Interacties tussen Federation Framework, server, adapter en toewijzingsengine

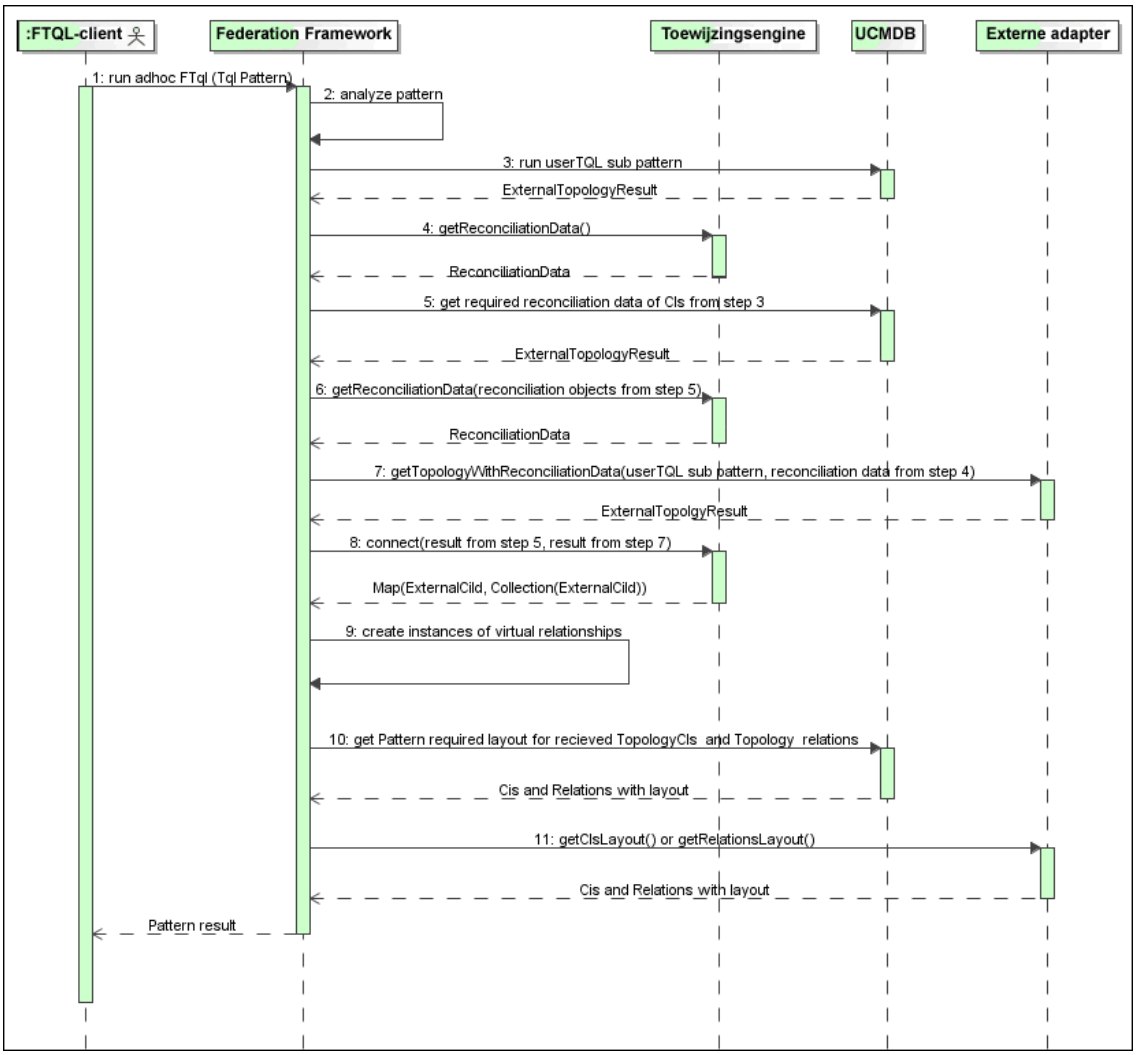
De volgende diagrammen illustreren de interacties tussen het Federation Framework, UCMDB-server, de adapter en de toewijzingsengine. De federated TQL-query in de voorbeelddiagrammen heeft slechts één virtuele relatie, zodat alleen UCMDB en één externe gegevensopslaglocatie zijn betrokken bij de federated TQL-query.

In dit gedeelte vindt u de volgende onderwerpen:

In het eerste diagram begint de berekening in de UCMDB en in het tweede diagram in de externe adapter. Elke stap in het diagram bevat referenties naar de juiste methodeaanroep van de adapter- of toewijzingsengine-interface.

De berekening start bij het servereinde

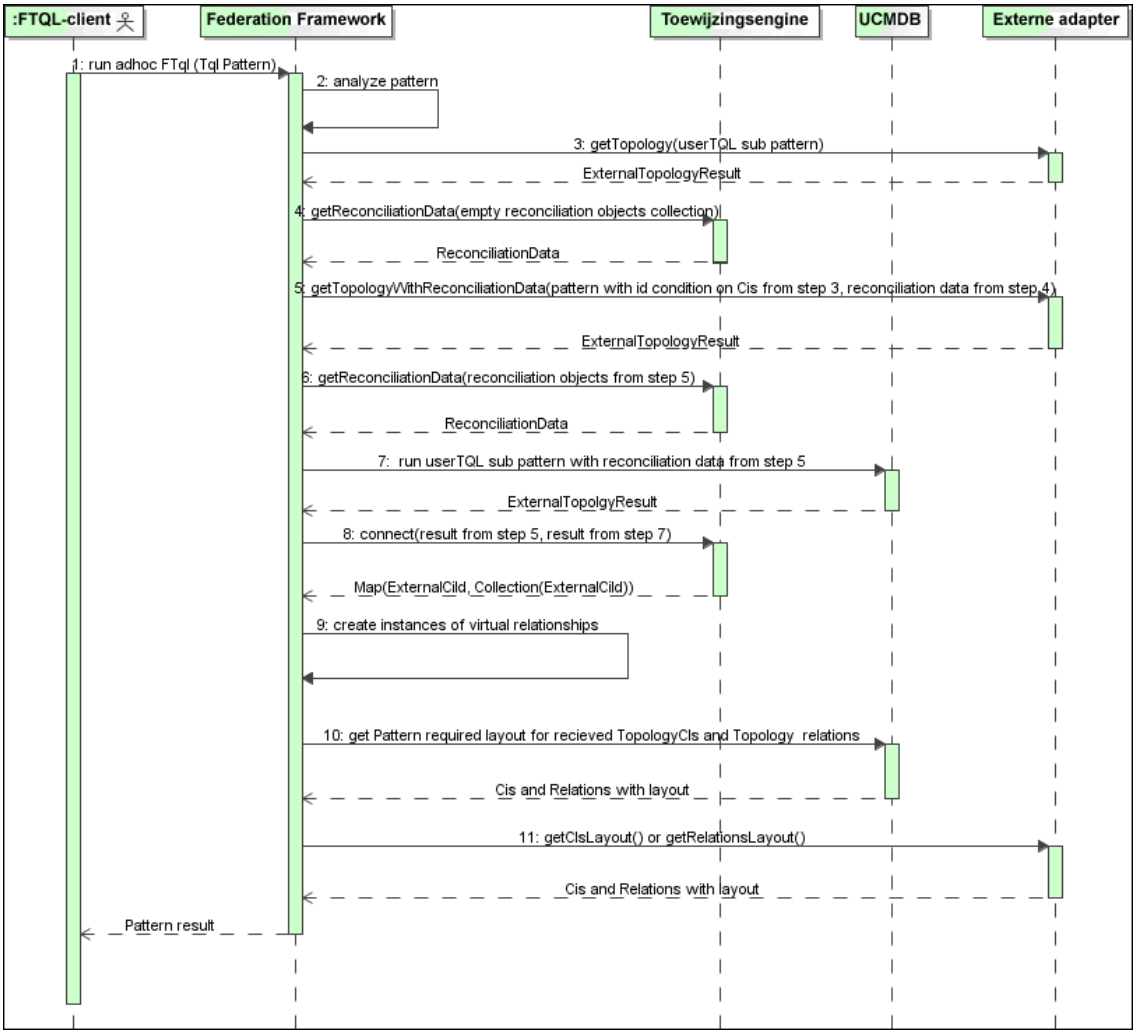
Het volgende stroomdiagram illustreert de interactie tussen het Federation Framework, UCMDB, de adapter en de toewijzingsengine. De federated TQL-query in het voorbeelddiagram heeft slechts één virtuele relatie, zodat alleen UCMDB en één externe gegevensopslaglocatie zijn betrokken bij de federated TQL-query.



De cijfers in deze afbeelding worden hieronder verklaard:

Cijfer	Uitleg
1	Het Federation Framework ontvangt een aanroep voor een federated TQL-berekening.
2	Het Federation Framework analyseert de adapter, vindt de virtuele relatie en verdeelt de oorspronkelijke TQL in twee subadapters: een voor UCMDB en een voor de externe gegevensopslaglocatie.
3	Het Federation Framework verzoekt om de topologie van de sub-TQL in UCMDB.
4	Na ontvangst van de topologieresultaten roept het Federation Framework de juiste toewijzingsengine aan voor de huidige virtuele relatie en verzoekt het om afstemmingsgegevens. De parameter `reconciliationObject` is op dat moment nog leeg, omdat er nog geen voorwaarde is toegevoegd aan afstemmingsgegevens in deze aanroep. De geretourneerde afstemmingsgegevens bepalen welke gegevens nodig zijn om de afstemming-CI's in UCMDB te koppelen aan de externe gegevensopslaglocatie. Er zijn drie typen afstemmingsgegevens: • **IdReconciliationData.** CI's worden afgestemd volgens hun ID. • **PropertyReconciliationData.** CI's worden afgestemd volgens de eigenschappen van een van de CI's. • **TopologyReconciliationData.** CI's worden afgestemd volgens de topologie (voor het afstemmen van knooppunt-CI's is bijvoorbeeld ook het IP-adres van **IP** nodig).
5	Het Federation Framework verzoekt om afstemmingsgegevens voor de CI's van de virtuele relatie-einden die in stap "3" boven zijn ontvangen van UCMDB.
6	Het Federation Framework roept de toewijzingsengine aan om de afstemmingsgegevens op te halen. In deze staat (in tegenstelling tot stap "3" boven) ontvangt de toewijzingsengine de afstemmingsobjecten uit stap "5" boven als parameters. De toewijzingsengine vertaalt het ontvangen afstemmingsobject volgens de voorwaarde van de afstemmingsgegevens.
7	Het Federation Framework verzoekt om de topologie van de sub-TQL uit de externe gegevensopslaglocatie. De externe adapter ontvangt de afstemmingsgegevens uit stap"6" boven als een parameter.
8	Het Federation Framework roept de toewijzingsengine aan om de ontvangen resultaten met elkaar te verbinden. De parameter `firstResult` is het externe topologieresultaat dat werd ontvangen van UCMDB in stap "5" boven en de parameter `secondResult` is het externe topologieresultaat dat werd ontvangen van de externe adapter in stap "7" boven. De toewijzingsengine retourneert een toewijzing waarbij de externe CI-ID van de eerste gegevensopslaglocatie (UCMDB in dit geval) is toegewezen aan de externe CI-ID's uit de tweede (externe) gegevensopslaglocatie.
9	Voor elke toewijzing maakt het Federation Framework een virtuele relatie aan.
10	Na de berekening van de federated TQL-queryresultaten (alleen in de topologiestatus) haalt het Federation Framework de oorspronkelijke TQL-indeling voor de resulterende CI's en relaties op uit de juiste gegevensopslaglocaties.

De berekening start bij het einde van de externe adapter



De cijfers in deze afbeelding worden hieronder verklaard:

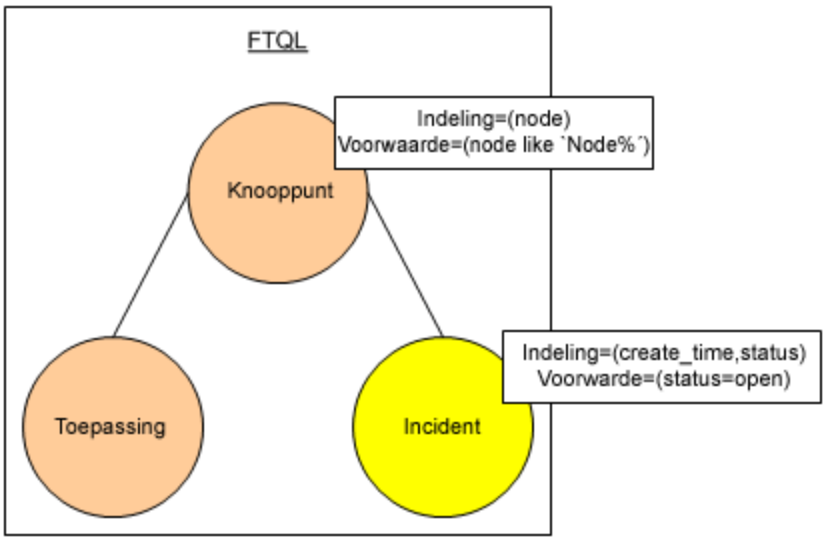
Cijfer	Uitleg
1	Het Federation Framework ontvangt een aanroep voor een federated TQL-berekening.
2	Het Federation Framework analyseert de adapter, vindt de virtuele relatie en verdeelt de oorspronkelijke TQL in twee subadapters: een voor UCMDB en een voor de externe gegevensopslaglocatie.
3	Het Federation Framework verzoekt om de topologie van de sub-TQL uit de externe gegevensadapter. Het geretourneerde `ExternalTopologyResult` mag geen afstemmingsobject bevatten, omdat de afstemmingsgegevens geen onderdeel zijn van het verzoek.
4	Na ontvangst van de topologieresultaten roept het Federation Framework de juiste toewijzingsengine aan met de huidige virtuele relatie en verzoekt het om

Cijfer	Uitleg
	afstemmingsgegevens. De parameter `reconciliationObjects` is op dat moment nog leeg, omdat er nog geen voorwaarde is toegevoegd aan afstemmingsgegevens in deze aanroep. De geretourneerde afstemmingsgegevens bepalen welke gegevens nodig zijn om de afstemming-CI's in UCMDB te koppelen aan de externe gegevensopslaglocatie. Er zijn drie typen afstemmingsgegevens: • **IdReconciliationData.** CI's worden afgestemd volgens hun ID. • **PropertyReconciliationData.** CI's worden afgestemd volgens de eigenschappen van een van de CI's. • **TopologyReconciliationData.** CI's worden afgestemd volgens de topologie (voor het afstemmen van knooppunt-CI's is bijvoorbeeld ook het IP-adres van **IP** nodig).
5	Het Federation Framework verzoekt om afstemmingsobjecten voor de CI's die waren ontvangen in stap 3 uit de externe gegevensopslaglocatie. Het Federation Framework roept de methode **getTopologyWithReconciliationData()** in de externe adapter aan, waarbij de aangevraagde topologie één knooppunt heeft met de in stap 3 ontvangen CI's en de ID-status en de afstemmingsgegevens uit stap 4.
6	Het Federation Framework roept de toewijzingsengine aan om de afstemmingsgegevens op te halen. In deze staat (in tegenstelling tot stap 3) ontvangt de toewijzingsengine de afstemmingsobjecten uit stap 5 als parameters. De toewijzingsengine vertaalt het ontvangen afstemmingsobject volgens de voorwaarde van de afstemmingsgegevens.
7	Het Federation Framework verzoekt om de topologie van de sub-TQL met afstemmingsgegevens uit stap 6 van UCMDB.
8	Het Federation Framework roept de toewijzingsengine aan om de ontvangen resultaten met elkaar te verbinden. De parameter `firstResult` is het externe topologieresultaat dat werd ontvangen van de externe adapter in stap 5 en de parameter `secondResult` is het externe topologieresultaat dat werd ontvangen van UCMDB in stap 7. De toewijzingsengine retourneert een toewijzing waarbij de externe CI-ID van de eerste gegevensopslaglocatie (de externe gegevensopslaglocatie in dit geval) is toegewezen aan de externe CI-ID's uit de tweede gegevensopslaglocatie (UCMDB).
9	Voor elke toewijzing maakt het Federation Framework een virtuele relatie aan.
10	Na de berekening van de federated TQL-queryresultaten (alleen in de topologiestatus) haalt het Federation Framework de oorspronkelijke TQL-indeling voor de resulterende CI's en relaties op uit de juiste gegevensopslaglocaties.

Voorbeeld van Federation Framework-stroom voor federated TQL-query's

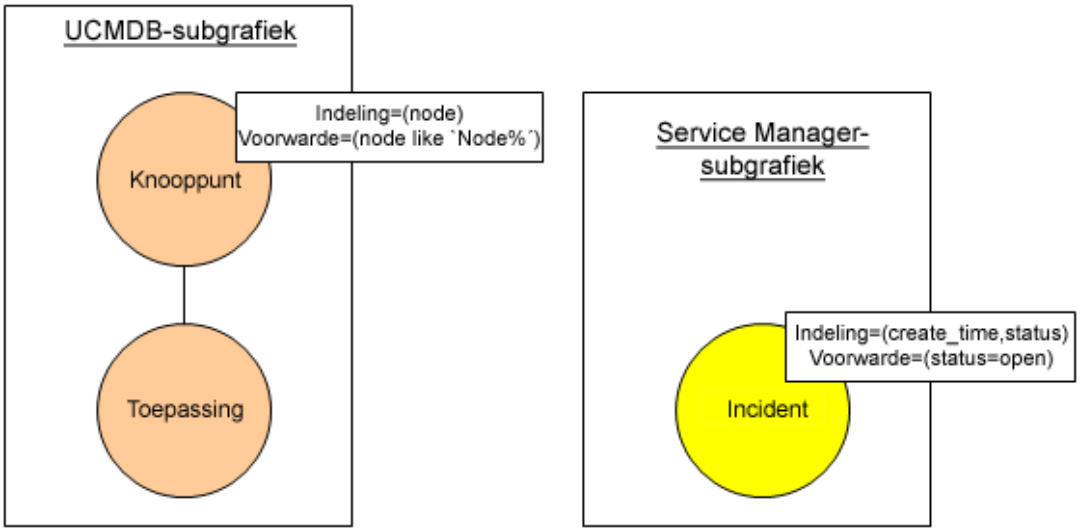
Dit voorbeeld illustreert hoe u alle open incidenten op specifieke knooppunten weergeeft. De gegevensopslaglocatie ServiceCenter is de externe gegevensopslaglocatie. De exemplaren van het knooppunt worden opgeslagen in UCMDB en de exemplaren van de incidenten worden opgeslagen in ServiceCenter. Aangenomen wordt dat de `knooppunt`- en `ip_address`-eigenschappen van de host en IP nodig zijn om de exemplaren van de incidenten te verbinden met

het juiste knooppunt. Dit zijn afstemmingseigenschappen die de knooppunten uit ServiceCenter identificeren in UCMDB.

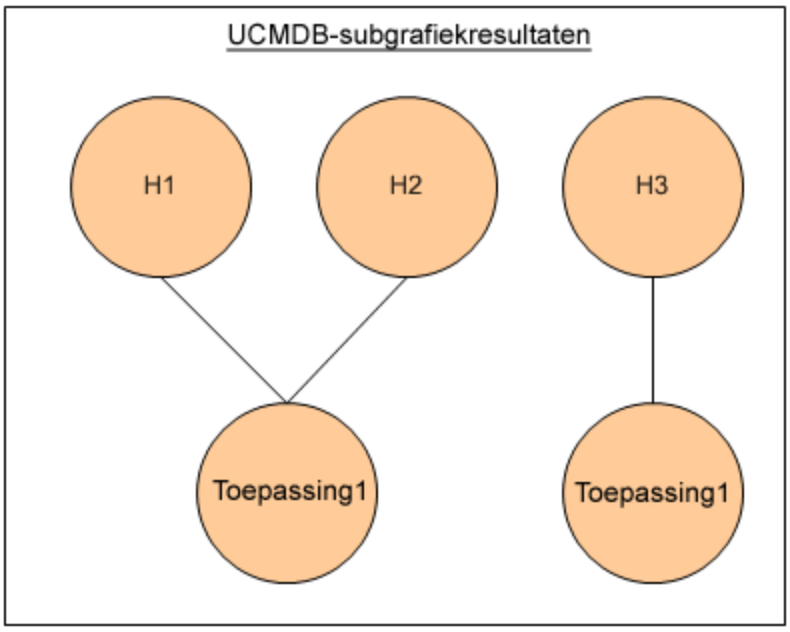


Opmerking: Voor attribuut-federation wordt de methode **getTopology** van de adapter aangeroepen. De afstemmingsgegevens worden aangepast in de gebruiker-TQL (in dit geval het CI-element).

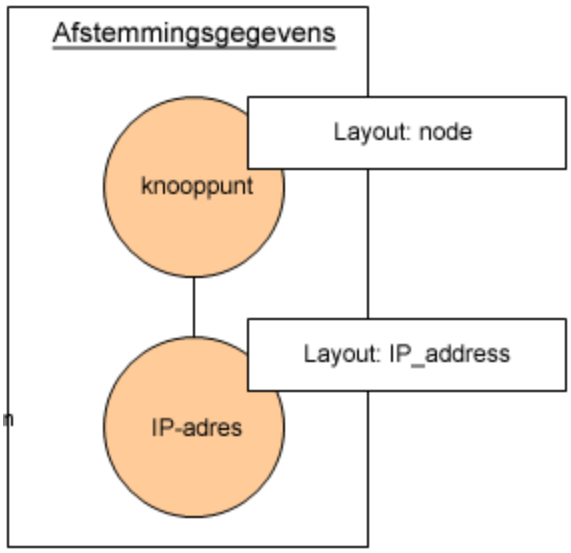
1. Na analyse van de adapter herkent het Federation Framework de virtuele relatie tussen `knooppunt` en `incident` en splitst het de federated TQL-query op in twee subgrafieken:



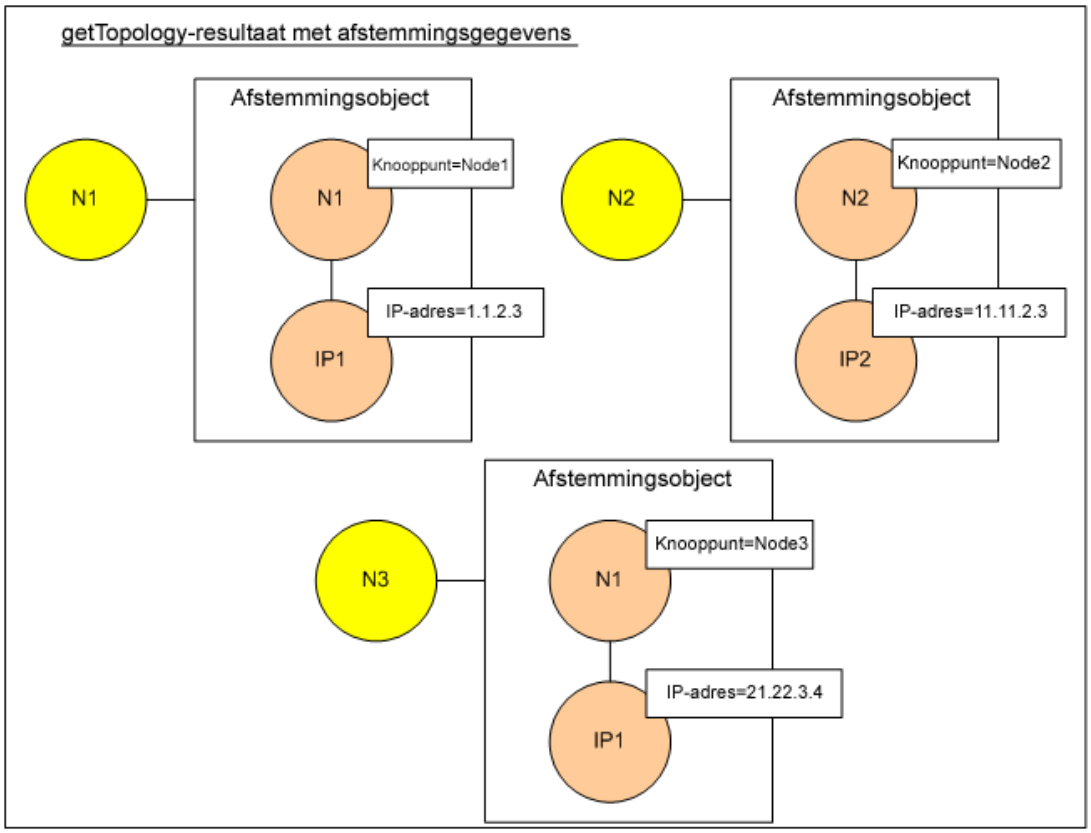
2. Het Federation Framework voert de UCMDB-subgrafiek uit om de topologie aan te vragen en ontvangt de volgende resultaten:



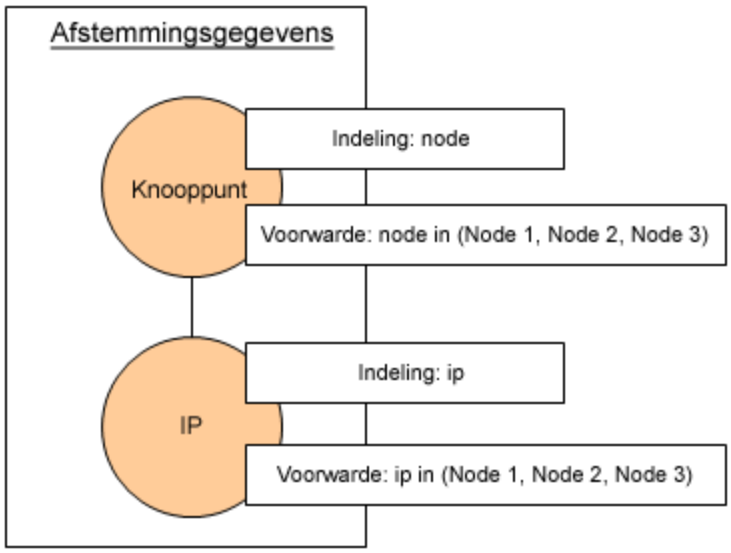
3. Het Federation Framework vraagt de juiste toewijzingsengine om de afstemmingsgegevens voor de eerste gegevensopslaglocatie (UCMDB), die de informatie bevatten die nodig zijn om de ontvangen gegevens uit twee gegevensopslaglocaties met elkaar te verbinden. In dit geval zijn de afstemmingsgegevens:



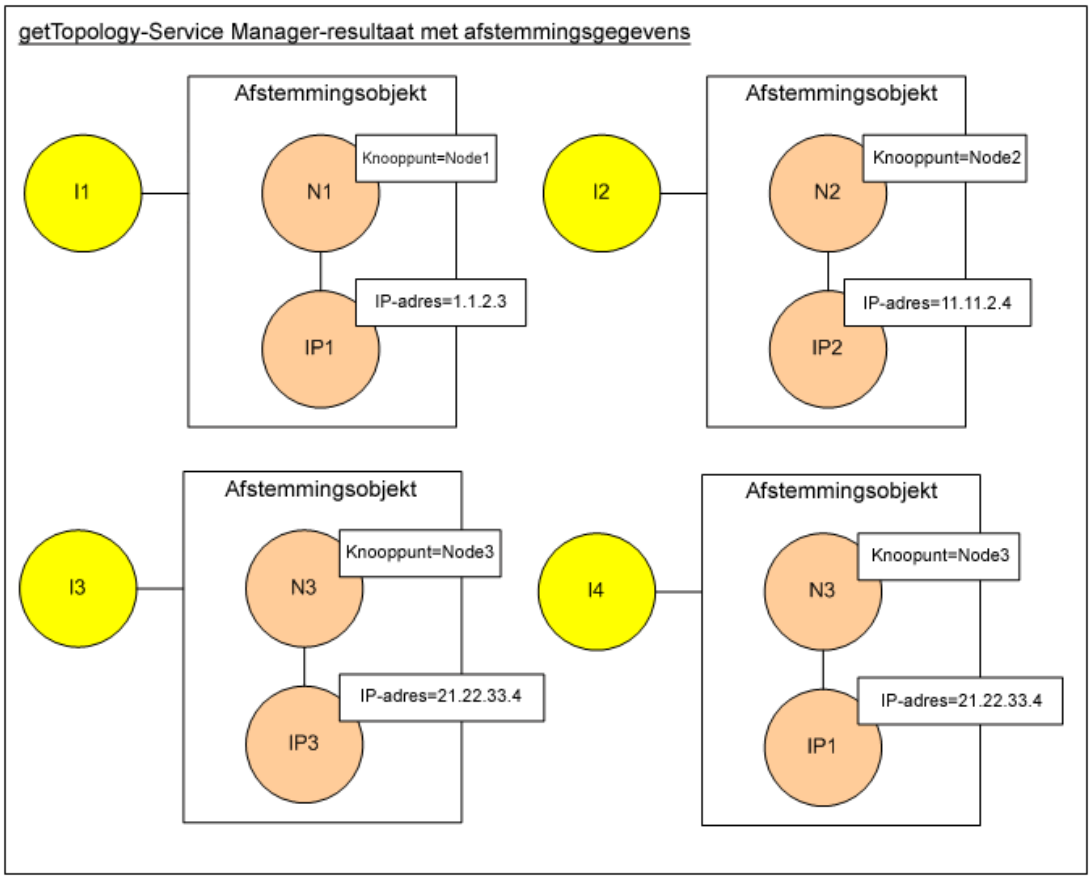
4. Het Federation Framework maakt een topologie-query aan met de knooppunt- en ID-voorwaarden uit het vorige resultaat (`knooppunt` in H1, H2, H3) en voert deze query uit met de vereiste afstemmingsgegevens in UCMDB. Het resultaat bevat knooppunt-CI's die van belang zijn voor de ID-voorwaarde en het betreffende afstemmingsobject voor elk CI:



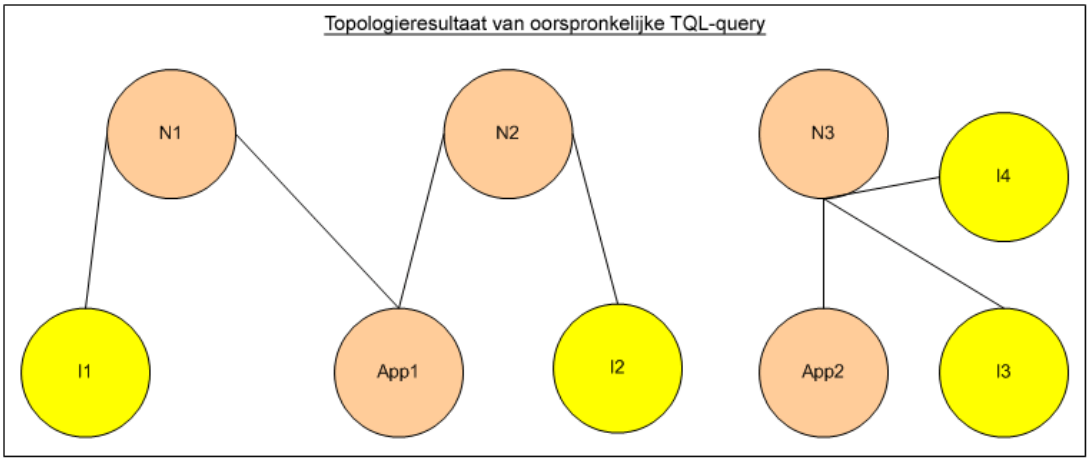
5. De afstemmingsgegevens voor ServiceCenter moeten een voorwaarde voor `knooppunt` en `ip` bevatten die is afgeleid van de afstemmingsobjecten die zijn ontvangen van UCMDB:



6. Het Federation Framework voert de subgrafiek van ServiceCenter uit met de afstemmingsgegevens om de topologie en de betreffende afstemmingsobjecten aan te vragen, en ontvangt de volgende resultaten:



7. Het resultaat na verbinding in de toewijzingsengine en het aanmaken van een virtuele relatie is:



8. Het Federation Framework vraagt om de oorspronkelijke TQL-indeling voor ontvangen exemplaren uit UCMDB en ServiceCenter.

Federation Framework-stroom voor vulling

In dit gedeelte vindt u de volgende onderwerpen:

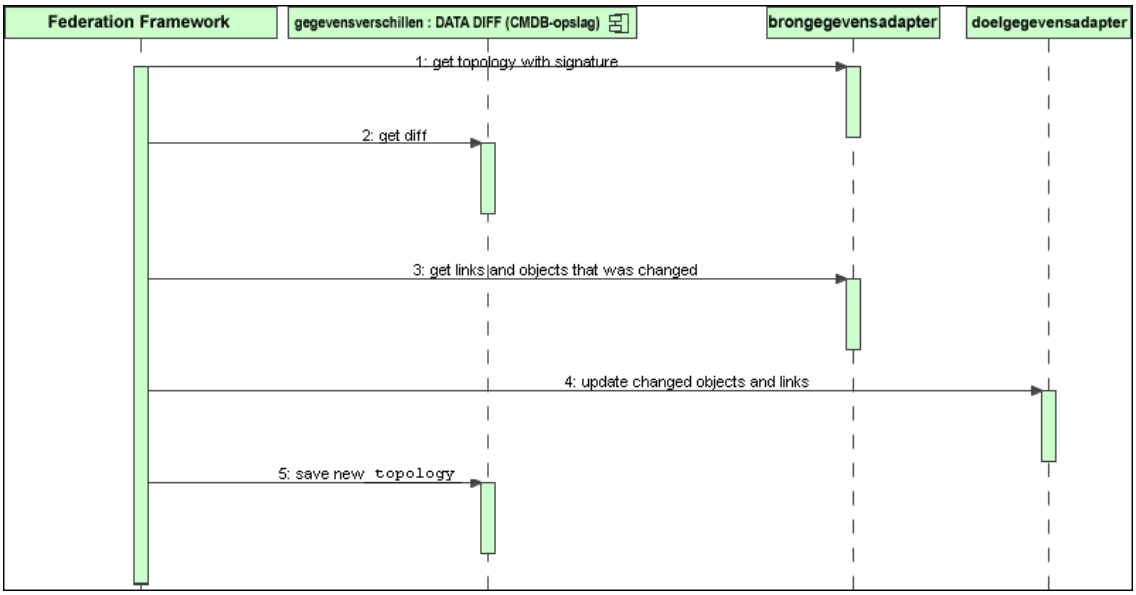
- "Definities en termen" beneden
- "Stroomdiagram" beneden

Definities en termen

Handtekening. Duidt de status van de eigenschappen in het CI aan. Als de eigenschapswaarden in een CI worden aangepast, moet de CI-handtekening ook worden gewijzigd. Aan de hand van de CI-handtekening kan worden bepaald of een CI is aangepast zonder het ophalen en vergelijken van alle CI-eigenschappen. Zowel het CI als de CI-handtekening worden geleverd door de betreffende adapter. De adapter is verantwoordelijk voor het wijzigen van de CI-handtekening als de CI-eigenschappen worden aangepast.

Stroomdiagram

Het volgende diagram illustreert de interactie tussen het Federation Framework en de bron- en doeladapters in een vullingstroom.



1. Het Federation Framework ontvangt de topologie voor het query-resultaat van de bronadapter. De adapter herkent de query aan de naam en voert deze uit in de externe gegevensopslaglocatie. Het topologieresultaat bevat de ID en handtekening voor elk CI en elke relatie in het resultaat. De ID is de logische ID die het CI definieert als uniek in de externe gegevensopslaglocatie. De handtekening moet worden gewijzigd als het CI of de relatie is aangepast.
2. Het Federation Framework gebruikt handtekeningen om zojuist ontvangen topologie-query-resultaten te vergelijken met de opgeslagen resultaten en zo te bepalen welke CI's zijn veranderd.
3. Nadat het Federation Framework heeft ontdekt dat de CI's en relaties zijn aangepast, roept het de bronadapter met de ID's van de aangepaste CI's en relaties aan als een parameter om hun volledige indeling op te kunnen halen.
4. Het Federation Framework verzendt de update naar de doeladapter. De doeladapter werkt de externe gegevensbron bij met de ontvangen gegevens.
5. Na de update slaat het Federation Framework het laatste query-resultaat op.

Adapter-interfaces

In dit gedeelte vindt u de volgende onderwerpen:

Definities en termen

Externe relatie. De relatie tussen twee externe CI-typen die worden ondersteund door dezelfde adapter.

Adapter-interfaces voor federated TQL-query's

Gebruik op de volgende manier de juiste adapter-interface voor elke adapter.

- De topologie-interface voor **One Node** wordt gebruikt als de adapter geen externe relaties ondersteunt. De adapter is dan niet bedoeld voor het ontvangen van een verzoek met meer dan één extern CI. Alle OneNode-interfaces zijn gemaakt om de werkstroom te vereenvoudigen. Wanneer u een uitgebreidere query nodig heeft, gebruikt u de `DataAdapter`-interface.
- Een **DataAdapter-interface** wordt gebruikt voor het definiëren van adapters die complexe federated query's ondersteunen. Het afstemmingsverzoek in deze adapters is een onderdeel van de enkelvoudige QueryDefinition-parameter. Deze adapters kunnen ook worden gebruikt voor vulling.

OneNode-interfaces

De volgende interfaces hebben verschillende typen afstemmingsgegevens:

- **OneNodeTopologyIdReconciliationDataAdapter.** Gebruik dit als de adapter een **TQL van één knooppunt** ondersteunt en de afstemming tussen gegevensopslaglocaties wordt berekend door de ID.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Gebruik dit als de adapter een **TQL van één knooppunt** ondersteunt en de afstemming tussen gegevensopslaglocaties wordt berekend door de eigenschappen van één ID.
- **OneNodeTopologyDataAdapter.** Gebruik dit als de adapter een **TQL van één knooppunt** ondersteunt en de afstemming tussen gegevensopslaglocaties via topologie wordt afgehandeld.

Gegevensadapter-interfaces

- **DataAdapter.** Gebruik deze adapter om complexe federated TQL-query's te ondersteunen. Maakt een optimale diversiteit mogelijk.
- **PopulateDataAdapter.** Gebruik deze adapter om complexe federated TQL-query's en vullingstromen te ondersteunen. In een vullingstroom haalt deze adapter de volledige gegevensset op en laat hij de probe de verschillen sinds de laatste uitvoering van de taak filteren.
- **PopulateChangesDataAdapter.** Gebruik deze adapter om complexe federated TQL-query's en

vullingstromen te ondersteunen. In een vullingstroom ondersteunt deze adapter het ophalen van uitsluitend de veranderingen die hebben plaatsgevonden sinds de laatste uitvoering van de taak.

Opmerking: Bij het ontwikkelen van een adapter die grote gegevenssets kan retourneren, is het van belang om segmentering toe te laten door de ChunkGetter Interface te implementeren. Raadpleeg het Java-document van de specifieke adapter voor meer informatie.

Aanvullende interfaces

- **SortResultDataAdapter.** Gebruik dit als u de resulterende CI's in de externe gegevensopslaglocatie kunt sorteren.
- **FunctionalLayoutDataAdapter.** Gebruik dit als u de functionele indeling in de externe gegevensopslaglocatie kunt berekenen.

Adapter-interfaces voor synchronisatie

- **SourceDataAdapter.** Gebruik dit voor bronadapters in vullingstromen.
- **TargetDataAdapter.** Gebruik voor doeladapters in datapush-stromen.

Fouten opsporen in adapterbronnen

In deze taak wordt beschreven hoe u de JMX-console gebruikt voor het aanmaken, weergeven en verwijderen van bronnen voor de adapterstatus (alle bronnen die zijn aangemaakt met de methoden voor bronmanipulatie in de DataAdapterEnvironment-interface, die worden opgeslagen in de UCMDB-database of de Probe-database) ten behoeve van foutopsporing en ontwikkeling.

1. Start de webbrowser en voer het adres van de server als volgt in:
   - Voor de UCMDB-server: `http://localhost:8080/jmx-console.`
   - Voor de probe: `http://localhost:1977`

   Wellicht moet u zich aanmelden met een gebruikersnaam en wachtwoord (de standaardinstellingen zijn sysadmin/sysadmin).
2. Ga als volgt te werk om de weergavepagina van JMX MBEAN te openen:
   - Op de UCMDB-server: klik op **UCMDB:service=FCMDB Adapter State Resource Services**
   - Op de probe: Klik op **type=AdapterStateResources**
3. Voer waarden in voor de bewerkingen die u wilt gebruiken en klik op **Invoke**.

Een adapter voor een nieuwe externe gegevensbron toevoegen

Deze taak legt uit hoe een adapter moet worden gedefinieerd om een nieuwe externe gegevensbron te ondersteunen.

Deze taak omvat de onderstaande stappen:

1. ## Vereisten

   Modelondersteunde adapterklassen voor CI's en relaties in het UCMDB-gegevensmodel. Als adapterontwikkelaar dient u:

   - kennis te bezitten over de hiërarchie van de UCMDB-CI-typen om te begrijpen hoe externe CIT's zijn gerelateerd aan de UCMDB-CIT's
   - de externe CIT's te modelleren in het klassemodel van UCMDB
   - de definities voor nieuwe CI-typen en hun relaties toe te voegen
   - geldige relaties te definiëren in het UCMDB-klassemodel voor de geldige relaties tussen interne adapterklassen. (De CIT's kunnen op elk niveau van de UCMDB-klassemodelstructuur worden geplaatst.)

   De modellering moet overeenkomen, ongeacht het federation-type (real-time of replicatie). Meer informatie over het toevoegen van nieuwe CIT-definities aan het UCMDB-klassemodel kunt u vinden in "Werken met de CI-kiezer" in de HP Universal CMDB – Handleiding Modeling.

   Voeg dit CIT toe aan de ondersteunde klassen met ondersteunde attributen en de afstemmingsregel voor dit CIT om ervoor te zorgen dat de adapter federated attributen op CIT's ondersteunt.

2. ## Definiëren van geldige relaties voor virtuele relaties

   Opmerking: dit gedeelte is alleen van belang voor federation.

   Voor het ophalen van federated CIT's die zijn verbonden met lokale CMDB-CIT's moet er een geldige koppelingsdefinitie bestaan tussen de twee CIT's in de CMDB.

   a. Maak een XML-bestand voor geldige koppelingen aan dat deze koppelingen bevat (als deze nog niet bestaan).

   b. Voeg het XML-bestand met de koppelingen toe aan het adapterpakket in de map **\validlinks**. Zie "Pakketbeheer" in de *HP Universal CMDB – Handleiding Beheer* voor meer informatie over dit onderwerp.

Voorbeeld van een definitie van een geldige relatie:

In het volgende voorbeeld is de relatie van type `containment` tussen exemplaren van type `node` tot exemplaren van type `myclass1` een geldige relatiedefinitie.

```
<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. ## Een adapterconfiguratie definiëren

   a. Ga naar **Adapterbeheer**.

   b. Klik op de knop **Nieuwe bron maken**  .

   c. In het dialoogvenster 'Nieuwe adapter' selecteert u **Integratie** en **Java-adapter**.

   d. Rechtsklik op de aangemaakte adapter en selecteer **Adapterbron bewerken** in het snelmenu.

   e. Pas de volgende XML-tags aan:

```
<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="newAdapterIdName"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
description="Adapterbeschrijving" schemaVersion="9.0"
displayName="Nieuwe schermnaam adapter">>

<deletable>true</deletable>

<discoveredClasses>

<discoveredClass>link</discoveredClass>

<discoveredClass>object</discoveredClass>

</discoveredClasses>

<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
 AdapterService">

<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
 AdapterServiceParams" enableAging="true"
enableDebugging="false" enableRecording=
"false" autoDeleteOnErrors="success" recordResult="false"
maxThreads="1" patternType="java_adapter"
maxThreadRuntime="25200000">
```

```
<className>com.yourCompany.adapter.MyAdapter.MyAdapterClass
<className>

</params>

<destinationInfo
className="com.hp.ucmdb.discovery.probe.tasks.BaseDestinationDat-
a">

<!-- check -->

<destinationData name="adapterId"
description="">${ADAPTER.adapter_id}</destinationData>

<destinationData name="attributeValues"
description="">${SOURCE.attribute_values}</destinationData>

<destinationData name="credentialsId"
description="">${SOURCE.credentials_id}</destinationData>

<destinationData name="destinationId"
description="">${SOURCE.destination_id}</destinationData>

</destinationInfo>

<resultMechanism isEnabled="true">

<autoDeleteCITs isEnabled="true">

<CIT>link</CIT>

<CIT>object</CIT>

</autoDeleteCITs>

</resultMechanism>

</taskInfo>

<adapterInfo>

<adapter-capabilities>

<support-federated-query>

<!--<supported-classes/> <!—see the section about supported
classes-->

<topologie>

<pattern-topology /> <!—or <one-node-topology> -->

</topology>

</support-federated-query>

<!--<support-replicatioin-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>
```

Zie "Tags en eigenschappen XML-configuratie " op pagina 183 voor meer informatie over XML-tags.

4. ## Ondersteunde klassen definiëren

Definieer ondersteunde klassen door de adaptercode te gebruiken door het implementeren van de methode *getSupportedClasses()* of door het XML-patroonbestand te gebruiken.

```
<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>
```

naam	De naam van het CI-type.
is-derived	Geeft aan of deze definitie inclusief alle onderliggende items is
is-reconciliation-	Geeft aan of deze klasse wordt gebruikt voor afstemming

supported	
is-id-reconciliation-supported	Geeft aan of deze klasse wordt gebruikt voor ID-afstemming
federation-not-supported	Geeft aan dat dit CIT niet bestemd is voor federation (bepaalde CIT's blokkeren, bijvoorbeeld een CIT die uitsluitend is gedefinieerd voor federation)
<supported-conditions>	Geeft aan wat de ondersteunde voorwaarden zijn op elk attribuut

5. ## De adapter implementeren

   Selecteer de juiste implementatieklasse van de adapter volgens zijn gedefinieerde mogelijkheden. De implementatieklasse van de adapter implementeert de juiste interfaces volgens gedefinieerde mogelijkheden.

   Ondersteuning van adapterafstemming kan worden gedefinieerd op basis van **global_id**. In dat geval moet **global_id** worden gedefinieerd als onderdeel van de afstemmingsattributen in de ondersteunde adapterklassen. Als ondersteuning van adapterafstemming wordt gedefinieerd op basis van **global_id**, moet **getTopologyWithReconciliationData()** de **global_id** retourneren als onderdeel van de afstemmingsobjecteigenschappen. De UCMDB gebruikt **global_id** voor afstemming van federation-resultaten voor een CIT in plaats van de identificatieregel.

6. ## Afstemmingsregels definiëren of de toewijzingsengine implementeren

   Als uw adapter federated TQL-query's ondersteunt, heeft u drie mogelijkheden om uw toewijzingsengine te definiëren:

   - De standaard CMDB 9.0x-toewijzingsengine gebruiken, die gebruikmaakt van de interne afstemmingsregels voor toewijzingen van de CMDB. Om dit te gebruiken, laat u de XML-tag **<default-mapping-engine/>>** leeg.

     Zie "Bestand reconciliation_types.txt" op pagina 134 voor meer informatie over dit onderwerp.

   - De CMDB 8.0x-toewijzingsengine gebruiken. Hiervoor gebruikt u de volgende XML-tag: **<default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>**

     Zie "Bestand reconciliation_rules.txt (voor achterwaartse compatibiliteit)" op pagina 134 voor meer informatie over dit onderwerp.

   - Schrijf uw eigen toewijzingsengine door het implementeren van de toewijzingsengine-interface en de JAR bij de rest van de adaptercode te plaatsen. Hiervoor gebruikt u de volgende XML-tag: **<default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>**

7. ## Voor implementatie vereiste JAR's toevoegen aan het klassepad

   Om uw klassen te implementeren, voegt u het bestand **federation_api.jar** toe aan het

klassepad van uw code-editor.

8. **De adapter uitrollen**

   Het adapterpakket uitrollen. Zie "Pakketbeheer" in de *HP Universal CMDB – Handleiding Beheer* voor algemene informatie over het uitrollen van een pakket.

   Het pakket moet de volgende entiteiten bevatten:

   - Nieuwe CIT-definitie (optioneel):
   - Wordt alleen gebruikt als de adapter nieuwe CI-typen ondersteunt die nog niet bestaan in UCMDB.
   - De nieuwe CIT-definities zijn te vinden in de map `class` in het pakket.
   - Nieuwe definitie gegevenstype (optioneel):
   - Wordt alleen gebruikt als de nieuwe CIT's nieuwe gegevenstypen vereisen.
   - De nieuwe gegevenstypedefinities zijn te vinden in de map `typedef` in het pakket.
   - Nieuwe definitie geldige relaties (optioneel):
   - Wordt alleen gebruikt als de adapter federated TQL ondersteunt.
   - De nieuwe definities van geldige relaties zijn te vinden in de map `validlinks` in het pakket.
   - Het XML-bestand voor patroonconfiguratie moet worden geplaatst in de map `discoveryPatterns` in het pakket.
   - **Beschrijver.** Definieert de pakketdefinities.
   - Plaats uw gecompileerde klassen (meestal een JAR-bestand) in het pakket onder de map **adapterCode\<adapter id>**.

     Opmerking: de mapnaam `adapter id` heeft dezelfde waarde als in de adapterconfiguratie.

   - Als u uw eigen configuratiebestand maakt, moet u het bestand plaatsen in het pakket onder de map **adapterCode\<adapter id>**.

9. **De adapter bijwerken**

   Aanpassingen van niet-binaire bestanden kunnen worden doorgevoerd in de module Adapterbeheer. Als u de configuratiebestanden in de module Adapterbeheer aanpast, wordt de adapter opnieuw geladen met de nieuwe configuraties.

   Bijwerken kan ook door de bestanden in het pakket aan te passen (zowel de binaire als de niet-binaire bestanden) en het pakket vervolgens opnieuw uit te rollen in Pakketbeheer. Zie "Een pakket uitrollen" in de *HP Universal CMDB – Handleiding Beheer*.

De toewijzingsengine implementeren

De configuratie van de toewijzingsengine hangt af van de toewijzingsengine die u gebruikt.

Deze taak omvat de onderstaande stappen:

- "Het bestand reconciliation_types.txt configureren (voor de UCMDB 9.0x-toewijzingsengine)" beneden
- "Het bestand reconciliation_rules.txt configureren (voor de UCMDB 8.0x-toewijzingsengine)" beneden

1. ## Het bestand reconciliation_types.txt configureren (voor de UCMDB 9.0x-toewijzingsengine)

   Het bestand wordt gebruikt om te definiëren welke CI-typen voor afstemming worden gebruikt in de adapter.

   Schrijf elk CI-type dat wordt gebruikt voor afstemming op een enkele regel, zoals hieronder:

   ```
   node business_application
   ```

   Plaats het bestand in het adapterpakket in de map **adapterCode\<AdapterID>\META-INF**. Ter ondersteuning van ID-afstemming (afstemming op basis van ID-toewijzing tussen de CMDB-ID in de CMDB en een waarde in de externe database) moet u een speciaal CMDB-kenmerk, te weten **cmdb_id**, toewijzen aan een kolom in de database van het tekenreekstype (char, varchar) of byte[] (raw/bytes).

2. ## Het bestand reconciliation_rules.txt configureren (voor de UCMDB 8.0x-toewijzingsengine)

   Met dit bestand worden de afstemmingsregels geconfigureerd. Elke rij in het bestand staat voor een regel. Bijvoorbeeld:

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

   De parameter **reconcilition_type** is gevuld met het type CI waarop de afstemming wordt uitgevoerd (de UCMDB-klassenaam die is verbonden met de federated klasse in de TQL).

   De parameter **expression** is de logica die bepaalt of twee afstemmingsobjecten gelijk zijn (een afstemmingsobject van de UCMDB-zijde en de andere van de federated adapter-zijde).

   De expressie is samengesteld uit OR's en AND's.

   De conventie met betrekking tot attribuutnamen in het expressiedeel is **[className].[attributeName]**. Bijvoorbeeld: het attribuut **ip_address** in de klasse **ip** wordt als **ip.ip_address** geschreven.

   U kunt geordende vergelijkingen definiëren. De geordende vergelijking controleert de eerste OR-subexpressie. Als twee afstemmingsobjecten de waarde op de attributen van de subexpressie hebben en dit als 'false' wordt geretourneerd (de afstemmingsobjecten zijn niet gelijk) dan wordt de tweede OR-subexpressie niet vergeleken.

   Voor een geordende vergelijking gebruikt u **ordered expression** in plaats van **expression**.

   De circonflexe (**^**) wordt gebruikt om vergelijkingen niet hoofdlettergevoelig uit te voeren.

   De overige parameters (**end1_type**, **end2_type**, and **link_type**) worden alleen gebruikt als de afstemmingsgegevens twee knooppunten bevatten van het afstemmingstype (de topologische afstemmingsgegevens). In dit geval zijn de afstemmingsgegevens **end1_type -(link_type)> end2_type**.

De relevante indeling hoeft niet te worden toegevoegd, aangezien deze van de expressie wordt overgenomen.

Voor afstemming op basis van UCMDB-ID gebruikt u **cmdb_id** als attribuutnaam in de expressie.

Plaats het bestand in het adapterpakket in de map **adapterCode\<AdapterID>\ META-INF**.

Voorbeelden:

- U kunt een afstemmingsregel alleen voor een knooppunt-CIT toevoegen. Dit is omdat alleen knooppunt-CIT's een geldige relatie hebben met externe CIT's. Een knooppunt-CI in de CMDB wordt bijvoorbeeld vergeleken met een knooppunt-CI in ServiceCenter via het attribuut `node.name` of via het attribuut `ip_address.name`.
- De afstemmingsregel is in dit geval een topologieregel en de expressie is geordend. De regel voert de volgende controles uit op de CI's die onder de vergelijking vallen:
  - Als het attribuut `node.name` gelijk is, komt de regel overeen met de knooppunten.
  - Als het attribuut `node.name` niet gelijk is, komt de regel niet overeen met de knooppunten.
  - Als het attribuut `node.name` null is in een van de vergeleken CI's, controleert de regel het attribuut `ip_address.name`. Als het attribuut`ip_address.name` gelijk is, komt de regel overeen met de knooppunten.

Een voorbeeldadapter maken

Dit voorbeeld laat zien hoe u een voorbeeldadapter maakt. Deze taak omvat de onderstaande stappen:

- "Adapterlogica selecteren" beneden
- "Het project laden" beneden

1. ### Adapterlogica selecteren

   Wanneer u een adapter implementeert, moet u bepalen hoe om te gaan met de voorwaardenlogica in de implementatie (eigenschapsvoorwaarden, ID-voorwaarden, afstemmingsvoorwaarden en koppelingsvoorwaarden).

   a. Laad alle gegevens in het adaptergeheugen en laat het de benodigde CI-exemplaren selecteren of filteren.

   b. Converteer alle voorwaarden naar de taal van de gegevensbron en laat die de gegevens selecteren en filteren. Bijvoorbeeld:

      - Converteer de voorwaarde naar een SQL-query.
      - Converteer de voorwaarde naar een Java-API-filterobject.

   c. Filter sommige gegevens op de externe service en laat het selecteren en filteren van de rest over aan de adapter.

   In het MyAdapter-voorbeeld is de logica in optie *a* gebruikt.

2. ### Het project laden

Kopieer de bestanden uit de map **C:\hp\UCMDB\UCMDBServer\tools\adapter-dev-kit\SampleAdapters** volg de instructies in de readme-bestanden.

Opmerking: wanneer u een adapter gebruikt met grote gegevenssets moet u wellicht caching en indexering gebruiken om de prestaties van federation te verbeteren.

U vindt de online javadocs-documentatie in:

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

Tags en eigenschappen XML-configuratie

id="newAdapterIdName"		Definieert de echte naam van de adapter. Wordt gebruikt voor zoeken in logboeken en mappen
displayName="New Adapter Display Name"		Definieert de weergavenaam van de adapter zoals deze verschijnt in de UI.
<className>...</className>		Definieert de adapterinterface die de Java-klasse implementeert.
<category >My Category</category>		Definieert de categorie van de adapter.
<parameters>		Definieert de eigenschappen voor de configuratie die beschikbaar zijn in de UI bij het opzetten van een nieuw integratiepunt.
	naam	De naam van de eigenschap (meestal door code gebruikt)
	description	De weergavehint van de eigenschap
	type	Tekenreeks of geheel getal (gebruik geldige waarden met tekenreeks voor boolean).
	display-name	De naam van de eigenschap in de UI.
	mandatory	Geeft aan of deze configuratie-eigenschap verplicht is voor de gebruiker.
	order-index	De plaatsingsvolgorde van de eigenschap (klein = bovenaan)
	valid-values	Een lijst met mogelijke geldige waarden, gescheiden door ';'-tekens (bijvoorbeeld valid-values="Oracle;SQLServer;MySQL" of valid-values="True;False").
<adapterInfo>		Bevat de definitie van de statische instellingen en mogelijkheden van de adapter.
	<support-federated-query>	Definieert deze adapter als geschikt voor

		federation.
	<one-node-topology>	De mogelijkheid tot het federeren van query's met een enkel federated query-knooppunt.
	<pattern-topology>	De mogelijkheid tot het federeren van complexe query's.
	<support-replication-data>	Definieert de mogelijkheid om datapush- en vullingstromen uit te voeren.
	<source>	Deze adapter kan worden gebruikt voor vullingstromen.
	<push-back-ids>	Hiermee wordt de algemene ID van het CI geretourneerd naar de kolom global_id van de tabel (moet worden gedefinieerd in orm.xml). Het gedrag kan worden overschreven door de invoegtoepassing **FcmdbPluginPushBackIds** te implementeren.
	<changes-source>	Deze adapter kan worden gebruikt voor vullingverandering-stromen.
	<target>	Deze adapter kan worden gebruikt voor datapush-stromen.
	<default-mapping-engine>	Maakt de definitie van een toewijzingsengine voor de adapter mogelijk (standaard gebruikt de adapter de standaard toewijzingsengine). Voor een andere toewijzingsengine voert u de te implementeren klassenaam van de toewijzingsengine in (voor de UCMDB 8.0x-toewijzingsengine gebruikt u: **com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine**)
	<removedAttributes>	Voor een gedwongen verwijdering van een specifiek attribuut uit het resultaat.
	<full-population-days-interval>	Geeft aan wanneer een volledige vullingstaak moet worden uitgevoerd in plaats van een differentiële taak (om de ‘x’ dagen). Gebruikt het verouderingsmechanisme samen met de veranderingstroom.
	<adapter-settings>	De lijst met instellingen van de adapter.
	<list.attributes.for.set>	Hiermee wordt bepaald welke attributen de vorige waarde (indien aanwezig) overschrijven.

Hoofdstuk 6

Push-adapters ontwikkelen

In dit hoofdstuk vindt u de volgende informatie:

Push-adapters ontwikkelen - overzicht

De algemene push-adapter biedt een platform dat snelle ontwikkeling van integraties mogelijk maakt, waarmee UCMDB 9.0x-gegevens naar externe gegevensopslagplaatsen (databases en applicaties van derden) worden gepusht. Voor de ontwikkeling van een aangepaste integratie op basis van een algemene push-adapter is het volgende nodig:

- Een XML-toewijzingsbestand tussen de CI-koppelingstypen van UCMDB en de externe gegevensitems.
- Een Jython-script om de gegevensitems naar de externe gegevensopslagplaats te pushen.

Differentiële synchronisatie

De push-adapter kan differentiële synchronisatie alleen ondersteunen als de functie **DiscoveryMain** een object retourneert waarmee de **DataPushResults**-interface wordt geïmplementeerd. Deze interface bevat de toewijzingen tussen de ID's die het Jython-script van de XML ontvangt, en de ID's die het Jython-script op de externe computer aanmaakt. De laatste ID's zijn van het type **ExternalId**.

De opdracht **ExternalIdUtil.restoreExternal**, die de ID van het CI in de CMDB als parameter ontvangt, herstelt de externe ID op basis van de ID van het CI in de CMDB. Deze opdracht kan bijvoorbeeld worden gebruikt tijdens het uitvoeren van differentiële synchronisatie en er wordt een koppeling ontvangen waarvan een van de uiteinden niet in de bulk is opgenomen (deze was al gesynchroniseerd).

Als de methode **DiscoveryMain** in het Jython-script waarop de push-adapter is gebaseerd, een leeg **ObjectStateHolderVector**-exemplaar retourneert, wordt er geen differentiële synchronisatie op de adapter ondersteund. Dit houdt in dat zelfs wanneer een differentiële synchronisatietaak

wordt uitgevoerd, in werkelijkheid een volledige synchronisatie wordt uitgevoerd. Daarom kunnen er geen gegevens worden bijgewerkt of verwijderd in het externe systeem, aangezien alle gegevens tijdens elke synchronisatie aan de CMDB worden toegevoegd.

Toewijzingsbestanden voorbereiden

Opmerking: U kunt alle CI's en relaties rechtstreeks en zonder toewijzing uit de CMDB ophalen door geen **mappings.xml**-bestand te maken. Daardoor worden alle CI's en relaties met al hun attributen geretourneerd.

Toewijzingsbestanden kunnen op twee verschillende manieren worden voorbereid:

- U kunt één globaal toewijzingsbestand voorbereiden.

  Alle toewijzingen worden in één bestand met de naam **mappings.xml** geplaatst.

- U kunt voor elke push-query een apart bestand voorbereiden.

  Elk toewijzingsbestand wordt **<query name>.xml** genoemd.

Zie "Schema toewijzingsbestand" op pagina 194 voor meer informatie over dit onderwerp.

Deze taak omvat de onderstaande stappen:

- "Toewijzingsbestanden aanmaken" beneden
- "CI's toewijzen" beneden
- "Koppelingen toewijzen " op volgende pagina

1. ## Toewijzingsbestanden aanmaken

   De structuur van toewijzingsbestanden wordt als volgt aangemaakt:

```
<?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. ## CI's toewijzen

   CMDB-CI-typen kunnen op twee manieren worden toegewezen:

   - Wijs een CI-type zodanig toe dat CI's van dat type en alle overgenomen typen op dezelfde manier worden toegewezen.

```
<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>
```

- Wijs een CI-type zodanig toe dat alleen CI's van dat type worden verwerkt. CI's van overgenomen typen worden dan alleen verewerkt als hun type ook is toegewezen (op een van de twee manieren):

```
<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>
```

Een CI-type dat indirect wordt toegewezen (een van de bovenliggende CI's wordt toegewezen met behulp van **source_ci_type_tree**), kan ook zijn bovenliggende toewijzing overschrijven door het op te nemen in zijn eigen **source_ci_type_tree** of **source_ci_type**.

Het verdient aanbeveling waar mogelijk **source_ci_type_tree** te gebruiken. Anders worden resulterende CI's van een CI-type dat niet in de toewijzingsbestanden voorkomt, niet overgebracht naar het Jython-script.

3. ## Koppelingen toewijzen

Er zijn twee manieren om koppelingen toe te wijzen.

- Wijs een koppeling toe die tevens alle koppelingstypen toewijst die overnemen van die specifieke koppeling:

```
<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>
```

- Wijs een koppeling toe die tevens alleen dat specifieke koppelingstype toewijst en niet de koppelingstypen die daarvan overnemen:

```
<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>
```

Jython-scripts schrijven

Het toewijzingsscript is een gewoon Jython-script waarin de regels voor Jython-scripts moeten worden gevolgd. Zie "Jython-adapters ontwikkelen" op pagina 40 voor meer informatie over dit onderwerp.

Het script moet de functie **DiscoveryMain** bevatten. Met deze functie kan een leeg **OSHVResult**- of een **DataPushResults**-exemplaar worden geretourneerd in geval van succes.

Om fouten te rapporteren moet het script een uitzondering genereren, bijvoorbeeld:

```
raise Exception('Failed to insert to remote UCMDB using
TopologyUpdateService. See log of the remote UCMDB')
```

In de functie DiscoveryMain kunnen de gegevensitems die moeten worden gepusht of verwijderd uit de externe applicatie, als volgt worden verkregen:

```
# get add/update/delete result objects (in XML format) from the
Framework
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
```

The client object to the external application can be obtained as follows:

```
oracleClient = Framework.createClient()
```

Het clientobject voor de externe applicatie kan als volgt worden verkregen:

```
oracleClient = Framework.createClient()
```

Voor dit clientobject wordt automatisch gebruikgemaakt van de ID van de aanmeldingsgegevens, de hostnaam en het poortnummer die door de adapter zijn doorgegeven via Framework.

Als u de verbindingsparameters moet gebruiken die u voor de adapter hebt gedefinieerd (zie stap "Bewerk het bestand discoveryPatterns\push_adapter.xml." in "Adapterpakketten samenstellen " op pagina 193 voor meer informatie), gebruikt u de volgende code:

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

Bijvoorbeeld:

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

In dit gedeelte vindt u ook de volgende onderwerpen:

- "Werken met de resultaten van de toewijzing" beneden
- "Testverbinding in het script afhandelen" op pagina 191

Werken met de resultaten van de toewijzing

Met de algemene push-adapter worden XML-strings aangemaakt waarmee de gegevens worden beschreven die moeten worden toegevoegd, bijgewerkt of verwijderd uit het doelsysteem. Het Jython-script moet deze XML analyseren en vervolgens moet de toevoeg-, bijwerk- of verwijderbewerking op het doel worden uitgevoerd.

In de XML van de toevoegbewerking die het Jython-script ontvangt, is het attribuut `mamId` voor de objecten en koppelingen altijd de UCMDB-ID van het oorspronkelijke object of de oorspronkelijke koppeling, voordat type, attribuut of overige informatie ervan werd gewijzigd in het schema van het externe systeem.

In de XML van de bijwerk- of verwijderbewerkingen bevat het atttribuut `mamId` van elk object of elke koppeling de stringweergave van dezelfde `ExternalId` die door het Jython-script werd geretourneerd bij de vorige synchronisatie.

In de XML gebruikt het `id`-attribuut van een CI de `cmdbId` als een externe ID of de `ExternalId` van dat CI als het CI een `ExternalId` heeft gekregen toen het CI naar het script werd verzonden. De velden `end1Id` en `end2Id` van de koppeling bevatten voor elk van de uiteinden van de koppeling de `cmdbId` als externe ID of de `ExternalId` van het uiteinde van die koppeling als het CI aan dat uiteinde een `ExternalId` heeft gekregen toen het naar het script werd verzonden.

Tijdens het verwerken van de CI's in het Jython-script is de retourwaarde van het script een toewijzing tussen de CMDB-ID van het CI en de gegeven ID (de ID die aan elk CI in het script is gegeven). Als een CI voor het eerst wordt gepusht, is de ID die is opgenomen in de XML van dat CI, de CMDB-ID. Als een CI niet voor het eerst werd gepusht, is de CI-ID dezelfde als de ID die aan dat CI in het script werd gegeven toen deze voor het eerst werd gepusht.

De ID wordt als volgt opgehaald uit het XML-script van het CI:

1. Vanuit het CI-element in de XML: haal de ID op uit het id-attribuut. Bijvoorbeeld: `id = objectElement.getAttributeValue('id')`.
2. Na het ophalen van de ID uit de XML: herstel de ID uit het attribuut (string). Bijvoorbeeld: `objectId = CmdbObjectID.Factory.restoreObjectID(id)`.
3. Controleer of de `objectId` die in de vorige stap werd opgehaald, de CMDB-ID is. U kunt dat doen door te controleren of de `objectId` de nieuwe ID heeft gekregen die verstrekt werd door het script. Als dat het geval is, is de geretourneerde ID niet de CMDB-ID. Een voorbeeld: `newId = objectId.getPropertyValue(<de naam van het id-attribuut dat door het script is verstrekt>)`.

   Als `newId` null is, is de ID die werd geretourneerd in de XML, een CMDB-ID.
4. Als de ID een CMDB-ID is (dat wil zeggen, `newId` is null), voert u de volgende opdracht uit (is de ID geen CMDB-ID, ga dan verder met stap 5):
   a. Maak een eigenschap voor het CI aan dat de nieuwe ID bevat. Een voorbeeld: `propArray =[TypesFactory.createProperty('<de baam van het id-attribuut dat door het script werd verstrekt>', '<new id>')]`.
   b. Maak een `externaId` voor dat CI aan. Bijvoorbeeld: `cmdbId = extI.getPropertyValue('internal_id')`

```
className = extI.getType()
externalId = ExternalIdFactory.createExternalCiId(className,
propArray)
```

c. Wijs de CMDB-ID toe aan de nieuw gemaakte externalId (en retourneer in de volgende stap die toewijzing naar de adapter). Bijvoorbeeld: `objectMappings.put(cmdbId, externalId)`

d. Wanneer alle CI's en koppelingen zijn toegewezen:

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

5. Als de ID de nieuwe ID is (dat wil zeggen, `newId` is niet null), is de `externalId` de `newId`.

Voorbeeld van het XML-resultaat

```
<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>
```

Opmerking: Als `datatype="BYTE"`, is de waarde van het geretourneerde resultaat een **String** die wordt gegenereerd als: `new String([het bytematrix-attribuut])`. Het `byte[]-object` kan worden gereconstrueerd door: `<de ontvangen String>.getBytes()`. Er kan een verschil in standaardlandinstelling bestaan tussen de server en de probe. In dat geval vindt de reconsructie plaats op op basis van de standaardlandinstelling van de server.

Testverbinding in het script afhandelen

Een Jython-script kan worden aangeroepen om de verbinding met een externe applicatie te testen. In dit geval is het bestemmingsattribuut `testConnectiontrue`. Dit attribuut kan als volgt van het Framework worden verkregen:

```
testConnection = Framework.getTriggerCIData('testConnection')
```

Wanneer uitvoering plaatsvindt in de testverbindingsmodus, moet een uitzondering worden gegenereerd als geen verbinding met de externe applicatie tot stand kan worden gebracht. Als de verbinding wel lukt, moet de functie **DiscoveryMain** een leeg **OSHVResult** retourneren.

Differentiële synchronisatie ondersteunen

Belangrijk: Als u differentiële synchronisatie implementeert op een bestaande adapter die in versie 9.00 of 9.01 is aangemaakt, moet u het bestand push-adapter.zip van versie 9.02 of hoger gebruiken om uw adapterpakket opnieuw aan te maken. Zie "Adapterpakketten samenstellen " op pagina 193 voor meer informatie over dit onderwerp.

Met deze taak wordt de push-adapter ingeschakeld om differentiële synchronisatie uit te voeren. Zie "Differentiële synchronisatie " op pagina 185 voor meer informatie over dit onderwerp.

Met het Jython-script wordt het object **DataPushResults** geretourneerd dat twee Java-toewijzingen bevat: een voor toewijzingen van object-ID's (sleutels en waarden zijn objecten van

het type ExternalCiId) en een voor koppelings-ID's (sleutels en waarden zijn objecten van het type ExternalRelationId).

- Voeg de volgende **from**-instructies aan uw Jython-script toe:

```
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
```

- Gebruik de standaardklasse **DataPushResultsFactory** om het **DataPushResults** van de functie **DiscoveryMain** te verkrijgen.

```
# Create the UpdateResult object

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

- Gebruik de volgende opdrachten om Java-toewijzingen voor het object **DataPushResults** aan te maken:

```
# Prepare the maps to store the mappings if IDs

objectMappings = HashMap()

linkMappings = HashMap()
```

- Gebruik de klasse **ExternalIdFactory** om de volgende ExternalId-ID's aan te maken:
  - ExternalId voor objecten of koppelingen die afkomstig zijn van een CMDB (zo zijn alle CI's in een toevoegbewerking afkomstig van de CMDB):

```
externaCIlId = ExternalIdFactory.createExternalCmdbCiId(ciType,
ciIDAsString)

externalRelationId =
ExternalIdFactory.createExternalCmdbRelationId(linkType,
end1ExternalCIId, end2ExternalCIId, linkIDAsString)
```

  - ExternalId voor objecten of koppelingen die niet afkomstig zijn van een CMDB (elke bijwerk- en verwijderbewerking bevat doorgaans dergelijke objecten):

```
myIDField = TypesFactory.createProperty("systemID", "1")

myExternalId = ExternalIdFactory.createExternalCiId(type,
myIDField)
```

> **Opmerking:** als het Jython-script bestaande informatie heeft bijgewerkt en de ID van het object (of de koppeling) verandert, moet u een toewijzing tussen de vorige externe ID en de nieuwe ID retourneren.

- Gebruik de methode **restoreCmdbCiIDString** of **restoreCmdbRelationIDString** van de klasse **ExternalIdFactory** om de UCMDB ID-string op te halen van een externe ID van een object of koppeling die afkomstig is van de UCMDB.

- Gebruik de methoden **restoreExternalCiId** en **restoreExternalRelationId** van de klasse **ExternalIdUtil** om het object **ExternalId** van de attribuutwaarde mamId van de XML van de bijwerk- of verwijderbewerkingen te herstellen.

  Opmerking: ExternalId-objecten vormen in werkelijkheid een matrix van eigenschappen. Dit houdt in dat u een **ExternalId**-object kunt gebruiken om informatie, die u later nodig hebt en waarmee de gegevens in het externe systeem worden geïdentificeerd, op te slaan.

Adapterpakketten samenstellen

1. Pak de inhoud van **C:\hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip** uit in een tijdelijke map. Het bestand **sql_queries** dat zich in het adapterpakket bevindt in **adapterCode > PushAdapter > sqlTablesCreation**, bevat de query's voor het maken van tabellen in een nieuw schema in Oracle voor het testen van de adapter. De tabellen behoren bij het bestand **adapterCode\<adapter ID>\mappings\mappings.xml**.

   Opmerking: Het bestand **sql_queries** is voor de adapter niet nodig. Het is slechts een voorbeeld.

2. Bewerk het bestand **discoveryPatterns\push_adapter.xml**.

   a. Wijzig de tag <pattern> met een nieuwe ID en weergegeven label. Vervang:

   ```
   <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">
   ```

   met:

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

   b. Werk de parameterlijst bij, zodat de lijst met parameters de vereiste verbindingsattributen weergeeft. Verwijder het attribuut **probeName** niet.

3. Wijzig de naam van de map **adapterCode\PushAdapter** met de adapter-ID die wordt gebruikt in stap (bijvoorbeeld **adapterCode\MyPushAdapter**).

4. Het bestand **discoveryScript** bevat een **script pushScript.py**-script waarmee de CI's en koppelingen in een externe Orcale-database worden ingevoegd. Vervang **discoveryScripts\pushScript.py** met het script dat u hebt geschreven (zie "Jython-scripts schrijven " op pagina 188 voor meer informatie). Als u de naam van het script wijzigt, moet de eigenschap `jythonScript.name` in **adapterCode\<adapter ID>\push.properties** dienovereenkomstig worden bijgewerkt.

5. Het bestand **adapterCode\<adapter ID>\mappings\mappings.xml**, dat zich bevindt in **adapterCode\<adapter ID>\mappings**, is een voorbeeld van een toewijzingsbestand dat een toewijzing bevat van:

- knooppunt-CI-type met alle CI-typen die hiervan overnemen
- UNIX-CI-type zonder de CI-typen die hiervan overnemen
- Afhankelijkheidskoppeling met alle koppelingstypen die hiervan overnemen
- Talk-koppelingstype zonder de koppelingstypen die hiervan overnemen

Dit toewijzingsvoorbeeld behoort bij het voorbeeld van de tabellen die in Oracle zijn gemaakt in het bestand **sql_queries** (zie stap 1).

Vervang het bestand **adapterCode\<adapter ID>\mappings\mappings.xml** met de toewijzingsbestanden die u hebt voorbereid (zie "Toewijzingsbestanden voorbereiden" op pagina 186 voor meer informatie.

Als u een toewijzingsbestand voor elke TQL-methode wilt gebruiken, wijst u de naam van de bijbehorende TQL aan elk XML-bestand toe, gevolgd door **.xml**. In dit geval wordt het bestand **mappings.xml** als standaard gebruikt, indien geen specifiek toewijzingsbestand voor de huidige TQL-naam is gevonden. De naam van het standaardtoewijzingsbestand kan worden aangepast door de eigenschap `mappingFile.default` te wijzigen in **adapterCode\<adapter ID>\push.properties**.

Schema toewijzingsbestand

Naam en pad van element	Beschrijving	Attributen
Integratie	Hiermee wordt de toewijzingsinhoud van het bestand gedefinieerd. Moet het buitenste blok in het bestand zijn, met uitzondering van de beginregel en opmerkingen.	
info (integration)	Hiermee wordt informatie gedefinieerd over de gegevensopslagplaatse-n die worden geïntegreerd.	
source (integration > info)	Hiermee wordt informatie over de opslagplaats van brongegevens gedefinieerd.	1. **Naam:** type **Beschrijving:** Naam van de opslagplaats van de brongegevens **Is verplicht?:** Vereist **Type:** String 2. **Naam:** versions **Beschrijving:** Versie(s) van de opslagplaatsen van de brongegevens

Naam en pad van element	Beschrijving	Attributen
		Is verplicht?: Vereist **Type:** String 3. **Naam:** vendor **Beschrijving:** Leverancier van de opslagplaats van de brongegevens **Is verplicht?:** Vereist **Type:** String
target (integration > info)	Hiermee wordt informatie over de opslagplaats van de doelgegevens gedefinieerd.	1. **Naam:** type **Beschrijving:** Naam van de opslagplaats van de brongegevens **Is verplicht?:** Vereist **Type:** String 2. **Naam:** versions **Beschrijving:** Versie(s) van de opslagplaatsen van de doelgegevens **Is verplicht?:** Vereist **Type:** String 3. **Naam:** vendor **Beschrijving:** Leverancier van de opslagplaats van de brongegevens **Is verplicht?:** Vereist **Type:** String
targetcis (integration)	Containerelement voor alle CIT-toewijzingen.	
source_ci_type_tree (integration > targetcis)	Hiermee wordt een bron-CIT en alle CI-typen die daarvan overnemen gedefinieerd.	1. **Naam:** name **Beschrijving:** Naam van het bron-CIT **Is verplicht?:** Vereist **Type:** String 2. **Naam:** mode **Beschrijving:** Het bijwerkingstype dat is vereist voor het huidige CI-type **Is verplicht?:** Vereist **Type:** Een van de volgende strings: a. **insert:** Gebruik deze string alleen als het CI nog niet bestaat. b. **update:** Gebruik deze string alleen als van het CI bekend is dat het bestaat. c. **update_else_insert:** Als het CI bestaat, werkt u het bij. Anders maakt u een nieuw CI aan. d. **ignore:** Doe niets met dit CI-type.

Naam en pad van element	Beschrijving	Attributen
source_ci_type (integration > targetcis)	Hiermee wordt een bron-CIT gedefinieerd zonder de CI-typen die daarvan overnemen.	1. **Naam:** name **Beschrijving:** Naam van het bron-CIT **Is required?:** Vereist **Type:** String 2. **Naam:** mode **Beschrijving:** Het bijwerkingstype dat is vereist voor het huidige CI-type **Is verplicht?:** Vereist **Type:** Een van de volgende strings: a. **insert:** Gebruik deze string alleen als het CI nog niet bestaat. b. **update:** Gebruik deze string alleen als van het CI bekend is dat het bestaat. c. **update_else_insert:** Als het CI bestaat, werkt u het bij. Anders maakt u een nieuw CI aan. d. **ignore:** Doe niets met dit CI-type.
target_ci_type (integration > targetcis > source_ci_type -OF- integration > targetcis > source_ci_type_ tree)	Hiermee wordt een doel-CIT gedefinieerd.	1. **Naam:** name **Beschrijving:** Typenaam van doel-CI **Is verplicht?:** Vereist **Type:** String 2. **Naam:** schema **Beschrijving:** De naam van het schema waarmee dit CI-type op het doel wordt opgeslagen **Is verplicht?:** Niet vereist **Type:** String 3. **Naam:** namespace **Beschrijving:**Hiermee wordt de naamruimte van dit CI-type op het doel aangegeven. **Is verplicht?:** Niet vereist **Type:** String
targetprimarykey (integration > targetcis > source_ci_type) -OF- (integration > targetcis > source_ci_type_tree -OF- (integration >	Hiermee worden de primaire sleutelattributen van het doel-CIT geïdentificeerd.	

Naam en pad van element	Beschrijving	Attributen
targetrelations > link) -OF- (integration > targetrelations > source_link_type_ tree)		
pkey (integration > targetcis> source_ci_ type > targetprimarykey -OF- integration > targetcis > source_ci_type_tree > targetprimarykey -OF- (integration > targetrelations > link > targetprimarykey) -OF- integration > targetrelations > source_link_type_tree > targetprimarykey)	Hiermee wordt één primair sleutelattribuut geïdentificeerd. Alleen vereist als modus **update** of **insert_else_update** is.	
target_attribute (integration > targetcis> source_ci_ type -OF- integration > targetcis > source_ci_type_tree -OF- integration > targetrelations > link -OF-	Hiernee wordt het attribuut van het doel-CIT gedefinieerd.	1. **Naam:** name **Beschrijving:**Naam van het attribuut van het doel-CIT. **Is verplicht?:** Vereist **Type:** String 2. **Naam:** datatype **Beschrijving:** Gegevenstype van het het attribuut van het doel-CIT. **Is verplicht?:** Vereist **Type:** String 3. **Naam:** length **Beschrijving:** Voor gegevenstypen string en teken is dit de grootte van een geheel getal van het doelattribuut **Is verplicht?:** Niet vereist

Naam en pad van element	Beschrijving	Attributen
integration > targetrelations > source_link_type_ tree)		**Type.** Geheel getal 4. **Naam.** option **Beschrijving.** De conversiefunctie die op de waarde moet worden toegepast **Is vereist.** False **Type.** Een van de volgende strings: a. **uppercase** – Converteren naar hoofdletters b. **lowercase** – Converteren naar kleine letters Als dit attribuut leeg is, wordt geen conversiefunctie toegepast
map (integration > targetcis > source_ci_ type > target_attribute -OF- integration > targetcis > source_ci_type_tree > target_attribute) -OF- (integration > targetrelations > link > target_attribute -OF- integration > targetrelations > source_link_type_tree > target_attribute)	Hiermee wordt opgegeven hoe de attribuutwaarde van het bron-CIT wordt verkregen.	1. **Naam.** type **Beschrijving.** Het type toewijzing tussen de bron- en de doelwaarden **Is vereist.** Vereist **Type.** Een van de volgende strings: a. **direct** – Hiermee wordt een 1-op-1 toewijzing van de waarde van het bronattribuut aan de waarde van het doelattribuut opgegeven b. **compoundstring –** Subelementen worden in één string samengevoegd en de waarde van het doelattribuut wordt ingesteld c. **childattr** – Subelementen zijn een of meer attributen van een onderliggend CIT. Onderliggende CIT's worden gedefinieerd als CIT's met **composition**- of **containment**-relatie d. **constant** – Statische string 2. **Naam.** value **Beschrijving.** Constante string voor type=**constant** **Is vereist.** Alleen vereist wanneer type=**constant** **Type.** String 3. **Naam.** attr **Beschrijving.** Naam van bronattribuut voor type=**direct** **Is vereist.** Alleen vereist wanneer type=**direct** **Type.** String

Naam en pad van element	Beschrijving	Attributen
aggregation (integration > targetcis > source_ci_type > target_attribute > map -OF- integration > targetcis > source_ci_type_tree > target_attribute > map -OF- (integration > targetrelations > link > target_attribute > map -OF- integration > targetrelations > source_link_type_tree > target_attribute > map) Alleen geldig wanneer het type van de kaart **childattr** is	Hiermee wordt opgegeven hoe de attribuutwaarden van het onderliggende CI voor het bron-CI in één waarde worden gecombineerd voor toewijzing aan het attribuut van het doel-CI. Optioneel.	**Naam:** type **Beschrijving.** Het type aggregatiefunctie **Is verplicht?:** Vereist **Type.** Een van de volgende strings: • **csv** – Hiermee worden alle opgenomen waarden in een door komma's gescheiden lijst samengevoegd (numeriek of string/teken) • **count** – Hiermee wordt een numeriek aantal van alle opgenomen waarden geretourneerd • **sum –** Hiermee wordt een numeriek aantal van alle opgenomen waarden geretourneerd • **average** – Hiermee wordt een numeriek gemiddelde van alle opgenomen waarden geretourneerd • **min** – Hiermee wordt de laagste opgenomen waarde van het type numeriek/teken geretourneerd • **max** – Hiermee wordt de hoogste opgenomen waarde van het type numeriek/teken geretourneerd
source_child_ci_type (integration > targetcis> source_ci_type > target_attribute > map -OF- integration > targetcis > source_ci_type_tree > target_attribute > map -OF- (integration > targetrelations > link > target_attribute > map	Geeft aan van welk verbonden CI het onderliggend attribuut wordt overgenomen.	1. **Naam.** name **Beschrijving.** Type van het onderliggend CI **Is vereist.** Vereist **Type.** String 2. **Name.** source_attribute **Beschrijving.** Attribuut van het onderliggend CI dat wordt toegewezen. **Is vereist.** Alleen vereist als het **childAttr**-aggregatietype (dat zich op hetzelfde pad bevindt) niet gelijk is aan count. **Type.** String

Naam en pad van element	Beschrijving	Attributen
-OF- integration > targetrelations > source_link_type_tree > target_attribute > map) Alleen geldig wanneer het type van de kaart **childattr** is.		
validation (integration > targetcis > source_ci_type > target_attribute > map -OF- integration > targetcis > source_ci_type_tree > target_attribute > map -OF- (integration > targetrelations > link > target_attribute > map -OF- integration > targetrelations > source_link_type_tree > target_attribute > map) Alleen geldig wanneer het type van de kaart **childattr** is.	Hiermee is uitfilteren mogelijk van de onderliggende CI's van het bron-CI op basis van attribuutwaarden. Wordt gebruikt met het aggregatiesubelement om granulariteit te bereiken van exact welke onderliggende attributen aan de attribuutwaarde van het doel-CIT worden toegewezen. Optioneel.	1. **Naam.** minlength **Beschrijving.** Hiermee worden strings uitgesloten die korter zijn dan de opgegeven waarde **Is verplicht?:** Niet vereist **Type.** Geheel getal 2. **Naam.** maxlength **Beschrijving.** Hiermee worden strings uitgesloten die langer zijn dan de opgegeven waarde **Is verplicht?:** Niet vereist **Type.** Geheel getal 3. **Naam.** minvalue **Beschrijving.** Hiermee worden getallen uitgesloten die kleiner zijn dan de opgegeven waarde **Is verplicht?:** Niet vereist **Type.** Numeriek 4. **Naam.** maxvalue **Beschrijving.** Hiermee worden getallen uitgesloten die groter zijn dan de opgegeven waarde **Is verplicht?:** Niet vereist **Type.** Numeriek
targetrelations (integration)	Containerelement voor alle relatietoewijzingen. Optioneel.	
source_link_type_tree (integration >	Hiermee wordt een bronrelatietype zonder de typen die daarvan	1. **Naam:** name **Beschrijving.** Naam bronrelatie **Is required?:**Required

Naam en pad van element	Beschrijving	Attributen
targetrelations)	overnemen, toegewezen aan een doelrelatie. Alleen verplicht als **targetrelation** aanwezig is	**Type.** String 2. **Naam:** target_link_type **Beschrijving.** Naam doelrelatie **Is verplicht?:** Vereist **Type.** String 3. **Naam:** nameSpace **Beschrijving:** De naamruimte voor de koppeling die op het doel wordt aangemaakt **Is verplicht?:** Niet vereist **Type:** String 4. **Naam:** mode **Beschrijving:** Het bijwerkingstype dat is vereist voor de huidige koppeling **Is verplicht?:** Vereist **Type:** Een van de volgende strings: ▪ **insert** – Gebruik deze string alleen als het CI nog niet bestaat ▪ **update** – Gebruik deze string alleen als van het CI bekend is dat het bestaat ▪ **update_else_insert** – Als het CI bestaat, werkt u het bij. Anders maakt u een nieuw CI aan ▪ **ignore** – Doe niets met dit CI-type 5. **Naam:** source_ci_type_end1 **Beschrijving:** CI-type End1 van bronrelatie. **Is verplicht?:** Vereist **Type:** String 6. **Naam:** source_ci_type_end2 **Beschrijving:** CI-type End2 van bronrelatie. **Is verplicht?:** Vereist **Type:** String
link (integration > targetrelations)	Hiermee wordt een bronrelatie aan een doelrelatie toegewezen. Alleen verplicht als **targetrelation** aanwezig is	1. **Naam:** source_link_type **Beschrijving:** Naam bronrelatie **Is verplicht?:** Vereist **Type:** String 2. **Naam:** target_link_type **Beschrijving:** Naam doelrelatie

Naam en pad van element	Beschrijving	Attributen
		Is verplicht?: Vereist **Type:** String 3. **Naam:** nameSpace **Beschrijving:** De naamruimte voor de koppeling die op het doel wordt aangemaakt **Is verplicht?:** Niet vereist **Type:** String 4. **Naam:** mode **Beschrijving:** Het bijwerkingstype dat is vereist voor de huidige koppeling **Is verplicht?:** Vereist **Type:** Een van de volgende strings: ▪ **insert** – Gebruik deze string alleen als het CI nog niet bestaat ▪ **update** – Gebruik deze string alleen als van het CI bekend is dat het bestaat ▪ **update_else_insert** – Als het CI bestaat, werkt u het bij. Anders maakt u een nieuw CI aan ▪ **ignore** – Doe niets met dit CI-type 5. **Naam:** source_ci_type_end1 **Description:** CI-type End1 van bronrelatie **Is verplicht?:** Vereist **Type:** String 6. **Naam:** source_ci_type_end2 **Beschrijving:** CI-type End2 van bronrelatie **Is verplicht?:** Vereist **Type:** String
target_ci_type_end1 (integration > targetrelations > link -OF- integration > targetrelations > source_link_type_ tree)	CI-type End1 van doelrelatie.	1. **Naam:** name **Beschrijving:**Naam van het CI-type End1 van de doelrelatie **Is verplicht?:** Vereist **Type:** String 2. **Naam:** superclass **Beschrijving:** Naam van de superklasse van het CI-type End1 **Is verplicht?:** Niet vereist **Type:** String

Naam en pad van element	Beschrijving	Attributen
target_ci_type_end2 (integration > targetrelations > link -OF- integration > targetrelations > source_link_type_ tree)	CI-type End2 van doelrelatie.	1. **Naam:** name **Beschrijving:** Naam van het CI-type End2 van de doelrelatie **Is verplicht?:** Vereist **Type:** String 2. **Naam:** superclass **Beschrijving:** Naam van de superklasse van het CI-type End2 **Is verplicht?:** Niet vereist **Type:** String

Schema toewijzingsresultaten

Naam en pad van element	Beschrijving	Attributen
root	Het beginpunt van het resultaatdocument.	
data (root)	Het beginpunt van de gegevens zelf.	
objects (root > data)	Het hoofdelement voor de objecten die moeten worden bijgewerkt.	
Object (root > data > objects)	Hiermee wordt het bijwerken beschreven voor één object en alle bijbehorende attributen.	1. **Naam:** name **Beschrijving:** Naam van het CI-type **Is verplicht?:** Vereist **Type:** String 2. **Naam:** mode **Beschrijving:** Het bijwerkingstype dat is vereist voor het huidige CI-type **Is verplicht?:** Vereist **Type:** Een van de volgende strings: a. **insert** – Gebruik deze string alleen als het CI nog niet bestaat. b. **update** – Gebruik deze string alleen als van het CI bekend is dat het bestaat. c. **update_else_insert** – Als het CI bestaat, werkt u het bij. Anders maakt u een nieuw CI aan. d. **ignore** – Doe niets met dit CI-type.

Naam en pad van element	Beschrijving	Attributen
		3. **Naam:** operation **Beschrijving:** De bewerking die met dit CI moet worden uitgevoerd **Is verplicht?:** Vereist **Type:** Een van de volgende strings: a. **add** – Het CI moet worden toegevoegd b. **update** – Het CI moet worden bijgewerkt c. **delete** – Het CI moet worden verwijderd Als geen waarde wordt ingesteld, wordt de standaardwaarde van **add** gebruikt. 4. **Naam:** mamId **Beschrijving:** De ID van het object in de bron-CMDB **Is verplicht?:** Vereist **Type:** String
field (root > data > objects> Object -OF- root > data > links > link)	Hiermee wordt de waarde van één veld voor een object beschreven. De tekst van het veld is de nieuwe waarde in het veld en als het veld een koppeling bevat, is de waarde de ID van een van de einden. Elk eind-ID wordt als een object weergegeven (onder <objects>).	1. **Naam:** name **Beschrijving:** Naam van het veld **Is verplicht?:** Vereist **Type:** String 2. **Naam:** key **Beschrijving:** Hiermee wordt opgegeven of dit veld een sleutel is voor het object **Is verplicht?:** Vereist **Type:**Boolean 3. **Naam:** datatype **Beschrijving:** Het type van het veld **Is verplicht?:** Vereist **Type:** String 4. **Naam:** length **Beschrijving:** Voor gegevenstypen string/teken is dit de grootte van een geheel getal van het doelattribuut **Is verplicht?:**Niet vereist **Type:** Geheel getal
links (root > data)	Het hoofdelement voor de koppelingen die moeten worden bijgewerkt.	1. **Naam:** targetRelationshipClass **Beschrijving:** De naam van de relatie (koppeling) in het doelsysteem **Is verplicht?:** Vereist **Type:** String 2. **Naam:** targetParent

Naam en pad van element	Beschrijving	Attributen
		Beschrijving: Het type van het eerste eind van de koppeling (bovenliggend) **Is verplicht?:** Vereist **Type:** String 3. **Naam:** targetChild **Beschrijving:** Het type van het tweede eind van de koppeling (onderliggend) **Is verplicht?:** Required **Type:** String 4. **Naam:** mode **Beschrijving:** Het bijwerkingstype dat is vereist voor het huidige CI-type. **Is required?:** Vereist **Type:** Een van de volgende strings: a. **insert** – Gebruik deze string alleen als het CI nog niet bestaat. b. **update** – Gebruik deze string alleen als van het CI bekend is dat het bestaat. c. **update_else_insert** – Als het CI bestaat, werkt u het bij. Anders maakt u een nieuw CI aan. d. **ignore** – Doe niets met dit CI-type. 5. **Naam:** operation **Beschrijving:** De bewerking die met dit CI moet worden uitgevoerd **Is verplicht?:** Vereist **Type:** Een van de volgende strings: a. add – Het CI moet worden toegevoegd b. update – Het CI moet worden bijgewerkt c. delete – Het CI moet worden verwijderd Als geen waarde wordt ingesteld, wordt de standaardwaarde van **add** gebruikt. 6. **Naam:** mamId **Beschrijving:** De ID van het object in de bron-CMDB **Is verplicht?:** Vereist **Type:** String

Hoofdstuk 7

Uitgebreide algemene push-adapters ontwikkelen

In dit hoofdstuk vindt u de volgende informatie:

Uitgebreide push-adapters ontwikkelen - overzicht

Een uitgebreide push-adapter is een gegevensstructuur die het TQL-queryresultaat voorstelt. Met elke adapter die op basis van de uitgebreide push-adapter is gemaakt, wordt deze gegevensstructuur verwerkt en naar het vereiste doel gepusht.

De gegevensstructuur wordt **ResultTreeNode (RTN)** genoemd. De RTN wordt aangemaakt op basis van het toewijzingsbestand van de adapter en de resultaten van de TQL-query. De query's voor de uitgebreide push-adapter moeten op een hoofdelement zijn gebaseerd. Dat wil zeggen dat de query één queryknooppunt moet bevatten met elementnaam **root** of een of meer relatie-elementen die beginnen met het voorvoegsel **root**. Dit CI of deze relatie dient als het hoofdelement van de query. Zie "Datapush" in de *HP Universal CMDB – Handleiding Data Flow Management* voor meer informatie over dit onderwerp.

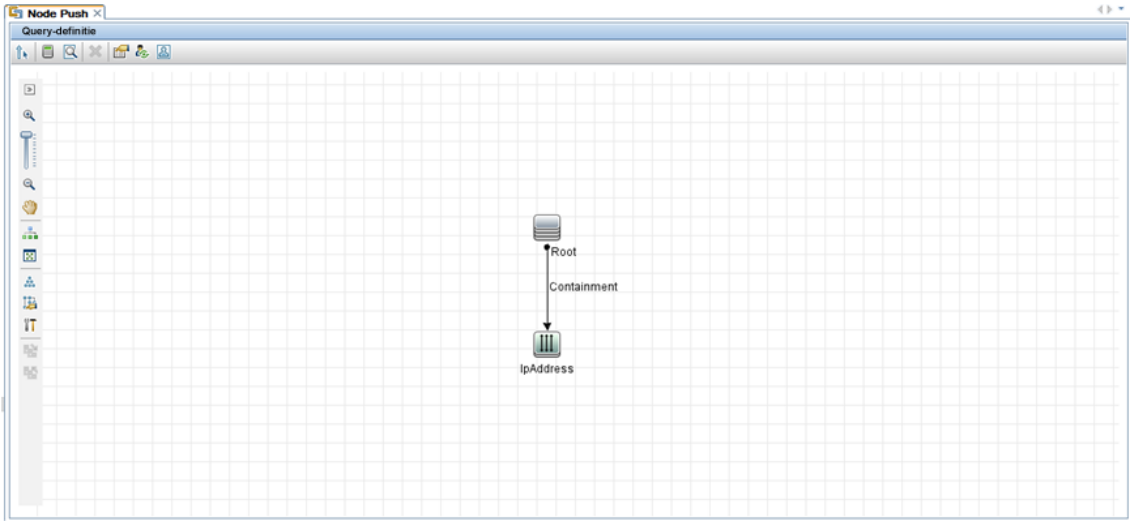
Er zijn twee basisstappen betrokken bij het ontwikkelen van een uitgebreide push-adapter:

1. De PushConnector-interface implementeren – met deze interface worden de gegevens ontvangen die moeten worden toegevoegd, bijgewerkt en verwijderd als een lijst met RTN's en wordt de push naar het doel uitgevoerd.
2. Het toewijzingsbestand aanmaken – met het toewijzingsbestand wordt het aanmaken van de RTN-structuur bepaald door CI's en attributen van het TQL-resultaat toe te wijzen.

Toewijzingsbestand

Met het volgend voorbeeld wordt getoond hoe het toewijzingsbestand moet worden aangemaakt.

In dit voorbeeld wordt een push van knooppunt en IP-adres gesimuleerd. We maken als volgt een TQL-query aan met de naam: **Node Push**, als volgt:



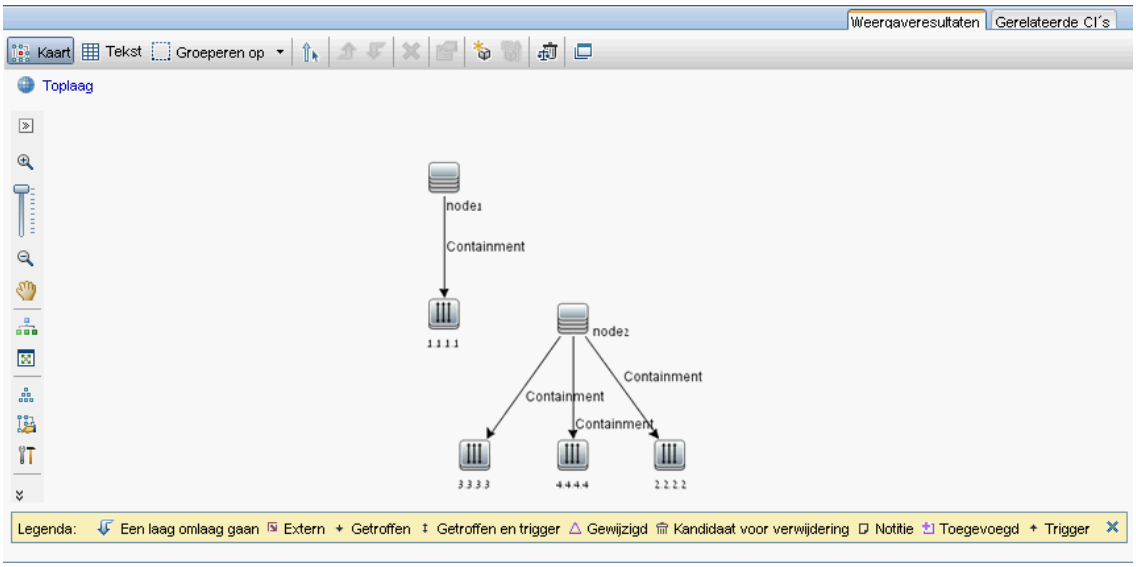
In het toewijzingsbestand maken we twee doel-CI-typen aan: **Computer** en **IP**. Computer heeft één variabele en twee attributen. IP heeft één attribuut.

Het volgende attribuut is het XML-toewijzingsbestand:

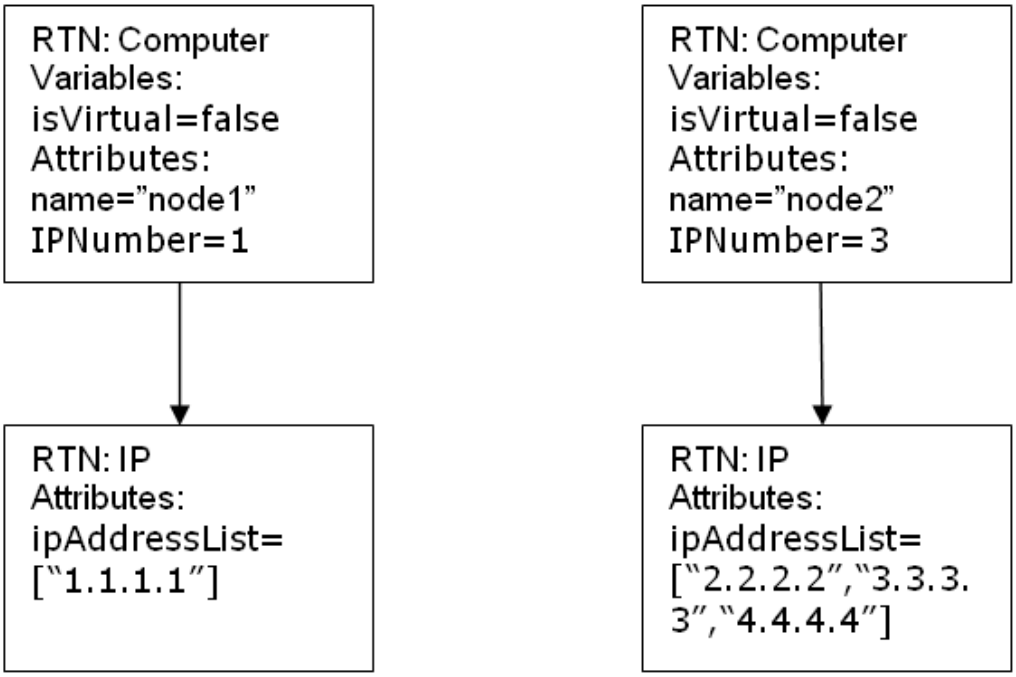
```
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://schemas.hp.com/ucmdb/1/pushAdap
ter">
        <info>
            <source name="UCMDB" versions="10.0" vendor="HP" />
            <target name="PushProduct" versions="9.3" vendor="HP" />
        </info>
        <import>
            <!—imports the Groovy script file. -->
            <scriptFile path="mappings.scripts.PushFunctions" />
        </import>
<targetcis>
<source_instance_type query-name="Node Push" root-element-name="Root">
 <target_ci_type name="Computer" is-
    valid="(Root['root_iscandidatefordeletion']==null) ? true :
    !Root['root_iscandidatefordeletion']">
  <variable name="isVirtual" datatype="BOOLEAN"
    value="PushFunctions.isVirtual(Root['root_class']) " />

  <target_mapping name="name" datatype="String" value="Root[`name’] " />
  <target_mapping name="ipNumber" datatype="INTEGER"
    value="Root.IpAddress.size()" />
  <target_mapping name="description" datatype="STRING" value=" PushFunctions
    .getDescription(isVirtual)" />
  <target_ci_type name="IP">
    <target_mapping name="ipAddressList" datatype="STRING_LIST"
    value="Root.IpAddress*.getAt(`name’) " />
</target_ci_type>
</target_ci_type>
</targetcis>
</integration>
```

De queryresultaten worden als volgt weergegeven:



Hier is de RTN-lijst op basis van dit toewijzingsbestand gemaakt:



Elk hoofdexemplaar wordt afzonderlijk met het toewijzingsbestand toegewezen. In dit voorbeeld ontvangt de PushConnector dus een lijst met twee RTN-hoofdelementen.

Opmerking: De vorige push-adapter had de mogelijkheid om een algemene toewijzing voor een CI-type aan te maken. De nieuwe push-adaptertoewijzing is per TQL-query. Tijdens de uitvoering van een pushtaak waarin een query met de naam **x** wordt gebruikt, zoekt de adapter naar het relevante toewijzingsbestand (het bestand dat het volgende attribuut heeft: query-name=x).

U kunt de waarden in het toewijzingsbestand berekenen met groovy-scripttaal. Zie "Groovy Traveler" beneden voor meer informatie over dit onderwerp.

Groovy Traveler

Op de volgende manier kan toegang tot de TQL-queryresultaten worden verkregen:

- Met **Root[attr]** wordt het attribuut **attr** van het hoofdelement geretourneerd.
- Met **Root.Query_Element_Name** wordt een lijst geretourneerd met de CI-exemplaren met de naam Query_Element_Name in TQL en die aan het huidige hoofd-CI worden gekoppeld.
- Met **Root.Query_Element_Name[2][attr]** wordt het attribuut **attr** van de derde Query_Element_Name geretourneerd die aan het huidige hoofd-CI wordt gekoppeld.
- Met **Root.Query_Element_Name*.getAt(attr)** wordt een lijst geretourneerd met de attributen **attr** van de CI-exemplaren met de naam Query_Element_Name in TQL en die aan het huidige hoofd-CI worden gekoppeld.

Er zijn aanvullende attributen die door de Groovy Traveler kunnen worden aangeroepen:

- **cmdb_id** – retourneert de UCMDB-ID van het CI of de relatie als een string.
- **external_cmdb_id** – retourneert de externe ID van het CI of de relatie als een string.
- **Element_type** – retourneert het elementtype van het CI of de relatie als een string.

De tag import:

```
<import>

<scriptFile path="mappings.scripts.PushFunctions"/>

</import>
```

Dit betekent dat we een import declareren voor alle groovy-scripts in het toewijzingsbestand. In dit voorbeeld is **PushFunctions** een groovy-scriptbestand dat bepaalde statische functies bevat, waartoe toegang kan worden verkregen tijdens de toewijzing (dat wil zeggen: value=" PushFunctions.foo()")

source_instance_type

De toewijzing vindt per TQL plaats en de waarde query-name is de gerelateerde TQL van de huidige toewijzing. '*' betekent dat dit toewijzingsbestand is gekoppeld aan alle TQL-query's die beginnen met het voorvoegsel: **Node Push**.

```
<source_instance_type query-name="Node Push*" root-element-
name="Root">
```

Met de tag source_instance_type wordt het hoofdelement aangeduid dat we toewijzen.

root-element-name moet exact hetzelfde zijn als de naam van het hoofdelement in de TQL.

target_ci_type

Deze tag wordt gebruikt om de RTN aan te maken.

Met het naamattribuut wordt de naam target_ci_type voorgesteld: name=Computer

Het attribuut **is-valid** is een Boolean waarde die tijdens de toewijzing wordt berekend als het huidige target_ci geldig is. Ongeldige target_ci_types worden niet aan de RTN toegevoegd. In dit voorbeeld willen we geen exemplaar van target_ci_type maken waarvoor het attribuut **root_ iscandidatefordeletion** in UCMDB waar is.

target_ci_type kan variabelen hebben die tijdens de toewijzing worden berekend:

```
<variable name="vSerialNo" datatype="STRING" value="Root['serial_
number']"/>
```

De variabele **vSerialNo** krijgt de waarde van **serial_number** van het huidige hoofdelement.

Het attribuut van de RTN wordt aangemaakt door de tag **target_mapping**. Het resultaat van de uitvoering van het groovy-script in het veld **value**, wordt aan het RTN-attribuut toegewezen.

```
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
```

Met **SerialNo** wordt de waarde van de variabele **vSerialNo** toegewezen.

Een target_ci_type kan als volgt worden gedefinieerd als onderliggende waarde van een ander target_ci_type:

```
<target_ci_type name="Portfolio">

<variable name="vSerialNo" datatype="STRING" value="Root['global_id']
"/>

<target_mapping name="CMDBId" datatype="STRING" value="globalId"/>

<target_ci_type name=Asset">

<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>

</target_ci_type>

</target_ci_type>
```

De RTN **Portfolio** heeft een onderliggende RTN met de naam **Asset**.

for-each-source-ci

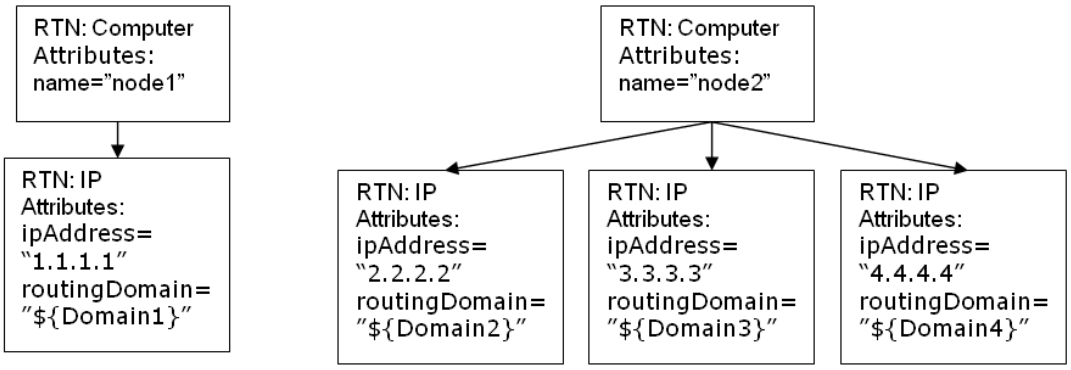
Met deze tag worden de specifieke CI's van het hoofdexemplaar weergegeven. Het heeft de volgende velden:

- source-cis=" " – de lijst met CI's waarvoor een doel-CI wordt aangemaakt. Deze lijst wordt door de Groovy Traveler in het veld **Root.IpAddress** aangemaakt.
- count-index=" " – een variabele die de index van het CI in de huidige herhaling van de lus bevat.
- var-name=" " – de naam van het CI in de huidige herhaling van de lus.

We gaan nu het voorbeeldtoewijzingsbestand wijzigen:

```
<target_ci_type name="Computer">
  <target_mapping name="name" datatype="String" value="Root[`name'] " />
   <for-each-source-ci count-index="i" var-name="currIP" source-cis="Root.IpAddress" >
     <target_ci_type name="IP">
         <target_mapping name="ipAddress" datatype="STRING"
     value="Root.IpAddress[i][`name']"/>
         <target_mapping name="routingDomain" datatype="STRING"
     value="currIP['routing_domain']"/>
   </target_ci_type>
 </for-each-source-ci>
 </target_ci_type>
```

De RTN-lijst die op basis van dit toewijzingsbestand wordt aangemaakt, ziet er als volgt uit:



dynamic_mapping

Met deze tag wordt de mogelijkheid toegevoegd om een toewijzing van gegevens vanuit de doelgegevensopslag aan te maken tijdens het aanmaken van de RTN-structuur.

Voorbeeld: Stel dat het doel een database is met een tabel met de naam **Computer** die de kolom **id** bevat en de kolom **name** die aan **Node.name** in UCMDB is gecorreleerd. Beide kolommen zijn uniek. De database heeft ook een tabel met de naam **IP** die een sleutel heeft die verwijst naar de **parentID** in de tabel Computer. Met de 'dynamic_mapping' kan een toewijzing worden aangemaakt die de naam en ID opslaat als <name,id>. Op basis van deze toewijzing kan de Adapter ID's vergelijken met computers en kan de juiste waarde naar het attribuut parentID in de tabel IP worden gepusht. U kunt deze toewijzing gebruiken om een waarde toe te wijzen aan het attribuut parentID tijdens het aanmaken van de RTN.

De toewijzing wordt bepaald door de **map_property**. De dynamic_mapping wordt eenmaal voor elk segment uitgevoerd.

```
<dynamic_mapping name="IdByName " keys-unique="true">
```

Het attribuut **name** stelt de naam van de toewijzing voor. Met het attribuut **keys-unique** wordt aangegeven of de sleutels uniek zijn (elke sleutel wordt aan één waarde toegewezen of aan een set waarden).

De naam van de toewijzing in dit voorbeeld is **IdByName** en heeft unieke sleutels. Om toegang te krijgen tot de toewijzing in het script, voert u de volgende opdracht uit:

```
DynamicMapHolder.getMap('IdByName')
```

Er wordt een verwijzing naar die toewijzing geretourneerd.

Met de tag **map_property** wordt de eigenschap aangemaakt waarop de toewijzing wordt gebaseerd.

Voorbeeld:

```
<map_property property-name="SQLQuery" datatype="STRING"
property-value="SELECT name, id FROM Computer"/>
```

In dit voorbeeld is de naam van de eigenschap **SQLQuery** en is de waarde ervan een SQL-instructie waarmee de toewijzing wordt gemaakt. Met de implementatie van de methoden **retrieveUniqueMapping** en **retrieveNonUniqueMapping** voor de PushConnector-interface wordt de werkelijke inhoud van de geretourneerde toewijzing bepaald.

Globale variabelen

De volgende globale variabelen zijn toegankelijk voor het groovy-script in het toewijzingsbestand:

- **Topology** – Type: Topologie. Een exemplaar van de topologie van het huidige segment.
- **QueryDefinition** - Type: QueryDefinition. Een exemplaar van de querydefinitie van de huidige TQL.
- **OutputCI** – Type: ResultTreeNode. De RTN van het hoofdelement in de huidige structuurtoewijzing.
- **ClassModel** – Type: ClassModel. Een exemplaar van het klassemodel.
- **CustomerInformation** – Type: CustomerInformation. Informatie over de klant die de taak uitvoert.
- **Logger** – Type: DataAdapterLogger. Dit logprogramma is beschikbaar voor het schrijven van logboeken naar het UCMDB-logframework.

Groovy-scripts schrijven

In dit gedeelte maken we het bestand **PushFunctions.groovy**. Dit bestand bevat statische functies die worden gebruikt tijdens de toewijzing van het hoofdexemplaar.

```
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.
        }
}
```

PushConnector-interface implementeren

De implementatie moet de volgende basisstappen ondersteunen:

```
public class PushExampleAdapter implements PushConnector

{

public UpdateResult pushTreeNodes(ResultTreeNodeActionData
resultTreeNodes, QueryDefinition queryDefinition) throws
DataAccessException{

// 1. build an UpdateResult instance - the UpdateResult is used to
return mappings between the sent ids to the actual ids that entered
the data store.
// Also has an update status which allows to pass errors to the
statistics in the UI.
// 2. handle the data:
// a. handle data to add. Can be retrieved by:
resultTreeNodeActionData.getDataToAdd();
//  b. handle data to update.
//  c. handle data to delete.
// 3. Return the Update result.
        }

public void start(PushDataAdapterEnvironment env) throws
DataAccessException{
        // this method is called when the integration point created, or when
the adapter is reloaded
        //(i.e after changing one of the mapping files
        // and pressing 'save').

}



public void testConnection(PushDataAdapterEnvironment env) throws
DataAccessException {
                // this method is called when pressing the 'test connection' button
in the
                //creation of the integration point.
                // For example if we push data to RDBMS this method can create a
connection
                //to the database and will run a dummy SQL statement.
```

```
                // If it fails it writes an error message to the log and throws an
exception.
       }



Map<Object, Object> retrieveUniqueMapping(MappingQuery mappingQuery){
//This method will create the map according to the given mappingQuery.
It will be called in the
// mapping stage of the adapter execution, before the 'UpdateResult
pushTreeNodes' method.
// This method is called when the 'keys-unique' attribute of the
'dynamic_mapping' tag is true.
}

Map<Object, Set<Object>> retrieveNonUniqueMapping(MappingQuery
mappingQuery){
 // This method is called when the 'keys-unique' attribute of the
'dynamic_mapping' tag is false.
//  In this case a key can be mapped to several values.
}
}
```

Adapterpakketten samenstellen

Het adapterpakket moet de volgende mappen bevatten:

- **adapterCode.** Maak onder deze map een map met de naam **PushExampleAdapter**, die het .jar-bestand bevat dat is gemaakt op basis van PushExampleAdapter.java. Deze map bevat ook een map met de naam **mappings**. Hierin kunt u het eerder gemaakte toewijzingsbestand **computerIPMapping.xml** plaatsen. De map bevat ook een andere map met de naam **scripts** die het bestand **PushFunctions.groovy** bevat.
- **discoveryConfigFiles.** Voor configuratiebestanden zoals de foutcodes die worden gebruikt bij het rapporteren van een fout met UpdateResult. In dit voorbeeld is de map leeg.
- **discoveryPatterns.** Voor **push_example_adapter.xml**.
- **tql.** Voor de TQL-query die voor het voorbeeld is gemaakt. Deze map is optioneel, maar wanneer het pakket wordt uitgerold, wordt de TQL automatisch aangemaakt.

Schema toewijzingsbestand

Naam en pad van element	Beschrijving	Attributen
Integratie	Hiermee wordt de toewijzingsinhoud van het bestand gedefinieerd. Moet het buitenste blok in het bestand zijn, met uitzondering van de beginregel en opmerkingen.	

Naam en pad van element	Beschrijving	Attributen
info (integration)	Hiermee wordt informatie gedefinieerd over de gegevensopslagplaatsen die worden geïntegreerd.	
source (info)	Het bronproduct	**Naam:** name **Beschrijving:** de naam van het product **Is vereist?:** vereist **Type:** String
		Naam: vendor **Beschrijving:** de leverancier van het product **Is vereist?:** vereist **Type:** String
		Naam: versions **Beschrijving:** de versie van het product **Is vereist?:** vereist **Type:** decimaal
target (info)	Het doelproduct	**Naam:** name **Beschrijving:** de naam van het product **Is vereist?:** vereist **Type:** String
		Naam: vendor **Beschrijving:** de leverancier van het product **Is vereist?:** vereist **Type:** String
		Naam: versions **Beschrijving:** de versie van het product **Is vereist?:** vereist

Naam en pad van element	Beschrijving	Attributen
		Type: decimaal
Import (integration)	Een containerelement voor geïmporteerde scriptbestanden.	
scriptFile (integration>import)	Hiermee wordt het te importeren groovy-scriptbestand gedefinieerd.	Naam: padbeschrijving: het pad van het scriptbestand Is vereist?: vereist Type: string
Targetcis (integration)	Een containerelement voor doel-CI-typen.	
Source_instance_type (integration>targetcis)	Hiermee wordt het CI-exemplaartype van de bron gedefinieerd. Moet het hoofdelement zijn zoals is gedefinieerd in de TQL.	**Naam:** query-name. **Beschrijving:** De naam van de relevante TQL **Is vereist?:** vereist **Type:** String
		Naam: name **Beschrijving:** Het hoofdelement zoals is gedefinieerd in de TQL. **Is vereist?:** vereist **Type:** String
dynamic_mapping (integration>targetcis> source_instance_type)	Hiermee wordt een toewijzing gedefinieerd die eenmaal per segment wordt aangemaakt.	**Naam:** name **Beschrijving:** de naam van de toewijzing **Is vereist?:** vereist **Type:** String
		Naam: keys-unique **Beschrijving:** Geeft aan of de sleutels uniek zijn of niet. **Is vereist?:** vereist **Type:** boolean

Naam en pad van element	Beschrijving	Attributen
target_ci_type (integration>targetcis> source_instance_type) -OF- (integration>targetcis> source_instance_type> for-each-source-ci)	Hiermee wordt het doel-CI-type gedefinieerd dat aan de RTN moet worden toegevoegd.	**Naam:** name **Beschrijving:** De naam van het doel-CI-type **Is vereist?:** vereist **Type:** String
		Naam: is-valid **Beschrijving:** controleert of het huidige doel-CI geldig is op basis van het gegeven script. **Is vereist?:** niet vereist **Type:** String (groovy-script)
Variabele (target_ci_type)	Hiermee wordt een variabele gedefinieerd voor het doel-CI-type.	**Naam:** name **Beschrijving:** de naam van de variabele **Is vereist?:** vereist **Type:** String
		Naam: datatype **Beschrijving:** het gegevenstype van de variabele. **Is vereist?:** vereist **Type:** type-enum (kan een van de volgende waarden zijn: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_ LIST, STRING_LIST)
		Naam: value **Beschrijving:** De

Naam en pad van element	Beschrijving	Attributen
		waarde die aan de variabele moet worden toegewezen **Is vereist?:** vereist **Type:** groovy-script
target_mapping (target_ci_type)	Hiermee wordt een attribuut gedefinieerd voor het doel-CI-type.	**Naam:** name **Beschrijving:** de naam van het attribuut **Is vereist?:** vereist **Type:** String
		Naam: datatype **Beschrijving:** het gegevenstype van de variabele. **Is vereist?:** vereist **Type:** type-enum (kan een van de volgende waarden zijn: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Naam: value **Beschrijving:** De waarde die aan de variabele moet worden toegewezen **Is vereist?:** vereist **Type:** groovy-script
		Naam: ignore-on-null **Beschrijving:** Als de doeltoewijzingswaarde null is en dit attribuut is TRUE, moet u het attribuut negeren **Is vereist?:** niet

Naam en pad van element	Beschrijving	Attributen
		vereist **Type:** Boolean (groovy-script)
before-mapping (target_ci_type)	Een groovy-script dat wordt uitgevoerd vóór de toewijzing van het doel-CI-type.	
after-mapping (target_ci_type)	Een groovy-script dat wordt uitgevoerd na de toewijzing van het doel-CI-type.	
for-each-source-ci (target_ci_type)	Hiermee wordt een herhaling van specifieke CI's van het hoofdexemplaar gedefinieerd.	**Naam:** count-index **Beschrijving:** de index van het huidige herhaalde CI **Is vereist?:** vereist **Type:** String
		Naam: var-name **Beschrijving:** de variabele waarmee naar het huidige herhaalde CI wordt verwezen. **Is vereist?:** niet vereist **Type:** String
		Naam: source-cis **Beschrijving:** De CI-naam in de query die moet worden herhaald **Is vereist?:** vereist **Type:** String (groovy-script)

API's gebruiken

Hoofdstuk 8

Inleiding tot API's

In dit hoofdstuk vindt u de volgende informatie:

Overzicht van API's

De volgende API's worden meegeleverd bij HP Universal CMDB:

- **UCMDB Java API.** Hiermee wordt uitgelegd hoe eigen tools of tools van derden met de Java API gegevens en berekeningen kunnen extraheren en gegevens naar de UCMDB (Universal Configuration Management Database) kunnen schrijven. Zie "HP Universal CMDB API" op pagina 222 voor meer informatie over dit onderwerp.
- **UCMDB Web Service API.** Hiermee kunnen configuratie-itemdefinities en topologische relaties worden geschreven naar de UCMDB (Universal Configuration Management Database) en kunnen query's worden uitgevoerd op de informatie met TQL-query's en ad-hoc query's. Zie "HP Universal CMDB Web Service API" op pagina 230 voor meer informatie over dit onderwerp.
- **Data Flow-beheer Web Service API.** Hiermee kunt u probes, taken, triggers en referenties voor Data Flow-beheer beheren. Zie "API Data Flow-beheer" op pagina 288 voor meer informatie over dit onderwerp.

Opmerking: Om de waarde van de API-documentatie ten volle te benutten, is het raadzaam gebruik te maken van de online documentatie. De PDF-versie biedt niet de koppelingen naar de API-documentatie die in HTML-indeling wordt gegenereerd.

Hoofdstuk 9

HP Universal CMDB API

In dit hoofdstuk vindt u de volgende informatie:

Conventies

In dit hoofdtuk worden de volgende conventies gehanteerd:

- **UCMDB** verwijst naar de Universal Configuration Management Database zelf. HP Universal CMDB verwijst naar de applicatie.
- Voor elementen en methode-argumenten in UCMDB worden hoofdletters en kleine letters op dezelfde manier als in de interfaces gebruikt.

Zie HP UCMDB API Reference voor volledige documentatie van de beschikbare API's.

De bestanden bevinden zich in de volgende map:

\\<UCMDB-hoofdmap>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

HP Universal CMDB API gebruiken

Opmerking: Gebruik dit hoofdstuk in combinatie met de API Javadoc, die beschikbaar is in de online Documentatiebibliotheek.

De HP Universal CMDB API wordt gebruikt om applicaties te integreren met de Universal CMDB (CMDB). Met de API worden methoden verschaft voor het volgende:

- CI's en relaties in de CMDB toevoegen, verwijderen en bijwerken
- Informatie over het klassemodel ophalen
- informatie uit de UCMDB-geschiedenis ophalen

- What-if-scenario's uitvoeren
- Informatie over configuratie-items en relaties ophalen

Voor methoden voor het ophalen van informatie over configuratie-items en relaties wordt over het algemeen de TQL (Topology Query Language) gebruikt. Zie "Topology Query Language" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.

Gebruikers van de HP Universal CMDB API moeten vertrouwd zijn met het volgende:

- De Java-programmeertaal
- HP Universal CMDB

In dit gedeelte vindt u de volgende onderwerpen:

- "Toepassingen van de API" beneden
- "Rechten" beneden

Toepassingen van de API

De API wordt gebruikt om aan een aantal bedrijfsvereisten te voldoen. Met een systeem van derden kan bijvoorbeeld een query worden uitgevoerd op het klassemodel voor informatie over beschikbare configuratie-items (CI's). Zie "Use cases" op pagina 227 voor meer use cases.

Rechten

De beheerder verstrekt aanmeldingsgegevens om verbinding te maken met de API. Voor de API-client zijn de gebruikersnaam en het wachtwoord van een integratiegebruiker vereist die is gedefinieerd in de CMDB. Deze gebruikers zijn geen echte mensen die de CMDB gebruiken, maar zijn applicaties die verbinding maken met de CMDB.

> **Let op:** De API-client kan ook met normale gebruikers werken zo lang deze over have API-verificatierechten beschikken. Deze optie wordt echter niet aanbevolen.

Zie "Integratiegebruikers aanmaken" op pagina 225 voor meer informatie over dit onderwerp.

Algemene structuur van een applicatie

Er is slechts één statische fabrieksinstelling, de UcmdbServiceFactory. Deze fabrieksinstelling vormt het beginpunt voor een applicatie. Met de UcmdbServiceFactory worden getServiceProvider-methoden beschikbaar gesteld. Met deze methoden wordt een exemplaar van de interface **UcmdbServiceProvider** geretourneerd.

De client maakt andere objecten met interfacemethoden aan. Om bijvoorbeeld een nieuwe query-definitie aan te maken kan de client het volgende doen:

1. De queryservice van het CMDB-hoofdserviceobject ophalen
2. Een query-fabrieksobject van het serviceobject ophalen
3. Een nieuwe query-definitie van de fabrieksinstellingen ophalen

```
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");
```

Dit zijn de services die beschikbaar zijn via **UcmdbService**:

Servicemethoden	Toepassing
getClassModelService	Informatie over typen CI's en relaties
getConfigurationService	Beheer van infrastructuurinstellingen, voor serverconfiguratie
getDDMConfigurationService	Het systeem van Data Flow-beheer configureren
getDDMManagementService	De voortgang, resultaten en fouten van het Data Flow-beheersysteem analyseren en bekijken
getHistoryService	Informatie over geschiedenis van gecontroleerde CI's (wijzigingen, verwijderingen, enzovoort)
getImpactAnalysisService	Impactanalysescenario uitvoeren (ook bekend als **correlatie**)
getQueryManagementService	Toegang tot query's beheren: query's opslaan en verwijderen en bestaande query's weergeven. Biedt ook query-validatie en onderzoekt discovery van afhankelijkheden.
getResourceBundleManagementService	Bronnen voorzien van label (bundelservices). Maakt ook expliciet aanmaken van nieuwe labels en verwijdering van labels uit alle bronnen met labels mogelijk.
getStateService	Biedt services voor statusbeheer (weergeven, toevoegen, verwijderen, enzovoort)
getSoftwareSignatureService	Software-items definiëren die moeten worden gedetecteerd door het Discovery and Dependency Management-systeem
getSnapshotService	Biedt services voor beheer van momentopnames (ophalen, opslaan, vergelijken, enzovoort)
getTopologyQueryService	Informatie over het IT-universum ophalen

Servicemethoden	Toepassing
getTopologyUpdateService	Informatie in het IT-universum wijzigen
getViewService	Uitvoeringsservice (definitie uitvoeren, opgeslagen uitvoeren) en beheerservice (opslaan, verwijderen, bestaande weergeven) bekijken. Biedt ook weergavevalidatie en discovery van afhankelijkheden.
getViewArchiveService	Archiveringsservices resultaat bekijken. Maakt mogelijk dat het huidige weergaveresultaat wordt opgeslagen en eerder opgeslagen resultaten worden opgehaald.
SystemHealthService	Biedt services voor systeemstatus (elementaire indicatoren voor systeemprestaties, meetgegevens voor capaciteit en beschikbaarheid)

De client communiceert met de server via HTTP.

Jar-bestand van API in het klassepad plaatsen

Voor gebruik van deze API-set is het bestand **ucmdb-api.jar** vereist. U kunt het bestand downloaden door `http://<localhost>:8080` in een webbrowser in te voeren, waarbij `localhost` de computer is waarop UCMDB is geïnstalleerd, en vervolgens te klikken op de koppeling **API Client Download**.

Plaats het jar-bestand in het klassepad alvorens uw applicatie te compileren of uit te voeren.

Opmerking: Om de UCMDB Java API Jar te kunnen gebruiken, moet JRE-versie 6 of hoger zijn geïnstalleerd.

Integratiegebruikers aanmaken

U kunt een exclusieve gebruiker aanmaken voor integraties tussen andere producten en UCMDB. Deze gebruiker kan dan een product inschakelen dat gebruikmaakt van de SDK van de UCMDB-client voor verificatie in de server-SDK en zo de API's uitvoeren. Voor applicaties die met deze API-set zijn geschreven, moet aanmelding plaatsvinden met aanmeldingsgegevens van integratiegebruikers.

Let op: Het is ook mogelijk om verbinding te maken met een normale UCMDB-gebruiker (bijvoorbeeld admin); maar deze optie wordt niet aanbevolen. Om verbinding met een UCMDB-gebruiker te kunnen maken, moet u de gebruiker API-verificatierechten verlenen.

Een integratiegebruiker maken:

1. Start de webbrowser en voer het adres van de server als volgt in:

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

Wellicht moet u zich aanmelden met een gebruikersnaam en wachtwoord (de standaardinstellingen zijn sysadmin/sysadmin).

2. Onder `UCMDB` klikt u op **service=UCMDB Authorization Services**.
3. Zoek naar de bewerking **createUser**. Bij deze methode worden de volgende parameters geaccepteerd:
   - **customerId**. De klant-ID.
   - **username**. De naam van de integratiegebruiker.
   - **userDisplayName**. De weergavenaam van de integratiegebruiker.
   - **userLoginName**. De aanmeldingsnaam van de integratiegebruiker.
   - **wachtwoord**. Het wachtwoord van de integratiegebruiker.
4. Klik op **Aanroepen**.
5. Zoek in een single-tenant omgeving de methode **setRolesForUser** en voer de volgende parameters in:
   - **userName**. De naam van de integratiegebruiker.
   - **roles.** SuperAdmin.

   Klik op **Aanroepen**.
6. Zoek in een a multi-tenant omgeving de methode **grantRolesToUserForAllTenants** en voer de volgende parameters in om de rol toe te wijzen in verbinding met alle tenants:
   - **userName**. De naam van de integratiegebruiker.
   - **roles.** SuperAdmin.

   Klik op **Aanroepen**.

   Om de rol in verbinding met specifieke tenants toe te wijzen, kunt u ook de methode **grantRolesToUserForTenants** aanroepen met dezelfde parameterwaarden voor gebruikersnaam en rollen. Voer de vereiste tenants in voor de parameter **tenantNames**.
7. Maak meer gebruikers aan of sluit de JMX-console.
8. Meld u als beheerder bij de UCMDB aan.
9. Voer **Pakketbeheer** op het tabblad **Beheer** uit.
10. Klik op het pictogram **Aangepast pakket aanmaken**.
11. Voer een naam voor het nieuwe pakket in en klik op **Volgende**.
12. Klik op het tabblad Bronselectie onder **Instellingen** op **Gebruikers**.
13. Selecteer een gebruiker of gebruikers die u met de JMX-console hebt aangemaakt.
14. Klik op **Volgende** en vervolgens op **Voltooien**. Uw nieuwe pakket wordt weergegeven in de lijst Pakketnaam in Pakketbeheer.
15. Rol het pakket uit voor de gebruikers die de API-applicaties zullen uitvoeren.

    Zie "Een pakket uitrollen" in de *HP Universal CMDB – Handleiding Beheer*.

Opmerking:

de integratiegebruiker wordt gedefinieerd op klantbasis. Als u een krachtigere integratiegebruiker wilt maken die bestemd is voor verschillende klanten, gebruikt u een **systemUser** met de vlag **isSuperIntegrationUser** ingesteld op **true**. Gebruik de methoden **systemUser** (**removeUser**, **resetPassword**, **UserAuthenticate**, enzovoort).

Er worden twee systeemgebruikers standaard meegeleverd. Het wordt aanbevolen na de installatie hun wachtwoorden te wijzigen met behulp van de methode **resetPassword**.

- **sysadmin**/**sysadmin**.
- **UISysadmin**/**UISysadmin** (Dit is tevens de Super Integration User **SuperIntegrationUser**).

  Als u het UISysadmin-wachtwoord wijzigt met behulp van **resetPassword**, moet u de volgende methode uitvoeren: Ga in de JMX-console naar de service **UCMDB-UI:name=UCMDB Integration**. Voer **setCMDBSuperIntegrationUser** uit met de gebruikersnaam en het nieuwe wachtwoord van de integratiegebruiker.

HP Universal CMDB API-referentie

voor volledige documentatie van de beschikbare API's.

De bestanden bevinden zich in de volgende map:

\\<UCMDB-hoofdmap>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Use cases

In de volgende use cases wordt uitgegaan van twee systemen:

- HP Universal CMDB-server
- Een systeem van derden dat een opslagplaats van configuratie-items bevat

In dit gedeelte vindt u de volgende onderwerpen:

De CMDB vullen

Use cases:

- Met een assetbeheersysteem van derden wordt de CMDB bijgewerkt met informatie die alleen in assetbeheer beschikbaar is
- Met een aantal systemen van derden wordt de CMDB gevuld om een centrale CMDB aan te maken waarmee wijzigingen kunnen worden bijgehouden en een impactanalyse kan worden

uitgevoerd.

- Met een systeem van derden worden configuratie-items en relaties aangemaakt in overeenstemming met bedrijfslogica van derden om gebruik te maken van de query-mogelijkheden van UCMDB

Query's uitvoeren op de CMDB

Use cases:

- Met een systeem van derden worden de configuratie-items en relaties waarmee het SAP-systeem wordt vertegenwoordigd, opgehaald door de resultaten van de SAP TQL op te halen.
- Met een systeem van derden wordt de lijst met Oracle-servers opgehaald die gedurende de laatste vijf uur zijn toegevoegd of gewijzigd
- Met een systeem van derden wordt de lijst met servers opgehaald waarvan de hostnaam de substring `lab` bevat.
- Met een systeem van derden worden de elementen gezocht die betrekking hebben op een bepaald CI door de bijbehorende buren op te halen

Query uitvoeren op het klassemodel

Use cases:

- Een systeem van derden maakt mogelijk dat gebruikers de set gegevens kunnen opgeven die uit de CMDB moeten worden opgehaald. Er kan een gebruikersinterface worden samengesteld op basis van het klassemodel om gebruikers de mogelijke eigenschappen te tonen en hen om vereiste gegevens te vragen. De gebruiker kan vervolgens de informatie kiezen die moet worden opgehaald.
- Met een systeem van derden wordt het klassemodel onderzocht wanneer de gebruiker geen toegang kan krijgen tot de CMDB-gebruikersinterface.

Wijzigingsimpact analyseren

Use case.

Met een systeem van derden wordt een lijst uitgevoerd van de bedrijfsservices die kunnen worden beïnvloed door een wijziging op een opgegeven host.

Voorbeelden

Zie de volgende codevoorbeelden:

- 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

Deze bestanden bevinden zich in de volgende map:

\\<UCMDB-hoofdmap>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples

Hoofdstuk 10

HP Universal CMDB Web Service API

In dit hoofdstuk vindt u de volgende informatie:

Conventies

In dit hoofdtuk worden de volgende conventies gehanteerd:

- **UCMDB** verwijst naar de Universal Configuration Management Database zelf. HP Universal CMDB verwijst naar de applicatie.
- Voor elementen en methode-argumenten in UCMDB worden hoofdletters en kleine letters op dezelfde manier als in het schema gebruikt. Een element of een argument voor een methode wordt niet in hoofdletters geschreven. Een `relation` is bijvoorbeeld een element van het type `Relation` dat aan een methode wordt doorgegeven.

Raadpleeg de HP UCMDB Web Service API Reference voor volledige documentatie over de aanvraag- en responsstructuren. De bestanden bevinden zich in de volgende map:

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

HP Universal CMDB Web Service API - overzicht

Opmerking: Gebruik dit hoofdstuk in combinatie met de UCMDB-schemadocumentatie, die beschikbaar is in de online Documentatiebibliotheek.

De HP Universal CMDB Web Service API wordt gebruikt om applicaties te integreren met HP Universal CMDB (UCMDB). Met de API worden methoden verschaft voor het volgende:

- CI's en relaties in de CMDB toevoegen, verwijderen en bijwerken
- Informatie over het klassemodel ophalen
- Impactanalysen ophalen
- Informatie over configuratie-items en relaties ophalen
- Referenties beheren: weergeven, toevoegen, bijwerken en verwijderen
- Taken beheren: status weergeven, activeren en deactiveren
- Probe-bereiken beheren: weergeven, toevoegen en bijwerken
- Triggers beheren: een trigger-CI toevoegen of verwijderen en een trigger-TQL toevoegen, verwijderen of uitschakelen
- Algemene gegevens in domeinen en probes weergeven

Voor methoden voor het ophalen van informatie over configuratie-items en relaties wordt over het algemeen de TQL (Topology Query Language) gebruikt. Zie "Topology Query Language" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.

Gebruikers van de HP Universal CMDB Web Service API moeten vertrouwd zijn met het volgende:

- De SOAP-specificatie
- Een objectgeoriënteerde programmeertaal, zoals C++, C# of Java
- HP Universal CMDB
- Data Flow-beheer

In dit gedeelte vindt u de volgende onderwerpen:

Toepassingen van de API

De API wordt gebruikt om aan een aantal bedrijfsvereisten te voldoen. Bijvoorbeeld:

- Een systeem van derden kan een query uitvoeren op het klassemodel voor informatie over beschikbare configuratie-items (CI's).
- Met een tool voor assetbeheer van derden kan de CMDB worden bijgewerkt met informatie die alleen beschikbaar is voor die tool. Hierdoor worden de bijbehorende gegevens verbonden met gegevens die door HP-applicaties zijn verzameld.
- Met een aantal systemen van derden kan de CMDB worden gevuld om een centrale CMDB aan

te maken, waarmee wijzigingen kunnen worden bijgehouden en een impactanalyse kan worden uitgevoerd.

- Met een systeem van derden kunnen entiteiten en relaties worden aangemaakt volgens de bedrijfslogica. Vervolgens kunnen de gegevens naar de CMDB worden geschreven, zodat kan worden geprofiteerd van de query-mogelijkheden van de CMDB.
- Andere systemen, zoals het Release Control-systeem (CCM), kunnen de impactanalysemethoden gebruiken voor wijzigingsanalyse.

Rechten

Voor toegang tot het WSDL-bestand voor de webservice gaat u naar: **http://localhost:8080/axis2/services/UcmdbService?wsdl**. U moet gebruikersreferenties voor de serverbeheerder opgeven om het WSDL-bestand te bekijken.

De gebruiker moet over het algemene actierecht **Oude API uitvoeren** beschikken om zich aan te melden.

In de volgende tabel worden de aanvullende vereiste rechten weergegeven voor elke Web Service API-opdracht:

Web Service API-opdracht	Vereiste rechten
addCIsAndRelations deleteCIsAndRelations updateCIsAndRelations	Algemene actie: **Gegevens bijwerken**
executeTopologyQueryByName(AdHoc) executeTopologyQueryByNameWithParameters(AdHoc) executeTopologyQueryWithParameters(AdHoc)	Algemene actie: **Query uitvoeren per definitie** Voor elke query: recht **Weergeven**
getTopologyQueryExistingResultByName getTopologyQueryResultCountByName releaseChunks pullTopologyMapChunks getCINeighbours getFilteredCIsByType getCIsById getCIsByType getRelationsById	Algemene actie: **CI´s weergeven** Voor elke query: recht **Weergeven**
getQueryNameOfView	Algemene actie: **CI´s weergeven** Voor elke weergave: recht **Weergeven**
getChangedCIs	Algemene actie: **Geschiedenis weergeven**, **CI's weergeven**

Web Service API-opdracht	Vereiste rechten
calculateImpact getImpactPath getImpactRulesByGroupName getImpactRulesByNamePrefix	Algemene actie: **Impactanalyse uitvoeren**
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Geen

HP Universal CMDB Web Service API-referentie

Raadpleeg de HP UCMDB Web Service API Reference voor volledige documentatie over de aanvraag- en responsstructuren. De bestanden bevinden zich in de volgende map:

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

Webservice aanroepen

U gebruikt SOAP-standaardprogrammeermethoden in de HP Universal CMDB Web Service om servermethoden te kunnen aanroepen. Als de instructie niet kan worden geparseerd of als er een probleem is met het aanroepen van de methode, genereren de API-methoden een `SoapFault`-uitzondering. Wanneer een `SoapFault`-uitzondering wordt gegenereerd, worden in UCMDB een of meer velden voor foutberichten, foutcodes en uitzonderingsberichten gevuld. Als er geen fout is, worden de resultaten van de aanroep geretourneerd.

SOAP-programmeurs kunnen toegang krijgen tot de WSDL via:

http://<server>[:port]/axis2/services/UcmdbService?wsdl

De poortspecificatie is alleen nodig voor niet-standaardinstallaties. Raadpleeg de systeembeheerder voor het juiste poortnummer.

De URL voor het aanroepen van de service is:

http://<server>[:port]/axis2/services/UcmdbService

Voorbeelden van verbinding maken met de CMDB vindt u in "Use cases" op pagina 261.

Query's uitvoeren op de CMDB

Er wordt een query uitgevoerd op de CMDB met de API's die worden beschreven in "Query-methoden van UCMDB" op pagina 244.

De query's en de geretourneerde CMDB-elementen bevatten altijd echte UMDB -ID's.

Zie "Query-voorbeeld" op pagina 264 voor voorbeelden van het gebruik van de query-methoden.

In dit gedeelte vindt u de volgende onderwerpen:

Just-in-time responsberekening

Voor alle query-methoden worden de waarden die door de query-methode worden aangevraagd, berekend met de UMDB-server wanneer de aanvraag wordt ontvangen. De resultaten worden op basis van de laatste gegevens geretourneerd. Het resultaat wordt altijd berekend op het moment dat de aanvraag wordt ontvangen, zelfs als de TQL-query actief is en er een eerder berekend resultaat is. Daarom kunnen de aan de clientapplicatie geretourneerde resultaten van de uitvoering van een query afwijken van de resultaten van dezelfde query die in de gebruikersinterface wordt weergegeven.

Tip: als de resultaten van een bepaalde query in uw applicatie meerdere malen worden gebruikt en de gegevens waarschijnlijk niet aanzienlijk zullen veranderen tussen de toepassingen van de resultaatgegevens, kunt u de prestaties verbeteren door de gegevens in de clientapplicatie op te slaan in plaats van de query herhaaldelijk uit te voeren.

Grote responses verwerken

De respons op een query omvat altijd de structuren van de gegevens die door de query-methode zijn aangevraagd, zelfs als er geen werkelijke gegevens worden verzonden. Bij vele methoden waarbij de gegevens een verzameling of kaart vormen, omvat de respons ook de structuur `ChunkInfo`, die bestaat uit `chunksKey` en `numberOfChunks`. Met het veld `numberOfChunks` wordt het aantal segmenten aangegeven met gegevens die moeten worden opgehaald.

De maximale overdrachtsgrootte van gegevens wordt door de systeembeheerder ingesteld. Als de via de query geretourneerde gegevens de maximale grootte overschrijden, bevatten de gegevensstructuren in de eerste respons geen informatie van betekenis en is de waarde van het veld `numberOfChunks2` of groter. Als de gegevens het maximum niet overschrijden, is het veld `numberOfChunks` 0 (nul) en worden de gegevens in de eerste respons verzonden. Daarom moet u bij de verwerking van een respons de waarde `numberOfChunks` eerst controleren. Als deze groter is dan `1`, negeert u de gegevens in de overdracht en vraagt u de segmenten van gegevens aan. Gebruik anders de gegevens in de respons.

Zie "pullTopologyMapChunks" op pagina 252 en "releaseChunks" op pagina 254 voor informatie over het verwerken van gesegmenteerde gegevens.

Te retourneren eigenschappen opgeven

CI's en relaties hebben in het algemeen veel eigenschappen. Sommige methoden waarbij verzamelingen of grafieken van deze items worden geretourneerd, accepteren invoerparameters waarmee wordt opgegeven welke eigenschapswaarden moeten worden geretourneerd met elk item dat overeenkomt met de query. Met de CMDB worden geen lege eigenschappen geretourneerd. Daarom kan de respons op een query minder eigenschappen hebben dan in de query zijn

aangevraagd.

In dit gedeelte worden de typen sets beschreven waarmee de te retourneren eigenschappen worden opgegeven.

Er kan op twee manieren naar eigenschappen worden verwezen:

- Op basis van naam
- Op basis van de naam van vooraf gedefinieerde eigenschapsregels. Vooraf gedefinieerde eigenschapsregels worden gebruikt door de CMDB om een lijst met echte eigenschapsnamen aan te maken.

Wanneer een applicatie op basis van naam verwijst naar eigenschappen, wordt een `PropertiesList`-element doorgegeven.

Tip: Gebruik indien mogelijk `PropertiesList` om de namen op te geven van de eigenschappen waarin u bent geïnteresseerd, in plaats van een regelgebaseerde set. Als vooraf gedefinieerde eigenschapsregels worden gebruikt, worden vrijwel altijd meer eigenschappen dan benodigd geretourneerd. Bovendien gaat dit ten koste van de prestaties.

Er zijn twee typen vooraf gedefinieerde eigenschappen: kwalificatoreigenschappen en eenvoudige eigenschappen.

- **Kwalificatoreigenschappen.** Gebruik deze wanneer de clientapplicatie een `QualifierProperties`-element (een lijst met kwalificatoren die op eigenschappen kunnen worden toegepast) moet doorgeven. Met de CMDB wordt de lijst met kwalificatoren geconverteerd die door de clientapplicatie worden doorgegeven aan de lijst met eigenschappen waarop minimaal een van de kwalificatoren van toepassing is. De waarden van deze eigenschappen worden geretourneerd met het element `CI` of `Relation`.
- **Eenvoudige eigenschappen.** Om eenvoudige regelgebaseerde eigenschappen te gebruiken geeft de clientapplicatie een `SimplePredefinedProperty`- of `SimpleTypedPredefinedProperty`-element door. Deze elementen bevatten de naam van de regel op basis waarvan de CMDB de lijst met te retourneren eigenschappen genereert. De regels die in een `SimplePredefinedProperty`- of `SimpleTypedPredefinedProperty`-element kunnen worden opgegeven, zijn `CONCRETE`, `DERIVED` en `NAMING`.

Concrete eigenschappen

Concrete eigenschappen zijn de set met eigenschappen die voor het opgegeven CIT zijn gedefinieerd. De eigenschappen die zijn toegevoegd door afgeleide klassen, worden niet geretourneerd voor exemplaren van deze afgeleide klassen.

Een verzameling van door een methode geretourneerde klassen kan bestaan uit exemplaren van een CIT dat is opgegeven in de methodeaanroep, en exemplaren van CIT's die overerven van dat CIT. De afgeleide CIT's overerven de eigenschappen van het opgegeven CIT. Daarnaast breiden de afgeleide CIT's het bovenliggende CIT uit door eigenschappen toe te voegen.

Voorbeeld van concrete eigenschappen:

CIT `T1` heeft eigenschappen `P1` en `P2`. CIT `T11` overerft van `T1` en breidt `T1` uit met eigenschappen `P21` en `P22`.

De verzameling van CI's van het type `T1` omvat de exemplaren van `T1` en `T11`. De concrete eigenschappen van alle exemplaren in deze verzameling zijn `P1` en `P2`.

Afgeleide eigenschappen

Afgeleide eigenschappen vormen de set met eigenschappen die zijn gedefinieerd voor het opgegeven CIT en vormen voor elk afgeleid CIT de eigenschappen die door het afgeleide CIT zijn toegevoegd.

Voorbeeld van afgeleide eigenschappen:

Als we verdergaan met het voorbeeld van concrete eigenschappen, zijn de afgeleide eigenschappen van exemplaren van `T1P1` en `P2`. De afgeleide eigenschappen van exemplaren van `T11` zijn `P1`, `P2`, `P21` en `P22`.

Naamgevingseigenschappen

De naamgevingseigenschappen zijn `display_label` en `data_name`.

Andere specificatie-elementen voor eigenschappen

- **PredefinedProperties**

  `PredefinedProperties` kunnen een `QualifierProperties`-element en een `SimplePredefinedProperty`-element bevatten voor elk van de andere mogelijke regels. Een `PredefinedProperties`-set bevat niet noodzakelijkerwijs alle typen van lijsten.

- **PredefinedTypedProperties**

  `PredefinedTypedProperties` wordt gebruikt om een andere set met eigenschappen op elk CIT toe te passen. `PredefinedTypedProperties` kan een `QualifierProperties`-element en een `SimpleTypedPredefinedProperty`-element bevatten voor elk van de andere toepasselijke regels. Omdat `PredefinedTypedProperties` op elk CIT afzonderlijk wordt toegepast, zijn afgeleide eigenschappen niet relevant. Een `PredefinedProperties`-set bevat niet noodzakelijkerwijs alle toepasselijke typen van lijsten.

- **CustomProperties**

  `CustomProperties` kunnen elke combinatie van de basis-`PropertiesList`en de regelgebaseerde eigenschappenlijsten bevatten. Het eigenschappenfilter is het totaal van alle eigenschappen die door alle lijsten zijn geretourneerd.

- **CustomTypedProperties**

  `CustomTypedProperties` kan elke combinatie van de basis-`PropertiesList` en de van toepassing zijnde regelgebaseerde eigenschappenlijsten bevatten. Het eigenschappenfilter is het totaal van alle eigenschappen die door alle lijsten zijn geretourneerd.

- **TypedProperties**

  `TypedProperties` wordt gebruikt om een andere set met eigenschappen voor elk CIT te parseren. `TypedProperties` is een verzameling paren die zijn samengesteld uit typenamen

en eigenschappensets van alle typen. Elke eigenschappenset wordt op slechts één overeenkomend type toegepast.

De UCMDB bijwerken

U werkt de CMDB bij met de API's voor bijwerken. Zie "UCMDB-bijwerkmethoden" op pagina 254 voor meer informatie over de API-methoden. Zie "Bijwerkvoorbeeld" op pagina 275 voor voorbeelden van het gebruik van de bijwerkmethoden.

Deze taak omvat de onderstaande stappen:

- "De UCMDB bijwerken" boven
- "Gebruik van ID-typen met bijwerkmethoden" beneden

UCMDB-bijwerkparameters

In dit onderwerp worden de parameters beschreven die alleen door de bijwerkmethoden van de service worden gebruikt. Raadpleeg schema documentation. voor meer informatie.

- **CIsAndRelationsUpdates**

  Het `CIsAndRelationsUpdates`-type bestaat uit `CIsForUpdate`, `relationsForUpdate`, `referencedRelations` en `referencedCIs`. Een `CIsAndRelationsUpdates`-exemplaar bevat niet noodzakelijkerwijs alle drie de elementen.

  `CIsForUpdate` is een CI-verzameling. `relationsForUpdate` is een `Relations`-verzameling. De elementen `CI` en `relation` in de verzamelingen hebben een `props`-element. Wanneer een CI of relatie wordt aangemaakt, moeten eigenschappen die het attribuut `required` of het attribuut `key` in de definitie CI-type hebben, met waarden worden gevuld. De items in deze verzamelingen worden bijgewerkt of aangemaakt met de methode.

  `referencedCIs` en `referencedRelations` zijn verzamelingen CI's die al zijn gedefinieerd in de CMDB. De elementen in de verzameling worden geïdentificeerd met een tijdelijke ID in combinatie met alle sleuteleigenschappen. Deze items worden gebruikt om de identiteiten van CI's en relaties voor bijwerken om te zetten. Ze worden nooit aangemaakt of bijgewerkt door de methode.

  Elk van de `CI`- en `relation`-elementen in deze verzamelingen hebben een eigenschappenverzameling. Nieuwe items worden aangemaakt met de eigenschappenwaarden in deze verzamelingen.

Gebruik van ID-typen met bijwerkmethoden

Hierna worden ID-CIT's en CI's en relaties beschreven. Wanneer de ID geen echte CMDB -ID is, zijn de type- en sleutelattributen vereist.

- **Configuratie-items verwijderen of bijwerken**

  Er kan een tijdelijke of lege ID door de client worden gebruikt bij het aanroepen van een methode om een item te verwijderen of bij te werken. In dit geval moeten het CI-type en de "Sleutelattributen"sleutelattributen waarmee het CI wordt geïdentificeerd, worden ingesteld.

- **Relaties verwijderen of bijwerken**

  Wanneer relaties worden verwijderd of bijgewerkt, kan de relatie-ID leeg, tijdelijk of echt zijn.

Als de ID van een CI tijdelijk is, moet het CI in de verzameling `referencedCIs` worden doorgegeven en moeten de bijbehorende sleutelattributen worden opgegeven. Zie referencedCIs in "CIsAndRelationsUpdates" op vorige pagina voor meer informatie.

- **Nieuwe configuratie-items invoegen in de CMDB**

  Het is mogelijk een lege ID of een tijdelijke ID te gebruiken om een nieuw CI in te voegen. Als de ID echter leeg is, kan de server de echte CMDB -ID niet retourneren in de structuur `createIDsMap` omdat er geen `clientID` is. Zie "addCIsAndRelations" op pagina 255 en "Query-methoden van UCMDB" op pagina 244 voor meer informatie over dit onderwerp.

- **Nieuwe relaties invoegen in de CMDB**

  De relatie-ID kan tijdelijk of leeg zijn. Als de relatie echter nieuw is, maar de configuratie-items aan beide einden van de relatie al in de CMDB zijn gedefinieerd, moeten deze reeds aanwezige CI's door een echte CMDB -ID worden geïdentificeerd of in een `referencedCIs`-verzameling worden opgegeven.

Query's uitvoeren op het UCMDB-klassemodel

Met de klassemodelmethoden wordt informatie over CIT's en relaties geretourneerd. Het klassemodel wordt geconfigureerd met CI-typebeheer. Zie "CI-typebeheer" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.

Zie "Klassemodelvoorbeeld" op pagina 280 voor voorbeelden van het gebruik van de klassemodelmethoden.

Dit gedeelte bevat informatie over de volgende methoden waarmee informatie over CIT's en relaties wordt geretourneerd:

- "getClassAncestors" beneden
- "getAllClassesHierarchy" op volgende pagina
- "getCmdbClassDefinition" op volgende pagina

getClassAncestors

Met de methode `getClassAncestors` wordt het pad opgehaald tussen het opgegeven CIT en het beginpunt van het CIT, inclusief het beginpunt.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240
className	De typenaam. Zie "Typenaam" op pagina 241 voor meer informatie over dit onderwerp.

Uitvoer

Parameter	Opmerking
classHierarchy	Een verzameling paren van klassenamen en naam van bovenliggende klasse.
comments	Alleen voor intern gebruik.

getAllClassesHierarchy

Met de methode `getAllClassesHierarchy` wordt de gehele boomstructuur van het klassemodel opgehaald.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op volgende pagina voor meer informatie.

Uitvoer

Parameter	Opmerking
classesHierarchy	Een verzameling paren van klassenamen en naam van bovenliggende klasse.
comments	Alleen voor intern gebruik.

getCmdbClassDefinition

Met de methode `getCmdbClassDefinition` wordt informatie over de opgegeven klasse opgehaald.

Als u getCmdbClassDefinition gebruikt om de sleutelattributen op te halen, moet u ook een query uitvoeren op de bovenliggende klassen tot aan de basisklasse. Met `getCmdbClassDefinition` worden als sleutelattributen alleen die attributen geïdentificeerd waarvan de ID_ATTRIBUTE in de klassedefinitie is opgegeven met `className`. Overgenomen sleutelattributen worden niet herkend als sleutelattributen van de opgegeven klasse. Daarom is de volledige lijst met sleutelattributen van de opgegeven klasse de verzameling van alle sleutels van de klasse en alle bijbehorende bovenliggende klassen tot aan de hoofdklasse.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op volgende pagina voor meer informatie.
className	De typenaam. Zie "Algemene UCMDB-parameters " op volgende pagina voor meer informatie over dit onderwerp.

Uitvoer

Parameter	Opmerking
cmdbClass	De klassedefinitie, bestaande uit `name`, `classType`, `displayLabel`, `description`, `parentName`, kwalificatoren en attributen.
comments	Alleen voor intern gebruik.

Query uitvoeren voor impactanalyse

Met de`Identifier` in de impactanalysemethoden wordt verwezen naar de responsgegevens van de service. Deze is uniek voor de huidige respons en wordt na tien minuten inactiviteit verwijderd uit de geheugencache van de server.

Zie "Impactanalysevoorbeeld" op pagina 281 voor voorbeelden van het gebruik van de impactanalysemethoden.

Algemene UCMDB-parameters

In dit gedeelte worden de meest gebruikte parameters van de servicemethoden beschreven. Zie de schema documentation. voor meer informatie.

In dit gedeelte vindt u de volgende onderwerpen:

CmdbContext

Voor alle UCMDB Web Service API-serviceaanroepen is een `CmdbContext`-argument vereist. `CmdbContext` is een `callerApplication`-string waarmee de applicatie wordt geïdentificeerd waarmee de service wordt aangeroepen. `CmdbContext` wordt gebruikt voor registratie in logboeken en oplossen van problemen.

ID

Elk `CI` en elke relatie heeft een `ID`-veld. Het bestaat uit een hoofdlettergevoelige ID-string en een optionele `temp`-vlag, waarmee wordt aangegeven of de ID tijdelijk is.

Sleutelattributen

Om een `CI` of `Relation` in bepaalde contexten te identificeren kunnen sleutelattributen worden gebruikt in plaats van een CMDB-ID. Sleutelattributen zijn attributen waarvan ID_ATTRIBUTE is ingesteld in de klassedefinitie.

In de gebruikersinterface bevindt zich een sleutelpictogram naast de sleutelattributen in de lijst met attributen van het type configuratie-item. Zie "Het dialoogvenster Attribuut toevoegen/bewerken" in de HP Universal CMDB – Handleiding Modeling voor meer informatie over dit onderwerp. Zie "getCmdbClassDefinition" op pagina 239 voor informatie over het identificeren van de sleutelattributen in de API-clientapplicatie.

ID-typen

Een `ID`-element kan een echte ID of een tijdelijke ID bevatten.

Aan een echte ID wordt een string toegewezen door de CMDB. Hiermee wordt een entiteit in de database geïdentificeerd. Een tijdelijke ID kan een willekeurige string zijn die uniek is in de huidige aanvraag.

Een tijdelijke ID kan door de client worden toegewezen en stelt vaak de ID van het CI voor zoals die door de client wordt opgeslagen. Met een tijdelijke ID wordt niet noodzakelijkerwijs een entiteit vertegenwoordigd die al in de CMDB is aangemaakt. Wanneer een tijdelijke ID door de client wordt doorgegeven, als de CMDB een bestaand gegevensconfiguratie-item kan identificeren met de CI-sleuteleigenschappen, wordt dat CI gebruikt zoals het uitkomt voor de context alsof het met een echte ID is geïdentificeerd.

CIProperties

Een `CIProperties`-element bestaat uit verzamelingen en elke verzameling bevat een sequentie van naam-/waarde-elementen waarmee eigenschappen worden opgegeven van het type dat wordt aangegeven door de verzamelingsnaam. Geen enkele verzameling is vereist. Het element `CIProperties` kan dus elke combinatie van verzamelingen bevatten.

`CIProperties` worden gebruikt door `CI`- en `Relation`-elementen. Zie "Configuratie-item (CI)" op volgende pagina en "Relatie" op volgende pagina voor meer informatie.

De eigenschappenverzamelingen zijn:

- `dateProps` - verzameling van `DateProp`-elementen
- `doubleProps` - verzameling van`DoubleProp`-elementen
- `floatProps` - verzameling van `FloatProp`-elementen
- `intListProps` - verzameling van `IntListProp`-elementen
- `intProps` - verzameling van `IntProp`-elementen
- `strProps` - verzameling van `StrProp`-elementen
- `strListProps` - verzameling van `StrListProp`-elementen
- `longProps` - verzameling van `LongProp`-elementen
- `bytesProps` - verzameling van `BytesProp`-elementen
- `xmlProps` - verzameling van `XmlProp`-elementen

Typenaam

De typenaam is de klassenaam van een configuratie-itemtype of relatietype. De typenaam wordt gebruikt in code om naar de klasse te verwijzen. De typenaam mag niet worden verward met de weergegeven naam, die in de gebruikersinterface wordt weergegeven waar de klasse wordt vermeld, maar die geen betekenis heeft in code.

Configuratie-item (CI)

Een `CI`-element bestaat uit een `ID`, een `type` en een `props`-verzameling.

Wanneer "UCMDB-bijwerkmethoden" worden gebruikt om een `CI` bij te werken, kan het `ID`-element een echte CMDB-ID of een door de client toegewezen tijdelijke ID bevatten. Als een tijdelijke ID wordt gebruikt, stelt u de vlag `temp` in op waar. Wanneer een item wordt verwijderd, kan de `ID` leeg zijn. Bij "Query-methoden van UCMDB" worden echte `ID`'s gebruikt als invoerparameters en worden echte `ID`'s in de query-resultaten geretourneerd.

Het `type` kan een willekeurige typenaam zijn die is gedefinieerd in CI-typebeheer. Zie "CI-typebeheer" op pagina 1 in de HP Universal CMDB – Handleiding Modeling voor meer informatie over dit onderwerp.

Het element `props` is een CIProperties-verzameling. Zie "Algemene UCMDB-parameters " op pagina 240 voor meer informatie over dit onderwerp.

Relatie

Een `Relation` is een entiteit waarmee twee configuratie-items worden gekoppeld. Een `Relation`-element bestaat uit een `ID`, een `type`, de ID's van de twee items die worden gekoppeld `(end1ID` en `end2ID)` en een `props`-verzameling.

Wanneer "UCMDB-bijwerkmethoden" worden gebruikt om een `Relation` bij te werken, kan de waarde van de ID van de `Relation` een echte CMDB-ID of een tijdelijke ID zijn. Wanneer een item wordt verwijderd, kan de `ID` leeg zijn. Bij "Query-methoden van UCMDB" worden echte `ID`'s als invoerparameters gebruikt en worden echte `ID`'s in de query-resultaten geretourneerd.

Het relatietype is de `Type Name` van de UCMDB-klasse waaruit de relatie wordt geïnstantieerd. Het type kan een van de relatietypen zijn die zijn gedefinieerd in de CMDB. Zie "Query's uitvoeren op het UCMDB-klassemodel " op pagina 238 voor meer informatie over klassen of typen.

Zie "CI-typebeheer" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.

De twee ID's voor relatie-eind mogen geen lege ID's zijn omdat ze worden gebruikt om de ID van de huidige relatie aan te maken. Ze kunnen beide echter tijdelijke ID's toegewezen hebben gekregen door de client.

Het element `props` is een CIProperties-verzameling. Zie "CIProperties" op vorige pagina voor meer informatie over dit onderwerp.

UCMDB-uitvoerparameters

In dit gedeelte worden de meest gebruikte uitvoerparameters van de servicemethoden beschreven. Zie de schema documentation. voor meer informatie.

In dit gedeelte vindt u de volgende onderwerpen:

- "CIs" op volgende pagina
- "ShallowRelation" op volgende pagina
- "Topology" op volgende pagina
- "CINode" op volgende pagina

- "RelationNode" beneden
- "TopologyMap" beneden
- "ChunkInfo" beneden

CIs

`CIs` vormen een verzameling CI-elementen.

ShallowRelation

Een `ShallowRelation` is een entiteit waarmee twee configuratie-items worden gekoppeld, en die bestaat uit een `ID`, een `type` en de ID's van de twee items die worden gekoppeld `(end1ID` en `end2ID)`. Het relatietype is de `Typenaam` van de CMDB-klasse waaruit de relatie wordt geïnstantieerd. Het type kan een van de relatietypen zijn die zijn gedefinieerd in de CMDB.

Topology

`Topology` is een grafiek van `CI`-elementen en relaties. Een `Topology` bestaat uit een verzameling `CIs` en een verzameling `Relations` die een of meer `Relation`-elementen bevatten.

CINode

`CINode` bestaat uit een verzameling `CIs` met een `label`. Het `label` in de `CINode` is het label dat in het knooppunt is gedefinieerd van de TQL die in de query wordt gebruikt.

RelationNode

`RelationNode` is een set `Relation`s-verzamelingen met een `label`. Het `label` in de `RelationNode` is het label dat in het knooppunt is gedefinieerd van de TQL die in de query wordt gebruikt.

TopologyMap

`TopologyMap` is de uitvoer van een query-berekening die overeenkomt met een TQL-query. De `labels` in de `TopologyMap` zijn de knooppuntlabels die in de TQL zijn gedefinieerd die in de query wordt gebruikt.

De gegevens van `TopologyMap` worden in de volgende vorm geretourneerd:

- `CINodes`. Dit is een of meer `CINode`s (zie "CINode" boven).
- `relationNodes`. Dit is een of meer `RelationNode`s (zie "RelationNode" boven).

Met de `labels` in deze twee structuren worden de lijsten met configuratie-items en relaties geordend.

ChunkInfo

Wanneer een query een grote hoeveelheid gegevens oplevert, worden de gegevens op de server opgeslagen en worden ze verdeeld in segmenten. De informatie die de client gebruikt om de gesegmenteerde gegevens op te halen, bevindt zich in de structuur `ChunkInfo` die door de query wordt geretourneerd. `ChunkInfo` bestaat uit `numberOfChunks` die moeten worden opgehaald, en de `chunksKey`. De `chunksKey` is een unieke ID van de gegevens op de server voor deze specifieke query-aanroep.

Zie "Grote responses verwerken" op pagina 234 voor meer informatie.

Query-methoden van UCMDB

Dit gedeelte bevat informatie over de volgende methoden:

executeTopologyQueryByNameWithParameters

Met de methode `executeTopologyQueryByNameWithParameters` wordt een `topologyMap`-element opgehaald dat overeenkomt met de opgegeven geparametriseerde query.

De waarden voor de query-parameters worden doorgegeven in het argument `parameterizedNodes`. Voor de opgegeven TQL moeten unieke labels voor elke `CINode` en elke `relationNode` zijn gedefinieerd. Anders kan de methode niet worden aangeroepen.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
queryName	De naam van de geparametriseerde TQL in de CMDB waarvoor de kaart moet worden opgehaald.
parameterizedNodes	De voorwaarden waaraan elk knooppunt moet voldoen om in de query-resultaten te worden opgenomen.
queryTypedProperties	Een verzameling van sets met eigenschappen die moeten worden opgehaald voor items van een specifiek type configuratie-item.

Uitvoer

Parameter	Opmerking
topologyMap	Zie "TopologyMap" op pagina 243 voor meer informatie.
chunkInfo	Zie voor meer informatie: "ChunkInfo" op pagina 243 en "Grote responses verwerken" op pagina 234.

executeTopologyQueryWithParameters

De methode `executeTopologyQueryWithParameters` haalt een `topologyMap`-element op dat overeenkomt met de geparametriseerde query.

De query wordt doorgegeven in het argument `queryXML`. De waarden voor de query-parameters worden doorgegeven in het argument `parameterizedNodes`. Voor de TQL moeten unieke labels zijn gedefinieerd voor elke `CINode` en elke `relationNode`.

De methode `executeTopologyQueryWithParameters` wordt gebruikt om ad-hoc query's door te geven in plaats van een query te openen die is gedefinieerd in de CMDB. U kunt deze methode gebruiken wanneer u geen toegang hebt tot de UCMDB-gebruikersinterface om een query te definiëren of wanneer u de query niet in de database wilt opslaan.

Ga als volgt te werk om een geëxporteerde TQL te gebruiken als invoer voor deze methode:

1. Open de webbrowser en voer het volgende adres in:
   http://localhost:8080/jmx-console.

   Wellicht zult u zich moeten aanmelden met een gebruikersnaam en wachtwoord. De standaardinstelling is **sysadmin**/**sysadmin**
2. Klik op **UCMDB:service=TQL Services**.
3. Zoek naar de bewerking **exportTql**.
   - Typ **1** in het parametervak **customerId**.
   - Voer in het parametervak **patternName** een geldige TQL-naam in.
4. Klik op **Aanroepen**.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
queryXML	Een XML-string die een TQL zonder brontags vertegenwoordigt.
parameterizedNodes	De voorwaarden waaraan elk knooppunt moet voldoen om in de query-resultaten te worden opgenomen.

Uitvoer

Parameter	Opmerking
topologyMap	Zie "TopologyMap" op pagina 243 voor meer informatie.
chunkInfo	Zie "ChunkInfo" op pagina 243 en "Grote responses verwerken" op pagina 234 voor meer informatie over dit onderwerp.

getChangedCIs

Met de methode `getChangedCIs` worden de wijzigingsgegevens geretourneerd voor alle CI's die betrekking hebben op de opgegeven CI's.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
ids	De lijst met de ID's van de hoofd-CI's waarvan de gerelateerde CI's op wijzigingen worden gecontroleerd. Alleen echte CMDB-ID's zijn geldig in deze verzameling voor meer informatie.
fromDate	Het begin van de periode waarin moet worden gecontroleerd of CI's zijn gewijzigd.
toDate	Het einde van de periode waarin moet worden gecontroleerd of CI's zijn gewijzigd.

Uitvoer

Parameter	Opmerking
changeDataInfo	Nul of meer verzamelingen van `ChangedDataInfo`-elementen.

getCINeighbours

Met de methode `getCINeighbours` worden de directe buren van het opgegeven CI geretourneerd.

Als de query bijvoorbeeld wordt uitgevoerd op de buren van CI `A` en CI `A` bevat CI `B` dat CI `C` gebruikt, wordt CI `B` geretourneerd, maar CI `C` niet. Dat wil zeggen: alleen buren van het opgegeven type worden geretourneerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.

Parameter	Opmerking
ID	De ID van het CI waarmee de buren moeten worden opgehaald. Dit moet een echte CMDB-ID zijn.
neighbourType	De CIT-naam van de buren die moeten worden opgehaald. Buren van het opgegeven type en van typen die van dat type zijn afgeleid, worden geretourneerd. Zie "Typenaam" op pagina 241 voor meer informatie over dit onderwerp.
CIProperties	De gegevens die voor elk configuratie-item moeten worden geretourneerd (de zogenaamde Query-indeling in de gebruikersinterface). Zie "TypedProperties" op pagina 236 voor meer informatie.
relationProperties	De gegevens die voor elke relatie moeten worden geretourneerd (de zogenaamde Query-indeling in de gebruikersinterface). Zie "TypedProperties" op pagina 236

Uitvoer

Parameter	Opmerking
topologie	Zie "Topology" op pagina 243 voor meer informatie.
comments	Alleen voor intern gebruik.

getCIsByID

Met de methode `getCIsByID` worden configuratie-items op basis van de bijbehorende CMDB-ID's opgehaald.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
CIsTypedProperties	Een verzameling getypeerde eigenschappen. Zie "Andere specificatie-elementen voor eigenschappen" op pagina 236 voor meer informatie.
IDs	Alleen echte CMDB-ID's zijn geldig in deze verzameling voor meer informatie.

Uitvoer

Parameter	Opmerking
CI's	Verzameling van CI-elementen.
chunkInfo	Zie voor meer informatie: "ChunkInfo" op pagina 243 en "Grote responses verwerken" op pagina 234.

getClsByType

Met de methode `getCIsByType` wordt de verzameling geretourneerd van configuratie-items van het opgegeven type en van alle typen die overerven van het opgegeven type.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
type	De klassenaam. Zie "Typenaam" op pagina 241 voor meer informatie over dit onderwerp.
properties	De gegevens die voor elk configuratie-item moeten worden geretourneerd. Zie "CustomProperties" op pagina 236 voor meer informatie.

Uitvoer

Parameter	Opmerking
CI's	Verzameling van CI-elementen. voor meer informatie.
chunkInfo	Zie voor meer informatie: "ChunkInfo" op pagina 243 en "Grote responses verwerken" op pagina 234.

getFilteredClsByType

Met de methode `getFilteredCIsByType` worden de CI's van het opgegeven type opgehaald dat voldoet aan de voorwaarden die door de methode worden gebruikt. Een voorwaarde bestaat uit het volgende:

- Een naamveld met de naam van een eigenschap
- Een operatorveld met een vergelijkingsoperator
- Een optioneel waardeveld met een waarde of lijst met waarden

Samen vormen ze een boolean-expressie:
`<item>.property.value [operator] <condition>.value`

Als de voorwaardenaam bijvoorbeeld `root_actualdeletionperiod` is, de waarde van de voorwaarde `40` en de operator `Equal`, is de boolean-instructie:

`<item>.root_actualdeletionperiod.value = = 40`

Met de query worden alle items geretourneerd waarvan `root_actualdeletionperiod40` is, mits er geen andere voorwaarden zijn.

Als het argument `conditionsLogicalOperatorAND` is, worden de items geretourneerd die voldoen aan alle voorwaarden in de verzameling `conditions`. Als `conditionsLogicalOperatorOR` is, worden de items geretourneerd die minimaal aan een van de voorwaarden in de verzameling `conditions` voldoen.

In de volgende tabel worden de vergelijkingsoperatoren weergegeven:

Operator	Type voorwaarde/opmerkingen
ChangedDuring	Date Dit is een periodecontrole. De waarde van de voorwaarde wordt in uren opgegeven. Als de waarde van de datumeigenschap in de periode ligt waarin de methode wordt aangeroepen plus of minus de waarde van de voorwaarde, is de voorwaarde waar. Als de waarde van de voorwaarde bijvoorbeeld `24` is, is de voorwaarde waar als de waarde van de datumeigenschap tussen gisteren op dit moment en morgen op dit moment ligt. **Opmerking:** De naam `ChangedDuring` wordt aangehouden om achterwaartse compatibiliteit te behouden. In vorige versies werd de operator alleen gebruikt voor het aanmaken en wijzigen van tijd.
Gelijk aan	String en numeriek
EqualIgnoreCase	String
Greater	Numeriek
GreaterEqual	Numeriek
In	String, numeriek en lijst De waarde van de voorwaarde is een lijst. De voorwaarde is waar als de waarde van de eigenschap een van de waarden in de lijst is.
InList	Lijst De waarde van de voorwaarde en de waarde van de eigenschap zijn lijsten. De voorwaarde is waar als alle waarden in de lijst van de voorwaarde ook in de eigenschappenlijst van het item worden weergegeven. Er kunnen meer eigenschapswaarden zijn dan in de voorwaarde zijn opgegeven zonder dat dit van invloed is op de vraag of de waarde waar of onwaar is.
IsNull	String, numeriek en lijst De eigenschap van het item heeft geen waarde. Wanneer de operator `IsNull` wordt gebruikt, wordt de waarde van de voorwaarde genegeerd en kan deze soms leeg zijn.
Less	Numeriek
LessEqual	Numeriek
Lijkend op	String De waarde van de voorwaarde is een substring van de waarde van de eigenschapswaarde. De waarde van de voorwaarde moet tussen procenttekens (`%`) worden geplaatst. Bijvoorbeeld `%Bi%` komt overeen met `Bismarck` en `Baai van Biskaje`, maar niet met `bizon`.
LikeIgnoreCase	String

Operator	Type voorwaarde/opmerkingen
	U gebruikt de operator `LikeIgnoreCase` op dezelfde manier als u de operator `Like` gebruikt. De vergelijking is echter niet hoofdlettergevoelig. Daarom komt `%Bi%` wel overeen met `bizon`.
NotEqual	String en numeriek
UnchangedDuring	Date Dit is een periodecontrole. De waarde van de voorwaarde wordt in uren opgegeven. Als de waarde van de datumeigenschap in de periode ligt waarin de methode wordt aangeroepen plus of minus de waarde van de voorwaarde, is de voorwaarde onwaar. Als de waarde buiten de periode ligt, is de voorwaarde waar. Als de waarde van de voorwaarde bijvoorbeeld `24` is, is de voorwaarde waar als de waarde van de datumeigenschap vóór gisteren op dit moment of na morgen op dit moment ligt. **Opmerking:** De naam `UnchangedDuring` wordt aangehouden om achterwaartse compatibiliteit te behouden. In vorige versies werd de operator alleen gebruikt voor het aanmaken en wijzigen van tijd.

Voorbeeld van het instellen van een voorwaarde:

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.Equal);
```

Voorbeeld van het uitvoeren van een query voor overgenomen eigenschappen:

Het doel-CI is `sample` en heeft twee attributen, `name` en `size`. Met `sampleII` wordt het CI uitgebreid met twee attributen, `level` en `grade`. Met dit voorbeeld wordt een query ingesteld voor de eigenschappen van `sampleII` die zijn overgenomen van `sample` door ze op basis van naam op te geven.

```
GetFilteredCIsByType request = new GetFilteredCIsByType()
request.setCmdbContext(cmdbContext)
request.setType("sampleII")
CustomProperties customProperties = new CustomProperties();
PropertiesList propertiesList = new PropertiesList();
propertiesList.addPropertyName("name");
propertiesList.addPropertyName("size");
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties)
```

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
type	De klassenaam. Zie "Typenaam" op pagina 241 voor meer informatie over dit onderwerp. Het type kan een van de typen zijn die met CI-typebeheer zijn gedefinieerd. Zie "CI-typebeheer" in de *HP Universal CMDB – Handleiding Modeling* voor meer informatie over dit onderwerp.
properties	De gegevens die voor elk CI moeten worden geretourneerd (de zogenaamde Query-indeling in de gebruikersinterface). Zie "CustomProperties" op pagina 236 voor meer informatie.
conditions	Een verzameling van paren van namen en waarden en de operatoren die met elkaar zijn gerelateerd. Bijvoorbeeld `host_hostname like QA`.
conditionsLogicalOperator	• **AND.** Er moet aan alle voorwaarden worden voldaan. • **OR**. Er moet ministens aan één voorwaarde worden voldaan.

Uitvoer

Parameter	Opmerking
CI's	Verzameling van CI-elementen.
chunkInfo	Zie "ChunkInfo" op pagina 243 en "Grote responses verwerken" op pagina 234 voor meer informatie over dit onderwerp.

getQueryNameOfView

Met de methode `getQueryNameOfView` wordt de naam van de TQL opgehaald waarop de opgegeven weergave is gebaseerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
viewName	De naam van een weergave, dat wil zeggen: een subset van het klassemodel in de CMDB.

Uitvoer

Parameter	Opmerking
queryName	De naam van de TQL in de CMDB waarop de weergave wordt gebaseerd.

getTopologyQueryExistingResultByName

Met de methode `getTopologyQueryExistingResultByName` wordt het meest recente resultaat van het uitvoeren van de opgegeven TQL opgehaald. De TQL wordt niet uitgevoerd met de aanroep. Als er geen resultaten zijn van een vorige uitvoering, wordt niets geretourneerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
queryName	De naam van een TQL.
queryTypedProperties	Een verzameling van sets met eigenschappen die moeten worden opgehaald voor items van een specifiek type configuratie-item.

Uitvoer

Parameter	Opmerking
queryName	De naam van de TQL in de CMDB waarop de weergave wordt gebaseerd.

getTopologyQueryResultCountByName

Met de methode `getTopologyQueryResultCountByName` wordt het aantal exemplaren opgehaald van elk knooppunt dat overeenkomt met de opgegeven query.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
queryName	De naam van een TQL.
countInvisible	Indien waar, bevat de uitvoer CI's die als onzichtbaar in de query zijn gedefinieerd.

Uitvoer

Parameter	Opmerking
queryName	De naam van de TQL in de CMDB waarop de weergave wordt gebaseerd.

pullTopologyMapChunks

Met de methode `pullTopologyMapChunks` wordt een van de segmenten opgehaald die de respons op een methode bevatten.

Elk segment bevat een `topologyMap`-element dat deel uitmaakt van de respons. Het eerste segment heeft het nummer `1`, waardoor de teller van de ophaalsequentie wordt herhaald van 1 tot *<responsobject>*`.getChunkInfo().getNumberOfChunks()`.

Zie "ChunkInfo" op pagina 243 en "Query's uitvoeren op de CMDB" op pagina 233 voor meer informatie over dit onderwerp.

De clientapplicatie moet de deelkaarten kunnen verwerken. Zie het volgende voorbeeld van het afhandelen van een CI-verzameling en het voorbeeld van het samenvoegen van segmenten in een kaart in "Query-voorbeeld" op pagina 264.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
ChunkRequest	Het nummer van het segment dat moet worden opgehaald en de `ChunkInfo` die door de query-methode wordt geretourneerd.

Uitvoer

Parameter	Opmerking
topologyMap	Zie "TopologyMap" op pagina 243 voor meer informatie.
comments	Alleen voor intern gebruik.

Voorbeeld van het afhandelen van segmenten:

```
GetCIsByType request =
     new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
     ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1; j<=response.getChunkInfo().getNumberOfChunks(); j++){
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req =new
        PullTopologyMapChunks(cmdbContext,chunkRequest);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().sizeCINodeList()
;
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINode(m).getCIs
();
            for(int i=0 ; i < cis.sizeCIList() ; i++) {
                // your code to process the CIs
            }
```

```
        }
}

GetCIsByType request =
     new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
     ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1 ; j <= response.getChunkInfo().getNumberOfChunks() ;
j++) {
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req = new PullTopologyMapChunks
(cmdbContext,     chunkRequest);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().sizeCINodeList()
;
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINode(m).getCIs
();
            for(int i=0 ; i < cis.sizeCIList() ; i++) {
                // your code to process the CIs
            }
        }
}
```

releaseChunks

Met de methode `releaseChunks` wordt het geheugen vrijgemaakt van de segmenten die de gegevens van de query bevatten.

Tip: De server verwijdert de gegevens na tien minuten. Door deze methode om de gegevens te verwijderen aan te roepen zodra ze zijn gelezen, worden serverbronnen vrijgehouden.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
chunksKey	De ID van de gegevens op de server die zijn gesegmenteerd. De sleutel is een element van `ChunkInfo`.

UCMDB-bijwerkmethoden

Dit gedeelte bevat informatie over de volgende methoden:

addCIsAndRelations

Met de methode `addCIsAndRelations` worden CI's en relaties toegevoegd of bijgewerkt.

Als de CI's of de relaties niet aanwezig zijn in de CMDB, worden ze toegevoegd en worden de bijbehorende eigenschappen ingesteld op basis van de inhoud van het argument `CIsAndRelationsUpdates`.

Als de CI's of relaties wel aanwezig zijn in de CMDB, worden ze bijgewerkt met de nieuwe gegevens, als `updateExisting`**true** is.

Als `updateExisting`**false** is, kan `CIsAndRelationsUpdates` niet verwijzen naar bestaande configuratie-items of relaties. Elke poging te verwijzen naar bestaande items wanneer `updateExisting` onwaar is, resulteert in een uitzondering.

Als `updateExisting`**true** is, wordt de toevoeg- of bijwerkbewerking uitgevoerd zonder de CI's te valideren, ongeacht de waarde van `ignoreValidation`.

Als `updateExisiting`**false** is en `ignoreValidation`**true**, wordt de toevoegbewerking uitgevoerd zonder de CI's te valideren.

Als `updateExisiting`**false** is en `ignoreValidation`**false**, worden de CI's gevalideerd vóór de toevoegbewerking.

Relaties worden nooit gevalideerd.

`CreatedIDsMap` is een kaart of woordenlijst van het type `ClientIDToCmdbID` waarmee de tijdelijke ID's van de client worden verbonden met de overeenkomende echte CMDB-ID's.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
updateExisting	Stel in op *true* om items bij te werken die al aanwezig zijn in de CMDB. Stel in op *false* om een uitzondering te genereren als een item al aanwezig is.
CIsAndRelationsUpdates	De items die moeten worden bijgewerkt of aangemaakt. Zie "CIsAndRelationsUpdates" op pagina 237 voor meer informatie over dit onderwerp.
ignoreValidation	Indien waar, wordt geen controle uitgevoerd voordat de CMDB is bijgewerkt.

Uitvoer

Parameter	Opmerking
CreatedIDsMap	De toewijzing van client-ID's aan CMDB-ID's. Zie de beschrijving hierboven voor meer informatie.
comments	Alleen voor intern gebruik.

addCustomer

Met de methode `addCustomer` wordt een klant toegevoegd.

Invoer

Parameter	Opmerking
CustomerID	De numerieke ID van de klant.

deleteCIsAndRelations

Met de methode `deleteCIsAndRelations` worden de opgegeven configuratie-items en relaties verwijderd uit de CMDB.

Wanneer een CI wordt verwijderd en het CI één eind van een of meer `Relation`-items is, worden deze `Relation`-items ook verwijderd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
CIsAndRelationsUpdates	De items die moeten worden verwijderd. Zie "CIsAndRelationsUpdates" op pagina 237 voor meer informatie over dit onderwerp.

removeCustomer

Met de methode `removeCustomer` wordt een klantrecord verwijderd.

Invoer

Parameter	Opmerking
CustomerID	De numerieke ID van de klant.

updateCIsAndRelations

Met de methode `updateCIsAndRelations` worden de opgegeven CI's en relaties bijgewerkt.

Bij het bijwerken worden de eigenschapswaarden van het argument `CIsAndRelationsUpdates` gebruikt. Als een van de CI's of relaties niet aanwezig is in de CMDB, wordt een uitzondering gegenereerd.

`CreatedIDsMap` is een kaart of woordenlijst van het type `ClientIDToCmdbID` waarmee de tijdelijke ID's van de client worden verbonden met de overeenkomende echte CMDB-ID's.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
CIsAndRelationsUpdates	De items die moeten worden bijgewerkt. Zie "CIsAndRelationsUpdates" op pagina 237 voor meer informatie over dit onderwerp.
ignoreValidation	Indien waar, wordt geen controle uitgevoerd voordat de CMDB is bijgewerkt.

Uitvoer

Parameter	Opmerking
CreatedIDsMap	De toewijzing van client-ID's aan CMDB-ID's. Zie "addCIsAndRelations" op pagina 255 voor meer informatie over dit onderwerp.

UCMDB-impactanalysemethoden

Dit gedeelte bevat informatie over de volgende methoden:

- "calculateImpact" beneden
- "getImpactPath" op volgende pagina
- "getImpactRulesByNamePrefix" op volgende pagina

calculateImpact

Met de methode `calculateImpact` wordt berekend welke CI's worden beïnvloed door een bepaald CI in overeenstemming met de regels die zijn gedefinieerd in de CMDB.

Hiermee wordt het effect aangegeven van een activering van de regel door een event. De `identifier`-uitvoer van `calculateImpact` wordt gebruikt als invoer voor "getImpactPath" op volgende pagina.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
impactCategory	Het type event waarmee de regel wordt geactiveerd die wordt

Parameter	Opmerking
	gesimuleerd.
IDs	Een verzameling van ID-elementen. voor meer informatie.
impactRulesNames	Een verzameling van `ImpactRuleName`-elementen.
severity	De ernstgraad van het trigger-event.

Uitvoer

Parameter	Opmerking
impactTopology	Zie "Topology" op pagina 243 voor meer informatie.
identifier	De sleutel voor de serverrespons.

getImpactPath

Met de methode `getImpactPath` wordt de topologiegrafiek opgehaald van het pad tussen het CI dat wordt beïnvloed en het CI dat zelf beïnvloedt.

De `identifier`-uitvoer van "calculateImpact" op vorige pagina wordt gebruikt als het `identifier`-invoerargument van `getImpactPath`.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
identifier	De sleutel voor de serverrespons die is geretourneerd met calculateImpact.
relation	Een relatie die is gebaseerd op een van de "ShallowRelation"'s wordt geretourneerd door calculateImpact in het element `impactTopology`.

Uitvoer

Parameter	Opmerking
impactPathTopology	Een verzameling `CIs` en een verzameling `ImpactRelations`.
comments	Alleen voor intern gebruik.

Een `ImpactRelations`-element bestaat uit een `ID`, `type`, `end1ID`, `end2ID`, een `rule` en een `action`.

getImpactRulesByNamePrefix

Met de methode `getImpactRulesByNamePrefix` worden regels opgehaald met een voorvoegselfilter.

Deze methode is van toepassing op impactregels die worden benoemd met een voorvoegsel waarmee de toepasselijke context wordt aangegeven, bijvoorbeeld `SAP_myrule`, `ORA_myrule`, enzovoort. Met deze methode worden alle impactregelnamen gefilterd voor de namen die beginnen met het voorvoegsel dat is opgegeven door het argument `ruleNamePrefixFilter`.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
ruleNamePrefixFilter	Een string die de eerste letters van de regelnamen bevat die moeten overeenkomen.

Uitvoer

Parameter	Opmerking
impactRules	`impactRules` bestaat uit nul of meer `impactRule`s. Een `impactRule`, waarmee het effect van een wijziging wordt opgegeven, bestaat uit `ruleName`, `description`, `queryName` en `isActive`.

Actual State Web Service API

De Actual State Web Service API wordt voornamelijk gebruikt door de Service Manager voor het ophalen van Actual State-gegevens voor een specifieke CMDB-ID of algemene ID en een specifieke klant-ID. De API vindt een overeenkomende query in de map **Integration/SM Query** en voert de TQL uit met de CMDB-ID of algemene ID als voorwaarde, en retourneert de uitvoer van de query.

Web Service-URL: http://[naam_machine]:8080/axis2/services/ucmdbSMService

Web Service-schema: http://[naam_machine]:8080/axis2/services/ucmdbSMService?xsd=xsd0

Flow

Wanneer de API-methode wordt aangeroepen, probeert deze een geschikte Query te vinden in de map **Integration/SM Query**. De API-methode probeert het type van de aangevraagde CMDBID/GlobalID eerst te vergelijken met een van de query's in voornoemde map door te zoeken naar een **QueryElement** met de naam **Root**, en als dat niet wordt gevonden wordt een willekeurige **QueryNode** van hetzelfde type als de aangevraagde CMDBID/GlobalID gebruikt. Zodra er een geschikte Query en QueryNode zijn gevonden, wordt de CMDBID/GlobalID als voorwaarde doorgegeven aan de QueryNode en wordt de Query uitgevoerd. Vervolgens wordt het resultaat geretourneerd naar de aanroeper van de API.

Het resultaat bewerken met transformaties

In sommige gevallen kunt u extra transformaties toepassen op de resulterende XML, zoals het bij elkaar optellen van de grootten van alle schijven en dat totaal als extra attribuut toevoegen aan het CI. Als u extra transformaties wilt toevoegen aan de TQL-resultaten, plaatst u als volgt een bron

met de naam **[tql_name].xslt** in de adapterconfiguratie: **Adapterbeheer > ServiceDeskAdapter7-1 >Configuratiebestanden> [tql_name].xslt.**

Logboeken voor Actual State Web Service API

De logconfiguratie voor UCMDB bevindt zich hier: **UCMDBServer/Conf/log** in de diverse ***.properties**-bestanden.

U kunt als volgt logbestanden van de SM Actual State-stroom weergeven::

1. Open het bestand **cmdb_soaapi.properties** en wijzig het logniveau als volgt in DEBUG: **loglevel=DEBUG**.
2. Open het bestand **fcmdb.properties** en wijzig het logniveau als volgt in DEBUG: **loglevel=DEBUG**.
3. Wacht 1 minuut totdat de server de wijzigingen heeft opgehaald.
4. Voer de Actual State uit vanuit de SM.
5. Bekijk de volgende logbestanden in **UCMDBServer/Runtime/log**:
   - cmdb.soaapi.log
   - fcmdb.log

Werkelijke status van gerepliceerde CI's inschakelen na wijziging van hoofdmapcontext

Als u de hoofdmapcontext hebt gewijzigd die wordt gebruikt om toegang te krijgen tot UCMDB, moet u de volgende configuratiewijzigingen aanbrengen om de werkelijke status van gerepliceerde CI's in te schakelen:

1. Open onder **UCMDBServer\deploy\axis2\WEB-INF** het bestand **web.xml**.
2. Voeg de volgende parameter **servlet init** aan AxisServlet toe (plak deze vier regels na regel 28):

   ```
   <init-param>

   <param-name>axis2.find.context</param-name>

   <param-value>false</param-value>

   </init-param>
   ```

   Met deze instelling wordt voorkomen dat Axis2 de hoofdmapcontext probeert te berekenen en wordt aangegeven dat er expliciet naar moet worden gezocht in **axis2.xml**.
3. Open onder **UCMDBServer\deploy\axis2\WEB-INF\conf** het bestand **axis2.xml**.
4. Verwijder bij regel 58 opmerkingen uit de parameter **contextRoot** en bewerk deze als volgt:

   ```
   <parameter name="contextRoot" locked="false">test/axis2</parameter>
   ```

(hierin is **test** de nieuwe hoofdmapcontext in **cmdb.xml**).

Opmerking: Er staat geen slash aan het begin van **test/axis2**.

Use cases

In de volgende use cases wordt uitgegaan van twee systemen:

- HP Universal CMDB-server
- Een systeem van derden dat een opslagplaats van configuratie-items bevat

In dit gedeelte vindt u de volgende onderwerpen:

De CMDB vullen

Use cases:

- Met een assetbeheersysteem van derden wordt de CMDB bijgewerkt met informatie die alleen in assetbeheer beschikbaar is
- Met een aantal systemen van derden kan de CMDB worden gevuld om een centrale CMDB aan te maken, waarmee wijzigingen kunnen worden bijgehouden en een impactanalyse kan worden uitgevoerd.
- Met een systeem van derden worden configuratie-items en relaties aangemaakt in overeenstemming met bedrijfslogica van derden om gebruik te maken van de query-mogelijkheden van CMDB

Query uitvoeren op de CMDB

Use cases:

- Met een systeem van derden worden de configuratie-items en relaties waarmee het SAP-systeem wordt vertegenwoordigd, opgehaald door de resultaten van de SAP TQL op te halen
- Met een systeem van derden wordt de lijst met Oracle-servers opgehaald die gedurende de laatste vijf uur zijn toegevoegd of gewijzigd
- Met een systeem van derden wordt de lijst met servers opgehaald waarvan de hostnaam de substring *lab* bevat
- Met een systeem van derden worden de elementen gezocht die betrekking hebben op een bepaald CI door de bijbehorende buren op te halen

Query uitvoeren op het klassemodel

Use cases:

- Een systeem van derden maakt mogelijk dat gebruikers de set gegevens kunnen opgeven die uit de CMDB moeten worden opgehaald. Er kan een gebruikersinterface worden samengesteld op

basis van het klassemodel om gebruikers de mogelijke eigenschappen te tonen en hen om vereiste gegevens te vragen. De gebruiker kan vervolgens de informatie kiezen die moet worden opgehaald.

- Met een systeem van derden wordt het klassemodel onderzocht wanneer de gebruiker geen toegang kan krijgen tot de CMDB-gebruikersinterface.

Wijzigingsimpact analyseren

Use case.

Met een systeem van derden wordt een lijst uitgevoerd van de bedrijfsservices die kunnen worden beïnvloed door een wijziging op een opgegeven host.

Voorbeelden

In dit gedeelte vindt u de volgende onderwerpen:

Voorbeeldbasisklasse

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

import org.apache.axis2.transport.http.HttpTransportProperties;
import java.net.MalformedURLException;
import java.net.URL;

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

UcmdbService stub;
CmdbContext context;

public void initDemo() {
    try {
        setStub(createUcmdbService("admin", "admin"));
        setContext();
```

```
    } catch (Exception e) {
        //handle exception
    }
}

public UcmdbService getStub() {
    return stub;
}

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

public CmdbContext getContext() {
    return context;
}

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

//connection to service  - for axis2/jibx client
private static final String PROTOCOL = "http";
private static final String HOST_NAME = "host_name";
private static final int PORT = 8080;
private static final String FILE = "/axis2/services/UcmdbService";
protected UcmdbService createUcmdbService
    (String username, String password) throws Exception{
        URL url;
        UcmdbServiceStub serviceStub;

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

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

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

```
        }
        return serviceStub;
 }
}
```

Query-voorbeeld

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.query.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.props.*;
import java.rmi.RemoteException;

public class QueryDemo extends Demo{
    UcmdbService stub;
    CmdbContext context;


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

        simplePredefinedPropertyCollection.addSimplePredefinedProperty
            (simplePredefinedProperty);
        predefinedProperties.setSimplePredefinedProperties
            (simplePredefinedPropertyCollection);
        customProperties.setPredefinedProperties
(predefinedProperties);
        request.setProperties(customProperties);
        try {
            GetCIsByTypeResponse response =
                getStub().getCIsByType(request);
            TopologyMap map =
                getTopologyMapResultFromCIs
```

```
                    (response.getCIs(), response.getChunkInfo());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }


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

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

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

predefinedProperties1.
        setSimpleTypedPredefinedProperties
            (simplePredefinedPropertyCollection1);
        customProperties1.
            setPredefinedTypedProperties
                (predefinedProperties1);
```

```
        typedProperties1.setProperties(customProperties1);
        properties.addTypedProperties(typedProperties1);

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

        simplePredefinedPropertyCollection2.
            addSimpleTypedPredefinedProperty
                (simplePredefinedProperty2);

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

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


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

```
        IntCondition intCondition = new IntCondition();
        IntProp intProp = new IntProp();
        intProp.setName("int_attr1");

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

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

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

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

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

    public void executeTopologyQueryByNameDemo() {
```

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

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


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

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

```
        strProp.setValue("%2000%");
        strProps.addStrProp(strProp);
        parameters.setStrProps(strProps);
        hostParametrizedNode.setParameters(parameters);
        request.addParameterizedNodes(hostParametrizedNode);
        ParameterizedNode diskParametrizedNode =
            new ParameterizedNode();

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

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

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

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


    public void executeTopologyQueryWithParametersDemo() {
        ExecuteTopologyQueryWithParameters request =
            new ExecuteTopologyQueryWithParameters();
```

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


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

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

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


    public void getCINeighboursDemo() {
```

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

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

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

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

            (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
        SimpleTypedPredefinedPropertyCollection
            simplePredefinedPropertyCollection2 =
                new SimpleTypedPredefinedPropertyCollection();
        simplePredefinedPropertyCollection2.
            addSimpleTypedPredefinedProperty
                (simplePredefinedProperty2);
        predefinedProperties2.
```

```
            setSimpleTypedPredefinedProperties
                (simplePredefinedPropertyCollection2);
        customProperties2.setPredefinedTypedProperties
            (predefinedProperties2);
        typedProperties2.setProperties(customProperties2);
        relationsProperties.addTypedProperties(typedProperties2);
        request.setRelationProperties(relationsProperties);

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

//get Topology Map for chunked/non-chunked result
    private TopologyMap getTopologyMapResult(TopologyMap topologyMap,
ChunkInfo chunkInfo) {
        if(chunkInfo.getNumberOfChunks() == 0) {
            return topologyMap;
        } else {

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

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

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

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

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

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

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

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


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

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

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

            for(int j=0 ;
                    j < topologyMap.getRelationNodes
().sizeRelationNodeList() ;
                    j++) {
                RelationNode relationNode2 =
                    topologyMap.getRelationNodes().getRelationNode(j);
                if(relationNode2.getLabel().equals
```

```
(relationNode.getLabel())){
                    Relations relationsTOAdd =
relationNode.getRelations();
                    Relations relations =
                        mergeRelationsGroups
                        (topologyMap.getRelationNodes().
                            getRelationNode(j).getRelations(),
                            relationsTOAdd);
                    topologyMap.getRelationNodes().
                        getRelationNode(j).setRelations(relations);
                    alreadyExist = true;
                }
            }

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

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

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

Bijwerkvoorbeeld

```
import com.hp.ucmdb.generated.params.update.AddCIsAndRelations;

import
com.hp.ucmdb.generated.params.update.AddCIsAndRelationsResponse;

import com.hp.ucmdb.generated.params.update.UpdateCIsAndRelations;

import com.hp.ucmdb.generated.params.update.DeleteCIsAndRelations;

import com.hp.ucmdb.generated.services.UcmdbFault;

import com.hp.ucmdb.generated.types.*;

import com.hp.ucmdb.generated.types.update.CIsAndRelationsUpdates;
```

```
import com.hp.ucmdb.generated.types.update.ClientIDToCmdbID;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.List;
public class UpdateDemo extends Demo{
   public void getAddCIsAndRelationsDemo() {
      AddCIsAndRelations request = new AddCIsAndRelations();
      request.setCmdbContext(getContext());
      request.setUpdateExisting(true);
      CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
      CIs cis = new CIs();
      List<CI> listCI = new ArrayList<CI>();
      CI ci = new CI();
      ID id = new ID();
      id.setString("temp1");
      id.setTemp(true);
      ci.setID(id);
      ci.setType("host");
      CIProperties props = new CIProperties();
      StrProps strProps = new StrProps();
      StrProp strProp = new StrProp();
      strProp.setName("host_key");
      String value = "blabla";
      strProp.setValue(value);
      strProps.getStrProps().add(strProp);
      props.setStrProps(strProps);
      ci.setProps(props);
      listCI.add(ci);
      cis.setCIs(listCI);
      updates.setCIsForUpdate(cis);
      request.setCIsAndRelationsUpdates(updates);
      try {
```

```
        AddCIsAndRelationsResponse response  = getStub().addCIsAndRelations
(request);
        for(int i = 0  ;  i < response.getCreatedIDsMaps().size() ; i++) {
           ClientIDToCmdbID idsMap = response.getCreatedIDsMaps().get(i);
           //do something
        }
      } catch (RemoteException e) {
        //handle exception
      } catch (UcmdbFault e) {
        //handle exception
      }
   }
   public void getUpdateCIsAndRelationsDemo() {
      UpdateCIsAndRelations request = new UpdateCIsAndRelations();
      request.setCmdbContext(getContext());
      CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
      CIs cis = new CIs();
      List<CI> listCI = new ArrayList<CI>();
      CI ci = new CI();
      ID id = new ID();
      id.setString("temp1");
      id.setTemp(true);
      ci.setID(id);
      ci.setType("host");
      CIProperties props = new CIProperties();
      StrProps strProps = new StrProps();
      StrProp hostKeyProp = new StrProp();
      hostKeyProp.setName("host_key");
      String hostKeyValue = "blabla";
      hostKeyProp.setValue(hostKeyValue);
      strProps.getStrProps().add(hostKeyProp);
      StrProp hostOSProp = new StrProp();
      hostOSProp.setName("host_os");
      String hostOSValue = "winXP";
```

```
    hostOSProp.setValue(hostOSValue);
    strProps.getStrProps().add(hostOSProp);
    StrProp hostDNSProp = new StrProp();
    hostDNSProp.setName("host_dnsname");
    String hostDNSValue = "dnsname";
    hostDNSProp.setValue(hostDNSValue);
    strProps.getStrProps().add(hostDNSProp);
    props.setStrProps(strProps);
    ci.setProps(props);
    listCI.add(ci);
    cis.setCIs(listCI);
    updates.setCIsForUpdate(cis);
    request.setCIsAndRelationsUpdates(updates);
    try {
      getStub().updateCIsAndRelations(request);
    } catch (RemoteException e) {
      //handle exception
    } catch (UcmdbFault e) {
      //handle exception
    }
  }
  public void getDeleteCIsAndRelationsDemo() {
    DeleteCIsAndRelations request = new DeleteCIsAndRelations();
    request.setCmdbContext(getContext());
    CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
    CIs cis = new CIs();
    List<CI> listCI = new ArrayList<CI>();
    CI ci = new CI();
    ID id = new ID();
    id.setString("stam");
    id.setTemp(true);
    ci.setID(id);
    ci.setType("host");
```

```
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp1 = new StrProp();
        strProp1.setName("host_key");
        String value1 = "for_delete";
        strProp1.setValue(value1);
        strProps.getStrProps().add(strProp1);
        props.setStrProps(strProps);
        ci.setProps(props);
        listCI.add(ci);
        cis.setCIs(listCI);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
        try {
          getStub().deleteCIsAndRelations(request);
        } catch (RemoteException e) {
          //handle exception
        } catch (UcmdbFault e) {
          //handle exception
        }
    }
    public static void main(String[] args) {
      try {
        UpdateDemo demo = new UpdateDemo();
        demo.initDemo();
        demo.getAddCIsAndRelationsDemo();
       } catch(Exception e){
        System.out.println(e.getMessage());
        e.printStackTrace();
       }
    }
}
```

Klassemodelvoorbeeld

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.classmodel.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import
com.hp.ucmdb.generated.types.classmodel.UcmdbClassModelHierarchy;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClass;
import java.rmi.RemoteException;

public class ClassmodelDemo extends Demo{


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

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


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


    public void getCmdbClassDefinitionDemo() {
        GetCmdbClassDefinition request =
```

```
            new GetCmdbClassDefinition();
        request.setCmdbContext(getContext());
        request.setClassName("className");

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

Impactanalysevoorbeeld

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.impact.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.impact.*;
import java.rmi.RemoteException;

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

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

    request.setIDs(ids);
    //set impact category
```

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

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

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

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

try {
    GetImpactPathResponse response2 =
```

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

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

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

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

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

```
        }
}
}
```

Aanmeldingsgegevens toevoegen - voorbeeld

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.DiscoveryService;
import com.hp.ucmdb.generated.services.DiscoveryServiceStub;
import com.hp.ucmdb.generated.types.BytesProp;
import com.hp.ucmdb.generated.types.BytesProps;
import com.hp.ucmdb.generated.types.CIProperties;
import com.hp.ucmdb.generated.types.CmdbContext;
import com.hp.ucmdb.generated.types.StrList;
import com.hp.ucmdb.generated.types.StrProp;
import com.hp.ucmdb.generated.types.StrProps;

public class test {
    static final String HOST_NAME = "hostname";
    static final int PORT = 8080;
    private static final String PROTOCOL = "http";
    private static final String FILE =
"/axis2/services/DiscoveryService";

    private static final String PASSWORD = "admin";
    private static final String USERNAME = "admin";

    private static CmdbContext cmdbContext = new CmdbContext("ws
tests");

    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest("Range IPs
by ICMP", cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protcol
        addNTCMDCredentialsEntry();
    }

    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
```

```
        StrList domains =
            discoveryService.getDomainsNames(new
GetDomainsNamesRequest(cmdbContext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create
credentials");
            return;
        }
        String domainName = domains.getStrValue(0);
        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user",
newCredsProperties);
        setStringProperties("ntadminprotocol_ntdomain", "test doamin",
newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(new
AddCredentialsEntryRequest(domainName,  "ntadminprotocol",
newCredsProperties, cmdbContext));
        System.out.println("new credentials craeted for domain: " +
domainName + " in ntcmd protocol");
    }

    private static void setPasswordProperty(CIProperties
newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }

    private static void setStringProperties(String propertyName,
String value, CIProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }

    private static void getProbesInfo(DiscoveryService
discoveryService) throws Exception {
        GetDomainsNamesResponse result =
discoveryService.getDomainsNames(new GetDomainsNamesRequest
```

```
(cmdbContext ));
        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName = result.getDomainNames().getStrValue
(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(new
GetProbesNamesRequest(domainName, cmdbContext));
            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames
().sizeStrValueList(); i++) {
                String probeName = probesResult.getProbesNames
().getStrValue(i);
                // Check if connected
                IsProbeConnectedResponce connectedRequest =
                    discoveryService.isProbeConnected(new
IsProbeConnectedRequest(domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected
();
                // Do something ...
                System.out.println("probe " + probeName + "
isconnect=" + isConnected);
            }
        }
    }

    private static DiscoveryService getDiscoveryService() throws
Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub = new
DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth = new
HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty
(HTTPConstants.AUTHENTICATE,auth);

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

```
} // End class
```

Hoofdstuk 11

API Data Flow-beheer

In dit hoofdstuk vindt u de volgende informatie:

Data Flow-beheer API - overzicht

In dit hoofdstuk wordt beschreven hoe derden of aangepaste tools de HP Data Flow-beheer Web Service kunnen genereren om Data Flow-beheer te beheren.

Raadpleeg de *HP Discovery and Dependency Mapping Schema Reference* voor volledige documentatie over de beschikbare bewerkingen. De bestanden bevinden zich in de volgende map:

<UCMDB-hoofdmap>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

Conventies

In dit hoofdtuk worden de volgende conventies gehanteerd:

- Deze `Element`-stijl geeft aan dat een item een entiteit in de database is of een element zoals gedefinieerd in het schema, met inbegrip van structuren die zijn doorgegeven aan werden geretourneerd van methoden. Gewone tekst geeft aan dat er in algemene zin over het item wordt gesproken.
- Voor elementen en methode-argumenten in Data Flow-beheer worden hoofdletters en kleine letters op dezelfde manier als in het schema gebruikt. Dit betekent doorgaans dat een klassenaam of algemene verwijzing naar een exemplaar van de klasse in hoofdletters wordt geschreven. Een element of een argument voor een methode wordt niet in hoofdletters geschreven. Een `referentie` is bijvoorbeeld een element van het type `Credential` dat aan een methode wordt doorgegeven.

Data Flow-beheer Web Service

De HP Data Flow Management Web Service is een API voor het integreren van toepassingen met HP Universal CMDB. Met de API worden methoden verschaft voor het volgende:

- **Aanmeldingsgegevens beheren**. Weergeven, toevoegen, bijwerken en verwijderen.
- **Taken beheren**. Status weergeven, activeren en deactiveren.
- **Probe-bereiken beheren**. Weergeven, toevoegen en bijwerken.
- **Triggers beheren**. Een trigger-CI toevoegen of verwijderen en een trigger-TQL toevoegen, verwijderen of uitschakelen.
- **Algemene gegevens weergeven**. Gegevens op domeinen en probes.

Gebruikers van de HP Data Flow Management Web Service moeten vertrouwd zijn met het volgende:

- De SOAP-specificatie
- Een objectgeoriënteerde programmeertaal, zoals C++, C# of Java
- HP Universal CMDB
- Data Flow-beheer

Rechten

De beheerder verstrekt aanmeldingsgegevens om verbinding te maken met de webservice. De rechtenniveaus zijn Weergeven, Bijwerken en Uitvoeren. Als u de rechten wilt weergeven die voor elke bewerking vereist zijn, raadpleegt u de aanvraagdocumentatie van elke bewerking. Zie de *HP Discovery and Dependency Mapping Schema Reference*..

Webservice aanroepen

Met de HP Discovery and Dependency Mapping Web Service kunt u servermethoden aanroepen met behulp van SOAP-standaardprogrammeertechnieken. Als de instructie niet kan worden geparseerd of als er een probleem is met het aanroepen van de methode, genereren de API-methoden een `SoapFault`-uitzondering. Wanneer een `SoapFault`-uitzondering wordt gegenereerd, worden een of meer velden voor foutberichten, foutcodes en uitzonderingsberichten gevuld. Als er geen fout is, worden de resultaten van de aanroep geretourneerd.

De service aanroepen:

- `Protocol: http of https (afhankelijk van serverconfiguratie)`
- `URL: <UCMDB-server>:8080/axis2/services/DiscoveryService`
- `Standaardwachtwoord: "admin"`
- `Standaard gebruikersnaam: "admin"`

SOAP-programmeurs kunnen toegang krijgen tot de WSDL via:

- `axis2/services/DiscoveryService?wsdl`

Methoden voor Data Flow-beheer

Dit gedeelte bevat een lijst met webservicebewerkingen en een korte samenvatting van het gebruik ervan. Zie *HP Discovery and Dependency Mapping Schema Reference* voor volledige documentatie van de aanvraag en de respons van elke bewerking.

In dit gedeelte vindt u de volgende onderwerpen:

Gegevensstructuren

Enkele van de gegevensstructuren worden gebruikt in de Data Flow-beheer Web Service API.

CIProperties

`CIProperties` is een verzameling van verzamelingen. Elke verzameling bevat eigenschappen van een ander gegevenstype. Er kan bijvoorbeeld een `dateProps`-verzameling zijn, een `strListProps`-verzameling, een `xmlProps`-verzameling, enzovoort.

Elk type verzameling bevat afzonderlijke eigenschappen van het gegeven type. De namen van deze eigenschapelementen zijn hetzelfde als die van de container, maar dan in het enkelvoud. Zo bevat `datePropsdateProp`-elementen. Elke eigenschap is een naam-waarde paar.

Zie CIProperties in de *HP Discovery and Dependency Mapping Schema Reference*.

IPList

Een lijst met `IP`-elementen, waarvan elk een IPv4-adres bevat.

Zie IPList in de *HP Discovery and Dependency Mapping Schema Reference*.

IPRange

Een `IPRange` heeft twee elementen, te weten `Start` en `End`. Elk bevat een `Address`-element, dat uit een IPv4-adres bestaat.

Zie IPLRange in de *HP Discovery and Dependency Mapping Schema Reference*.

Scope

Twee `IPRanges`. `Exclude` is een verzameling `IPRanges` die niet in de taak moeten worden opgenomen. `Exclude` is een verzameling `IPRanges` die wel in de taak moeten worden opgenomen.

Zie Scope in de *HP Discovery and Dependency Mapping Schema Reference*.

Discovery-taakmethoden beheren

activateJob

Hiermee wordt de opgegeven taak geactiveerd.

Zie "Codevoorbeeld" op pagina 299.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	De naam van de taak.

deactivateJob

Hiermee wordt de opgegeven taak gedeactiveerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	De naam van de taak.

dispatchAdHocJob

Hiermee wordt een taak op de probe ad-hoc verzonden. De taak moet actief zijn en het opgegeven trigger-CI bevatten.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	De naam van de taak.
CIID	De ID van het trigger-CI.
ProbeName	De naam van de probe.
Time-out	In milliseconden

getDiscoveryJobsNames

Hiermee wordt de lijst met taaknamen geretourneerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.

Uitvoer

Parameter	Opmerking
strList	De lijst met taaknamen.

isJobActive

Hiermee wordt gecontroleerd of de taak actief is.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	De naam van de te controleren taak.

Uitvoer

Parameter	Opmerking
JobState	Waar als de taak actief is.

Triggermethoden beheren

addTriggerCI

Hiermee wordt een nieuw trigger-CI aan de opgegeven taak toegevoegd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	De naam van de taak.
CIID	De ID van het trigger-CI.

addTriggerTQL

Hiermee wordt een nieuwe trigger-TQL aan de opgegeven taak toegevoegd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	De naam van de taak.
TqlName	De naam van de toe te voegen TQL.

disableTriggerTQL

Hiermee wordt voorkomen dat de TQL de taak activeert. De taak wordt echter niet definitief verwijderd uit de lijst met query's waarmee de taak wordt geactiveerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.

Parameter	Opmerking
JobName	De naam van de taak.

removeTriggerCI

Hiermee wordt het opgegeven CI verwijderd uit de lijst met CI's waarmee de taak wordt geactiveerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	De naam van de taak.
CIID	De ID van het trigger-CI.

removeTriggerTQL

Hiermee wordt de opgegeven TQL verwijderd uit de lijst met query's waarmee de taak wordt geactiveerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	Verzameling van te controleren taaknamen.
CIID	De ID van de te verwijderen TQL.

setTriggerTQLProbesLimit

Hiermee worden de probes waarin de TQL actief is in de taak, beperkt tot de opgegeven lijst.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
JobName	De naam van de taak.
tqlName	De TQL-naam.
probesLimit	De lijst met probes waarvoor de TQL actief is.

Domein- en probegegevensmethoden

getDomainType

Hiermee wordt het domeintype geretourneerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	De naam van het domein.

Uitvoer

Parameter	Opmerking
domainType	Het domeintype.

getDomainsNames

Hiermee worden de namen van de huidige domeinen geretourneerd.

Zie "Codevoorbeeld" op pagina 299.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.

Uitvoer

Parameter	Opmerking
domainNames	De lijst met domeinnamen.

getProbeIPs

Hiermee worden de IP-adressen van de opgegeven probe geretourneerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het te controleren domein.
probeName	De naam van de probe die op dat domein wordt gebruikt.

Uitvoer

Parameter	Opmerking
probeIPs	De "IPList" van de adressen in de probe.

getProbesNames

Hiermee worden de namen van de probes in het opgegeven domein geretourneerd.

Zie "Codevoorbeeld" op pagina 299.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het te controleren domein.

Uitvoer

Parameter	Opmerking
probesName	De lijst met probes op het domein.

getProbeScope

Hiermee wordt de scope-definitie van de opgegeven probe geretourneerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het te controleren domein.
probeName	De naam van de probe.

Uitvoer

Parameter	Opmerking
probeScope	De "Scope" van de probe.

isProbeConnected

Hiermee wordt gecontroleerd of de opgegeven probe is verbonden.

Zie "Codevoorbeeld" op pagina 299.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het te controleren domein.
probeName	De te controleren probe.

Uitvoer

Parameter	Opmerking
isConnected	Waar als de probe verbonden is.

updateProbeScope

Hiermee wordt de scope van de opgegeven probe ingesteld, waarbij de bestaande scope wordt overschreven.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het domein.
probeName	De bij te werken probe.
newScope	De voor de probe in te stellen "Scope".

Methoden voor aanmeldingsgegevens

addCredentialsEntry

Hiermee worden aanmeldingsgegevens toegevoegd aan het opgegeven protocol voor het opgegeven domein.

Zie "Codevoorbeeld" op pagina 299.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het domein dat moeten worden bijgewerkt.
protocolName	De naam van het protocol.
credentialsEntryParameters	De "CIProperties"-verzameling van de nieuwe referenties.

Uitvoer

Parameter	Opmerking
credentialsEntryID	De CI-ID van de nieuwe referentie.

getCredentialsEntriesIDs

Hiermee worden de ID's geretourneerd van de aanmeldingsgegevens die voor het opgegeven protocol zijn gedefinieerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.

Parameter	Opmerking
domainName	Het domein waarvoor de referenties zijn bestemd.
protocolName	De naam van een protocol dat op dat domein wordt gebruikt.

Uitvoer

Parameter	Opmerking
credentialsEntryIDs	De lijst met referentie-ID's voor het protocol op het domein.

getCredentialsEntry

Hiermee worden de aanmeldingsgegevens geretourneerd die voor het opgegeven protocol zijn gedefinieerd. Versleutelde attributen worden leeg geretourneerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het domein waarvoor de referenties zijn bestemd.
protocolName	Een op dat domein gebruikt protocol.
credentialsEntryID	De op te halen referentie-ID.

Uitvoer

Parameter	Opmerking
credentialsEntryParameters	De "CIProperties"-verzameling van de referenties.

removeCredentialsEntry

Hiermee worden de opgegeven aanmeldingsgegevens uit het protocol verwijderd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het domein.
protocolName	Een op het domein gebruikt protocol.
credentialsEntryID	De ID van de te verwijderen referentie.

updateCredentialsEntry

Hiermee worden nieuwe waarden ingesteld voor eigenschappen van de opgegeven aanmeldingsgegevens.

De bestaande eigenschappen worden verwijderd en deze eigenschappen worden ingesteld. Elke eigenschap waarvan de waarde niet wordt ingesteld in deze aanroep, blijft ongedefinieerd.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
domainName	Het domein waar referenties moeten worden bijgewerkt.
protocolName	Een op het domein gebruikt protocol.
credentialsEntryID	De ID van de bij te werken referenties.
credentialsEntryParameters	De "CIProperties"-verzameling die wordt ingesteld als eigenschappen voor de referenties.

Methoden voor gegevensvernieuwing

rediscoverCIs

Hiermee worden de triggers gezocht waarmee de opgegeven CI-objecten zijn gedetecteerd, en worden deze triggers opnieuw uitgevoerd.

rediscoverCIs wordt asynchroon uitgevoerd. Roep **checkDiscoveryProgress** aan om te bepalen wanneer de heruitvoering van de discovery voltooid is.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
CmdbIDs	Verzameling ID's van de opnieuw te detecteren objecten.

Uitvoer

Parameter	Opmerking
isSucceed	Waar als het opnieuw detecteren van CI's is gelukt.

checkDiscoveryProgress

Hiermee wordt de voortgang van de meest recente **rediscoverCIs**-aanroep van de opgegeven ID's geretourneerd. De respons is een waarde van `0` tot `1`. Wanneer de respons `1` is, is de aanroep **rediscoverCIs** voltooid.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
CmdbIDs	Verzameling ID's van de opnieuw te detecteren objecten.

Uitvoer

Parameter	Opmerking
progress	Een voltooide taak heeft een voortgang van 1. Taken die nog niet zijn voltooid, zijn een breukgetal kleiner dan 1.

rediscoverViewCIs

Hiermee worden de triggers gezocht waarmee de gegevens zijn aangemaakt om de opgegeven weergave te vullen, en worden deze triggers opnieuw uitgevoerd.

rediscoverViewCIs wordt asynchroon uitgevoerd. Roep **checkViewDiscoveryProgress** aan om te bepalen wanneer de heruitvoering van de discovery voltooid is.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
viewName	De te controleren weergaven.

Uitvoer

Parameter	Opmerking
isSucceed	Waar als het opnieuw detecteren van CI's is gelukt.

checkViewDiscoveryProgress

Hiermee wordt de voortgang van de meest recente **rediscoverViewCIs**-aanroep in de opgegeven weergave geretourneerd. De respons is een waarde van 0 tot 1. Wanneer de respons 1 is, is de aanroep **rediscoverCIs** voltooid.

Invoer

Parameter	Opmerking
cmdbContext	Zie "CmdbContext" op pagina 240 voor meer informatie.
viewName	De verzameling te controleren weergaven.

Uitvoer

Parameter	Opmerking
progress	Een voltooide taak heeft een voortgang van 1. Taken die nog niet zijn voltooid, zijn een breukgetal kleiner dan 1.

Codevoorbeeld

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.*;
import com.hp.ucmdb.generated.types.*;
public class test {
    static final String HOST_NAME = "<my_hostname>";
    static final int PORT = 8080;
    private static final String PROTOCOL = "http";
    private static final String FILE =
"/axis2/services/DiscoveryService";

    private static final String PASSWORD = "<my_password>";
    private static final String USERNAME = "<my_username>";

    private static CmdbContext cmdbContext = new CmdbContext("ws
tests");

    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest(
            "Range IPs by ICMP", cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protcol
        addNTCMDCredentialsEntry();
    }

    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(
                new GetDomainsNamesRequest(cmdbContext)).
                getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create
credentials");
            return;
        }
        String domainName = domains.getStrValue(0);
        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();
```

```
        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user",
newCredsProperties);
        setStringProperties("ntadminprotocol_ntdomain",
            "test doamin", newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(
            new AddCredentialsEntryRequest(domainName,
            "ntadminprotocol",  newCredsProperties, cmdbContext));
        System.out.println("new credentials craeted for domain: " +
domainName + " in ntcmd protocol");
    }

    private static void setPasswordProperty(CIProperties
newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }

    private static void setStringProperties(String propertyName,
String value, CIProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }


    private static void getProbesInfo(DiscoveryService
discoveryService) throws Exception {
        GetDomainsNamesResponse result =
discoveryService.getDomainsNames(new GetDomainsNamesRequest
(cmdbContext ));
        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName =
            result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(
                    new GetProbesNamesRequest(domainName,
cmdbContext));
```

```
            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames
().sizeStrValueList(); i++) {
                String probeName = probesResult.getProbesNames
().getStrValue(i);
                // Check if connected
                IsProbeConnectedResponce connectedRequest =
                    discoveryService.isProbeConnected(
                        new IsProbeConnectedRequest(
                        domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected
();
                // Do something ...
                System.out.println("probe " + probeName + "
isconnect=" + isConnected);
            }
        }
    }

    private static DiscoveryService getDiscoveryService() throws
Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub =
                new DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth =
                new HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty(
                HTTPConstants.AUTHENTICATE,auth);

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


Java™
COMPATIBLE