HP Universal CMDB

Para los sistemas operativos Windows y Red Hat Enterprise

Versión de software: 10.00

Guía de referencia para el desarrollador

Fecha de publicación del documento: Junio de 2012

Fecha de lanzamiento del software: Junio de 2012

Avisos legales

Garantía

Las únicas garantías de los productos y servicios HP se exponen en el certificado de garantía que acompaña a dichos productos y servicios. El presente documento no debe interpretarse como una garantía adicional. HP no es responsable de omisiones, errores técnicos o de edición contenidos en el presente documento.

La información contenida en esta página está sujeta a cambios sin previo aviso.

Leyenda de derechos limitados

Software informático confidencial. Es necesario disponer de una licencia válida de HP para su posesión, uso o copia. De conformidad con FAR 12.211 y 12.212, el Gobierno estadounidense dispone de licencia de software informático de uso comercial, documentación del software informático e información técnica para elementos de uso comercial con arreglo a la licencia estándar para uso comercial del proveedor.

Aviso de copyright

Avisos de marcas comerciales

Adobe™ es una marca comercial de Adobe Systems Incorporated.

Microsoft® y Windows® son marcas comerciales registradas estadounidenses de Microsoft Corporation.

UNIX® es una marca comercial registrada de The Open Group.

Actualizaciones de la documentación

La página de título de este documento contiene la siguiente información de identificación:

- Número de versión del software, que indica la versión del software.
- Fecha de publicación del documento, que cambia cada vez que se actualiza el documento.
- Fecha de lanzamiento del software, que indica la fecha desde la que está disponible esta versión del software.

Para buscar actualizaciones recientes o verificar que está utilizando la edición más reciente de un documento, visite:

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

Este sitio requiere que esté registrado como usuario de HP Passport. Para registrarse y obtener un ID de HP Passport, visite:

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

O haga clic en el enlace **New user registration** (Registro de nuevos usuarios) de la página de registro de HP Passport.

Asimismo, recibirá ediciones actualizadas o nuevas si se suscribe al servicio de soporte del producto correspondiente. Póngase en contacto con su representante de ventas de HP para obtener más información.

Soporte

Visite el sitio web HP Software Support Online en:

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

Este sitio web proporciona información de contacto y detalles sobre los productos, servicios y soporte que ofrece HP Software.

HP Software Support Online brinda a los clientes la posibilidad de auto-resolución de problemas. Ofrece una forma rápida y eficaz de acceder a las herramientas de soporte técnico interactivo necesarias para gestionar su negocio. Como cliente preferente de soporte, puede beneficiarse de utilizar el sitio web de soporte para:

- Buscar los documentos de la Base de conocimiento que le interesen
- Enviar y realizar un seguimiento de los casos de soporte y las solicitudes de mejora
- Descargar revisiones de software
- Gestionar contratos de soporte
- Buscar contactos de soporte de HP
- Consultar la información sobre los servicios disponibles
- Participar en debates con otros clientes de software
- Investigar sobre formación de software y registrarse para recibirla

Para acceder a la mayor parte de las áreas de soporte es necesario que se registre como usuario de HP Passport. En muchos casos también será necesario disponer de un contrato de soporte. Para registrarse y obtener un ID de HP Passport, visite:

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

Para obtener más información sobre los niveles de acceso, visite:

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

Descargo de responsabilidad para la versión en PDF de la Ayuda en línea

Este documento es una versión en PDF de la Ayuda en línea. Este archivo PDF se incluye para que pueda imprimir con facilidad varios temas de la información de ayuda o leer la Ayuda en línea en formato PDF.

Nota: algunos temas no se convierten correctamente a PDF, lo que causa problemas de formato. Algunos elementos de la Ayuda en línea se han suprimido por completo de la versión PDF. Esos temas problemáticos se pueden imprimir correctamente dentro de la Ayuda en línea.

Contenido

Creación de adaptadores de detección e integración

Capítulo 1

Desarrollo y escritura de adaptadores

Este capítulo incluye:

Información general del desarrollo y escritura de adaptadores

Antes de comenzar la planificación real del desarrollo de los nuevos adaptadores, es importante comprender los procesos e interacciones habitualmente asociados a este desarrollo.

Las secciones siguientes le pueden ayudar a comprender lo que necesita saber y hacer para gestionar y ejecutar satisfactoriamente un proyecto de desarrollo de detección.

En este capítulo:

- Se supone un conocimiento práctico de HP Universal CMDB y cierta familiaridad básica con los elementos del sistema. Lo que significa una ayuda en el proceso de aprendizaje y que no proporciona una guía completa.
- Se abarcan las etapas de planificación, investigación e implementación del nuevo contenido detectado para HP Universal CMDB, junto con las directrices y consideraciones que hay que tener en cuenta.
- Debe proporcionarse información sobre las API de clave de Data Flow Management Framework. Para ver toda la documentación sobre las API disponibles, consulte la *Referencia de API de HP Universal CMDB Data Flow Management*. (Hay otras API no formales que, aunque se usan en los adaptadores de serie, pueden estar sujetas a cambios).

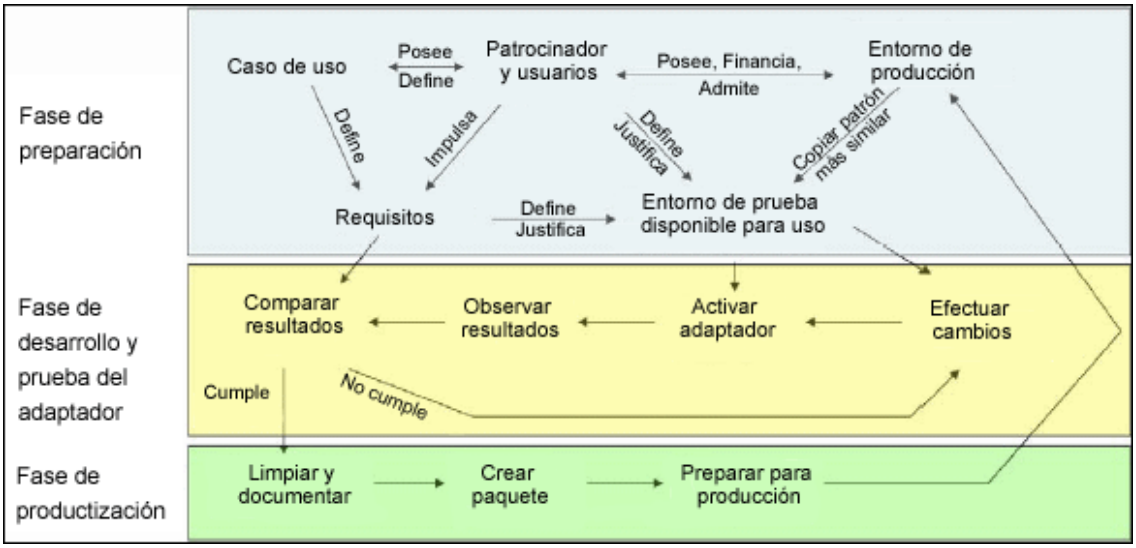
Creación de contenido

Esta sección incluye:

- "Ciclo de desarrollo del adaptador" abajo
- "Data Flow Management e integración" en la página 18
- "Asociación del valor de negocio con el desarrollo de detección" en la página 19
- "Investigación de los requisitos de integración" en la página 20

Ciclo de desarrollo del adaptador

En la ilustración siguiente se muestra un gráfico de flujo para la escritura de adaptadores. La mayor parte del tiempo se reserva a la sección central, que consiste en un bucle iterativo de desarrollo y prueba.



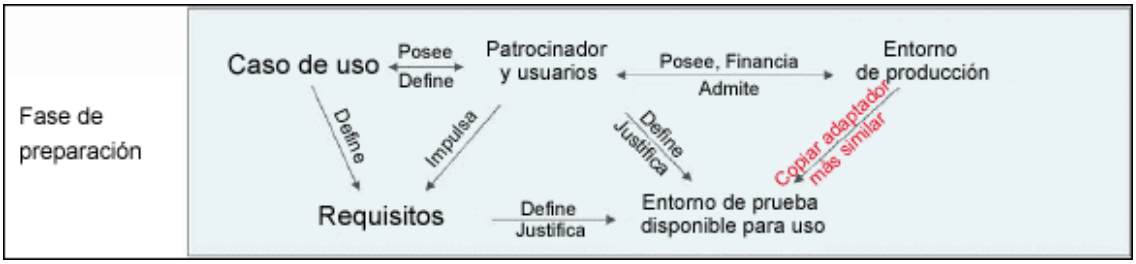
Cada fase del desarrollo del adaptador se construye sobre la anterior.

Cuando esté satisfecho con el aspecto y funcionamiento del adaptador, ya está listo para empaquetarlo. Con el Administrador de paquetes de UCMDB o exportando manualmente los componentes, cree un archivo de paquete *.zip. Como práctica recomendada, debería desplegar y probar este paquete en otro sistema de UCMDB antes de lanzarlo a producción para asegurarse de que todos los componentes están representados y correctamente empaquetados. Para obtener información detallada sobre el empaquetado, consulte "Administrador de paquetes" en *HP Universal CMDB – Guía de administración*.

En las secciones siguientes se amplía cada una de las fases, mostrando los pasos más críticos y las prácticas recomendadas:

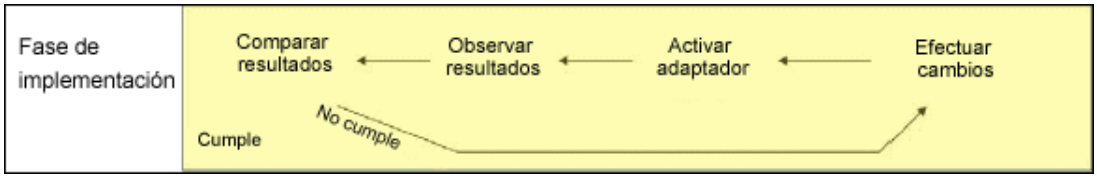
- "Fase de investigación y preparación" en la página siguiente
- "Desarrollo y prueba del adaptador" en la página siguiente
- "Empaquetado y conversión a producto del adaptador " en la página 17

Fase de investigación y preparación



La fase Investigación y preparación abarca las necesidades motrices de negocio y los casos de uso, y también considera la protección de las instalaciones necesarias para desarrollar y probar el adaptador.

1. Al planificar la modificación de un adaptador existente, el primer paso técnico es hacer una copia de seguridad del adaptador y asegurarse de que puede devolverlo a su estado original. Si piensa crear un nuevo adaptador, copie el adaptador más similar y guárdelo con un nombre apropiado. Para obtener más información, consulte "Panel Recursos" en *HP Universal CMDB – Guía de Administración de Data Flow*.
2. Investigue cómo el adaptador debería recopilar los datos:
   - Utilice herramientas o protocolos externos para obtener los datos
   - Desarrolle cómo el adaptador debería crear CI en función de los datos
   - Ahora sabe cómo debería ser un adaptador similar
3. Determine el adaptador más similar en función de:
   - Los mismos CI creados
   - Los mismos protocolos usados (SNMP)
   - El mismo tipo de destino (por tipo de sistema operativo, versiones, etc.)
4. Copie el paquete completo.
5. Descomprímalo en el espacio de trabajo y cambie el nombre de los archivos del adaptador (XML) y Jython (.py).



Desarrollo y prueba del adaptador

La fase Desarrollo y prueba del adaptador es un proceso sumamente iterativo. Cuando el adaptador empieza a tomar forma, comienza a realizar la prueba en los casos de uso finales, a realizar cambios, a probar de nuevo y a repetir este proceso hasta que el adaptador cumple con los requisitos.

Inicio y preparación de la copia

- Modifique las partes XML del adaptador: Nombre (ID) en línea 1, tipos de CI creados y nombre de la secuencia de comandos Jython llamada.
- Haga que la copia se ejecute con resultados idénticos al adaptador original.
- Defina como comentarios la mayor parte del código, especialmente el código que produzca los resultados críticos.

Desarrollo y prueba

- Utilice otro código de ejemplo para desarrollar cambios
- Pruebe el adaptador ejecutándolo
- Utilice una vista dedicada para validar los resultados complejos, busque validar los resultados sencillos

Empaquetado y conversión a producto del adaptador

La fase Empaquetado y conversión a producto del adaptador constituye la última fase del desarrollo. Como práctica recomendada, se debe realizar una revisión final para limpiar de restos de depuración, documentos y comentarios, para examinar las consideraciones de seguridad, etc., antes de desplazarse al empaquetado. Siempre debería disponer al menos de un documento Léame para explicar el funcionamiento interno del adaptador. Alguien (quizá el mismo usuario) puede tener que examinar este adaptador en el futuro y le resultará de gran ayuda incluso la documentación más limitada.

Limpieza y documentación

- Suprima la depuración
- Marque como comentarios todas las funciones y agregue algunos comentarios de apertura en la sección principal
- Cree un TQL de ejemplo y haga que lo pruebe el usuario

Crear paquete

- Exporte los adaptadores, el TQL y demás con el Administrador de paquetes. Para obtener más información, consulte "Administrador de paquetes" en *HP Universal CMDB – Guía de administración*.
- Compruebe cualquier dependencia entre el paquete y otros paquetes, por ejemplo, si los CI creados por esos paquetes son CI de entradas al adaptador.
- Utilice el Administrador de paquetes para crear un archivo zip de paquetes. Para obtener más información, consulte "Administrador de paquetes" en *HP Universal CMDB – Guía de administración*.

- Pruebe el despliegue quitando partes del nuevo contenido y volviendo a desplegarlo, o despliegue otro sistema de prueba.

Data Flow Management e integración

Los adaptadores DFM pueden integrarse con otros productos. Tenga en cuenta las siguientes definiciones:

- DFM recopila contenido específico de varios destinos.
- La integración recopila varios tipos de contenido de un sistema.

Tenga en cuenta que estas definiciones no distinguen entre los métodos de colección. Tampoco el DFM. El proceso del desarrollo de un nuevo adaptador es el mismo que el proceso de desarrollo de una nueva integración. Realiza la misma investigación, las mismas elecciones de adaptadores nuevos frente a existentes, escribe los adaptadores de la misma forma, etc. Solo cambian unas pocas cosas:

- La planificación final del adaptador. Los adaptadores de integración pueden ejecutarse con más frecuencia que los de detección, pero depende de los casos de uso.
- CI de entrada:
  - Integración: sin CI de activación para ejecutarse sin entrada: se pasa un nombre de archivo u origen mediante el parámetro del adaptador.
  - Discovery (detección): usa los CI de CMDB regulares como entrada.

En los proyectos de integración, casi siempre se debería reutilizar un adaptador existente. La dirección de la integración (desde HP Universal CMDB a otro producto, o desde otro producto a HP Universal CMDB) puede afectar a su enfoque al desarrollo. Puede copiar para sus propios usos de los paquetes de campos disponibles, utilizando técnicas probadas.

De HP Universal CMDB a otro proyecto:

- Cree un TQL que produzca los CI y las relaciones que se van a exportar.
- Utilice un adaptador de contenedor genérico para ejecutar el TQL y escriba los resultados en un archivo XML para que lo lea el producto externo.

> **Nota:** Para obtener ejemplos de paquetes de campos, póngase en contacto con HP Software Support.

Para integrar otro producto en HP Universal CMDB, dependiendo de cómo el otro producto muestra sus datos, el adaptador de integración actúa de manera diferente:

Tipo de integración	Ejemplo de referencia que se va a reutilizar
Acceder directamente a la base de datos del producto	HP ED
Leer en un archivo csv o xml producido por una exportación	HP ServiceCenter
Acceder a la API de un producto	BMC Atrium/Remedy

Asociación del valor de negocio con el desarrollo de detección

El caso de uso para desarrollar nuevo contenido de detección debe estar orientado por un caso de negocio y un plan para producir un valor de negocio. Es decir, el objetivo de asignar componentes del sistema a los CI y agregarlos a CMDB es proporcionar valor de negocio.

El contenido no siempre se va a utilizar para la asignación de aplicaciones, aunque es un paso intermedio habitual para muchos casos de uso. Con independencia del uso final del contenido, el plan debería responder a estas preguntas de este enfoque:

- ¿Quién es el consumidor? ¿Cómo debería actuar el consumidor en relación a la información proporcionada por los CI (y las relaciones entre ellos)? ¿Cuál es el contexto de negocio en el que se van a visualizar los CI y las relaciones? ¿Es el consumidor de estos CI una persona, un producto o ambos?
- Cuando existe la combinación perfecta de CI y relaciones en CMDB, ¿cómo piensa utilizarlos para producir un valor de negocio?
- ¿Cómo sería la asignación perfecta?
  - ¿Qué término describiría de manera más significativa las relaciones entre cada CI?
  - ¿Qué tipos de CI son los que habría que incluir en primer lugar?
  - ¿Cuál es el uso final y el usuario final de la asignación?
- ¿Cuál sería el diseño de informe perfecto?

Cuando se haya establecido la justificación del negocio, el paso siguiente consiste en plasmar el valor de negocio en un documento. Esto implica representar el mapa perfecto usando una herramienta de dibujo y comprender el impacto y las dependencias entre los CI, informes, cómo se realiza el seguimiento de los cambios, qué cambio es importante, la monitorización, la conformidad y el valor de negocio adicional, tal como requieren los casos de uso.

Este dibujo (o modelo) se conoce como **plano técnico**.

Por ejemplo, si es fundamental para la aplicación saber cuándo ha cambiado un determinado archivo de configuración, éste debe representarse y vincularse al CI apropiado (al que está relacionado) en el mapa dibujado.

Trabaje con un SME (experto en la materia del asunto) del área, que es el usuario final del contenido desarrollado. Este experto debería apuntar a las entidades críticas (los CI con atributos y relaciones) que deben existir en el CMDB para proporcionar valor de negocio.

Un método podría ser proporcionar un cuestionario al propietario de la aplicación (también el SME en este caso). El propietario debería poder especificar los objetivos y el plano técnico anterior. El propietario al menos debe proporcionar una arquitectura actual de la aplicación.

Debería representar únicamente datos críticos y no datos innecesarios: siempre podrá mejorar posteriormente el adaptador. El objetivo debería ser configurar una detección limitada que funcione y proporcione valor. La representación de grandes cantidades de datos produce mapas más impresionantes pero pueden resultar confusos y necesitar mucho tiempo en su desarrollo.

Cuando el modelo y el valor de negocio están claros, continúe al paso siguiente. Esta etapa se puede volver a revisar cuando se proporcione información más concreta en las etapas siguientes.

Investigación de los requisitos de integración

El requisito previo de esta etapa es un **plano técnico** de los CI y relaciones que el DFM tiene que detectar, que deberían incluir los atributos que se van a detectar. Para obtener más información, consulte "Información general del desarrollo y escritura de adaptadores" en la página 14.

Esta sección incluye los siguientes temas:

Modificación de un adaptador existente

Puede modificar un adaptador existente cuando existe un adaptador de campo o uno de serie, pero:

- no detecta atributos específicos que son necesarios
- no se está detectando un tipo de destino (OS) específico o se está detectando de manera incorrecta
- no se está detectando o creando una relación específica

Si un adaptador existente realiza parte del trabajo, pero no todo, su primer enfoque debería ser evaluar los adaptadores existentes y comprobar si uno de ellos hace más o menos lo que se precisa; si lo hace, puede modificar el adaptador existente.

También debería evaluar si está disponible algún adaptador de campo existente. Los adaptadores de campo son adaptadores de detección que están disponibles pero que no son de serie. Póngase en contacto con HP Software Support para recibir la lista actual de adaptadores de campo.

Escribir un nuevo adaptador

Hay que desarrollar un nuevo adaptador:

- Cuando es más rápido escribir un adaptador que introducir la información manualmente en CMDB (por lo general, desde 50 a 100 CI y relaciones aproximadamente) o que no sea una tarea que se realice una sola vez.
- Cuando la necesidad justifique el esfuerzo.
- Si los adaptadores de serie o de campo no están disponibles.
- Si los resultados se pueden reutilizar.
- Cuando el entorno de destino o sus datos están disponibles (no puede detectar lo que no puede ver).

Investigación del modelo

- Examine el modelo de clase de UCMDB (Administrador de tipos de CI) y haga corresponder las entidades y relaciones desde su **plano técnico** a los CIT existentes. Se recomienda encarecidamente adherirse al modelo actual para evitar posibles complicaciones durante la actualización de la versión. Si tiene que ampliar el modelo, debería crear nuevos CIT ya que una actualización puede sobrescribir los CIT de serie.
- Si algunas entidades, relaciones o atributos carecen del modelo actual, debería crearlos. Es preferible crear un paquete con estos CIT (que posteriormente también almacenará toda la detección, vistas y otros artefactos relacionados con este paquete), ya que tendrá que implementar estos CIT en cada instalación de HP Universal CMDB.

Investigación de tecnología

Cuando haya verificado que CMDB contiene los CI relevantes, la etapa siguiente consiste en decidir cómo recuperar estos datos de los sistemas relevantes.

La recuperación de los datos por lo general implica la utilización de un protocolo para acceder a una parte de gestión de la administración, los datos reales de la aplicación o archivos de configuración o bases de datos relacionadas con la aplicación. Cualquier origen de datos que pueda proporcionar información sobre un sistema es valioso. La investigación de tecnología requiere tanto un amplio conocimiento del sistema en cuestión y a veces creatividad.

Para las aplicaciones creadas internamente, puede resultar útil proporcionar un cuestionario al propietario de la aplicación. En este formulario el propietario debería enumerar todas las áreas de la aplicación que pueden proporcionar la información necesaria para el plano técnico y los valores de negocio. Esta información debería incluir (pero no estar limitada) las bases de datos de gestión, archivos de configuración, archivos de registros, interfaces de gestión, programas de administración, servicios web, mensajes o eventos enviados, entre otros.

En el caso de los productos diseñados, debería concentrarse en la documentación, foros o soporte del producto. Busque las guías de administración, guías de complementos e integraciones, guías de gestión, entre otras. Si todavía faltan datos en las interfaces de gestión, lea sobre los archivos de configuración de la aplicación, entradas del Registro, archivos de registro, registros de eventos NT y cualquier artefacto de la aplicación que controle su operación correcta.

Directivas para elegir formas de acceder a los datos

Relevancia: Seleccione los orígenes o una combinación de orígenes que proporcionan la mayoría de los datos. Si un único origen proporciona la mayor parte de información mientras que el resto de información está disperso o es difícil de acceder, pruebe a evaluar el valor de la información restante en comparación con el esfuerzo o riesgo de obtenerla. A veces se puede reducir el plano técnico si el valor o el coste no garantiza el esfuerzo invertido.

Reutilización: Si HP Universal CMDB ya incluye un soporte de protocolo de conexión específico, es un buen motivo para usarlo. Significa que DFM Framework puede proporcionar un cliente ya preparado y la configuración para la conexión. En caso contrario, es posible que tenga que invertir en desarrollo de infraestructura. Puede ver los protocolos de conexión de HP Universal CMDB actualmente admitidos: **Administración de Data Flow > Configuración de sonda de Data Flow > Panel Dominios y sondas**. Para obtener más información, consulte "Panel Dominios y sondas" en la página 1 en *HP Universal CMDB – Guía de Administración de Data Flow*.

Puede agregar nuevos protocolos añadiendo nuevos CI al modelo. Para obtener más información, póngase en contacto con HP Software Support.

Nota: Para acceder a los datos del Registro de Windows, puede usar WMI o NTCMD.

Seguridad: El acceso a la información suele requerir credenciales (nombre de usuario, contraseña), que se introducen en CMDB y se mantienen protegidas en todo el producto. Si es posible, y si la adición de seguridad no entra en conflicto con otros principios que haya establecido, elija la credencial o protocolo menos sensible que todavía responda a las necesidades de acceso. Por ejemplo, si la información está disponible tanto mediante JMX (interfaz de administración estándar, limitada) y Telnet, es preferible usar JMX ya que proporciona inherentemente acceso limitado y (por lo general) no permite el acceso a la plataforma subyacente.

Comodidad: Algunas interfaces de gestión pueden incluir funciones más avanzadas. Por ejemplo, puede ser más fácil enviar consultas (SQL, WMI) que desplazarse por árboles de información o crear expresiones regulares para análisis.

Audiencia del programador: Aquellos que en último término desarrollarán los adaptadores pueden estar inclinados hacia una determinada tecnología. También se puede tener en cuenta si dos tecnologías proporcionan casi la misma información al mismo coste en otros factores.

Resumen

El resultado de esta etapa es un documento que describe los métodos de acceso y la información relevante que se puede extraer para cada método. El documento también debería contener un mapa desde cada origen a cada dato del plano técnico relevante.

Cada método de acceso debería estar marcado de acuerdo con las instrucciones anteriores. Finalmente, debería tener ahora un plan en relación a qué orígenes se van a detectar y qué información se va a extraer de cada origen en el modelo del plano técnico (lo que ahora debería estar representado en el modelo de UCMDB correspondiente).

Desarrollo del contenido de integración

Antes de crear una nueva integración, debe comprender cuáles son los requisitos de dicha integración:

- ¿Debe la integración copiar datos en el CMDB? ¿Deben rastrearse los datos por el historial? ¿Es el origen no fiable?

  Se necesita **Rellenado**.

- ¿Debe la integración federar datos al instante para vistas y consultas TQL? ¿Es crítica la precisión de los cambios en los datos? ¿La cantidad de datos es demasiado grande para copiarse en CMDB, pero la cantidad de datos solicitada suele ser pequeña?

  Se necesita **Federación**.

- ¿Debe la integración insertar datos en orígenes de datos remotos?

  Se necesita **Inserción de datos**.

Nota: Los flujos Federación y Rellenado pueden configurarse en la misma integración para una máxima flexibilidad.

Para obtener información detallada sobre los distintos tipos de integraciones, consulte "Integration Studio" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Hay disponibles cuatro opciones diferentes para crear adaptadores de integración:

1. Adaptador Jython
   - El patrón de detección clásico
   - Escrito en Jython
   - Usado para el rellenado

   Para obtener más información, consulte "Desarrollo de los adaptadores Jython" en la página 39.
2. Adaptador Java
   - Adaptador que implementa una de las interfaces de adaptador en Federation Framework SDK.
   - Se puede usar para uno o varios procesos de Federación, Rellenado o Inserción de datos (dependiendo de la implementación requerida).
   - Escrito en su totalidad en Java, que permite escribir código que se conectará a cualquier origen o destino posible.
   - Adecuado para trabajos que se conectan a un único origen o destino de datos.

   Para obtener más información, consulte "Desarrollo de adaptadores Java" en la página 155.
3. Adaptador DB genérico
   - Un adaptador abstracto basado en el adaptador Java y utiliza Federation Framework SDK.
   - Permite la creación de adaptadores que se conectan a repositorios de datos externos.
   - Admite tanto la Federación como el Rellenado (con un complemento Java implementado para admitir los cambios).
   - Relativamente fácil de definir, ya que está basado sobre todo en XML y en archivos de configuración de propiedades.
   - La configuración principal está basada en un archivo **orm.xml** que establece relaciones entre las clases UCMDB y las columnas de la base de datos.
   - Adecuado para trabajos que se conectan a un único origen de datos.

   Para obtener más información, consulte "Desarrollo de adaptadores de bases de datos genéricas" en la página 83.
4. Adaptador de inserción genérico
   - Un adaptador abstracto basado en el adaptador Java (Federation Framework SDK) y el adaptador Jython.
   - Permite la creación de adaptadores que insertan datos a destinos remotos.
   - Relativamente fácil de definir, ya que solo se necesita definir la relación entre las clases UCMDB y XML, y una secuencia de comandos Jython que introduzca los datos en el destino.

- Adecuado para trabajos que se conectan a un único destino de datos.
- Utilizado para la inserción de datos.

  Para obtener más información, consulte "Desarrollo de los adaptadores de inserción" en la página 184.

En la tabla siguiente se muestran las características de cada adaptador:

Flujo/Adaptador	Adaptador Jython	Adaptador Java	Adaptador GDB	Adaptador de inserción
Rellenado	X	X	X	
Federación		X	X	
Inserción de datos		X		X

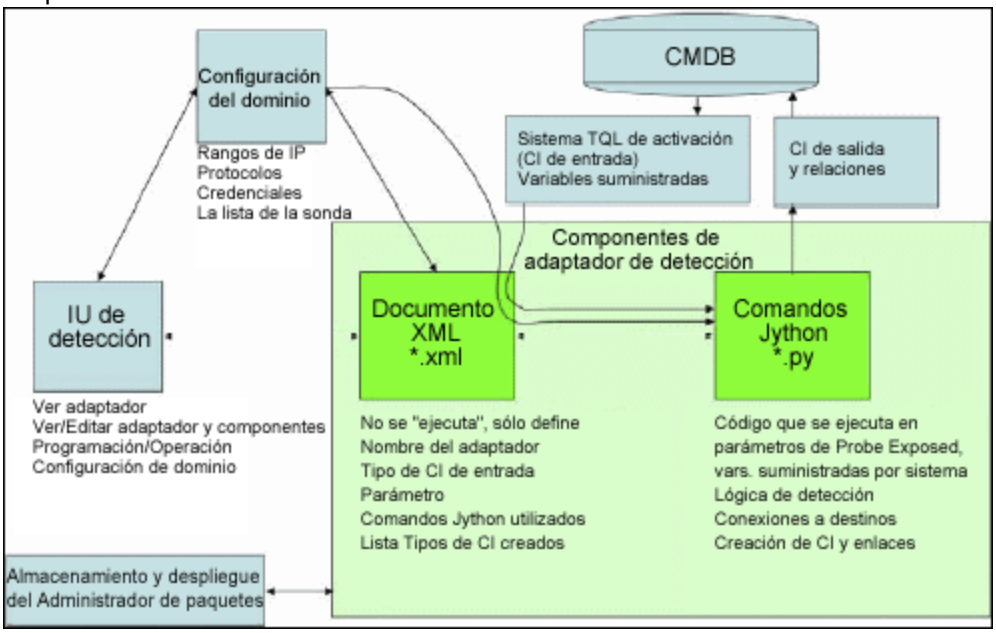
Desarrollo del contenido de detección

Esta sección incluye:

- "Adaptadores de detección y componentes relacionados " abajo
- "Separación de adaptadores" en la página siguiente

Adaptadores de detección y componentes relacionados

En el diagrama siguiente se muestran los componentes de un adaptador y los componentes que interactúan con la ejecución de la detección. Los componentes indicados en verde son los adaptadores reales y los componentes en azul son componentes que interactúan con los adaptadores.



Tenga en cuenta que la noción mínima de un adaptador es de dos archivos: un documento XML y una secuencia de comandos Jython. Discovery Framework, incluyendo los CI de entrada, credenciales y bibliotecas proporcionadas por el usuario, se expone al adaptador en tiempo de ejecución. Ambos componentes del adaptador de detección se administran mediante Data Flow Management. Se almacenan operativamente en el mismo CMDB; aunque sigue el paquete externo, no se hace referencia a él en el funcionamiento. El Administrador de paquetes permite la conservación de la nueva capacidad de contenido de detección e integración.

Un TQL proporciona los CI de entrada del adaptador y se exponen a la secuencia de comandos del adaptador en variables suministradas por el sistema. También se suministran parámetros del adaptador como datos de destino, por tanto puede configurar el funcionamiento del adaptador de acuerdo con la función específica del adaptador.

La aplicación DFM se utiliza para crear y probar nuevos adaptadores. Utilice las páginas Panel de control de detección, Administración de adaptador y Configuración de sonda de Data Flow durante la escritura del adaptador.

Los adaptadores se almacenan y se transportan como paquetes. La aplicación Administrador de paquetes y la consola JMX se utilizan para crear paquetes desde adaptadores recientemente creados y para implementar los adaptadores en nuevos sistemas.

Separación de adaptadores

Técnicamente, una detección completa podría definirse en un único adaptador. Sin embargo, el buen diseño exige que un sistema complejo esté separado en componentes más sencillos y más manejables.

A continuación se describen las directrices y prácticas recomendadas para dividir el proceso del adaptador:

- La detección debería realizarse en etapas. Cada etapa debería estar representada por un adaptador que debe representar un área o nivel del sistema. Los adaptadores deberían depender de la etapa o nivel anterior de detección para continuar la detección del sistema. Por ejemplo, el adaptador A está activado por un resultado TQL del servidor de aplicación y representa el nivel del servidor de aplicación. Como parte de esta representación, se representa un componente de conexión de JDBC. El adaptador B registra un componente de la conexión JDBC como un TQL de activación y utiliza el resultado del adaptador A para acceder al nivel de base de datos (por ejemplo, a través del atributo URL de JDBC) y representa el nivel de la base de datos.
- **El paradigma de conexión de dos fases**: La mayoría de los sistemas requieren credenciales para acceder a sus datos. Esto significa que una combinación de usuario/contraseña tiene que probarse con estos sistemas. El administrador de DFM proporciona la información de las credenciales de una forma segura al sistema y puede ofrecer varias credenciales de inicio de sesión priorizadas. Esto se conoce como **Diccionario de protocolos**. Si por cualquier motivo no se puede acceder al sistema, no hay motivo para seguir realizando la detección. Si la conexión se ha realizado correctamente, será preciso de establecer una forma de indicar qué conjunto de credenciales se ha usado correctamente para poder acceder a la detección en el futuro.

  Estas dos fases conducen a una separación de los dos adaptadores en los casos siguientes:

  - **Adaptador de conexión**: Es un adaptador que acepta un activador inicial y busca la existente de un agente remoto en ese activador. Lo hace probando todas las entradas del

Diccionario de protocolos que coincidan con el tipo de agente. Si se realiza correctamente, este adaptador proporciona como resultado un CI de agente remoto (SNMP, WMI, entre otros), que también apunta a la entrada correcta del Diccionario de protocolos para futuras conexiones. Este CI de agente forma entonces parte de un activador para el adaptador de contenido.

- **Adaptador de contenido**: La condición previa de este adaptador es la conexión correcta del adaptador anterior (condiciones previas especificadas por los TQL). Estos tipos de adaptadores ya no necesitan recorrer todo el Diccionario de protocolos ya que tienen una forma de obtener las credenciales correctas desde el CI de agente remoto y utilizarlas para iniciar sesión en el sistema detectado.

- En la división de detección también pueden influir las diferentes consideraciones de programación. Por ejemplo, un sistema solo puede consultarse durante las horas no laborables, por tanto aunque pudiera tener sentido unir el adaptador al mismo adaptador que detecta otro sistema, las diferentes programaciones implican que es preciso crear dos adaptadores.
- La detección de distintas interfaces de gestión o tecnologías para detectar el mismo sistema deberían situarse en adaptadores independientes. De esta forma, podrá activar el método de acceso apropiado para cada sistema u organización. Por ejemplo, algunas organizaciones tienen acceso WMI a los equipos, pero no tienen agentes SNMP instalados en ellos.

Implementación de un adaptador de detección

Una tarea DFM tiene el objetivo de acceder a sistemas remotos (o locales), modelando datos extraídos como CI y guardando los CI al CMDB. La tarea consta de los siguientes pasos:

1. **Crear un adaptador.**

   Puede configurar un archivo de adaptador que almacene el contexto, parámetros y tipos de resultado seleccionando las secuencias de comandos que van a formar parte del adaptador. Para obtener más información, consulte "Paso 1: Creación de un adaptador" en la página 29.

2. **Crear trabajo de detección**

   Puede configurar un trabajo con información de planificación y una consulta de activación. Para obtener más información, consulte "Paso 2: Asignación de un trabajo al adaptador" en la página 36.

3. **Editar código de detección.**

   Puede editar el código Jython o Java contenido en los archivos del adaptador y que se refiere a DFM Framework. Para obtener más información, consulte "Paso 3: Creación de código Jython" en la página 37.

Para escribir nuevos adaptadores, puede crear los componentes anteriores, cada uno de los cuales se vincula automáticamente al componente del paso anterior. Por ejemplo, una vez creado un trabajo y seleccionado el adaptador relevante, el archivo del adaptador se vincula al trabajo.

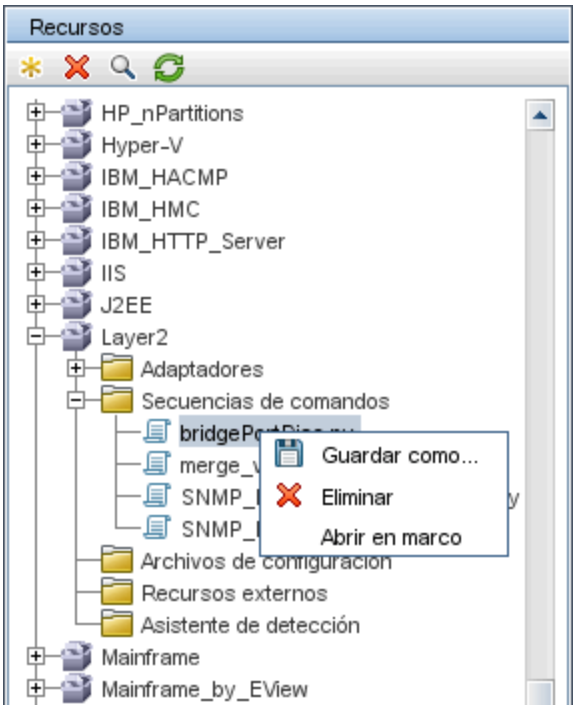
Código del adaptador

La implementación real de la conexión al sistema remoto, la consulta de los datos y su asignación como datos de CMDB lo realiza el código Jython. Por ejemplo, el código contiene la lógica para la conexión a una base de datos y su extracción de los datos. En este caso, el código espera recibir una URL de JDBC, un nombre de usuario, una contraseña, un puerto, etc. Estos parámetros son específicos para cada instancia de la base de datos que responde a la consulta de TQL. Puede

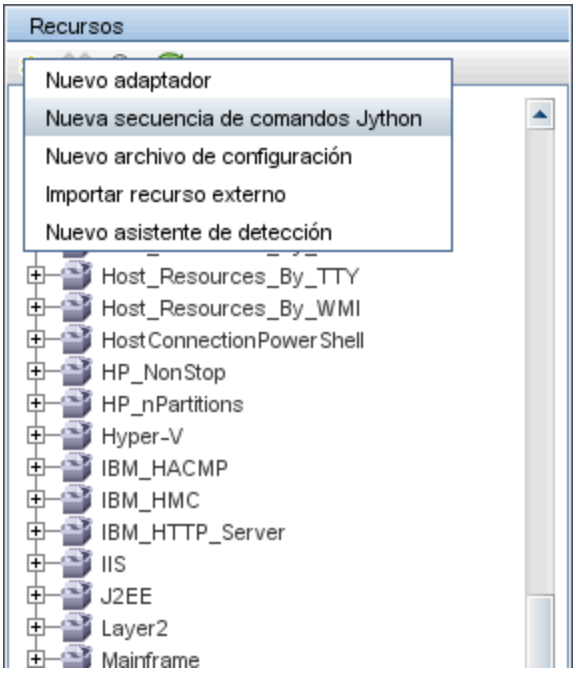
definir estas variables en el adaptador (en los datos del CI de activación) y , cuando se ejecuta el trabajo, estos detalles concretos se pasan al código para su ejecución.

El adaptador puede referirse a este código por un nombre de clase Java o un nombre de secuencias de comandos Jython. En esta sección, se tratará cómo escribir código DFM como secuencias de comandos Jython.

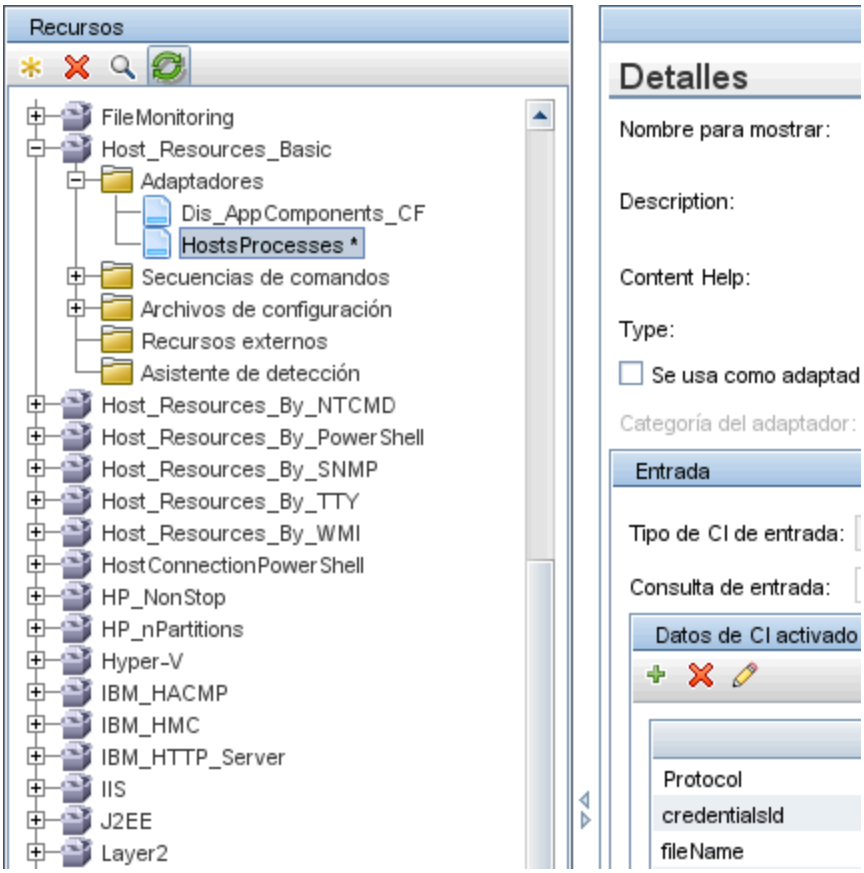
Un adaptador puede contener una lista de secuencias de comandos que se pueden utilizar al ejecutar la detección. Al crear un nuevo adaptador, por lo general creará una nueva secuencia de comandos y la asignará al adaptador. Una nueva secuencia de comandos incluye plantillas básicas, pero puede usar una de las otras secuencias de comandos como plantilla haciendo clic con el botón derecho sobre ella y seleccionando **Guardar como**:



Para obtener más información sobre cómo escribir nuevas secuencias de comandos Jython, consulte "Paso 3: Creación de código Jython" en la página 37. Puede agregar secuencias de comandos mediante el panel Recursos:



La lista de secuencias de comandos se ejecuta una tras otra, en el orden definido en el adaptador:



Nota: Se puede especificar una secuencia de comandos aunque otra secuencia de comandos

la esté utilizando únicamente como biblioteca. En este caso, debe definirse la secuencia de comandos de la biblioteca antes de que la utilice la secuencia de comandos. En este ejemplo, la secuencia de comandos `processdbutils.py` es una biblioteca usada por la última secuencia de comandos `host_processes.py`. Las bibliotecas se distinguen de las secuencias de comandos regulares ejecutables por la ausencia de la función `DiscoveryMain()`.

Paso 1: Creación de un adaptador

Se puede considerar al adaptador como la definición de una función. Esta función define una definición de entrada, ejecuta la lógica en la entrada, define la salida y proporciona un resultado.

Cada adaptador especifica una entrada y una salida: tanto la entrada como la salida son CI de activación específicamente definidos en el adaptador. El adaptador extrae los datos desde el CI de activación de entrada y pasa estos datos como parámetros al código. (Los datos procedentes de CI relacionados a veces se pasan también al código. Para obtener más información, consulte "Ventana CIs relacionados" en *HP Universal CMDB – Guía de Administración de Data Flow*). El código del adaptador es genérico, excepto esos parámetros específicos del CI de activación de entrada que se pasan al código.

Para obtener más información sobre los componentes de entrada, consulte "Conceptos de Universal Discovery" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Esta sección incluye los siguientes temas:

1. ## Definición de la entrada del adaptador(CIT de activación y consulta de entrada)

   Puede usar los componentes CIT de activación y Consulta de entrada para definir CI específicos como entrada del adaptador:

   - El CIT de activación define qué CIT se utiliza como entrada para el adaptador. Por ejemplo, para un adaptador que va a detectar direcciones IP, el CIT de entrada es Red.
   - La Consulta de entrada es una consulta regular y editable que define la consulta en CMDB. La Consulta de entrada define restricciones adicionales en el CIT (por ejemplo, si la tarea requiere un atributo `hostID` o `application_ip`) y puede definir más datos de CI, si lo necesita el adaptador.

     Si el adaptador requiere información adicional de los CI que están relacionados con el CI de activación, puede agregar nodos adicionales a la consulta de entrada. Para obtener más información, consulte "Ejemplo de definición de consulta de entrada" en la página siguiente y"Agregar nodos de consulta y relaciones a una consulta TQL" en la *HP Universal CMDB – Guía de modelado*.

- Los datos del CI de activación contienen toda la información requerida en el CI de activación, así como información de otros nodos de la consulta de entrada, si se definen. DFM usa variables para recuperar datos desde los CI. Cuando la tarea se descarga en la prueba, las variables de datos del CI de activación se reemplazan con valores reales que existen en los atributos para instancias de CI reales.

Ejemplo de definición de CIT de activación:

En este ejemplo, un CIT de activación define los CI de IP que se permiten en el adaptador.

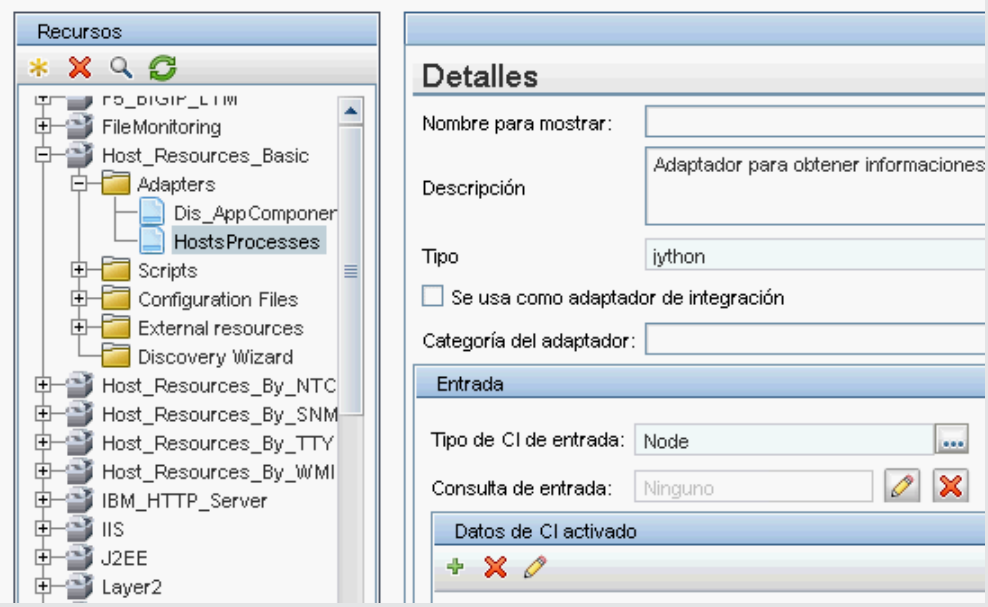
a. Acceda a **Administración de Data Flow > Administración de adaptador**. Seleccione el adaptador **HostProcesses** (**Paquetes > Host_Resources_Basic > Adaptadores > HostProcesses**).

b. Localice el cuadro Tipo de CI de entrada. Para obtener más información, consulte "Ficha Definición de adaptador" en *HP Universal CMDB – Guía de Administración de Data Flow*.

c. Haga clic en el botón para abrir el cuadro de diálogo Seleccionar la clase detectada. Para obtener más información, consulte "Cuadro de diálogo Seleccionar la clase detectada" en *HP Universal CMDB – Guía de Administración de Data Flow*

d. Seleccione el CIT.

En este ejemplo, se permite el CI de IP (Host) en el adaptador:



Ejemplo de definición de consulta de entrada

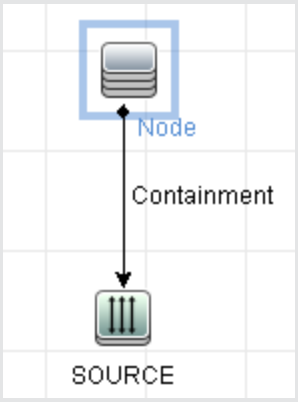
En este ejemplo, la consulta de entrada define que el CI `IpAddress` (configurado en el ejemplo anterior como un CIT de activación) debe conectarse al CI `Node`.

a. Acceda a **Administración de Data Flow > Administración de adaptador**. Localice el cuadro Consulta de entrada. Haga clic en el botón **Editar** para abrir el editor de

consultas de entrada. Para obtener más información, consulte "Ventana Editor de consultas de entrada" en *HP Universal CMDB – Guía de Administración de Data Flow*.

b. En el editor de consultas de entrada, llame **SOURCE** al nodo del CI de activación: haga clic con el botón derecho en el nodo y seleccione **Propiedades del nodo de consulta**. En el cuadro **Nombre del elemento**, cambie el nombre a **SOURCE**.

c. Agregue un CI `Node` y una relación `Containment` en el CI `IpAddress`. Para obtener más información sobre cómo trabajar con el editor de consultas de entrada, consulte "Ventana Editor de consultas de entrada" en *HP Universal CMDB – Guía de Administración de Data Flow*.



El CI **IpAddress** está conectado a un CI **Node**. El TQL de entrada consta de dos nodos, **Node** e **IpAddress**, con un vínculo entre ellos. El CI **IpAddress** se llama **SOURCE**.

Ejemplo de adición de variables a la consulta de entrada:

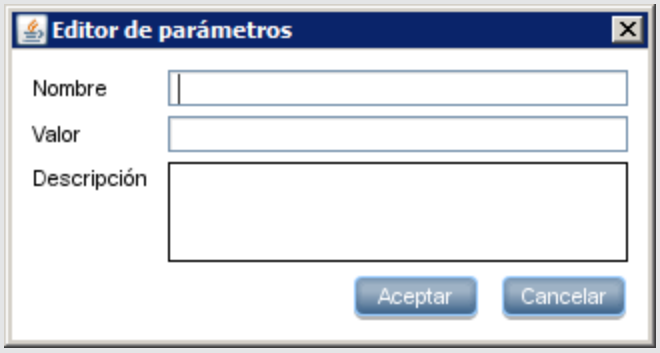
En este ejemplo, agrega las variables DIRECTORY y CONFIGURATION_FILE a la consulta de entrada creada en el ejemplo anterior. Estas variables ayudan a definir lo que se debe detectar, en este caso, para encontrar los archivos de configuración que residen en los host que están vinculados a las IP que hay que detectar.

a. Muestre la consulta de entrada creada en el ejemplo anterior.

Acceda a **Administración de Data Flow > Administración de adaptador**. Localice el panel Datos de CIs activados. Para obtener más información, consulte "Ficha Definición de adaptador" en *HP Universal CMDB – Guía de Administración de Data Flow*.

b. Agregue variables a la consulta de entrada. Para obtener más información, acceda a **Administración de Data Flow > Administración de adaptador**. Localice el panel Datos de CIs activados. Para obtener más información, consulte el campo Variables en "Ficha Definición de adaptador" en la *HP Universal CMDB – Guía de*

Administración de Data Flow.



Ejemplo de sustitución de variables con datos reales:

En este ejemplo, las variables sustituyen los datos del CI **IpAddress** por valores reales que existen en instancias del CI **IpAddress** reales del sistema.

Los datos de CI activados para el CI **IpAddress** incluyen una variable `fileName`. Esta variable permite la sustitución del nodo **CONFIGURATION_FILE** en el TQL de entrada con los valores reales del archivo de configuración ubicado en un host:

Datos de CI activado

Nombre	Valor
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}

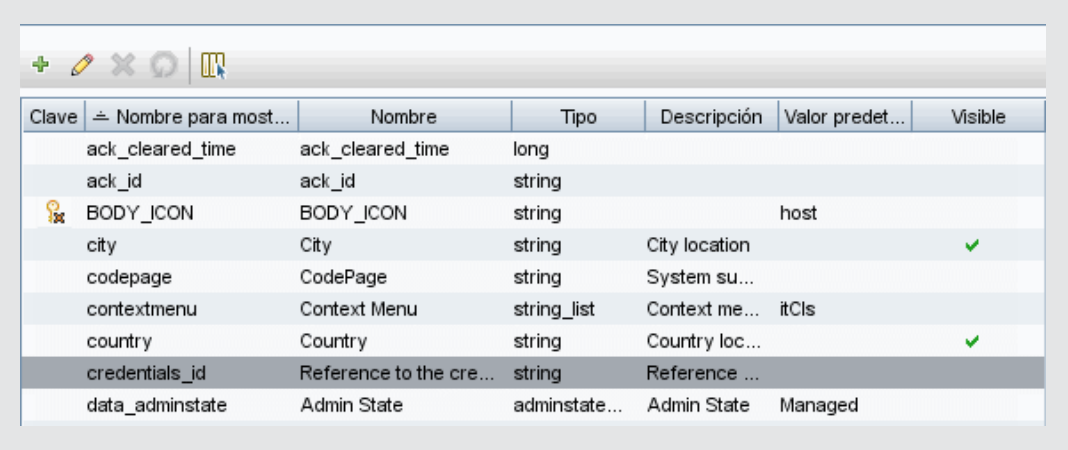
Los datos del CI de activación se cargan en la sonda con todas las variables sustituidas por valores reales. La secuencia de comandos del adaptador incluye un comando para utilizar DFM Framework para recuperar los valores reales de las variables definidas:

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

Las variables `fileName` y `path` usan los atributos `data_name` y `document_path` desde el nodo **CONFIGURATION_DOCUMENT** (definido en el Editor de consulta de entrada: consulte el ejemplo anterior).

Haga clic aquí para ver un ejemplo.

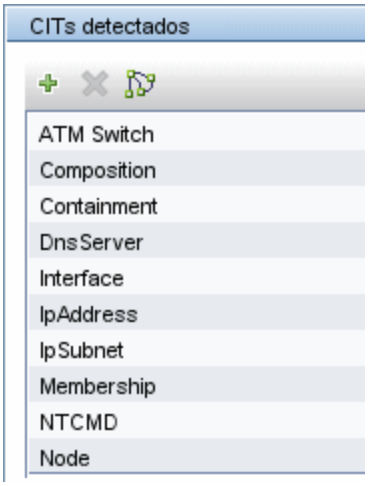
Las variables `Protocol`, `credentialsId` e `ip_address` usan los atributos `root_class`, `credentials_id` y `application_ip`:



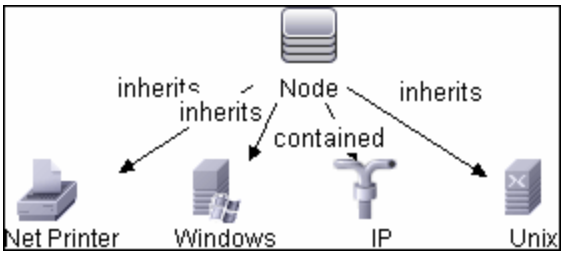
Clave	Nombre para most...	Nombre	Tipo	Descripción	Valor predet...	Visible
	ack_cleared_time	ack_cleared_time	long			
	ack_id	ack_id	string			
	BODY_ICON	BODY_ICON	string		host	
	city	City	string	City location		✔
	codepage	CodePage	string	System su...		
	contextmenu	Context Menu	string_list	Context me...	itCIs	
	country	Country	string	Country loc...		✔
	credentials_id	Reference to the cre...	string	Reference ...		
	data_adminstate	Admin State	adminstate...	Admin State	Managed	

2. ## Definición de la salida del adaptador

La salida del adaptador es una lista de CI detectados (**Administración de Data Flow > Administración de adaptador > pestaña Definición de adaptador > CIT detectados**) y los vínculos que hay entre ellos:



También puede ver los CIT como un mapa de topología,e s decir, los componentes y la forma en que se vinculan (haga clic en el botón **Ver CITs detectados en mapa**):



El código DFM devuelve los CI detectados (es decir, la secuencia de comandos Jython) en formato de `ObjectStateHolderVector` de UCMDB. Para obtener más información, consulte "Generación de resultados por la secuencia de comandos Jython" en la página 44.

Ejemplo de salida del adaptador:

En este ejemplo, define los CIT que van a formar parte de la salida del CI IP.

a. Acceda a **Administración de Data Flow > Administración de adaptador**.

b. En el panel Recursos, seleccione **Red > Adaptadores > NSLOOKUP_on_Probe**.

c. En la pestaña Definición de adaptador, localice el panel CITs detectados.

d. Se enumeran los CIT que van a formar parte de la salida del adaptador. Agregue CIT a la lista, o suprímalos de allí. Para obtener más información, consulte "Ficha Definición de adaptador" en *HP Universal CMDB – Guía de Administración de Data Flow*.

3. ## Reemplazo de los parámetros del adaptador

Para configurar un adaptador para más de un trabajo, puede reemplazar los parámetros del adaptador. Por ejemplo, el adaptador `SQL_NET_Dis_Connection` se utiliza en los trabajos `MSSQL Connection by SQL` y `Oracle Connection by SQL`.

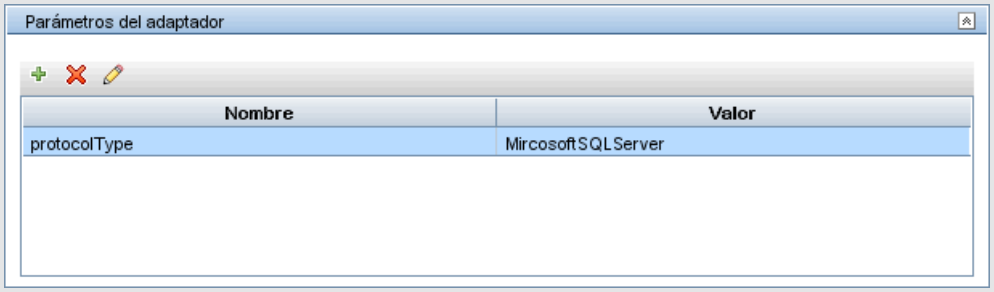
Ejemplo de reemplazo de un parámetro del adaptador:

En este ejemplo se muestra el reemplazo de un parámetro del adaptador de tal forma que se puede usar un adaptador para detectar bases de datos Microsoft SQL Server y Oracle.

a. Acceda a **Administración de Data Flow > Administración de adaptador**.

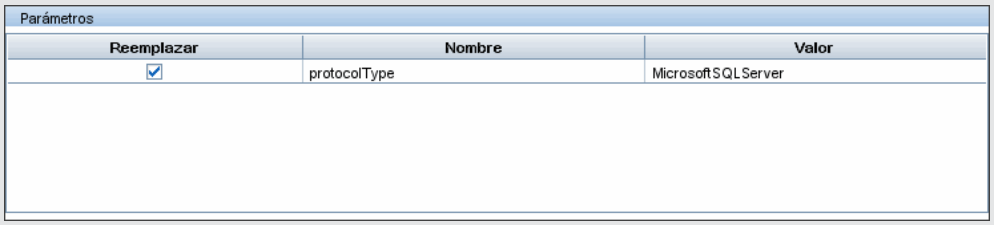
b. En el panel Recursos, seleccione **Database_Basic > Adaptadores > SQL_NET_Dis_Connection**.

c. En la pestaña Definición de adaptador, localice el panel **Parámetros del adaptador**. El parámetro `protocolType` tiene un valor de **all**:



d. Haga clic con el botón derecho en el adaptador **SQL_NET_Dis_Connection_MsSql** y seleccione **Ir a trabajo de detección > MSSQL Connection by SQL**.

e. Muestre la pestaña Propiedades. Localice el panel Parámetros:



El valor `all` es reemplazado por el valor `MicrosoftSQLServer`.

Nota: El trabajo **Oracle Connection by SQL** incluye el mismo parámetro pero el valor se reemplaza por un valor Oracle.

Para obtener más información sobre cómo agregar, eliminar o editar parámetros, consulte "Ficha Definición de adaptador" en la *HP Universal CMDB – Guía de Administración de Data Flow*.

DFM comienza a buscar instancias de Microsoft SQL Server de acuerdo con este parámetro.

4. ## Reemplazar selección de sonda: opcional

En el servidor UCMDB hay un mecanismo de distribución que toma los CI de activación recibidos por UCMDB y elige automáticamente la sonda que deberá ejecutar el trabajo para cada CI de activación según una de las opciones siguientes.

- **Para el tipo de CI de dirección IP:** tome la sonda definida para esta dirección IP.
- **Para el tipo de CI de software en ejecución:** utilice los atributos **application_ip** y **application_ip_domain** y elija la sonda definida para la dirección IP en el dominio pertinente.
- **Para otros tipos de CI:** tome la dirección IP del nodo según el nodo relacionado del CI (si existe).

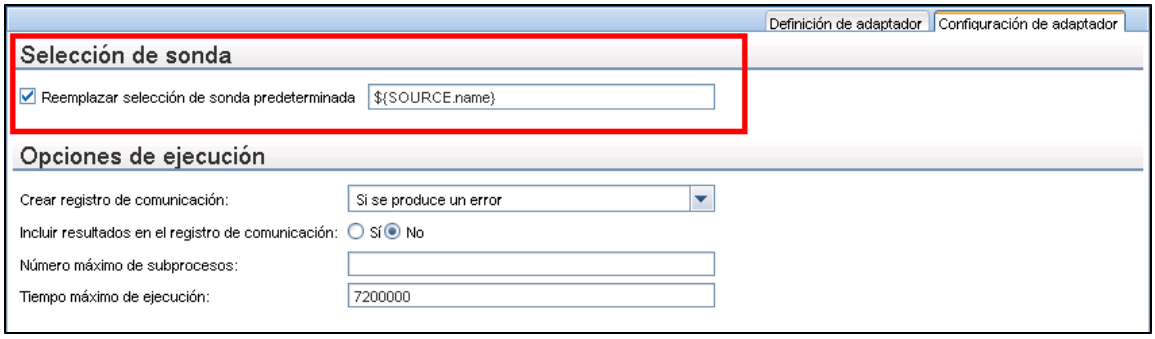
La selección automática de sonda se realiza según el nodo relacionado del CI. Tras obtener el nodo relacionado del CI, el mecanismo de distribución elige una de las direcciones IP del nodo y elige la sonda según las definiciones del ámbito de red de la sonda.

En los casos siguientes, tiene que especificar la sonda manualmente y no utilizar el mecanismo de distribución automática:

- Ya sabe qué sonda debe ejecutarse para el adaptador y no es necesario que el mecanismo de distribución automática seleccione la sonda (por ejemplo, si el CI de activación es la puerta de enlace de la sonda).
- La selección automática de sonda podría dar error. Esto puede ocurrir en las siguientes situaciones:
  - Un CI de activación no tiene un nodo relacionado (como el CIT de red).
  - Un nodo de CI de activación tiene varias direcciones IP, cada una perteneciente a una sonda distinta.

Para solucionar estos problemas puede especificar la sonda que se debe usar con el adaptador de la manera siguiente:

a. En la sección Selección de sonda, seleccione **Reemplazar selección de sonda predeterminada** como se muestra a continuación.



b. En el cuadro Sonda, especifique la sonda que desea usar para la tarea.

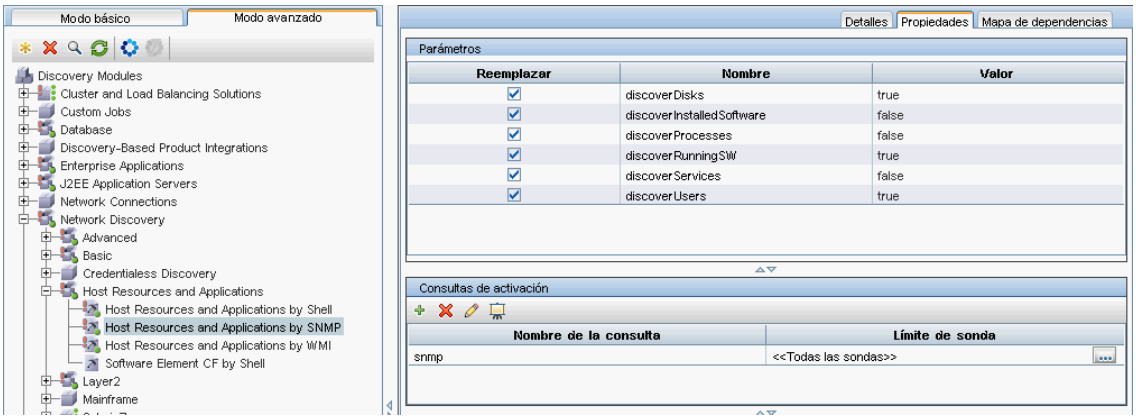
5. **Configure una ruta de clase para un proceso remoto: opcional**

Para obtener más información, consulte "Configurar ejecución de proceso remoto" en la página siguiente.

Paso 2: Asignación de un trabajo al adaptador

Cada adaptador tiene uno o más trabajos asociados que definen la directiva de ejecución. Los trabajos permiten la planificación del mismo adaptador de manera diferente en un conjunto distinto de CI activados y también permiten que se proporcionen distintos parámetros para cada conjunto.

Los trabajos aparecen en el árbol Módulos de detección y esta es la entidad que activa el usuario, como se muestra en la imagen siguiente.



Elegir una consulta de activación

Cada trabajo debe asociarse a los TQL de activación. Estos TQL de activación publican los resultados que se usan como CI de activación de entrada para el adaptador de este trabajo.

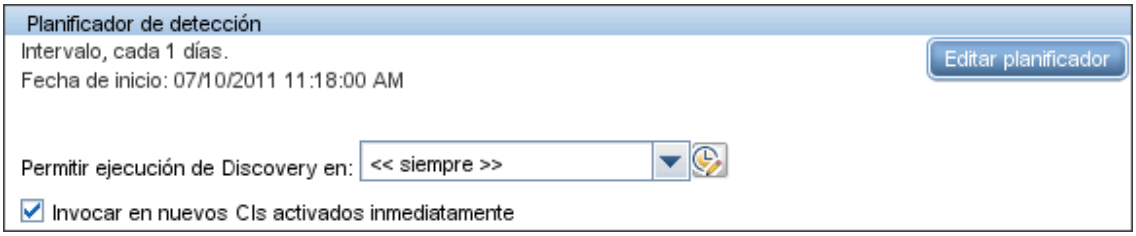
Un TQL de activación puede agregar restricciones a una consulta de entrada. Por ejemplo, si los resultados de una consulta de entrada son las IP conectadas a SNMP, los resultados de una consulta de activación pueden ser IP conectadas a SNMP dentro del rango 195.0.0.0-195.0.0.10.

Nota: Una consulta de activación debe hacer referencia a los mismos objetos a los que hace

referencia la consulta de entrada. Por ejemplo, si una consulta de entrada consulta IP que ejecutan SNMP, no puede definir una consulta de activación (para el mismo trabajo) para que consulte IP conectadas a un host, porque algunas de las IP pueden no estar conectadas a un objeto SNMP, tal como requiere la consulta de entrada.

Configurar información de planificación

La información de planificación de la sonda especifica cuándo ejecutar el código en CI de activación. Si se activa la casilla **Invocar en nuevos CIs activados inmediatamente**, el código también se ejecuta una vez en cada CI de activación cuando alcanza a la sonda, con independencia de la configuración de planificación futura.



Para cada ocurrencia planificada de cada trabajo, la sonda ejecuta el código en todos los CI de activación acumulados para dicho trabajo. Para obtener más información, consulte "Cuadro de diálogo Planificador de detección" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Reemplazar los parámetros del adaptador

Al configurar un trabajo, puede reemplazar los parámetros del adaptador. Para obtener más información, consulte "Reemplazo de los parámetros del adaptador" en la página 34.

Paso 3: Creación de código Jython

HP Universal CMDB utiliza secuencias de comandos Jython para la escritura de adaptadores. Por ejemplo, la secuencia de comandos `SNMP_Connection.py` la utiliza el adaptador `SNMP_NET_Dis_Connection` para probar y conectarse a equipos usando SNMP. Jython es un lenguaje basado en Python y con tecnología Java.

Para obtener más información sobre cómo trabajar en Jython, puede consultar estos sitios Web:

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

Para obtener más información, consulte "Creación de código Jython" en la página 39.

Configurar ejecución de proceso remoto

Puede ejecutar la detección para un trabajo de detección en un proceso independiente del proceso de la sonda de Data Flow.

Por ejemplo, puede ejecutar el trabajo en un proceso remoto independiente si el trabajo utiliza bibliotecas jar que son de una versión diferente de las bibliotecas de la sonda o que son incompatibles con las bibliotecas de la sonda.

También puede ejecutar el trabajo en un proceso remoto independiente si el trabajo consume potencialmente mucha memoria (aporta muchos datos) y preferirá aislar la sonda de problemas potenciales de falta de memoria (OutOMemory).

Para configurar un trabajo para ejecutarlo como un proceso remoto, defina los parámetros siguientes en el archivo de configuración de su adaptador:

Parámetro	Descripción
remoteJVMArgs	Parámetros de JVM para el proceso Java remoto.
runInSeparateProcess	Cuando se establece como **true**, el trabajo de detección se ejecuta en un proceso independiente.

remoteJVMClasspath	(Opcional) Permite la personalización de la ruta de clase del proceso remoto, reemplazando la ruta de clase de sonda predeterminada. Esto es útil si puede haber una incompatibilidad de versiones entre los archivos jar de la sonda y los archivos jar personalizados requeridos para la detección definida por el cliente.
	
	Si no se ha definido el parámetro **remoteJVMClasspath**, o se ha dejado vacío, se utiliza la ruta de clase de sonda predeterminada.
	
	Si desarrolla un trabajo de detección nuevo y desea asegurarse de que la versión de la biblioteca jar de la sonda no esté en conflicto con las bibliotecas jar del trabajo, debe utilizar al menos la ruta de clase mínima requerida para ejecutar la detección básica. La ruta de clase mínima está definida en **DataFlowProbe.properties**, en el parámetro **basic_discovery_minimal_classpath**.
	
	Ejemplos de personalización de **remoteJVMClasspath**:
	
	• Para agregar o agregar como prefijo archivos jar personalizados a la ruta de clase de sonda predeterminada, personalice el parámetro **remoteJVMClasspath** de la manera siguiente:
	
	`custom1.jar;%classpath%;custom2.jar -`
	
	En este caso, se coloca **custom1.jar** antes de la ruta de clase de sonda predeterminada y **custom2.jar** se agrega a la ruta de clase de la sonda.
	
	• Para utilizar la ruta de clase mínima, personalice el parámetro **remoteJVMClasspath** de la manera siguiente:
	
	`custom1.jar;%minimal_classpath%;custom2.jar`

Capítulo 2

Desarrollo de los adaptadores Jython

Este capítulo incluye:

Referencia de API de HP Data Flow Management

Para obtener toda la información sobre las API disponibles, consulte *Referencia de API de HP Universal CMDB Data Flow Management*. Estos archivos están ubicados en la carpeta siguiente:

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

Creación de código Jython

HP Universal CMDB utiliza secuencias de comandos Jython para la escritura de adaptadores. Por ejemplo, la secuencia de comandos `SNMP_Connection.py` la utiliza el adaptador `SNMP_NET_Dis_Connection` para probar y conectarse a equipos usando SNMP. Jython es un lenguaje basado en Python y con tecnología Java.

Para obtener más información sobre cómo trabajar en Jython, puede consultar estos sitios Web:

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

La sección siguiente describe la escritura real del código Jython dentro de DFM Framework. Esta sección aborda específicamente estos puntos de contactos entre la secuencia de comandos Jython y Framework al que llama, y también describe las bibliotecas Jython y utilidades que deberían usarse siempre que sea posible.

Nota:

- Las secuencias de comandos escritas para DFM deben ser compatibles con Jython versión 2.1.

- Para obtener toda la información sobre las API disponibles, consulte *Referencia de API de HP Universal CMDB Data Flow Management*.

Esta sección incluye los siguientes temas:

Utilización de archivos Java JAR externos dentro de Jython

Al desarrollar nuevas secuencias de comandos Jython, a veces se necesitan bibliotecas Java externas (archivos JAR) como archivos ejecutables de terceros como archivos de utilidad Java, archivos de conexión como archivos JAR de JDBC Driver o archivos ejecutables (por ejemplo, **nmap.exe** se usa para la detección sin credenciales).

Estos recursos deberían agruparse en el paquete debajo de la carpeta **Recursos externos**. Todos los recursos que se colocan en esta carpeta se envían automáticamente a la sonda que se conecta con el servidor de HP Universal CMDB.

Además, cuando se inicia la detección, todos los recursos de archivos JAR se cargan en la ruta de clase de Jython, haciendo que todas las clases que están dentro de ella se puedan importar y usar.

Ejecución del código

Una vez activado un trabajo, se descarga en la sonda una tarea con toda la información necesaria.

La sonda inicia la ejecución del código de DFM usando la información especificada en la tarea.

El flujo del código Jython inicia la ejecución desde una entrada principal de la secuencia de comandos, ejecuta el código para detectar los CI y proporciona los resultados de un vector de CI detectados.

Modificación de secuencias de comandos de serie

Al realizar modificaciones de secuencias de comandos de serie, realice solo cambios mínimos en ellas y coloque todos los métodos necesarios en una secuencia de comandos externa. Puede realizar el seguimiento de los cambios con más eficacia y, al pasar a una versión de HP Universal CMDB más reciente, el código no se sobrescribe.

Por ejemplo, la siguiente línea de código de una secuencia de comandos de serie llama a un método que calcula un nombre de servidor web de forma específica de aplicación:

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

En una secuencia de comandos externa está contenida la lógica más compleja que decide cómo calcular este nombre:

```
# 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
```

Guarde la secuencia de comandos externa en la carpeta Recursos externos. Para obtener más información, consulte "Panel Recursos" en *HP Universal CMDB – Guía de Administración de Data Flow*. Si agrega esta secuencia de comandos a un paquete, puede usarla también para otros trabajos. Para obtener más información sobre cómo trabajar con el Administrador de paquetes, consulte "Administrador de paquetes" en *HP Universal CMDB – Guía de administración*.

Durante la actualización, el cambio que se realice en una única línea de código se sobrescribe por la nueva versión de la secuencia de comandos de serie, por tanto habrá que sustituir la línea. Sin embargo, la secuencia de comandos externa no se sobrescribe.

Estructura del archivo Jython

El archivo Jython está compuesto por tres partes en un orden específico:

1. Importaciones
2. Función principal: DiscoveryMain
3. Definiciones de funciones (opcional)

A continuación se muestra un ejemplo de una secuencia de comandos Jython:

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

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

Importaciones

Las clases Jython se extienden por espacios de nombres jerárquicos. En la versión 7.0 o posterior, a diferencia de las versiones anteriores, no hay importaciones implícitas y por tanto cada clase que se utiliza debe importarse explícitamente. (Este cambio se realizó por motivos de rendimiento y para permitir una comprensión más sencilla de la secuencia de comandos de Jython al no ocultar ningún detalle necesario).

- Para importar una secuencia de comandos Jython:

  ```
  import logger
  ```

- Para importar una clase Java:

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

Función principal: DiscoveryMain

Cada archivo de secuencia de comandos ejecutable de Jython contiene una función principal: DiscoveryMain.

La función `DiscoveryMain` es la entrada principal a la secuencia de comandos; es la primera función que se ejecuta. La función principal puede llamar a otras funciones que se definen en las secuencias de comandos:

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

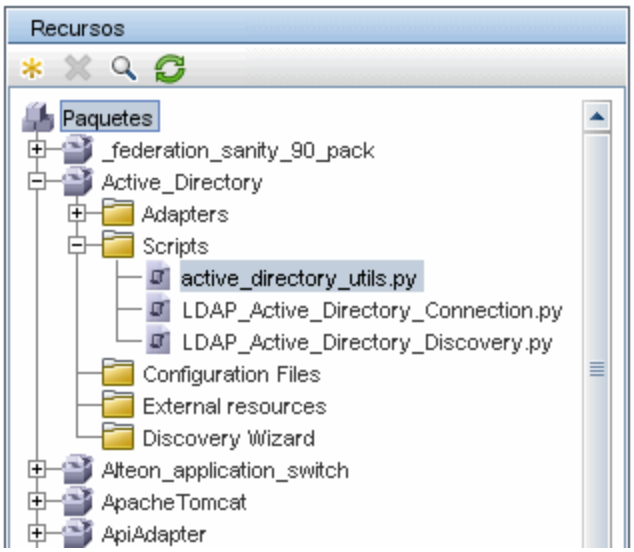
Se debe especificar el argumento Framework en la definición de la función principal. Este argumento lo utiliza la función principal para recuperar información que se requiere para ejecutar las secuencias de comandos (como información sobre parámetros y CI de activación) y también se pueden usar para informar sobre errores que tienen lugar durante la ejecución de la secuencia de comandos.

Puede crear una secuencia de comandos Jython sin ningún método principal. Dichas secuencias de comandos se usan como secuencias de comandos de biblioteca que se llaman desde otras secuencias de comandos.

Definición de funciones

Cada secuencia de comandos puede contener funciones adicionales que se llaman desde el código principal. Cada una de estas funciones puede llamar a otra función, que existe en la secuencia de

comandos actual o en otra (utilice la instrucción `import`). Tenga en cuenta que para usar otra secuencia de comandos, debe agregarla a la sección Scripts del paquete:



Ejemplo de una función que llama a otra función:

En el ejemplo siguiente, el código principal llama al método `doQueryOSUsers(..)` que llama a un método interno `doOSUserOSH(..)`:

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

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

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

Si esta secuencia de comandos es una biblioteca global relevante a varios adaptadores, puede agregarla a la lista de secuencia de comandos en el archivo de configuración `jythonGlobalLibs.xml`, en lugar de agregarla a cada adaptador (**Administración de adaptador > Panel Recursos > AutoDiscoveryContent > Archivos de configuración**).

Generación de resultados por la secuencia de comandos Jython

Cada secuencia de comandos Jython se ejecuta en un CI de activación específico y finaliza con resultados que devuelve el valor devuelto de la función `DiscoveryMain`.

El resultado de la secuencia de comandos es en realidad un grupo de CI y vínculos que se van a insertar o actualizar en CMDB. La secuencia de comandos devuelve este grupo de CI y vínculos en formato de `ObjectStateHolderVector`.

La clase `ObjectStateHolder` es una forma de representar un objeto o vínculo definido en CMDB. El objeto `ObjectStateHolder` contiene el nombre del CIT y una lista de atributos y sus valores. El objeto `ObjectStateHolderVector` es un vector de instancias de `ObjectStateHolder`.

Sintaxis de ObjectStateHolder

En esta sección se explica cómo generar los resultados de DFM en un modelo de UCMDB.

Ejemplo de configuración de atributos en los CI:

La clase ObjectStateHolder describe el gráfico de resultado de DFM. Cada CI y vínculo (relación) se coloca dentro de una instancia de la clase `ObjectStateHolder` en el siguiente ejemplo de código Jython:
siebel application server 1 appServerOSH = ObjectStateHolder('siebelappserver') 2 appServerOSH.setStringAttribute('data_name', sblsvrName) 3 appServerOSH.setStringAttribute ('application_ip', ip) 4 appServerOSH.setContainer (appServerHostOSH)

- En la línea 1 se crea un CI de tipo **siebelappserver**.
- En la línea 2 se crea un atributo denominado **data_name** con un valor de **sblsvrName**, que es un conjunto de variables Jython con el valor detectado para el nombre de servidor.
- En la línea 3 se establece un atributo que no es clave actualizado en CMDB.
- En la línea 4 se genera la contención (el resultado es un gráfico). Especifica que este servidor de aplicaciones se encuentra dentro de un host (otra clase `ObjectStateHolder` del ámbito).

Nota: Cada CI que notifica la secuencia de comandos Jython debe incluir valores para todos los atributos clave del tipo CI de los CI.

Ejemplo de relaciones (vínculos):

El siguiente ejemplo de vínculo explica cómo se representa el gráfico:
1 linkOSH = ObjectStateHolder('route') 2 linkOSH.setAttribute('link_end1', gatewayOSH) 3 linkOSH.setAttribute('link_end2', appServerOSH)

- En la línea 1 se crea el vínculo (que también es de la clase `ObjectStateHolder`. La única diferencia es que `route` es un tipo CI de vínculo).
- En las líneas 2 y 3 se especifican los nodos al final de cada vínculo. Esto se realiza usando los atributos **end1** y **end2** del vínculo que deben especificarse (porque son los atributos clave mínimos de cada vínculo). Los valores de los atributos son instancias de `ObjectStateHolder`. Para obtener más información sobre End 1 y End 2, consulte "Vínculo" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Precaución: un vínculo es direccional. Debería comprobar que los nodos End 1 y End 2 se corresponden con CIT válidos en cada extremo. Si los nodos no son válidos, el objeto de resultado no podrá realizar la validación y no se notificará correctamente. Para obtener más información, consulte "Relaciones de tipo de CI" en *HP Universal CMDB – Guía de modelado*.

Ejemplo de vector (CI de recopilación):

Después de crear objetos con atributos y vínculos con objetos en sus extremos, ahora hay que agruparlos. Para ellos, se agregan a una instancia `ObjectStateHolderVector`, de la forma siguiente:

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

Para obtener más información sobre cómo notificar este resultado compuesto a Framework para que se pueda enviar al servidor de CMDB, consulte el método sendObjects.

Cuando el gráfico de resultados se haya unido en una instancia `ObjectStateHolderVector`, debe devolverse a DFM Framework para que se inserte en CMDB. Esto se lleva a cabo devolviendo la instancia `ObjectStateHolderVector` como resultado de la función `DiscoveryMain()`.

Nota: Para obtener más información sobre cómo crear **OSH** para CIT comunes, consulte "modeling.py" en la página 76 en "Bibliotecas y utilidades Jython" en la página 74.

La instancia Framework

La instancia de Framework es el único argumento que se proporciona en la función principal de la secuencia de comandos Jython. Es una interfaz que se puede usar para recuperar la información que se requiere para ejecutar las secuencias de comandos (como información sobre CI de activación y parámetros del adaptador) y también se puede usar para notificar sobre errores que tienen lugar durante la ejecución de la secuencia de comandos. Para obtener más información, consulte "Referencia de API de HP Data Flow Management" en la página 39.

El uso correcto de la instancia de Framework consiste en pasarla como argumento a todos los métodos que la utilicen.

Ejemplo:

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

En esta sección se describen los usos más importantes de Framework:

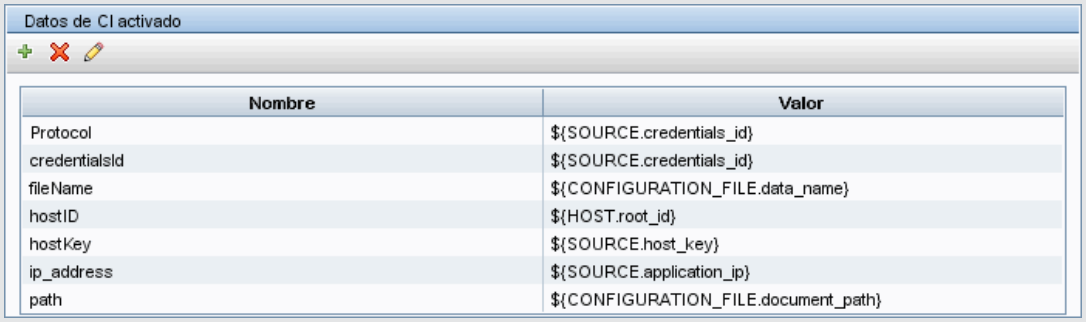
- "Framework.getTriggerCIData(String attributeName)" abajo
- "Framework.createClient(credentialsId, props)" abajo
- "Framework.getParameter (String parameterName)" en la página 48
- "Framework.reportError(mensaje de cadena) y Framework.reportWarning(mensaje de cadena)" en la página 48

Framework.getTriggerCIData(String attributeName)

Esta API proporciona el paso intermedio entre los datos del CI de activación definidos en el adaptador y la secuencia de comandos.

Ejemplo de recuperación de información de credenciales:

Solicita la siguiente información sobre datos del CI de activación:



Nombre	Valor
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}

Para recuperar la información de credenciales de la tarea, utilice esta API:

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

Framework.createClient(credentialsId, props)

Realice una conexión a un equipo remoto creando un objeto cliente y ejecutando comandos en dicho cliente. Para crear un cliente, recupere la clase `ClientFactory`. El método getClientFactory() recibe el tipo de protocolo de cliente solicitado. Las constantes del protocolo se definen en la clase ClientsConsts. Para obtener más información sobre credenciales y protocolos admitidos, consulte el manual *HP Universal CMDB Discovery and Integration Content Guide*.

Ejemplo de creación de una instancia Client para el Id. de credenciales:

Para crear una instancia `Client` para el Id. de credenciales:

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

Ahora puede usar la instancia `Client` para conectarse al equipo o aplicación pertinente.

Ejemplo de creación de un cliente WMI y de la ejecución de una consulta WMI:

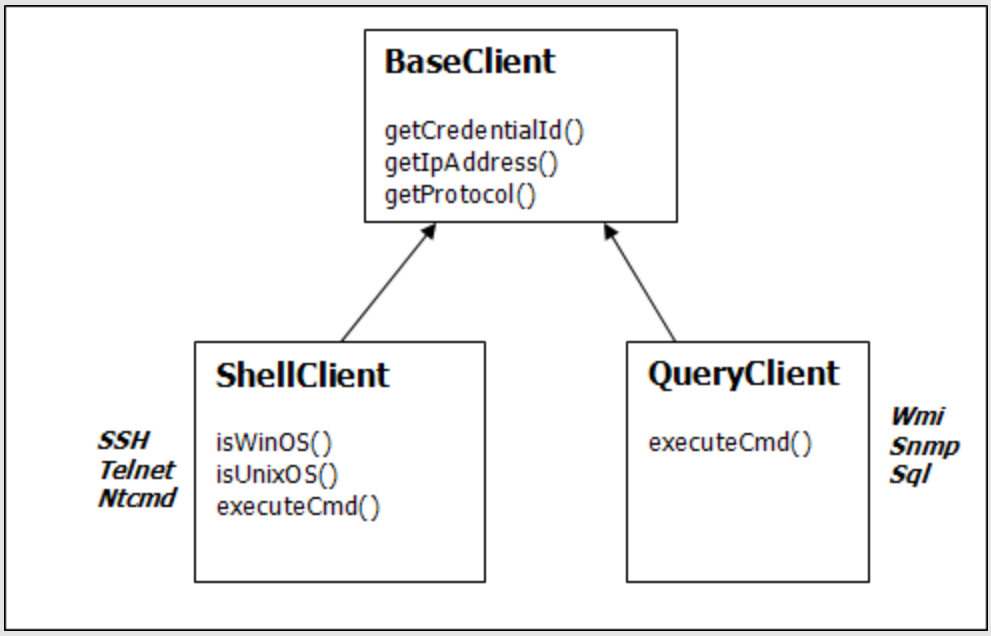
Para crear un cliente WMI y ejecutar una consulta WMI con el cliente:

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

FROM    Win32_LogicalMemoryConfiguration")
```

Nota: Para que la API `createClient()` funcione, agregue el siguiente parámetro a los parámetros de datos del CI de activación: **credentialsId = ${SOURCE.credentials_id}** en el panel de datos de CI activados. O bien, puede agregar manualmente el ID de credenciales al llamar a la función:
wmiClient = clientFactory().createClient(credentials_id).

En el diagrama siguiente se muestra la jerarquía de los clientes, con sus API habitualmente admitidos:

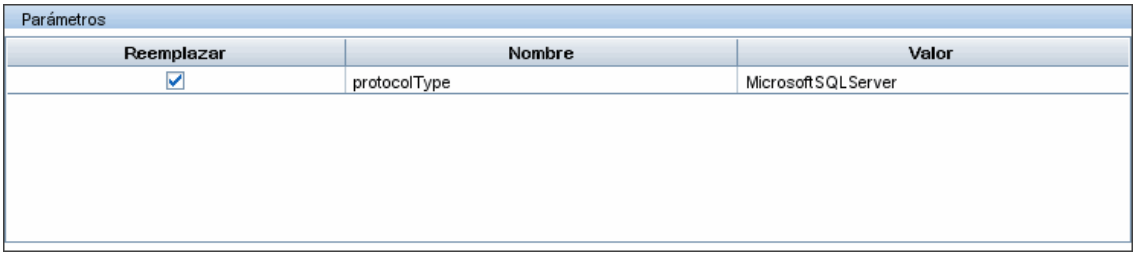


Para obtener más información sobre los clientes y sus API compatibles, consulte BaseClient, ShellClient, y QueryClient en *HP Discovery and Dependency Mapping Schema Reference*. Estos archivos están ubicados en la carpeta siguiente:

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

Framework.getParameter (String parameterName)

Además de recuperar información sobre el CI de activación, con frecuencia necesitará recuperar un valor de los parámetros del adaptador. Por ejemplo:



Ejemplo de recuperación de un valor del parámetro protocolType:

Para recuperar el valor del parámetro `protocolType` desde la secuencia de comandos Jython, utilice la API siguiente:

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

Framework.reportError(mensaje de cadena) y Framework.reportWarning (mensaje de cadena)

Durante la ejecución de una secuencia de comandos pueden ocurrir algunos errores (por ejemplo, fallo de conexión, problemas de hardware, tiempos de espera excedidos). Cuando se detectan estos errores, Framework puede notificar del problema. El mensaje que se notifica llega al servidor y se muestra al usuario.

Ejemplo de una notificación de error y mensaje:

En el ejemplo siguiente se ilustra el uso de la API `reportError(<Error Msg>)`:

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

Puede utilizar cualquiera de las API (`Framework.reportError(String message)`, `Framework.reportWarning(String message)`) para informar de un problema. La diferencia entre las dos API es que al informar de un error, la sonda guarda un archivo de registro de comunicaciones con todos los parámetros de la sesión en el sistema de archivos. De esta forma, podrá realizar el seguimiento de la sesión y comprender mejor el error.

Para obtener más información sobre los mensajes de error, consulte "Mensajes de error" en la página 78.

Encontrar las credenciales correctas (para adaptadores de conexión)

Un adaptador que intenta conectarse a un sistema remoto necesita probar todas las credenciales posibles. Uno de los parámetros necesarios al crear un cliente (a través de ClientFactory) es el ID de credenciales. La secuencia de comandos de credenciales obtiene acceso al posible conjunto de credenciales y prueba cada una de ellas usando el método `clientFactory.getAvailableProtocols()`. Cuando un conjunto de credenciales es correcto, el adaptador notifica un objeto de conexión de CI en el host de este CI de activación (con el Id. de credenciales que coincida con la dirección IP) a CMDB. Los adaptadores siguientes pueden usar directamente este CI de objeto de conexión para conectarse al conjunto de credenciales (es decir, los adaptadores no tienen que probar de nuevo todas las credenciales posibles).

En el ejemplo siguiente se muestra cómo obtener todas las entradas del protocolo SNMP. Obsérvese que aquí el IP se obtiene de los datos del CI de activación (`# Get the Trigger CI data values`).

La secuencia de comandos de conexión solicita todas las credenciales de protocolos posibles (`# Go over all the protocol credentials`) y las prueba en un bucle hasta que una sea satisfactoria (`resultVector`). Para obtener más información, consulte la entrada **paradigma de conexión de dos fases** en "Separación de adaptadores" en la página 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
```

Manejo de excepciones desde Java

Algunas clases Java lanzan una excepción tras un error. Se recomienda capturar la excepción y manejarla; en caso contrario, puede hacer que el adaptador finalice de forma inesperada.

Al capturar una excepción conocida, en la mayoría de los casos debería imprimir su seguimiento de la pila en el registro y enviar un mensaje adecuado a la interfaz de usuario, por ejemplo:

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

Si la excepción no es irrecuperable y la secuencia de comandos puede continuar, debería omitir la llamada al método `reportError()` y dejar que continúe la secuencia de comandos.

Soporte de la localización en los adaptadores Jython

La característica de configuración regional multilingüe permite a DFM funcionar con los distintos idiomas de los sistemas operativos y permitir las personalizaciones apropiadas en tiempo de ejecución.

Anteriormente, antes del Content Pack 3.00, DFM utilizaba una codificación especificada estáticamente para tratar la salida de todos los destinos de red. Sin embargo, este enfoque no es adecuado para una red TI multilingüe: para detectar hosts con distintos idiomas de los sistemas operativos, los administradores de la sonda tenían que volver a ejecutar manualmente los trabajos de DFM y varias veces con distintos parámetros de trabajo cada vez. Este procedimiento no solo producía una grave sobrecarga en la red sino que también evitaba varias funciones clave de DFM, como la invocación inmediata del trabajo en un CI de activación o la actualización automática de los datos en UCMDB por el Administrador de planificación.

De manera predeterminada se admiten los siguientes idiomas en la configuración regional: japonés, ruso y alemán. La configuración regional predeterminada es el inglés.

Esta sección incluye:

Adición de soporte para un nuevo idioma

En esta tarea se describe la manera de agregar soporte para un nuevo idioma.

Esta tarea incluye los siguientes pasos:

- "Agregar un paquete de recursos (archivos *.properties)" abajo
- "Declarar y registrar el objeto de idioma" abajo

1. ### Agregar un paquete de recursos (archivos *.properties)

   Agregue un paquete de recursos en función del trabajo que se va a realizar. En la tabla siguiente se enumeran los trabajos de DFM y el paquete de recursos que se utiliza en cada trabajo:

Trabajo	Nombre básico del paquete de recursos
Monitorización de archivos por shell	langFileMonitoring
Recursos y aplicaciones del host por shell	langHost_Resources_By_TTY, langTCP
Hosts por shell usando NSLOOKUP en servidor DNS	langNetwork
Conexión host por shell	langNetwork
Recopilar datos de red por shell o SNMP	langTCP
Recursos y aplicaciones del host por SNMP	langTCP
Conexión de Microsoft Exchange por NTCMD, topología de Microsoft Exchange por NTCMD	msExchange
Clúster MS por NTCMD	langMsCluster

   Para más información acerca de los paquetes, consulte "Trabajo con paquetes de recursos" en la página 54.

2. ### Declarar y registrar el objeto de idioma

   Para definir un nuevo idioma, agregue las dos líneas de código siguientes a la secuencia de comandos **shellutils.py**, que actualmente contiene la lista de todos los idiomas admitidos. La

secuencia de comandos se incluye en el paquete `AutoDiscoveryContent`. Para ver la secuencia de comandos, acceda a la ventana Administración de adaptador. Para obtener más información, consulte "Ventana Administración de adaptador" en HP Universal CMDB – Guía de Administración de Data Flow.

a. Declare el idioma de la forma siguiente:

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

Para obtener más información acerca del idioma de clase, consulte "Referencia de API" en la página 55. Para obtener más información sobre el objeto Class Locale, consulte http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html. Puede usar una configuración regional existente o definir una nueva.

b. Registre el idioma agregándolo a la colección siguiente:

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

Cambio del idioma predeterminado

Si el idioma del sistema operativo no se puede determinar, se utiliza el predeterminado. El idioma predeterminado se especifica en el archivo **shellutils.py**.

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

Para cambiar el idioma predeterminado, debe inicializar la variable DEFAULT_LANGUAGE con otro idioma. Para obtener más información, consulte "Adición de soporte para un nuevo idioma" en la página precedente.

Determinación del juego de caracteres para codificación

El juego de caracteres adecuado para la salida del comando de decodificación se determina en tiempo de ejecución. La solución multilingüe se basa en los hechos y suposiciones siguientes:

1. Es posible determinar el idioma del sistema operativo de manera independiente a la configuración regional, por ejemplo, ejecutando el comando **chcp** en Windows o el comando **locale** en Linux.
2. El idioma-codificación de la relación se conoce bien y se puede definir estáticamente. Por ejemplo, el ruso admite dos de las codificaciones más populares: `Cp866` y `Windows-1251`.
3. Se prefiere un juego de caracteres para cada idioma, por ejemplo, el juego de caracteres preferible para el ruso es `Cp866`. Esto significa que la mayoría de los comandos producen salidas en esta codificación.
4. La codificación en la que se proporciona la siguiente salida del comando es impredecible, pero es una de las posibles codificaciones de un idioma dado. Por ejemplo, al trabajar con un equipo Windows con una configuración local rusa, el sistema proporciona la salida del comando **ver** en Cp866, pero el comando **ipconfig** se proporciona en Windows-1251.
5. Un comando conocido produce palabras clave conocidas en su salida. Por ejemplo, el comando **ipconfig** contiene la forma traducida de la cadena **IP-Address**. Por tanto la salida

del comando **ipconfig** contiene **IP-Address** para el sistema operativo en inglés, IP-Адрес para el sistema operativo en ruso, **IP-Adresse** para el sistema operativo en alemán, y así sucesivamente.

Una vez detectado en qué idioma se produce la salida del comando (# 1), los posibles juegos de caracteres están limitados a uno o dos (# 2). Es más, se sabe las palabras clave contenidas en esta salida (# 5).

La solución, por tanto, es descodificar la salida del comando con una de las posibles codificaciones buscando una palabra clave en el resultado. Si se encuentra la palabra clave, se considera correcto el juego de caracteres actual.

Definición de un nuevo trabajo para operar con datos localizados

En esta tarea se describe cómo escribir un nuevo trabajo que puede operar con datos localizados.

Las secuencias de comandos por lo general ejecutan comandos y analizan su salida. Para recibir esta salida de comando de forma correctamente descodificada, hay que usar la API para la clase **ShellUtils**. Para obtener más información, consulte " Información general de la API de servicio web de HP Universal CMDB" en la página 230.

Por lo general el código adopta la forma siguiente:

```
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. Cree un cliente:

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

2. Cree una instancia de la clase **ShellUtils** y agréguela el idioma del sistema operativo. Si no se agrega el idioma, se utiliza el idioma predeterminado (por lo general, el inglés):

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

   Durante la inicialización del objeto, DFM detecta automáticamente el idioma del equipo y establece la codificación preferible para el objeto `Language` predefinido. La codificación preferible es la primera instancia que aparece en la lista de codificación.

3. Recupere el paquete de recursos apropiado desde **shellclient** mediante el método **getLanguageBundle**:

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

4. Recupere una palabra clave del paquete de recursos, adecuado para un comando particular:

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

5. Llame al método **executeCommandAndDecode** y pásele la palabra clave en el objeto **ShellUtils**:

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

   También se necesita el objeto ShellUtils object para vincular un usuario a la referencia de API (donde este método se describe de forma detallada).

6. Analice la salida de la manera habitual.

Descodificación de comandos sin una palabra clave

El método actual para la localización utiliza una palabra clave para decodificar todas las salidas de los comandos. Para obtener más información, consulte el paso "Recupere una palabra clave del paquete de recursos, adecuado para un comando particular:" en "Definición de un nuevo trabajo para operar con datos localizados" en la página precedente.

Sin embargo, otro enfoque utiliza una palabra clave para decodificar solo la primera salida del comando y, a continuación, descodifica comandos adicionales con el juego de caracteres usado para descodificar el primer comandos. Para ello, utilice los métodos **getCharsetName** y **useCharset** del objeto **ShellUtils**.

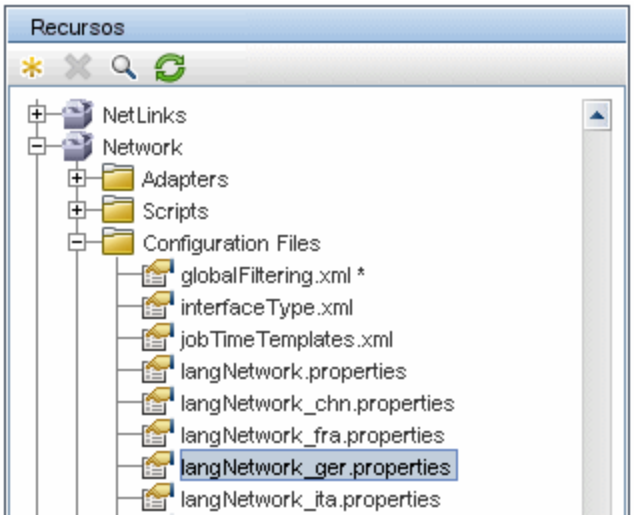
El caso de uso regular funciona de la forma siguiente:

1. Llame al método **executeCommandAndDecode** una vez.
2. Obtenga el nombre del último juego de caracteres usado con el método **getCharsetName**.
3. De manera predeterminada, haga que **shellUtils** utilice este juego de caracteres llamando al método **useCharset** en el objeto **ShellUtils**.
4. Llame al método **execCmd** de **ShellUtils** una o más veces. La salida se devuelve con el juego de caracteres especificado en el paso anterior. No tiene lugar ninguna operación de descodificación adicional.

Trabajo con paquetes de recursos

Un paquete de recursos es un archivo que lleva la extensión properties (***.properties**). Se puede considerar un archivo de propiedades con un diccionario que almacena datos en el formato `clave = valor`. Cada fila de un archivo de propiedades contiene una asociación `clave = valor`. La principal funcionalidad de un paquete de recursos es devolver un valor por su clave.

Los paquetes de recursos se localizan en el equipo de sondas: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles**. Se descargan desde el servidor de UCMDB como cualquier otro archivo de configuración. Se pueden editar, agregar o suprimir en la ventana Recursos. Para obtener más información, consulte "Panel Archivo de configuración" en HP Universal CMDB – Guía de Administración de Data Flow.



Al detectar un destino, DFM por lo general tiene que analizar el texto desde una salida de comando o contenido de archivo. Este análisis suele estar basado en una expresión regular. Los distintos idiomas requieren el uso de diferentes expresiones regulares para su análisis. Para que se escriba código una vez para todos los idiomas, se deben extraer todos los datos expecíficos del idioma en paquetes de recursos. Hay un paquete de recursos para cada idioma. (Aunque es posible que un paquete de recursos contenga datos para distintos idiomas, en DFM un paquete de recursos siempre contiene datos para un idioma).

La misma secuencia de comandos Jython no incluye datos específicos del idioma, codificados de forma rígida (por ejemplo, expresiones regulares específicas del idioma). La secuencia de comandos determina el idioma del sistema remoto, carga el paquete de recursos apropiado y obtiene todos los datos específicos del idioma por clave específica.

En DFM, los paquetes de recursos adoptan un formato de nombre específico: `<nombre_básico>_<identificador_idioma>.properties`, por ejemplo, `langNetwork_spa.properties`. (El paquete de recursos predeterminado adota el formato siguiente: `<nombre_básico>.properties`, por ejemplo, `langNetwork.properties`.)

El formato del `nombre_básico` refleja el propósito previsto de este paquete. Por ejemplo, **langMsCluster** significa el paquete de recursos contiene recursos específicos del idioma utilizados por los trabajos de MS Cluster.

El formato `identificador_idioma` es un acrónimo de tres letras utilizado para identificar el idioma. Por ejemplo, `rus` representa el idioma ruso y `ger` el idioma alemán. Este identificador de idioma se incluye en la declaración del objeto `Language`.

Referencia de API

Esta sección incluye:

- "Método getLanguageBundle" en la página siguiente
- "Campo osLanguage" en la página siguiente

Clase Language

Esta clase encapsula información sobre el idioma, como el postfijo del paquete de recursos, la codificación posible, entre otra.

Campos

Nombre	Descripción
configuración regional	Objeto Java que representa la configuración regional.
bundlePostfix	El postfijo del paquete de recursos. Este postfijo se utiliza en los nombres de los archivos de paquetes de recursos para identificar el idioma. Por ejemplo, el paquete **langNetwork_ger.properties** incluye el postfijo del paquete **ger**.
charsets	Los juegos de caracteres utilizados para codificar este idioma. Cada idioma puede tener varios juegos de caracteres. Por ejemplo, el idioma ruso se suele codificar con la codificación `Cp866` y `Windows-1251`.
wmiCodes	Lista de códigos WMI utilizados por el sistema operativo Microsoft Windows para identificar el idioma. Todos los códigos posibles se enumeran en http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (sección OSLanguage). Uno de los métodos para identificar el idioma del sistema operativo es consultar el sistema operativo de la clase WMI para la propiedad `OSLanguage`.
codepage	Página de códigos utilizada con un idioma específico. Por ejemplo, se utiliza `866` para los equipos en ruso y `437` para los equipos en inglés. Uno de los métodos para identificar el idioma del sistema operativo es recuperar su página de códigos predeterminada (por ejemplo, por el comando `OSLanguage`).

Método executeCommandAndDecode

Este método se ha previsto para su uso con las secuencias de comandos Jython de lógica de negocio. Encapsula la operación de descodificación y devuelve una salida descodificada de comandos.

Argumentos

Nombre	Descripción
cmd	El comando real que se va a ejecutar.
keyword	La palabra clave que se va a usar en la operación de descodificación.
framework	El objeto Framework pasado en todas las secuencias de comandos Jython

Nombre	Descripción
	ejecutables de DFM.
timeout	El tiempo de espera agotado del comando.
waitForTimeout	Especifica si el cliente debe esperar a que se agote el tiempo de espera.
useSudo	Especifica si se debe usar `sudo` (solo relevante para clientes de equipos UNIX).
language	Permite especificar directamente el idioma en lugar de detectarlo automáticamente.

Método getCharsetName

Este método devuelve el nombre del juego de caracteres últimamente utilizado.

Método useCharset

Este método establece el juego de caracteres en la instancia `ShellUtils`, que utilice este juego de caracteres para la descodificación de datos inicial.

Argumentos

Nombre	Descripción
charsetName	El nombre del juego de caracteres, por ejemplo, `windows-1251` o `UTF-8`.

Consulte también "Método getCharsetName " arriba

Método getLanguageBundle

Este método debería utilizarse para obtener el paquete de recursos correcto. Reemplaza a la API siguiente:

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

Argumentos

Nombre	Descripción
baseName	El nombre del paquete sin el sufijo de idioma, por ejemplo, `langNetwork`.
language	El objeto de idioma. Aquí debe pasarse `ShellUtils.osLanguage`.
framework	El objeto común Framework que se pasa en todas las secuencias de comandos Jython ejecutables de DFM.

Campo osLanguage

Este campo contiene un objeto que representa el idioma.

Trabajo con Discovery Analyzer

El uso previsto de la herramienta Discovery Analyzer es la depuración al desarrollar paquetes, secuencias de comandos o cualquier otro contenido. La herramienta ejecuta un trabajo en un destino remoto y devuelve los registros que contienen datos de información, advertencias y errores así como los resultados de los CI detectados.

Obsérvese que los resultados no siempre se notifican en la interfaz de usuario. Esto es debido a que los resultados se notifican de dos formas y solo se admite una de ellas. Además, el registro de comunicación no se admite en Eclipse.

Al ejecutar la herramienta en Eclipse, el archivo **DataFlowProbe.properties** (**C:\hp\UCMDB\DataFlowProbe\conf\DataFlowProbe.properties**) debe contener el siguiente parámetro establecido en **true**:

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

Para obtener más información, consulte "Ejecución de Discovery Analyzer desde Eclipse" en la página 64.

En todos los demás casos (cuando la herramienta se ejecuta desde el archivo **cmd** o mientras se está ejecutando la sonda) este indicador debe establecerse en **false**:

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

Tareas y registros

Un archivo de tareas contiene datos concernientes a la tarea que se va a ejecutar. La tarea consta de información como el nombre del trabajo y parámetros necesarios que definen el CI de activación, por ejemplo, la dirección de destino remota.

Un archivo de registro contiene información de la tarea así como los resultados de una ejecución específica, es decir, la comunicación detallada (incluyendo una respuesta) entre la sonda o Discovery Analyzer (el módulo que haya ejecutado la tarea) y el destino remoto.

Una tarea que se define por un archivo de tareas se puede ejecutar en un destino remoto, mientras que una tarea definida por un archivo de registro (que contiene los datos adicionales relativos a una ejecución específica) se puede ejecutar y también se puede reproducir (es decir, puede reproducir la misma ejecución documentada en el archivo de registro).

Registros

Los registros proporcionan información sobre la última ejecución, de la forma siguiente:

- **Registro general**. Este registro incluye todos los datos de información, errores y advertencias que han tenido lugar durante la ejecución.
- **Registro de comunicación**. Este registro contiene la comunicación detallada entre Discovery Analyzer y el destino remoto (incluyendo su respuesta). Tras la ejecución, el registro se guarda como un archivo de registro.
- **Registro de resultados**. Muestra una lista de los CI detectados. El tiempo de aparición de cada CI depende del diseño de los adaptadores y secuencias de comandos.

Puede guardar todos los registros juntos o por separado. Cuando guarde todos los registros, se guardan juntos con un único nombre.

Si reproduce un archivo de registros, los mismos datos se muestran en el archivo de comunicación, con la única diferencia del tiempo de ejecución.

Limitación: Los registros de comunicación y de resultados no están disponibles cuando se ejecuta Discovery Analyzer mediante Eclipse.

Esta sección incluye los siguientes pasos:

1. ## Requisitos previos

   - La sonda debe instalarse. (La herramienta Discovery Analyzer está instalada como parte del proceso de instalación de la sonda y comparte recursos con ella).
   - La sonda no tiene que estar ejecutándose mientras trabaja con Discovery Analyzer.

     Sin embargo, si la sonda ya se ejecuta en un servidor de UCMDB, todos los recursos necesarios ya están descargados en el sistema de archivos. Si la sonda no se ha ejecutado, puede cargar los recursos necesarios por Discovery Analyzer mediante el menú Configuración. Para obtener más información, consulte "Import Settings" en la página 63.
   - No es preciso instalar el servidor de CMDB.

2. ## Acceso a Discovery Analyzer

   Puede acceder a Discovery Analyzer de una de las siguientes formas:

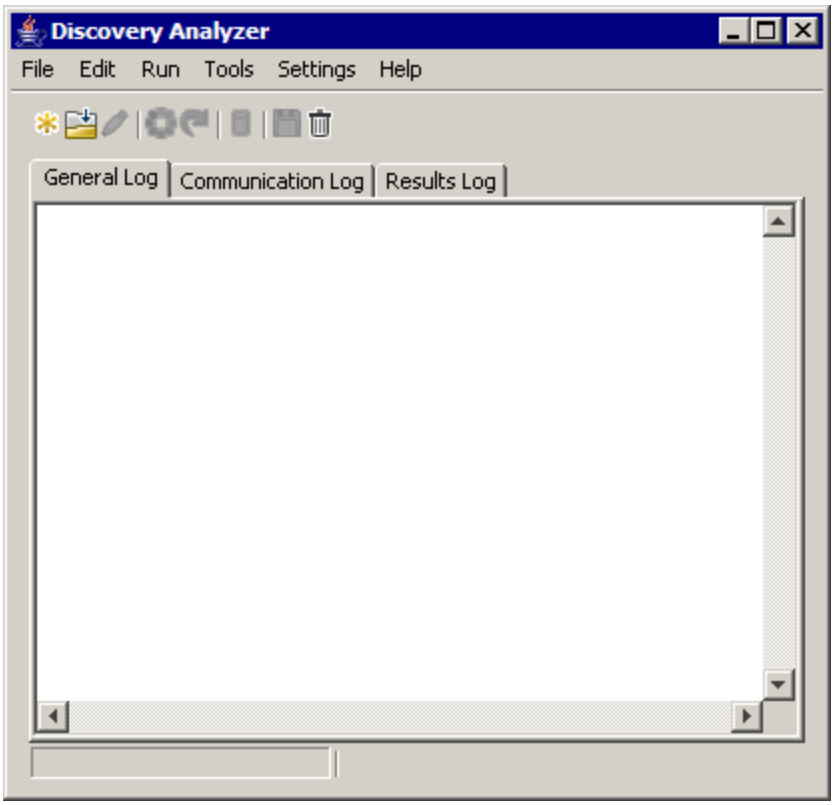
   - Cuando se trabaja con Eclipse.

     La instalación de la sonda incluye un espacio de trabajo Eclipse predeterminado situado en **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**. Este espacio de

trabajo incluye una secuencia de comandos Jyton para iniciar Discovery Analyzer (**startDiscoveryAnalyzerScript.py**) así como un vínculo a todas las secuencias de comandos DFM. Si inicia la herramienta de esta forma, puede localizar los puntos de interrupción dentro de las secuencias de comandos Jython para su depuración.

- Directamente, al hacer doble clic en el archivo de la siguiente carpeta: **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzer.cmd**. Para obtener más información, consulte la sección siguiente.

Se abre la ventana Discovery Analyzer:



3. **Definición de una tarea**

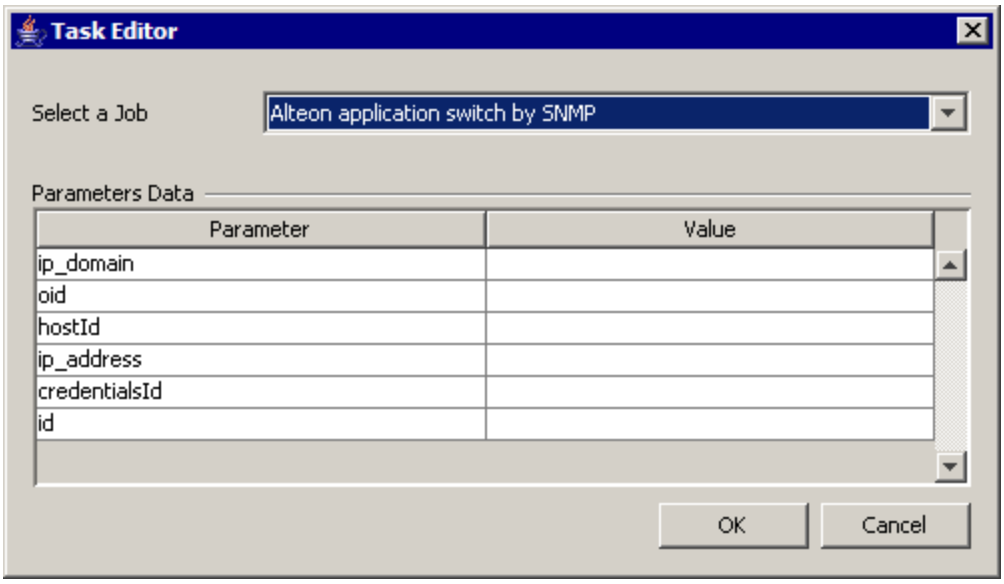
Puede definir una de la tarea usando uno de los siguientes métodos:

- Mediante la definición de una nueva tarea. Para obtener más información, consulte "Definición de una nueva tarea" abajo.
- Mediante la importación de una tarea desde un archivo de registro. Para obtener más información, consulte "Recuperación de un registro" en la página siguiente.
- Mediante la importación de una tarea guardada desde un archivo de tareas. Para obtener más información, consulte "Apertura de un archivo de tareas" en la página 62.
- Mediante la recuperación de un trabajo desde la base de datos interna de la sonda. Para obtener más información, consulte "Importación de una tarea desde la base de datos" en la página 62.

4. **Definición de una nueva tarea**

a. Muestre el Task Editor: Haga clic en el botón **New Task** .

Task Editor muestra una lista de los trabajos que existen actualmente en el sistema de archivos. La lista se actualiza cada vez que la sonda recibe tareas del servidor, o cuando los paquetes se implantan manualmente desde el menú Configuración.



b. Seleccione un trabajo.

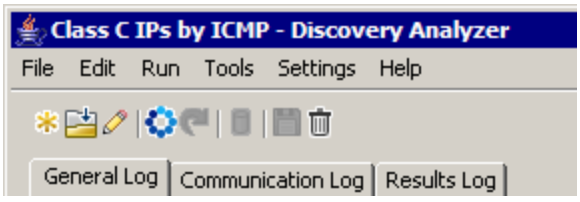
c. Introduzca los valores para todos los parámetros.

Los parámetros mostrados aquí son parámetros del adaptador DFM. Se pueden ver en el panel Discovery Pattern Parameters de la pestaña Pattern Signature. Para obtener más información, consulte "Ficha Definición de adaptador" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Todos los campos son obligatorios (a no ser que la secuencia de comandos del trabajo exija que el campo esté vacío).

En el caso de aquellos parámetros que requieran un Id. o un valor de entrada para el Id. de credenciales, puede usar los Id. creados aleatoriamente: haga clic con el botón derecho en el cuadro de valor y seleccione **Generate random CMDB ID** o **Credential Chooser**.

La tarea ahora está activa y el nombre de la tarea abierta se muestra en la barra de título:



d. Continúe con el procedimiento para definir una tarea. Para obtener más información, consulte "Almacenamiento de la tarea y de los registros" en la página siguiente.

5. Recuperación de un registro

Puede definir una tarea abriendo un archivo de registro que contiene datos relacionados con

una ejecución específica. Si se define una tarea de esta forma, puede reproducir la ejecución específica seleccionando la opción de reproducción. (Si se reproduce una tarea, las respuestas se reciben desde los datos almacenados en el archivo de registro y no desde el destino remoto).

Seleccione **File > Open Record**. Busque la carpeta en donde ha guardado el registro. El registro ahora está activo y el nombre de la tarea abierta se muestra en la barra de título.

Para obtener más información sobre cómo adquirir un registro de archivo, consulte "Registro de código DFM" en la página 73.

6. ### Apertura de un archivo de tareas

   Puede definir una tarea desde un archivo de tareas: Seleccione **File > Open Task**.

7. ### Importación de una tarea desde la base de datos

   Puede recuperar una tarea desde la base de datos de la sonda a condición de que esta ya se haya ejecutado y tenga tareas activas en su base de datos interna. Puede usar los valores del parámetro para definir la tarea.

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

   b. En el cuadro de diálogo que se abre, seleccione la tarea que desea ejecutar y haga clic en **OK**.

   c. Continúe con el procedimiento para definir una tarea. Para obtener más información, consulte "Almacenamiento de la tarea y de los registros" abajo.

8. ### Edición de una tarea

   Una vez definida la tarea, se muestra en la barra de título el nombre de la tarea (o el archivo). Ahora el archivo se puede editar.

   a. Seleccione **File > Edit Task**.

   b. Realice los cambios que desee en la tarea y haga clic en **OK**.

9. ### Almacenamiento de la tarea y de los registros

   Puede guardar los parámetros de las tareas: Seleccione **File > Save Task**.

   Las opciones siguientes están disponibles solo después de que se haya ejecutado una tarea:

   - Guarde un registro de la tarea. Puede guardar los parámetros de la tarea y los resultados de la ejecución de la tarea: Seleccione **File > Save Record**.
   - Guarde un registro de la tarea: Seleccione **File > Save General Log**.
   - Guarde los resultados: Seleccione **File > Save Results**.

10. ### Ejecución de la tarea

    El siguiente paso del procedimiento es ejecutar la tarea que ha creado.

    a. Importe el archivo de configuración de credenciales/rangos. Para obtener más información, consulte "Import Settings" en la página siguiente.

    b. Para ejecutar a tarea solo en un destino remoto, haga clic en el botón **Run Task**.

Discovery Analyzer ejecuta el trabajo y muestra información en los tres archivos de registro: **General**, **Communication** y **Results**.

c. Puede guardar los archivos de registros, ya sea juntos o por separado: Seleccione **File > Save General Log**, **Save Record**, **Save Results** o **Save All Logs**. Para obtener más información sobre los archivos de registro, consulte "Registros" en la página 58.

d. Si una tarea se recupera desde un archivo de registro, la ejecución que está documentada en este archivo se puede reproducir haciendo clic en el botón **Playback**. Se muestra el mismo registro de comunicación, pero el tiempo de ejecución se actualiza.

11. Envío de un resultado de la tarea al servidor

Si la ejecución de la tarea finaliza con resultados (es decir, la pestaña Results Log muestra una lista de CI detectados), puede enviar los resultados al servidor de UCMDB. Esto es útil, por ejemplo, si estuviera probando anteriormente una secuencia de comandos cuando el servidor estaba inactivo.

Nota: Puede enviar los resultados solo a un servidor de UCMDB que recibe las tareas desde la sonda que está instalada en el mismo equipo que Discovery Analyzer.

12. Import Settings

Para ejecutar las tareas en el archivo de registro de reproducción, debe importar el archivo **domainScopeDocument.bin**. Durante la importación, debe introducir una contraseña.

a. Inicie un navegador web y escriba la siguiente dirección URL: **http://localhost:8080/jmx-console**. Es posible que tenga que iniciar sesión con un nombre de usuario y una contraseña.

b. Haga clic en **UCMDB:service=DiscoveryManager** para abrir la página JMX MBEAN View.

c. Localice la operación **exportCredentialsAndRangesInformation**. Realice las siguientes operaciones:

- Introduzca el ID de cliente (el valor predeterminado es **1**).
- Introduzca un nombre para el archivo exportado.
- Escriba la contraseña.
- Establezca **isEncrypted** en **False**.

d. Haga clic en **Invoke** para exportar el archivo **domainScopeDocument.bin**.

Cuando el proceso de exportación finaliza de forma satisfactoria, el archivo se guarda en la ubicación siguiente: **C:\hp\UCMDB\UCMDBServer\conf\discovery\<customer_dir>**.

e. Copie el archivo **domainScopeDocument.bin** al sistema de archivos de la sonda de Data Flow e impórtelo seleccionado: **Configuración > Importar domainScopeDocument**.

Nota: Durante la importación del archivo **domainScopeDocument**, se le pedirá que indique una contraseña. Esta petición también se muestra después de cada reinicio de Discovery Analyzer y antes de que se ejecute la primera tarea o registro.

13. **Puntos de interrupción**

    Si ejecuta Discovery Analyzer desde la secuencia de comandos Python, puede agregar puntos de interrupción en la secuencia de comandos.

14. **Configuración de Eclipse**

    Para obtener más información sobre cómo ejecutar las secuencias de comandos Jython en modo de depuración, consulte "Ejecución de Discovery Analyzer desde Eclipse" abajo.

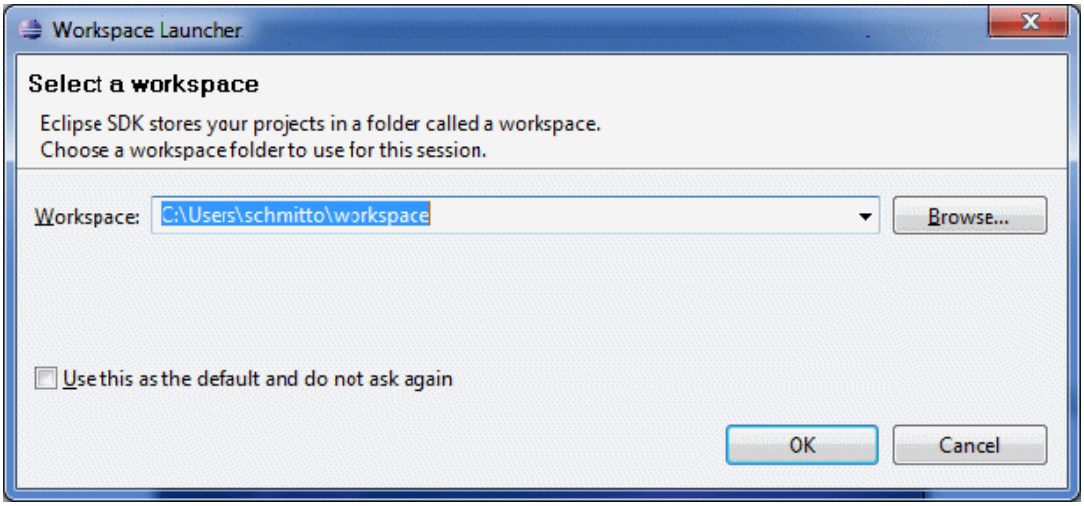
Ejecución de Discovery Analyzer desde Eclipse

En esta tarea se explica cómo configurar Eclipse para que pueda ejecutar las secuencias de comandos de Jython en modo de depuración, permitiendo por tanto una mayor visibilidad a los subprocesos de trabajo, CI de activación y resultados.

Esta sección incluye los siguientes pasos:

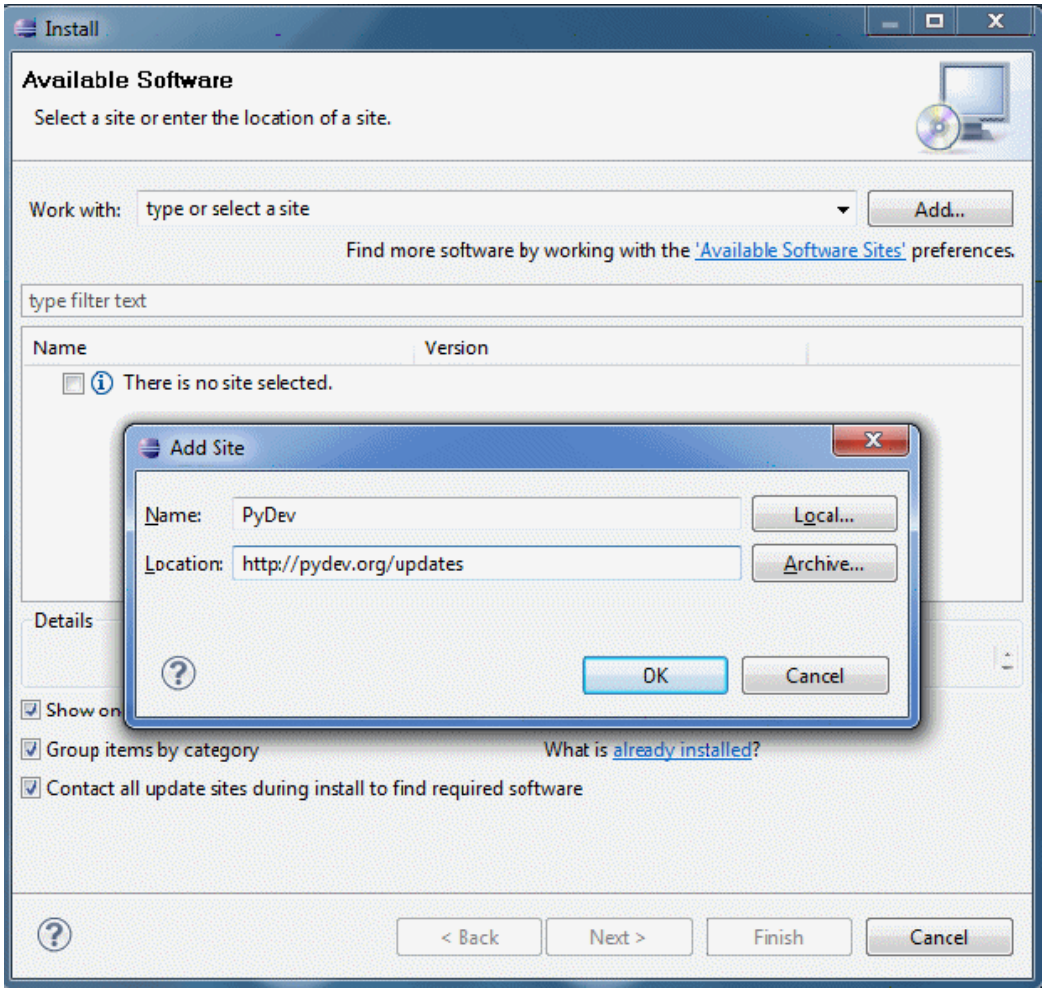
1. **Requisitos previos**
   - Instale la versión de Eclipse más reciente en el equipo. La aplicación está disponible en www.eclipse.org.
   - Compruebe que la sonda de Data Flow está instalado en el mismo equipo.
   - Compruebe que el parámetro **appilog.agent.local.discoveryAnalyzerFromEclipse** del archivo **DataFlowProbe.properties** está establecido en **true**.
2. Desempaquetar Eclipse e iniciarlo
3. **Configurar el espacio de trabajo predeterminado**

   Configure el espacio de trabajo predeterminado donde Eclipse va a guardar y almacenar todos los proyectos y datos relacionados.



4. ## Configurar las extensiones PyDev

   a. Acceda a **Help > Install New Software**, haga clic en **Add**, escriba un nombre para el complemento PyDev en el campo Location, agregue la dirección URL del sitio donde se va a descargar **pydev**: **http://pydev.org/updates**. Haga clic en **Aceptar**.

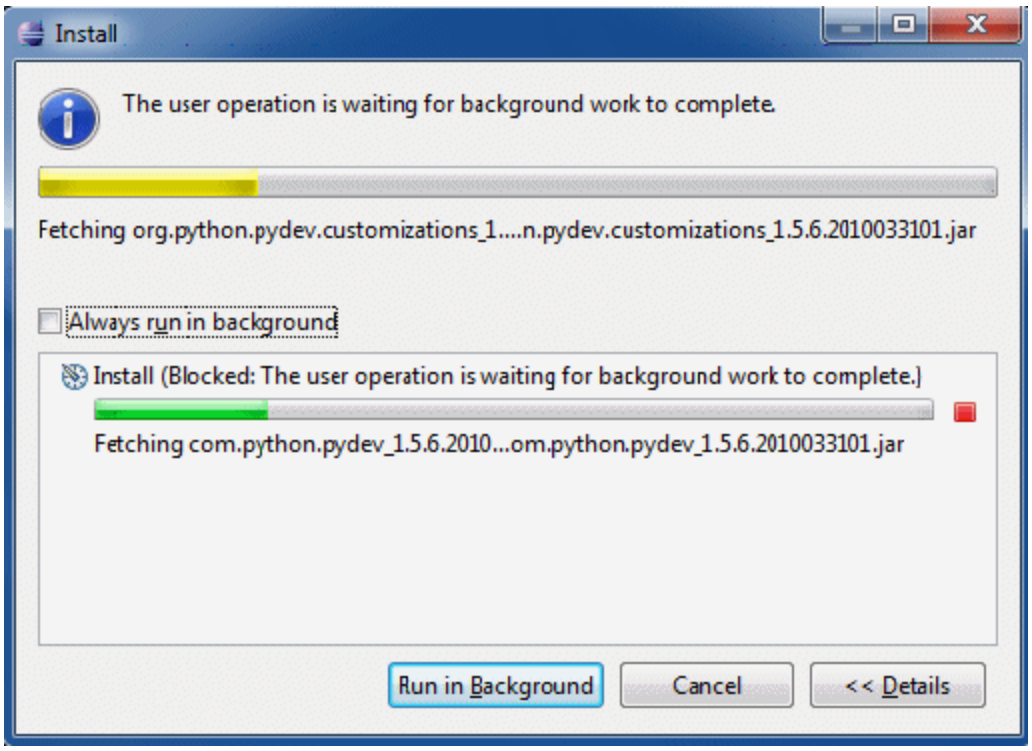


> **Nota:** Las extensiones PyDev y PyDev ya están fusionadas en un complemento, ya que las extensiones PyDev son ahora de código abierto. Para obtener información adicional, visite http://pydev.org.

b. En la ventana que se abre, seleccione **Pydev**. El segundo complemento es un complemento para la interfaz de usuario orientada a tareas. Haga clic en **Next**, compruebe los detalles de la instalación y vuelva a hacer clic en **Next**.

c. Acepte el contrato de licencia y haga clic en **Next**.

d. Pydev se instala. Si se le pregunta si desea instalar contenido sin firmar, confírmelo haciendo clic en **OK**.

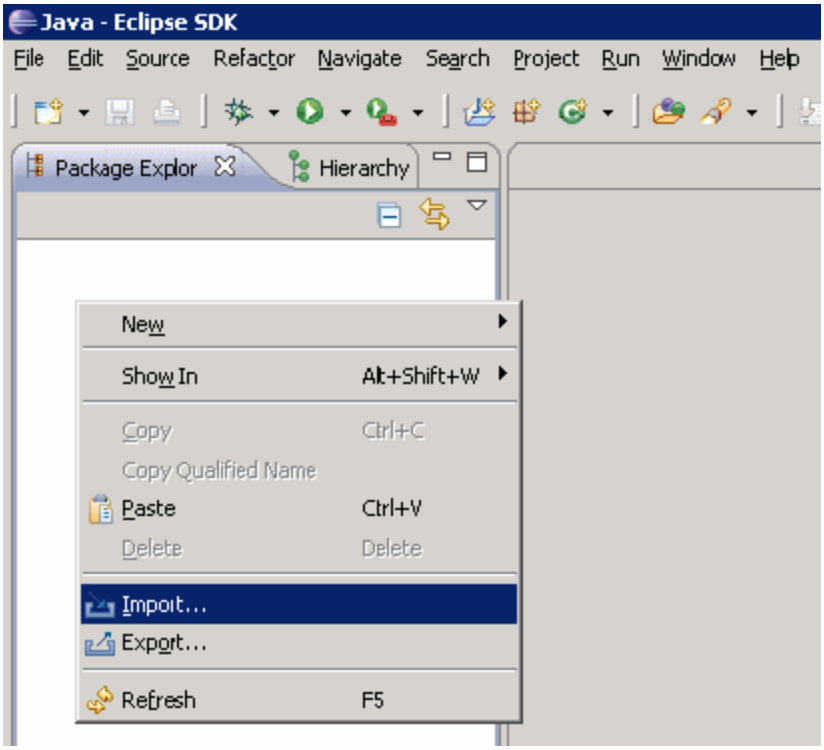


e. Reinicie Eclipse.

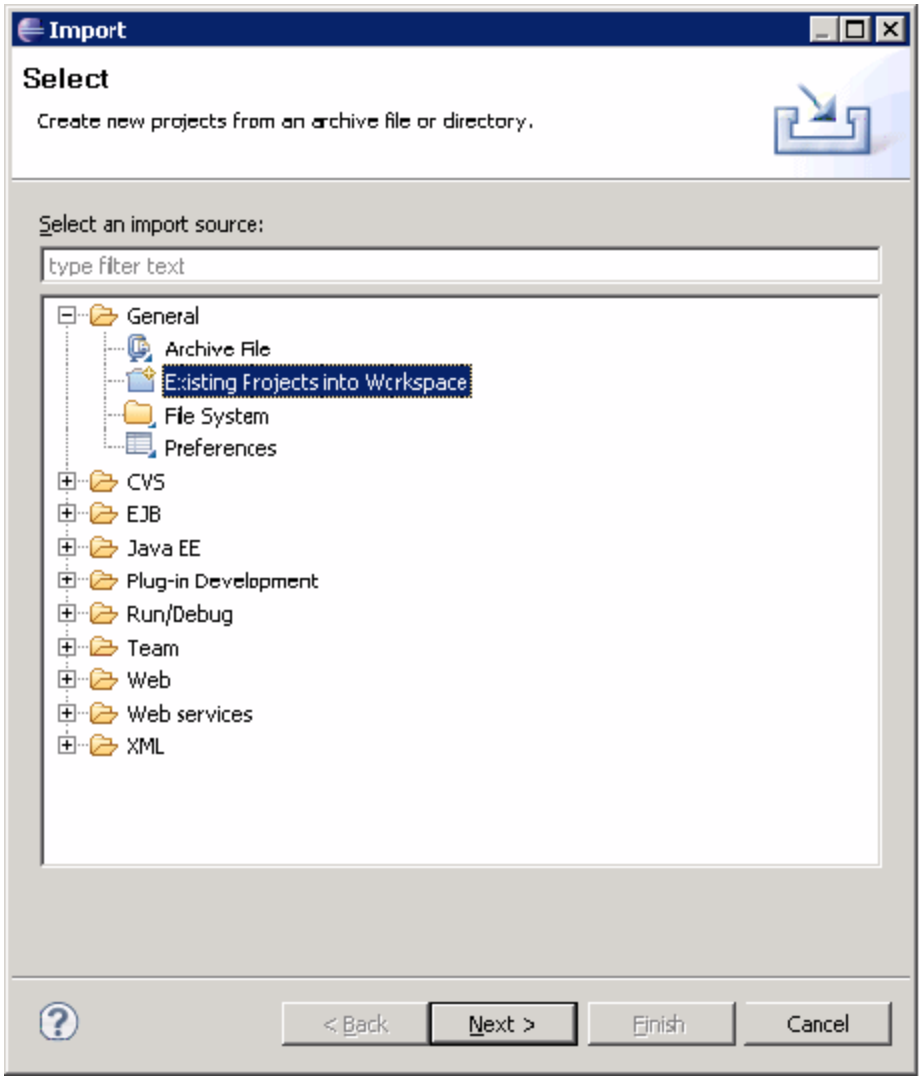
PyDev ahora está instalado en el IDE de Eclipse. Ahora ya dispone de nuevas perspectivas en Eclipse y el IDE puede interpretar secuencias de comandos de Python (resaltado de texto, opciones de configuración adicionales, etc.).

5. Configurar el espacio de trabajo de Discovery Analyzer

a. Importe los archivos necesarios: Haga clic con el botón derecho en el área en blanco de Package Explorer y haga clic en **Import** para importar el **discoveryAnalyzerWorkspace** preconfigurado, incluido con la instalación de la sonda.



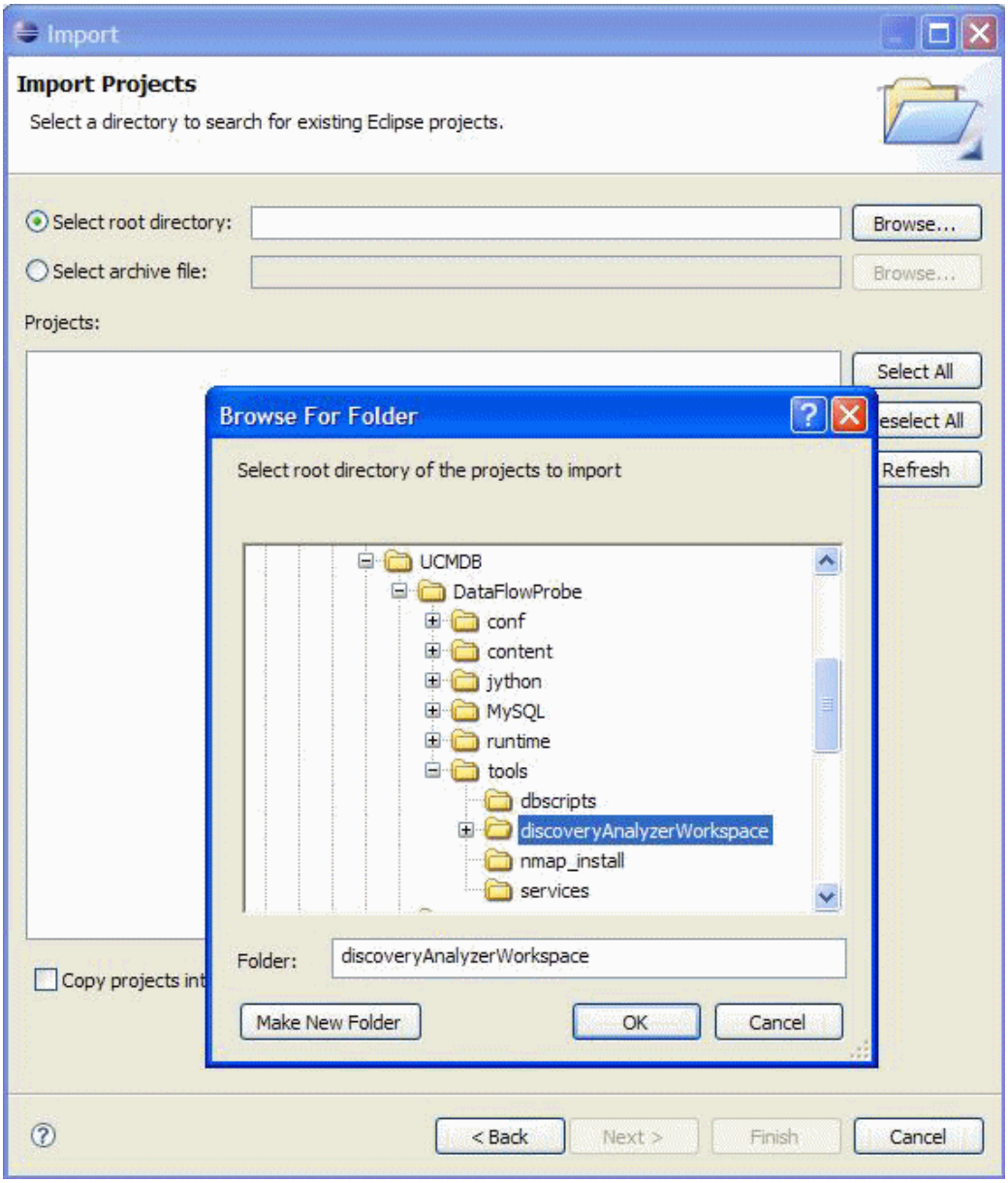
b. En **General**, seleccione **Existing projects into Workspace** para importar el proyecto en el espacio de trabajo de Eclipse.



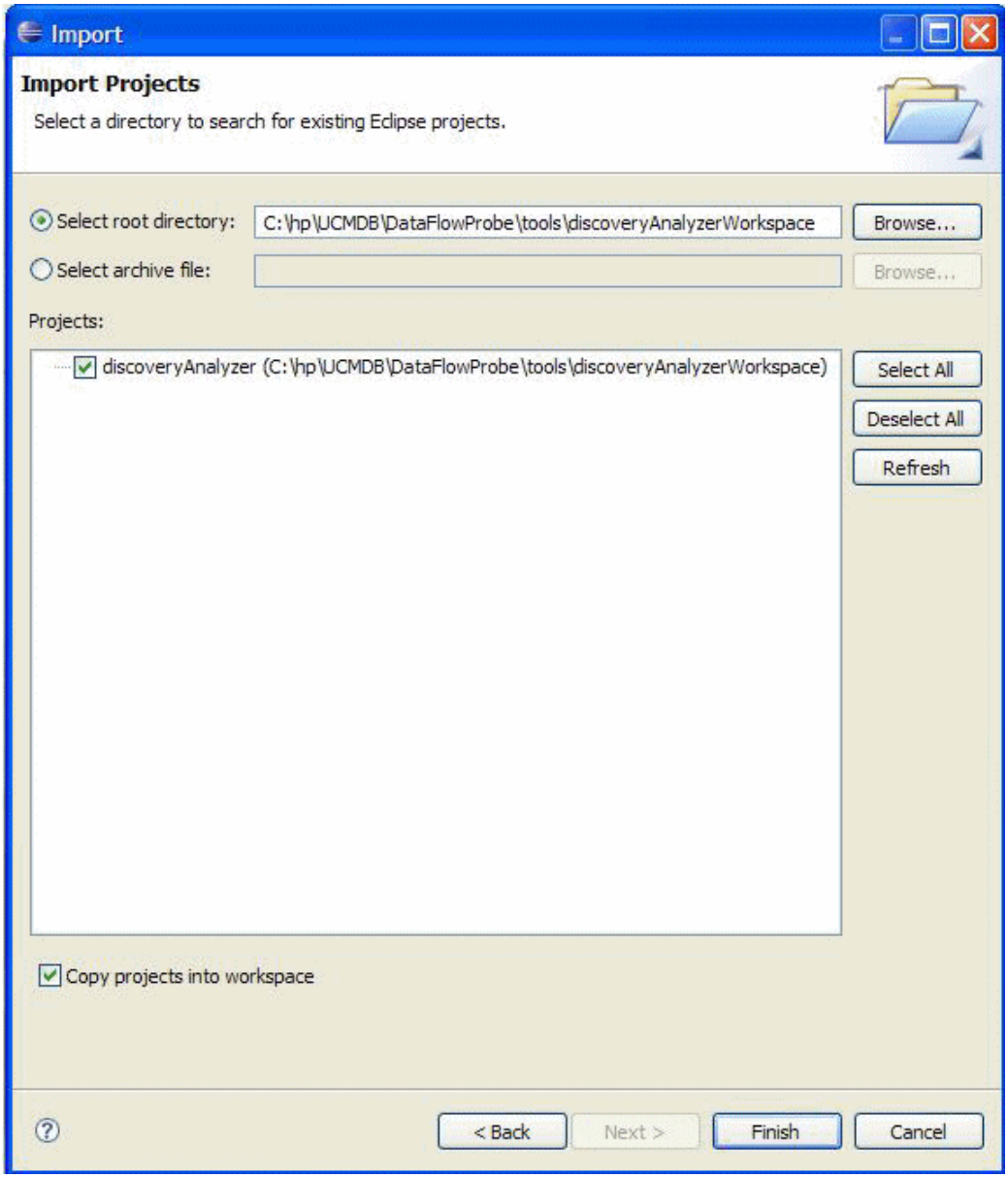
c. En **Select root directory**, seleccione el espacio de trabajo Analyzer, que generalmente se encuentra en:

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

d. Seleccione **Copy projects into workspace** para crear una copia verdadera del espacio de trabajo existente. Este es un paso importante: En caso de fallo, puede volver a importar el **discoveryAnalyserWorkspace** original.



e. Haga clic en **Finish** para iniciar la importación.



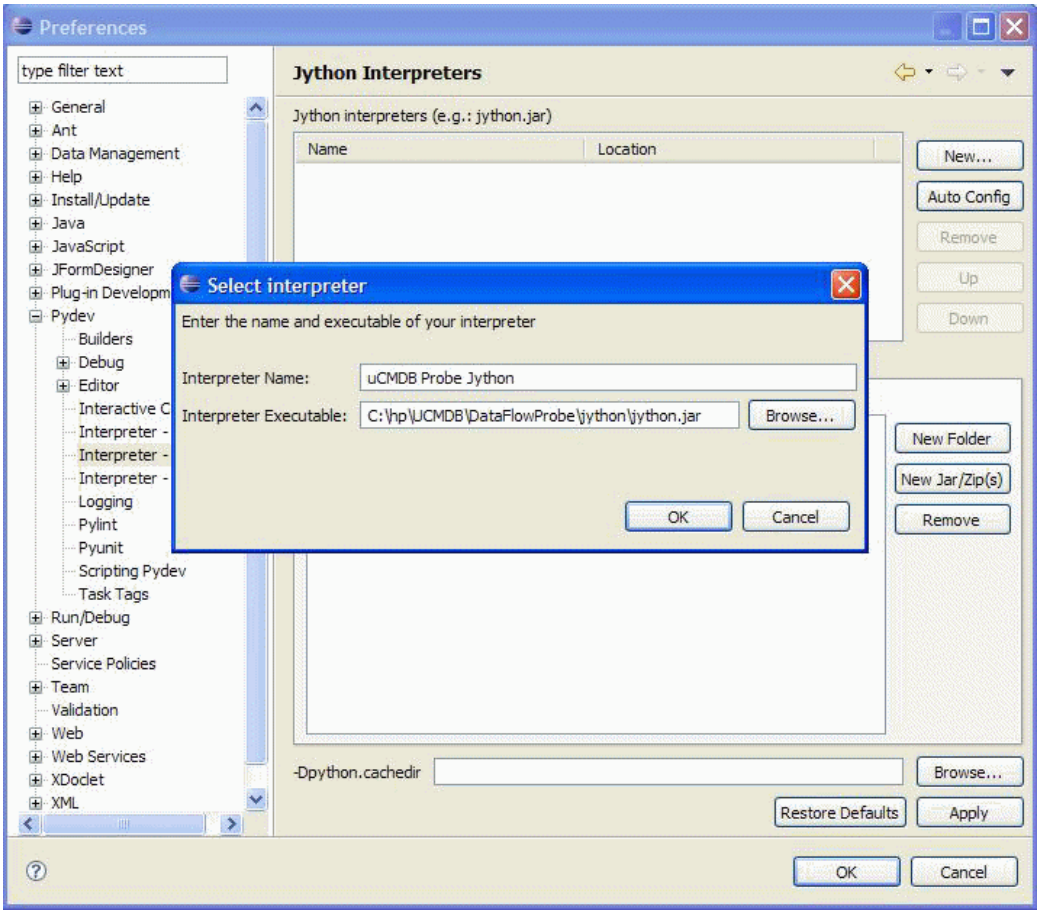
6. ## Configurar la ruta de clase y el intérprete

   a. Haga clic con el botón derecho en **discoveryAnalyzerWorkspace** y seleccione **Properties** para mostrar la configuración específica del proyecto.

   b. Vaya a **Pydev > Interpreter/Grammar** y haga clic en **Please configure an interpreter in the related preferences before proceeding**.

      En este paso se configura el mismo intérprete Jython que está usando la sonda para garantizar que las secuencias de comandos no se interpretan por una versión distinta de Jython.

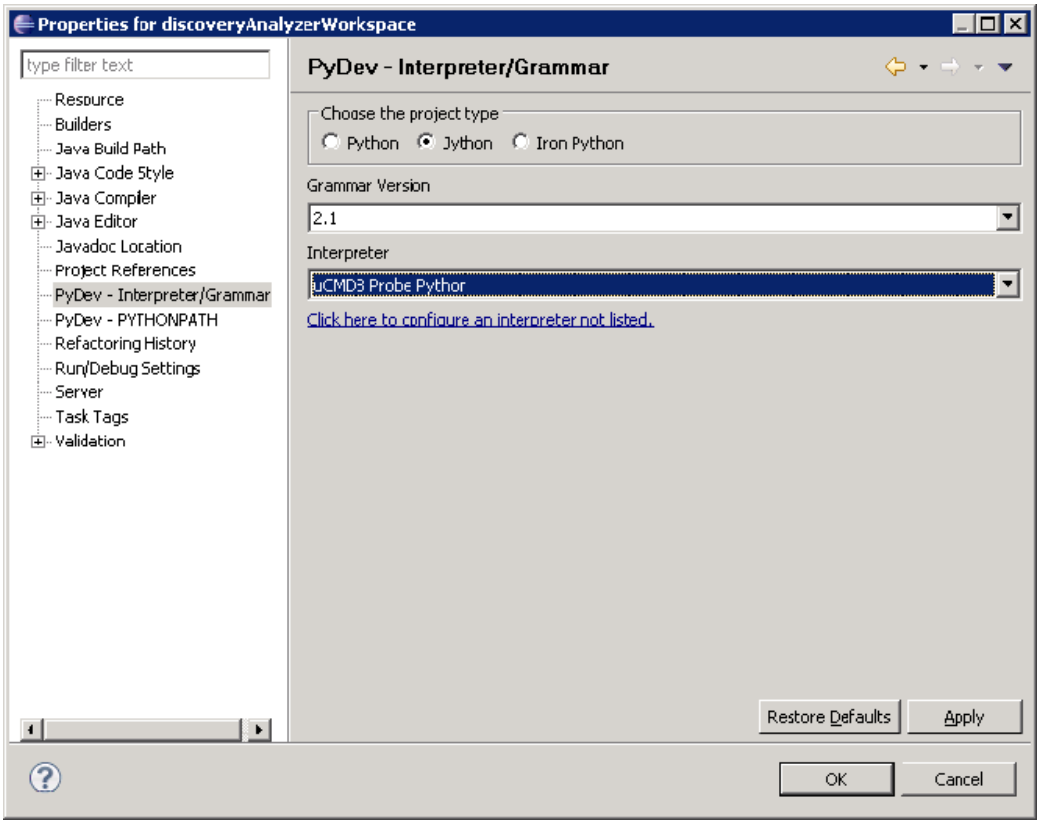
   c. Haga clic en **New**, escriba un nombre para el intérprete y seleccione el archivo en la carpeta siguiente: **C:\hp\UCMDB\DataFlowProbe\jython\jython.jar**.



d. Haga clic en **Aceptar**. Si se muestra una ventana en la que se le pregunta si desea seleccionar las carpetas que deben importarse a la ruta del sistema Python, no cambie nada (debería ser **C:\hp\UCMDB\DataFlowProbe\jython** y **C:\hp\UCMDB\DataFlowProbe\jython\lib**) y haga clic en **OK**.

e. Haga clic en **Apply** y, a continuación, en **OK**.

f. Haga clic en **Interpreter** y seleccione el intérprete que acaba de crear.



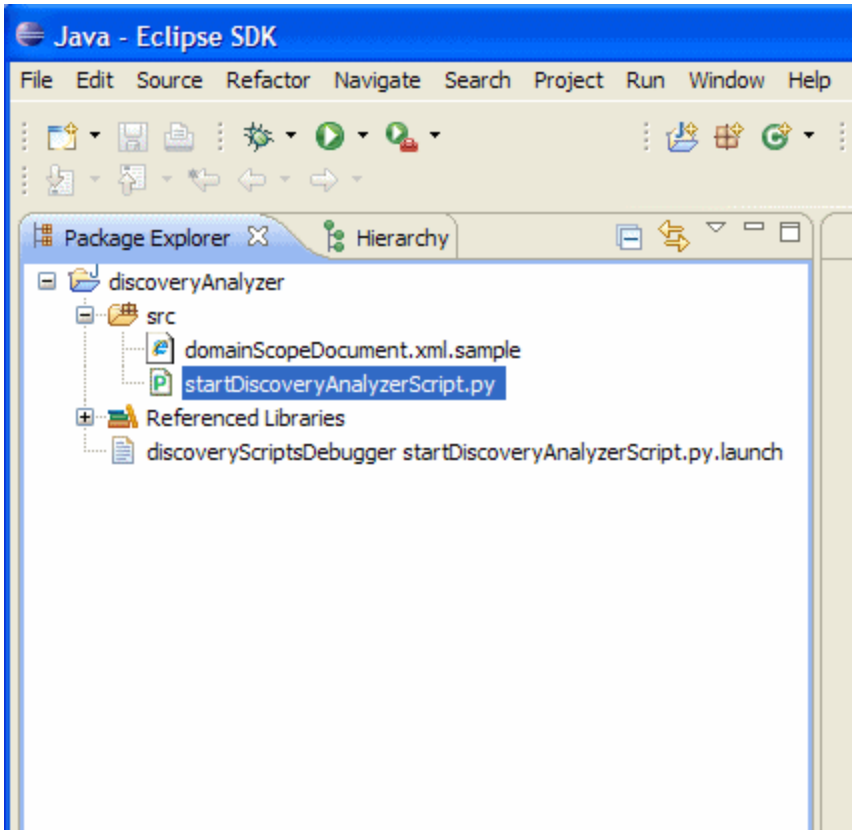
g. Haga clic en **Apply** y, a continuación, en **OK**.

El intérprete de Jython ahora es el mismo que el que está usando la sonda.

7. Ejecutar Discovery Analyzer

a. Agregue un punto de interrupción en la secuencia de comandos Jython para ser depurado.

b. Para iniciar Discovery Analyzer, seleccione **startDiscoveryAnalyzerScript.py** en el proyecto **discoveryAnalyzerWorkspace\src**. Haga clic con el botón derecho en el archivo y elija **Debug as > Jython run**.

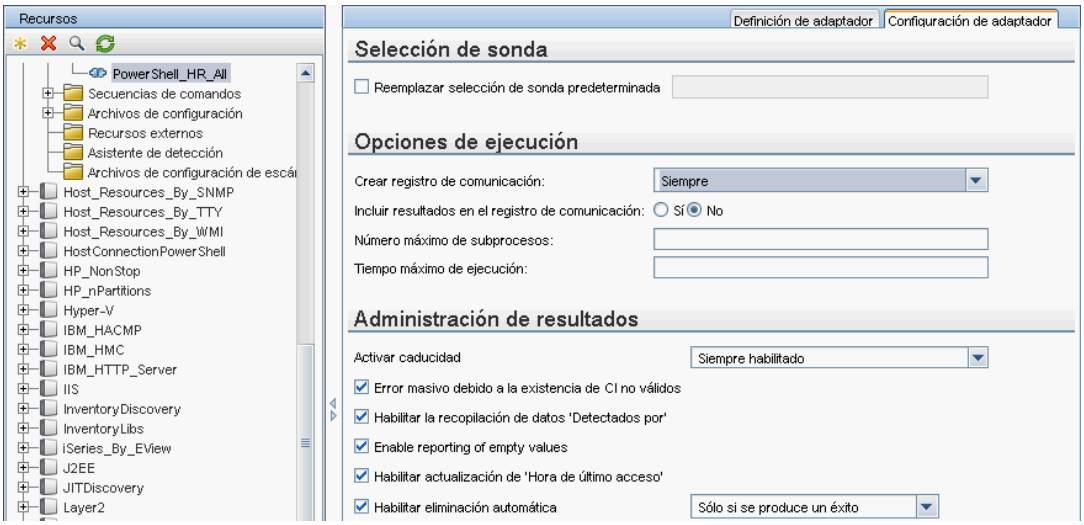


Registro de código DFM

Puede ser muy útil registrar una ejecución completa, incluyendo todos los parámetros, por ejemplo, al depurar y probar el código. En esta tarea se describe cómo registrar una ejecución completa con todas las variables relevantes. Es más, puede ver información de depuración adicional que por lo general no se imprime en los archivos de registros, incluso en el nivel de depuración.

Para registrar código DFM:

1. Acceda a **Administración de Data Flow > Panel de control de detección**. Haga clic con el botón derecho en el trabajo cuya ejecución debe registrarse y seleccione **Ir a adaptador** para abrir la aplicación Administración de adaptador.
2. Busque el panel **Opciones de ejecución** en la pestaña **Configuración de adaptador**, como se muestra a continuación.



3. Cambie el cuadro **Crear registro de comunicación** a **Siempre**. Para obtener más información acerca de cómo establecer las opciones de registro, consulte "Panel Opciones de ejecución" en HP Universal CMDB – Guía de Administración de Data Flow.

   El ejemplo siguiente es el archivo de registro XML que se creó cuando el trabajo `Host Connection by Shell` se ejecutó y el cuadro **Create communication logs** se estableció en **Siempre** o **Si se produce un error**:

Nombre del trabajo Datos de CI de activación

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

   En el ejemplo siguiente se muestra el mensaje y los parámetros del seguimiento de la pila:

Seguimiento de la pila

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

Bibliotecas y utilidades Jython

En los adaptadores se utilizan con mucha frecuencia varias secuencias de comandos de utilidades. Estas secuencias de comandos forman parte del paquete `AutoDiscovery` y están localizadas en: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** con las demás secuencias de comandos que se han descargado de la sonda.

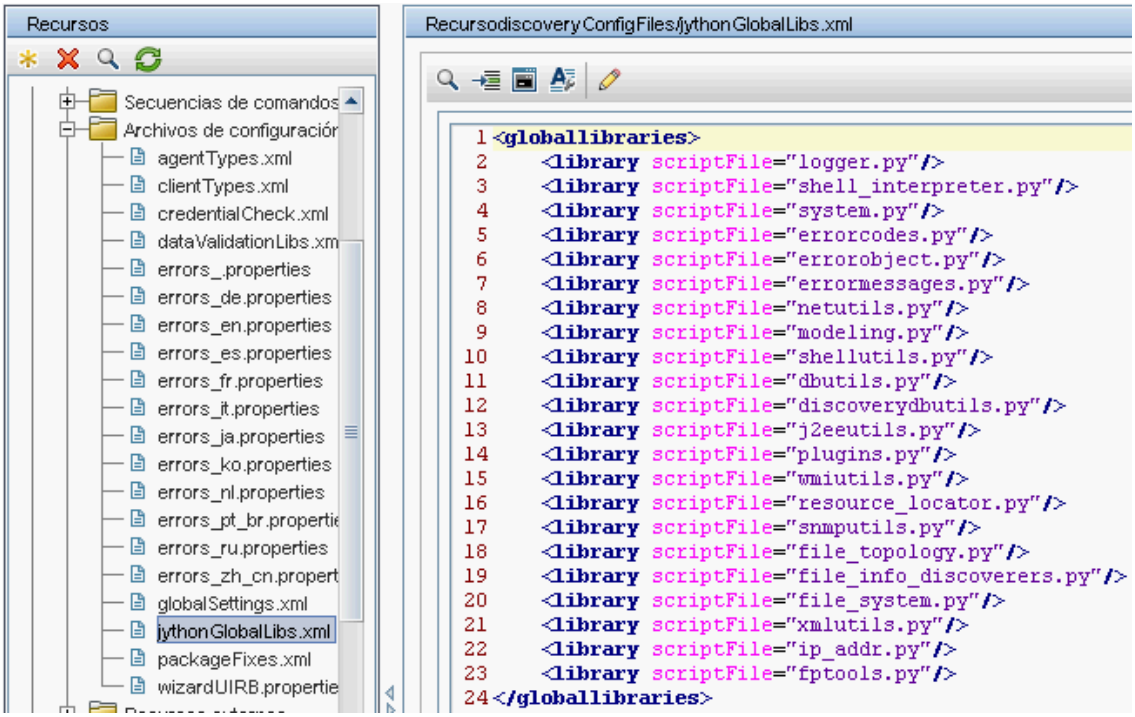
Nota: La carpeta `discoveryScript` se crea dinámicamente cuando la sonda comienza a

trabajar.

Para usar una de las secuencias de comandos de utilidad, agregue la siguiente línea de importación a la sección de importación de la secuencia de comandos:

```
import <nombre de secuencia de comandos>
```

La biblioteca AutoDiscovery Python contiene secuencias de comandos de la utilidad Jython. Estas secuencias de comandos de la biblioteca se consideran una biblioteca externa de DFM. Se definen en el archivo `jythonGlobalLibs.xml` (ubicado en la carpeta **Configuration Files**).



Todas las secuencias de comandos que aparecen en el archivo `jythonGlobalLibs.xml` se cargan de manera predeterminada en el inicio de la sonda, por tanto no es preciso usarlas explícitamente en la definición del adaptador.

Esta sección incluye los siguientes temas:

- "logger.py" abajo
- "modeling.py" en la página siguiente
- "netutils.py" en la página siguiente
- "shellutils.py" en la página 77

logger.py

La secuencia de comandos **logger.py** contiene utilidades de registro y funciones de la aplicación auxiliar para la generación de informes de errores. Puede llamar a sus API de depuración, información y error para escribir en los archivos de registro. Los mensajes de registro se registran en **C:\hp\UCMDB\DataFlowProbe\runtime\log**.

Los mensajes se introducen en el archivo de registro de acuerdo con el nivel de depuración definido por el anexador PATTERNS_DEBUG del archivo **C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties**. (De manera predeterminada, el nivel es `DEBUG`). Para obtener más información, consulte "Niveles de gravedad de errores" en la página 81.

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

Los mensajes de información y error también aparecen en la consola del símbolo del sistema.

Hay dos conjuntos de API:

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

El primer conjunto envía la concatenacion de todos sus argumentos de cadena al nivel de registro apropiado y el segundo conjunto envía la concatenación así como envía el seguimiento de la pila de la última excepción lanzada para proporcionar más información, por ejemplo:

```
logger.debug('encontró el resultado') logger.errorException('Error en
detección')
```

modeling.py

La secuencia de comandos **modeling.py** contiene API para crear hosts, IP, CI de proceso, etc. Estas API permiten la creación de objetos comunes y facilitan la lectura del código. Por ejemplo:

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

netutils.py

La biblioteca **netutils.py** se utiliza para recuperar información de la red y de TCP, como la recuperación de nombres del sistema operativo, comprobación de la validez de la dirección MAC, comprobación de la validad de la dirección IP, etc. Por ejemplo:

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

shellutils.py

La biblioteca **shellutils.py** proporciona una API para ejecutar comandos de shell y recuperar el estado final de un comando ejecutado, y permite la ejecución de varios comandos en función de dicho estado final. La biblioteca se inicializa con un cliente shell y utiliza el cliente para ejecutar comandos y recuperar resultados. Por ejemplo:

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

Capítulo 3

Mensajes de error

Este capítulo incluye:

Información general de los mensajes de error

Durante la detección, se pueden revelar muchos errores, por ejemplo, fallos de conexión, problemas de hardware, excepciones, tiempos de espera agotados, entre otros. DFM muestra estos errores en el panel de control de Discovery, tanto en modo básico como avanzado, siempre que el flujo de detección regular no se realice correctamente. Puede profundizar en el CI de activación que causó el problema para ver el mismo mensaje de error.

DFM diferencia entre errores que se pueden omitir a veces (por ejemplo, un host al que no se puede tener acceso) y errores que hay que resolver (por ejemplo, problemas de credenciales o configuración que falta o archivos DLL). Es más, DFM notifica los errores una vez, incluso si el mismo error tiene lugar en sucesivas ejecuciones, y notifica un error aunque solo tenga lugar una vez.

Al crear un paquete, puede añadir mensajes apropiados como recursos al paquete. Durante el despliegue del paquete, los mensajes también se despliegan en la ubicación correcta. Los mensajes deben cumplir con las convenciones, tal como se describe en "Convenciones para la escritura de errores" abajo.

DFM admite mensajes de error en varios idiomas. Puede traducir los mensajes que escriba para que aparezcan en el idioma local.

Para obtener información detallada sobre la búsqueda de errores, consulte "Panel Estado de detección" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Para obtener más información sobre cómo establecer los registros de comunicación, consulte "Panel Opciones de ejecución" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Convenciones para la escritura de errores

- Los errores se identifican por un código del mensaje de error y una matriz de argumentos (**int, String[]**). Una combinación de un código de mensaje y una matriz de argumentos define un error específico. La matriz de parámetros puede ser un valor nulo.
- Cada código de error se asigna a un **mensaje breve**, que es una cadena fija, y un **mensaje detallado**, que es una cadena de plantilla que contiene cero o más argumentos. Se asume la correspondencia entre el número de argumentos de la plantilla y el número real de parámetros.

Ejemplo de código de mensaje de error:

10234 puede representar un error con el mensaje breve:

```
Error de conexión
```

y el mensaje detallado:

```
No se puede conectar al protocolo {0} debido a un tiempo de espera agotado de {1} mseg
```

donde

{0} = el primer argumento: un nombre de protocolo

{1} = el segundo argumento: la duración del tiempo de espera agotado en mseg

Esta sección incluye también los siguientes temas:

- "Contenido de los archivos de propiedades" abajo
- "Archivo de propiedades de los mensajes de error" abajo
- "Convenciones de nomenclatura de configuración regional" abajo
- "Códigos de mensajes de error" en la página siguiente
- "Errores de contenido sin clasificar" en la página siguiente
- "Cambios en Framework" en la página 81

Contenido de los archivos de propiedades

Un archivo de propiedades debería contener dos claves para cada código de mensaje de error. Por ejemplo, para el error 45:

- **DDM_ERROR_MESSAGE_SHORT_45**. Descripción breve del error.
- **DDM_ERROR_MESSAGE_LONG_45**. Descripción larga del error (puede contener parámetros, por ejemplo, {**0**},{**1**}).

Archivo de propiedades de los mensajes de error

Un archivo de propiedades contiene una asignación entre un código de mensaje de error y dos mensajes (breve y detallado).

Cuando se despliega un archivo de propiedades, sus datos se fusionan con datos existentes, se añaden los nuevos códigos de mensaje mientras que se reemplazan los códigos de mensajes antiguos.

Los archivos de propiedades de infraestructura forman parte del paquete **AutoDiscoveryInfra**.

Convenciones de nomenclatura de configuración regional

- Para la configuración regional predeterminada: **<file name>.properties.errors**
- Para una configuración regional específica: **<file name>_xx.properties.errors**

  donde **xx** es la configuración regional (por ejemplo, **infraerr_fr.properties.errors** o **infraerr_en_us.properties.errors**).

Códigos de mensajes de error

Los siguientes códigos de error se incluyen de manera predeterminada con HP Universal CMDB. Puede agregar sus propios mensajes de error a esta lista.

Nombre de error	Código de error	Descripción
Interno	100-199	Se resuelven sobre todo en excepciones producidas durante las ejecuciones de las secuencias de comandos Jython
Conexión	200-299	Error en la conexión, sin agente en el equipo destino, destino no alcanzable, etc.
Relacionado con credenciales	300-399	Permiso denegado, intento de conexión bloqueado debido a la falta de credenciales
Tiempo de espera	400-499	Tiempo de espera agotado durante la conexión/comando
Comportamiento inesperado o no válido	500-599	Archivos de configuración que faltan, interrupciones inesperadas, etc.
Recuperación de información	600-699	Falta información en equipos de destino, error al solicitar información al agente, etc.
Relacionado con recursos	700-799	Errores relacionados con falta de memoria o clientes que no se han liberado correctamente
Análisis	800-899	Error al analizar texto
Codificación	900	Error en la entrada, codificación no admitida
Relacionado con SQL	901-903, 924	Errores recibidos en operaciones SQL
Relacionado con HTTP	904-909	Errores generados durante las conexiones HTTP, analizados desde códigos de error HTTP.
Específico de aplicación	910-923	Error notificado debido a problemas específicos de la aplicación, por ejemplo, versión LSOF incorrecta, no se encontraron gestores de cola, etc.

Errores de contenido sin clasificar

Para admitir contenido antiguo sin causar una regresión, la aplicación y los métodos relevantes de SDK gestionan errores del código de mensaje 100 (es decir, un error de secuencia de comandos sin clasificar) de manera diferente.

Estos errores no están agrupados (es decir, no se consideran errores del mismo tipo) por su código de mensaje, sino que se agrupan por el contenido del mensaje. Es decir, si una secuencia de

comandos notifica un error por métodos antiguos y obsoletos (con una cadena de mensajes y sin un código de error), todos los mensajes reciben el mismo código de error, pero en la aplicación o en métodos SDK relevantes, los distintos mensajes se muestran como errores diferentes.

Cambios en Framework

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

Los métodos siguientes se agregan a la interfaz:

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

Los siguientes métodos antiguos todavía se admiten por compatibilidad con versiones anteriores, pero se marcan como obsoletos:

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

Niveles de gravedad de errores

Cuando un adaptador finaliza la ejecución en un CI de activación, devuelve un estado. Si no se notifica de ningún error o advertencia, el estado es **Success** (Correcto).

Los niveles de gravedad se enumeran aquí desde el ámbito más restringido al más amplio:

Errores graves

Este nivel notifica errores graves como un problema con la infraestructura, archivos DLL que faltan o excepciones:

- Error al generar la tarea (no se encuentra la sonda, no se encuentran las variables, etc.)
- No es posible ejecutar la secuencia de comandos
- El procesamiento de los resultados produce un error en el servidor y los datos no se escriben en el CMDB

Errores

Este nivel notifica problemas que hacen que DFM no recupere los datos. Revise estos errores ya que por lo general requieren que se tome alguna medida (por ejemplo, aumentar el tiempo de espera, cambiar un intervalo, cambiar un parámetro, agregar otra credencial de usuario, etc.).

- En aquellos casos donde pueda ayudar la intervención del usuario, se informa de un error, ya sea un problema de credenciales o redes que pueda necesitar información adicional. (No son errores de detección sino de configuración).
- Error interno, por lo general debido a un comportamiento inesperado del equipo o aplicación detectado, por ejemplo, archivos de configuración que faltan, entre otros.

Advertencias

Cuando una ejecución se realiza de manera correcta pero pueden surgir problemas no graves que

deberían ser tenidos en cuenta, DFM marca la gravedad como **Advertencia**. Puede examinar esos CI para ver si faltan datos antes de comenzar una sesión de depuración más detallada. La **advertencia** puede incluir mensajes sobre la falta de un agente instalado en un host remoto, o que datos no válidos han causado que un atributo no se calcule correctamente.

- Agente de conexión que falta (SNMP, WMI)
- La detección se realiza correctamente, pero no se detecta toda la información disponible

Capítulo 4

Desarrollo de adaptadores de bases de datos genéricas

Este capítulo incluye:

Información general de los adaptadores de bases de datos genéricas

El propósito de la plataforma de adaptadores de bases de datos genéricas es crear adaptadores que puedan integrarse con sistemas de gestión de bases de datos relacionales (RDBMS) y ejecutar consultas de TQL y trabajos de rellenado en la base de datos. Los RDBMS admitidos por el adaptador de bases de datos genéricas son Oracle, Microsoft SQL Server y MySQL.

Esta versión de la implementación de adaptadores de bases de datos está basada en un estándar JPA (API de persistencia Java) con la biblioteca Hibernate ORM como proveedor de persistencia.

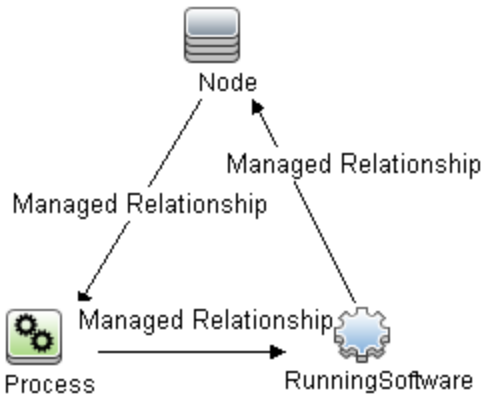
Consultas TQL para el adaptador de bases de datos genéricas

Para los trabajos de rellenado, cada diseño necesario de un CI debe comprobarse en el cuadro de diálogo Configuración de diseño en Modeling Studio. Para obtener más información, consulte "Cuadro de diálogo Propiedades de nodo de consulta/relación" en *HP Universal CMDB – Guía de modelado*. Es importante tener en cuenta que un CI puede necesitar un atributo para ser identificado y sin esos atributos, el CI no se podrá agregar a UCMDB.

Hay las limitaciones siguientes en las consultas TQL calculadas únicamente por el adaptador de bases de datos genéricas:

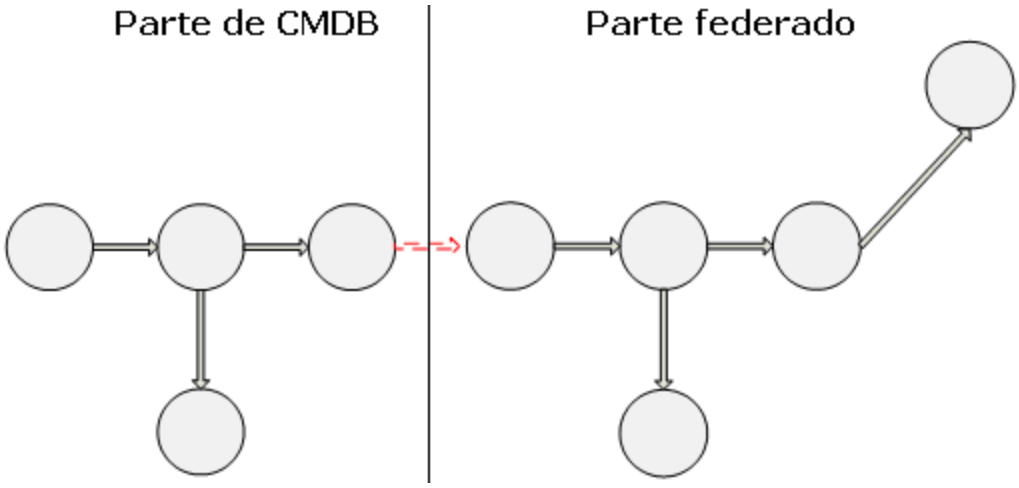
- no se admiten los subgráficos
- no se admiten las relaciones compuestas
- no se admiten los ciclos o partes de ciclos

  La siguiente consulta TQL es un ejemplo de un ciclo:



- No se admite el diseño de funciones.
- No se admite la cardinalidad 0..0.
- No se admite la relación `Join`.

- No se admiten las condiciones del calificador.
- Para conectar dos CI, debe haber una relación en forma de tabla o clave externa en el origen de la base de datos externa.



Reconciliación

La reconciliación se realiza como parte del cálculo de TQL en el adaptador. Para que tenga lugar la reconciliación, el CMDB se asigna a una entidad federada llamada CIT de reconciliación.

Asignación. Cada atributo de CMDB se asigna a una columna en el origen de datos.

Aunque la asignación se realiza directamente, también se admiten las funciones de transformación en los datos asignados. Puede agregar nuevas funciones mediante el código Java (por ejemplo, minúscula, mayúscula). El propósito de estas funciones es habilitar las conversiones de valor (valores almacenados en CMDB en un formato y en la base de datos federada en otro formato).

Nota:

- Para conectar CMDB y el origen de la base de datos externa, debe haber una asociación adecuada en la base de datos. Para obtener más información, consulte "Requisitos previos" en la página 88.
- También se admite la reconciliación con el Id. de CMDB
- También se admite la reconciliación con el Id. global.

Hibernate como proveedor de JPA

Hibernate es una herramienta de asignación relacional de objetos (OR), que permite la asignación de clases Java a tablas en varios tipos de bases de datos relacionales (por ejemplo, Oracle y Microsoft SQL Server). Para obtener más información, consulte "Limitaciones funcionales" en la página 153.

En una asignación elemental, cada clase Java está asignada a una única tabla. Una asignación más avanzada permite la asignación de herencia (como ocurre en la base de datos de CMDB).

Otras funciones admitidas incluyen la asignación de una clase a varias tablas, soporte para recopilaciones y asociaciones de tipos uno-a-uno, uno-a-varios y varios-a-uno. Para obtener más información, consulte "Asociaciones" en la página siguiente a continuación.

En el contexto que nos ocupa, no hay necesidad de crear clases Java. La asignación se define desde los CIT del modelo de clase de CMDB a las tablas de la base de datos.

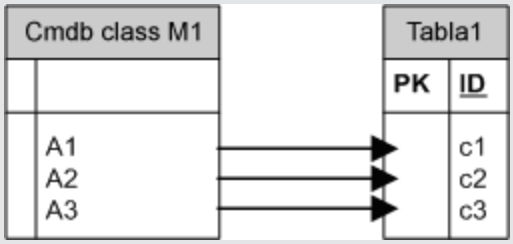
Esta sección incluye también los siguientes temas:

Ejemplos de asignación relacional de objetos

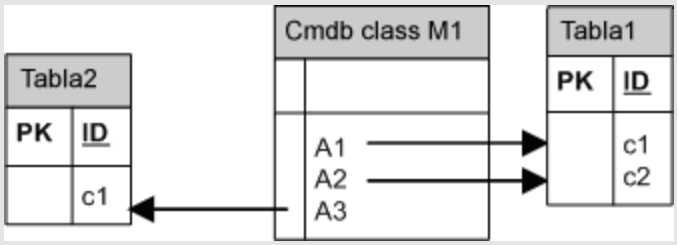
En los ejemplos siguientes se describe la asignación relacional de objetos:

Ejemplo de 1 clase de CMDB asignada a una tabla de la base de datos:

La clase `M1`, con atributos `A1`, `A2` y `A3`, se asigna a la tabla `1` columnas `c1`, `c2` y `c3`. Esto significa que cualquier instancia de `M1` tiene una fila correspondiente en la tabla `1`.

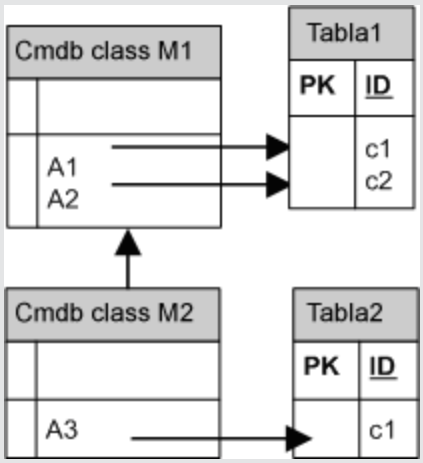


Ejemplo de 1 clase de CMDB asignada a dos tablas de la base de datos:



Ejemplo de herencia:

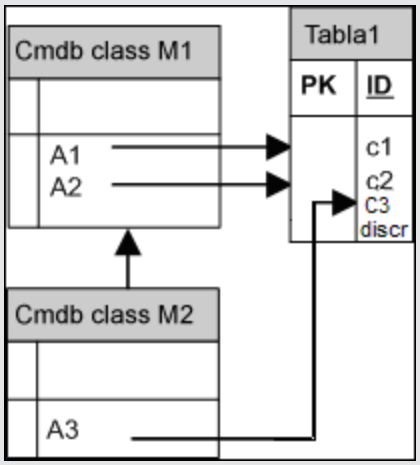
Este caso se utiliza en CMDB, donde cada clase tiene su propia tabla de base de datos.



Ejemplo de herencia de una única tabla con discriminador:

Una jerarquía completa de clases se asigna a una única tabla de la base de datos, cuyas columnas constan de un superconjunto de todos los atributos de las clases asignadas. La tabla también contiene una columna adicional (`Discriminator`), cuyo valor indica qué clase específica debería asignarse a esta entrada.

Cuando utiliza las capacidades del discriminador, no puede omitir una clase en la jerarquía; es decir, como C3 hereda de C2 y C2 hereda de C1, no puede definir solo C1 y C3, debe definir las tres clases.



Asociaciones

Hay tres tipos de asociaciones: uno-a-varios, varios-a-uno y varios-a-varios. Para realizar la conexión entre los distintos objetos de la base de datos, debe definirse una de estas asociaciones usando una columna de clave externa (para el caso uno-a-varios) o una tabla de asignación (para el caso varios-a-varios).

Facilidad de uso

Como el esquema JPA es muy amplio, se incluye un archivo XML simplificado para facilitar las definiciones.

El caso de uso para utilizar este archivo XML es como sigue: Los datos federados se modelan en una clase federada. Esta clase tiene relaciones varios-a-uno en una clase de CMDB no federada. Además, solo hay un posible tipo de relación entre la clase federada y la clase no federada.

Preparar la creación del adaptador

Esta tarea describe las preparaciones que son necesarias para crear un adaptador.

Nota: Puede ver ejemplos del adaptador de bases de datos genéricas en la API de UCMDB. Específicamente, la muestra del adaptador DDMi contiene un archivo **orm.xml** complicado, así como implementaciones para algunas interfaces de complementos.

Esta tarea incluye los siguientes pasos:

- "Requisitos previos" abajo
- "Crear un tipo de CI" en la página 90
- "Crear una relación" en la página 90

1. ## Requisitos previos

   Para validar que puede utilizar el adaptador de la base de datos con su base de datos, compruebe lo siguiente:

   - Las clases de reconciliación y sus atributos (también conocidos como multinodos) están en la base de datos. Por ejemplo, si la reconciliación se ejecuta por nombre de nodo, compruebe que hay una tabla que contiene una columna con nombres de nodos. Si la reconciliación se ejecuta de acuerdo con el nodo `cmdb_id`, compruebe que hay una columna con unos Id. de CMDB que coincidan con los Id. de CMDB de los nodos de CMDB. Para más información sobre la reconciliación, consulte "Reconciliación " en la página 85.

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

   - Para correlacionar dos CIT con una relación, debe haber datos de correlación entre las tablas de CIT. La correlación puede ser por columna de clave externa o por tabla de asignación. Por ejemplo, para establecer una correlación entre el nodo y el ticket, tiene que

haber una tabla de tickets que contenga el Id. del nodo, una columna en la tabla de nodos con el Id. de ticket que está conectado a él o una tabla de asignación cuyo `end1` sea el Id. de nodo y `end2` sea el Id. de ticket. Para obtener más información sobre los datos de correlación, consulte "Hibernate como proveedor de JPA " en la página 85.

En la tabla siguiente se muestra la columna `NODE_ID` de la clave externa:

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	Controladora de bus serie	Intel 82801EB USB Universal Host Controller
3581	2	Sistema	Intel 631xESB/6321ESB/3100 Chipset LPC
3581	3	Display (Visualización)	ATI ES1000
3581	4	Periférico del sistema base	HP ProLiant iLO 2 Legacy Support Function

- Cada CIT se puede asignar a una o más tablas. Para asignar un CIT a más de una tabla, compruebe que hay una tabla principal cuyas claves principales existen en las otras tablas, y que hay una única columna de valor.

  Por ejemplo, un ticket se asigna a dos tablas: `ticket1` y `ticket2`. La primera tabla tiene las columnas `c1` y `c2` y la segunda tabla tiene las columnas `c3` y `c4`. Para permitir que se consideren como una tabla, ambos deben tener la misma clave principal. Además, la primera clave de la tabla principal puede ser una columna de la segunda tabla.

  En el ejemplo siguiente, las tablas comparten la misma clave principal denominada `CARD_ID`:

CARD_ID	CARD_TYPE	CARD_NAME
1	Controladora de bus serie	Intel 82801EB USB Universal Host Controller
2	Sistema	Intel 631xESB/6321ESB/3100 Chipset LPC
3	Display (Visualización)	ATI ES1000
4	Periférico del sistema base	HP ProLiant iLO 2 Legacy Support Function

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(Controlador de host USB estándar)

CARD_ID	CARD_VENDOR
3	Hewlett-Packard Company
4	(Dispositivos del sistema estándar)
5	Hewlett-Packard Company

2. ## Crear un tipo de CI

   En este paso, va a crear un CIT que representa los datos del RDBMS (el origen de datos externo).

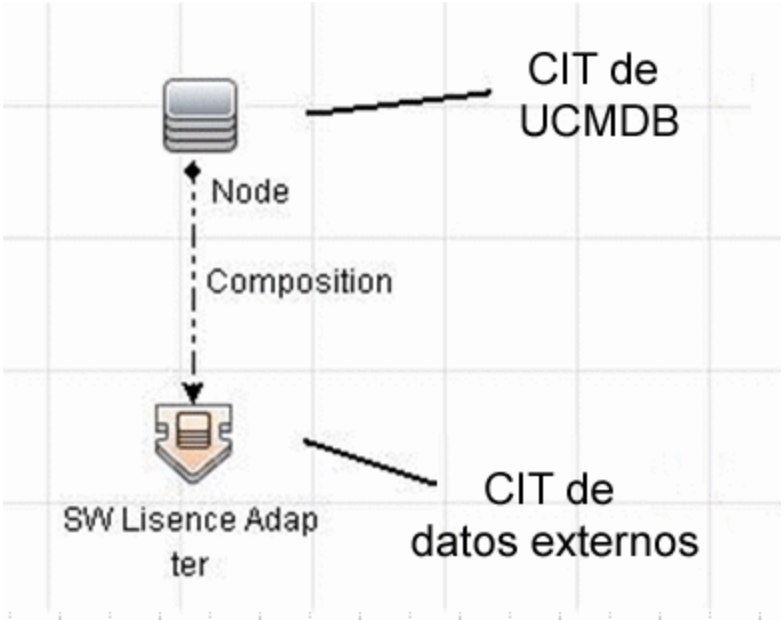
   a. En UCMDB, acceda al Administrador de tipos de CI y cree un nuevo tipo de CI. Para obtener más información, consulte "Crear un tipo de CI" en la página 1 en *HP Universal CMDB – Guía de modelado*.

   b. Agregue los atributos necesarios al CIT, como la hora del último acceso, proveedor, etc. Son los atributos que el adaptador recuperará del origen de datos externo e introducirá en las vistas de CMDB.

3. ## Crear una relación

   En este paso, agregará una relación entre el CIT de UCMDB y el nuevo CIT que representa los datos que se van a federar desde el origen de datos externo.

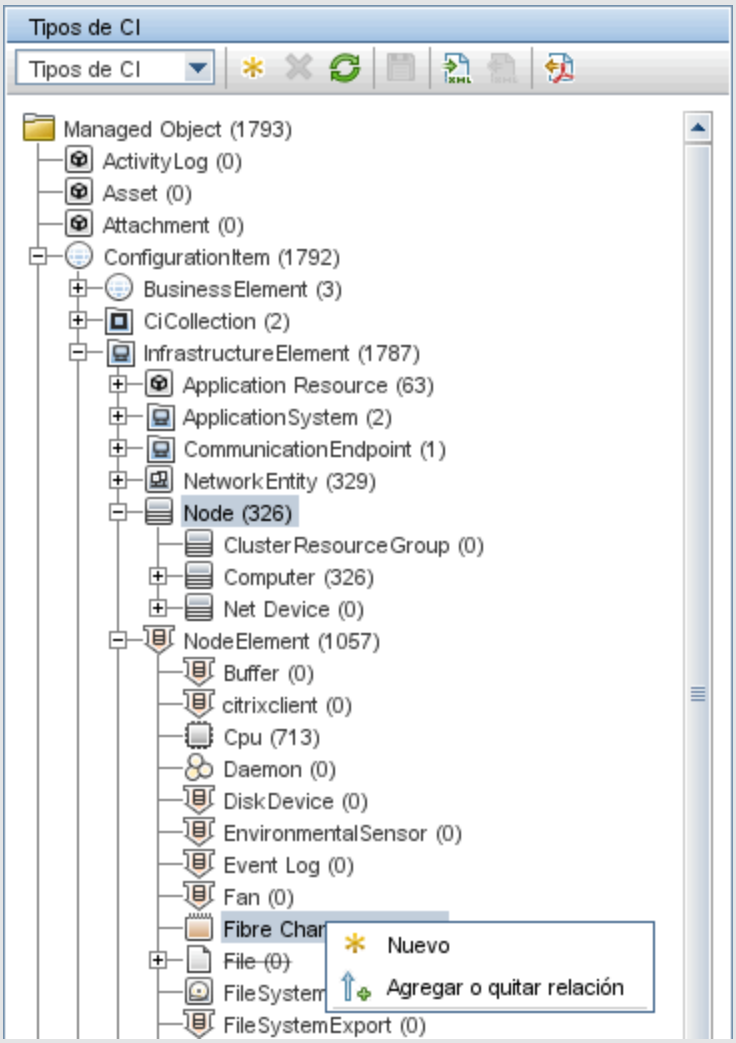
   Agregue relaciones adecuadas y válidas al nuevo CIT. Para obtener más información, consulte "Cuadro de diálogo Agregar o quitar relación" en la página 1 en *HP Universal CMDB – Guía de modelado*.



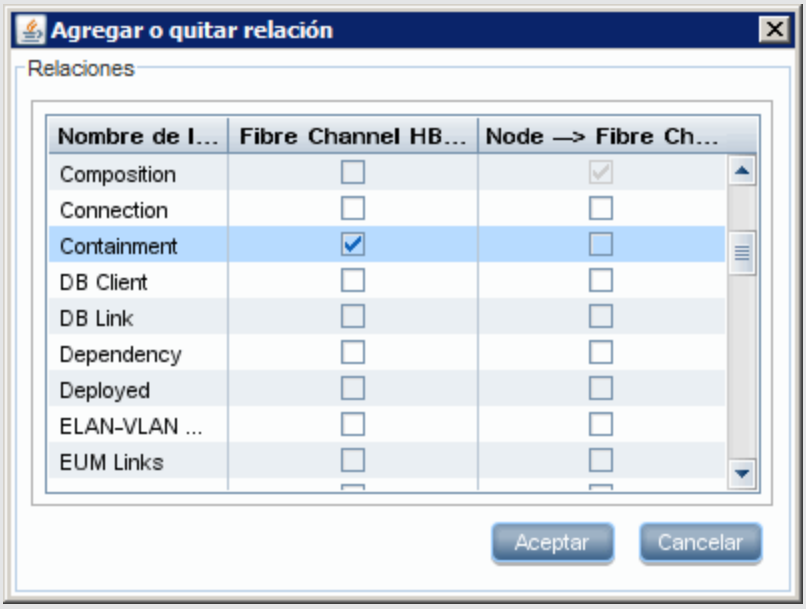
Nota: En esta etapa, todavía no puede ver los datos federados ni rellenar los datos externos, ya que todavía no ha definido el método para introducir los datos.

Ejemplo de la creación de una relación de contención:

a. En el Administrador de CIT, seleccione los dos CIT:



b. Cree una relación **Containment** entre los dos CIT:



Preparación del paquete de adaptadores

En este paso, localizará y configurará el paquete del adaptador de bases de datos genéricas.

1. Localice el paquete **db-adapter.zip** en la carpeta **C:\hp\UCMDB\UCMDBServer\content\adapters**
   .
2. Extraiga el paquete en un directorio temporal local.
3. Edite el archivo XML del adaptador:
   - Abra el archivo **discoveryPatterns\db_adapter.xml** en un editor de texto.
   - Localice el atributo **adapter id** y reemplace el nombre:

     ```
     <pattern id="MyAdapter" displayLabel="My Adapter"
     xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
     description="Descripción de patrón de detección"
     schemaVersion="9.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
     instance" displayName="UCMDB API Population">
     ```

     Si el adaptador admite el rellenado, debería agregarse la capacidad siguiente al elemento **<adapter-capabilities>**:

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

La etiqueta o ID del monitor en la lista de adaptadores en el panel Punto de integración de HP Universal CMDB.

Al crear un adaptador de BD genérico, no es necesario editar la etiqueta **changes-source** en la etiqueta **support-replicatioin-data**. Si se implementa el complemento **FcmdbPluginForSyncGetChangesTopology**, se devolverá la topología modificada de la última ejecución. Si no se implementa el complemento, se devolverá la topología completa y se realizará la eliminación automática según los CI devueltos.

Para obtener más información sobre cómo rellenar CMDB con datos, consulte "Página de Integration Studio " de *HP Universal CMDB – Guía de Administración de Data Flow*.

- Si el adaptador está utilizando el motor de asignación desde la versión 8.x (lo que significa que no está usando el nuevo motor de asignación de reconciliación), reemplace el siguiente elemento:

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

con:

```
<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>
```

- Localice la definición **category**:

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

Cambie el nombre de categoría **Generic** a la categoría que prefiera.

> **Nota:** Los adaptadores cuyas categorías se especifican como **Generic** no se enumeran en Integration Studio cuando crea un nuevo punto de integración.

- La conexión con la base de datos puede describirse mediante un nombre de usuario (esquema), contraseña, tipo de base de datos, nombre de máquina host de base de datos y nombre de base de datos o SID.

  Para este tipo de conexión, los parámetros tienen los elementos siguientes en la sección **parameter** del archivo XML del adaptador:

```
<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>
```

Nota: Esta es la configuración predeterminada. Por consiguiente, el archivo **db_ adapter.xml** ya contiene esta definición.

Hay situaciones en las que la conexión con la base de datos no puede configurarse de esta manera. Por ejemplo, la conexión con Oracle RAC o la conexión mediante un controlador de base de datos distinto del proporcionado con CMDB.

Para estas situaciones, puede describir la conexión utilizando el nombre de usuario (esquema), la contraseña y una cadena de URL de conexión.

Para definir esto, edite la sección de parámetros XML del adaptador de la manera siguiente:

```
<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>
```

Un ejemplo de una dirección URL que se conecta a un Oracle RAC mediante el controlador Data Direct con la configuración de fábrica es: **jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true**.

4. En el directorio temporal, abra la carpeta **adapterCode** y cambie el nombre de **GenericDBAdapter** al valor de **adapter id** que se usó en el paso anterior.

   Esta carpeta contiene los archivos jar que ejecutan la lógica de federación, por ejemplo, el nombre del adaptador, la consulta y las clases en CMDB y los campos de RDBMS que admite el adaptador.

5. Configure el adaptador como se precise. Para obtener más información, consulte "Configuración del adaptador - Método mínimo" abajo.

6. Cree un archivo `*.zip` con el mismo nombre que haya dado al atributo **adapter id**, tal como se ha descrito en el paso "Edite el archivo XML del adaptador:" en la página 92.

   Nota: El archivo **descriptor.xml** es un archivo predeterminado que existe en cada paquete.

7. Guarde el nuevo paquete que ha creado en el paso anterior. El directorio predeterminado para los adaptadores es: **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Configuración del adaptador - Método mínimo

El procedimiento siguiente describe un método de asignar el modelo de clase de CMDB a un RDBMS.

Estos archivos de configuración se encuentran en el paquete **db-adapter.zip** de la carpeta **C:\hp\UCMDB\UCMDBServer\content\adapters** que extrajo en el paso "Extraiga el paquete en un directorio temporal local." en la página 92 en "Preparación del paquete de adaptadores" en la página 92.

Nota: El archivo **orm.xml** que se ha generado automáticamente como resultado de ejecutar este método es un buen ejemplo que se puede usar al trabajar con el método avanzado.

Debe usar este método mínimo cuando tenga que:

- Federar/rellenar un único nodo, como un atributo de nodo.
- Mostrar las capacidades del adaptador de bases de datos genéricas.

Este método:

- admite solo la federación\rellenado de un nodo
- admite solo relaciones virtuales varios-a-uno

Esta tarea incluye los siguientes pasos:

- "Configurar el archivo adapter.conf" abajo
- "Configurar el archivo simplifiedConfiguration.xml" en la página siguiente

Configurar el archivo adapter.conf

En este paso, puede cambiar los ajustes del archivo `adapter.conf` para que el adaptador utilice el método de configuración simplificado.

1. Abra el archivo **adapter.conf** en un editor de texto.
2. Localice la siguiente línea: **use.simplified.xml.config=<true/false>**.
3. Cámbielo a **use.simplified.xml.config=true**.

Configurar el archivo simplifiedConfiguration.xml

En este paso, se configura el archivo **simplifiedConfiguration.xml** asignando el CIT de CMDB a los campos de la tabla del RDBMS.

1. Abra el archivo **simplifiedConfiguration.xml** en un editor de texto.

   Este archivo incluye una plantilla que se utiliza para cada entidad que se va a asignar.

   Nota: No edite el archivo **simplifiedConfiguration.xml** en ninguna versión del Bloc de notas de Microsoft Corporation. Utilice Notepad++, UltraEdit o cualquier otro editor de texto de otros fabricantes.

2. Haga cambios en los siguientes atributos:

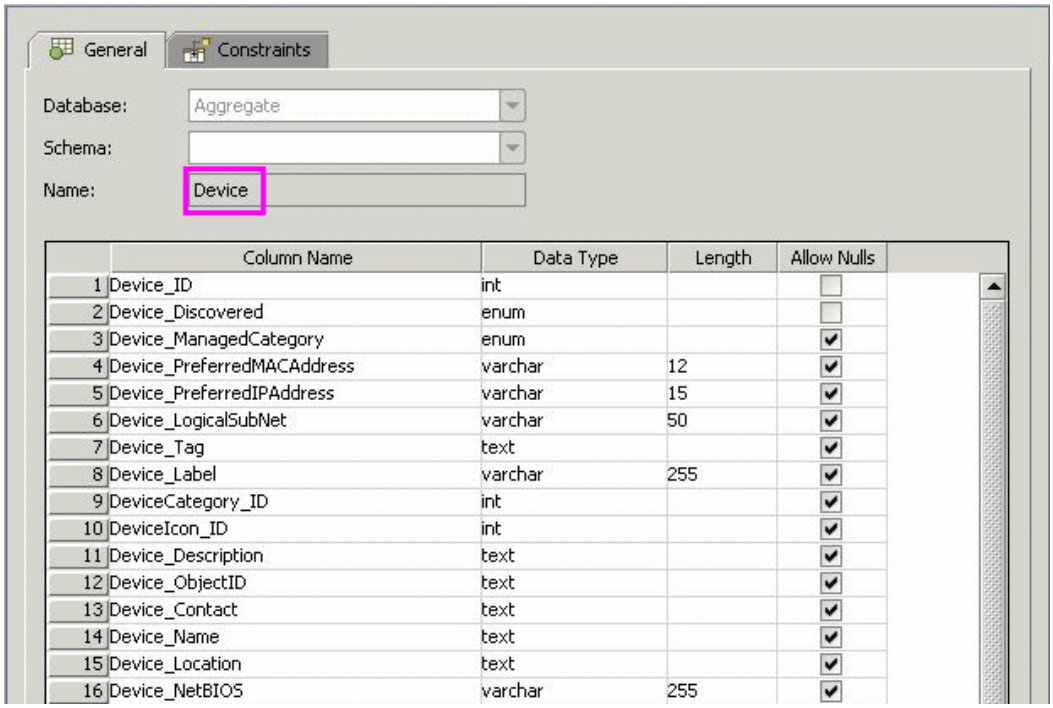
   - El nombre de CIT de UCMDB (cmdb-class-name) y el nombre de tabla correspondiente en el RDBMS (default-table-name):

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

     El atributo `cmdb-class-name` se toma del CIT node:

     El atributo `default-table-name` se toma de la tabla Device:

- El identificador único del RDBMS:

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

> **Nota:** la clave principal es equivalente al Id. de entidad en el archivo orm.xml.

- La regla de reconciliación (reconciliation-by-two-nodes):

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

- El atributo de reconciliación de UCMDB (`cmdb-attribute-name`) y del RDBMS (`column-name`):

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

- El nombre del CIT (`cmdb-class-name`) y el nombre de la tabla correspondiente en el RDBMS (`default-table-name`). Además, la relación de CMDB (`connected-cmdb-class-name`) y la relación del CIT (`link-class-name`):

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

- La clave principal y la clave externa:

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

- El identificador único del RDBMS:

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

- La asignación entre el atributo CMDB (`cmdb-attribute-name`) y el nombre de columna del RDBMS (`column-name`):

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

3. Guarde el archivo.

Ejemplo: Rellenado de un nodo y dirección IP utilizando el método simplificado

Este ejemplo muestra cómo rellenar un **Nodo** relacionado por un vínculo de tipo Containment con **Dirección IP** en UCMDB. RDBMS tiene una tabla denominada **simpleNode** que contiene datos en el nombre, el nodo y la dirección IP del equipo.

El contenido de la tabla **simpleNode** se muestra a continuación:

	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

El rellenado se realiza en tres fases, de la forma siguiente:

- "Crear simplifiedConfiguration.xml" abajo
- " Crear el TQL" en la página siguiente
- "Crear un punto de integración" en la página 100

Crear simplifiedConfiguration.xml

Cree **simplifiedConfiguration.xml** de la forma siguiente:

1. Cree una entidad **cmdb-class** de la forma siguiente:

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

   El tipo de CI es **node** y el nombre de tabla de RDBMS es **simpleNode**.

2. Establezca primary-key de la tabla de la forma siguiente:

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

   Esta clave principal es equivalente al Id. de entidad en el archivo **orm.xml**.

3. Establezca la regla **reconciliation-by-two-nodes** de la forma siguiente:

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

   Esta etiqueta define la relación entre los tipos de CI de **Node** e **IpAddress**. El tipo de relación es el vínculo de tipo Containment. Los dos tipos de CI conectados realizan la reconciliación. La asignación de atributos del nodo conectado (en este caso, IpAddress) se define en el atributo **connected-node**.

4. Agregue la condición **or** entre los atributos de reconciliación de la forma siguiente:

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

   Esta etiqueta define una relación OR entre los atributos de reconciliación, lo que significa que el primer atributo de reconciliación que sea **true** establecerá la regla de reconciliación entera en **true**.

5. Agregue los siguientes atributos:

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

   Esta etiqueta establece una asignación entre **node.name** en UCMDB y la columna **host_name** de la tabla **simpleNode**.

   Haga lo mismo con el atributo **data_note**:

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

Agregue el atributo de nodo conectado:

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

Esta etiqueta establece una asignación entre **ip_address.name** y la columna **ip_address** de la tabla **simpleNode**.

6. Cierre la etiqueta abierta por orden:

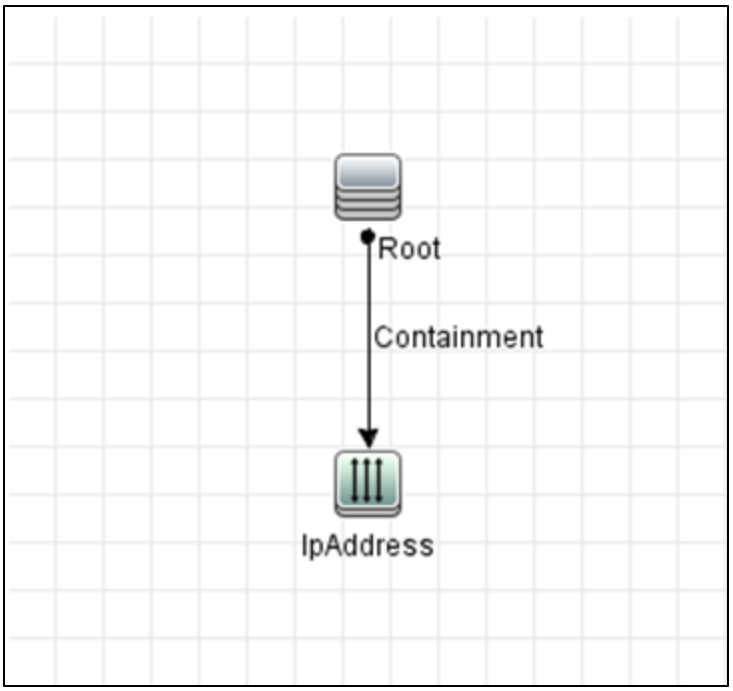
```
</or>

</reconciliation-by-two-nodes>

</cmdb-class>
```

Crear el TQL

El TQL será un **node** conectado por un vínculo de tipo Containment a **ip_address**. El nodo deberá estar marcado como **root**, como se muestra a continuación.



Para crear el TQL:

1. Vaya a Modelado > Modeling Studio.
2. Haga clic en el botón Nuevo y cree una consulta nueva.
3. Vaya a la ficha Tipos de CI y arrastre el tipo de CI Node y el tipo de CI IpAddress a la pantalla TQL.
4. Conecte **Node** e **IpAddress** con un vínculo de tipo Containment.

5. Haga clic con el botón derecho en el elemento **Node** y seleccione Propiedades del nodo de consulta.
6. Cambie **Nombre del elemento** a **Root**.
7. Vaya a la ficha **Diseño de elementos**. En **Condición de atributo**, seleccione **Atributos específicos**. Seleccione **Nombre** y **Nota** de la ventana Atributos disponibles y muévalos a la ventana Atributos específicos.
8. Haga clic con el botón derecho en el elemento **IpAddress** y seleccione Propiedades del nodo de consulta.
9. Vaya a la ficha **Diseño de elementos**. En **Condición de atributo**, seleccione **Atributos específicos**. Seleccione **Nombre** de la ventana Atributos disponibles y muévalo a la ventana Atributos específicos.
10. Guarde el TQL.

Crear un punto de integración

Cree el punto de integración de la forma siguiente:

1. Vaya a Administración de Data Flow > Integration Studio, y haga clic en el botón **Nuevo punto de integración**.
2. Inserte los detalles del punto de integración y haga clic en Aceptar.
3. En la ficha Rellenado, seleccione el botón **Nuevo trabajo de integración** y agregue el TQL creado previamente.
4. Guarde el punto de integración y haga clic en el botón Ejecutar sincronización completa.

Configuración del adaptador - Método avanzado

Estos archivos de configuración se encuentran en el paquete **db-adapter.zip** de la carpeta **C:\hp\UCMDB\UCMDBServer\content\adapters** que extrajo en el paso "Extraiga el paquete en un directorio temporal local." en la página 92 en "Preparación del paquete de adaptadores" en la página 92.

Esta tarea incluye los siguientes pasos:

- "Configuración del archivo orm.xml" abajo
- "Configurar el archivo reconciliation_types.txt" en la página 104
- "Configurar el archivo reconciliation_rules.txt " en la página 104

Configuración del archivo orm.xml

En este paso, asigna los CIT y las relaciones de CMDB a las tablas del RDBMS.

1. Abra el archivo **orm.xml** en un editor de texto.

   De manera predeterminada, este archivo contiene una plantilla que puede usar para asignar tantos CIT y relaciones como sea necesario.

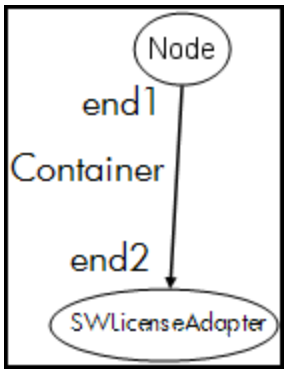
   Nota: No edite el archivo **orm.xml** en ninguna versión del Bloc de notas de Microsoft Corporation. Utilice Notepad++, UltraEdit o cualquier otro editor de texto de otros

fabricantes.

2. Realice los cambios en el archivo de acuerdo con las entidades de datos que se van a asignar. Para obtener más información, consulte los ejemplos siguientes.

   En el archivo **orm.xml** se pueden asignar los tipos de relaciones siguientes:

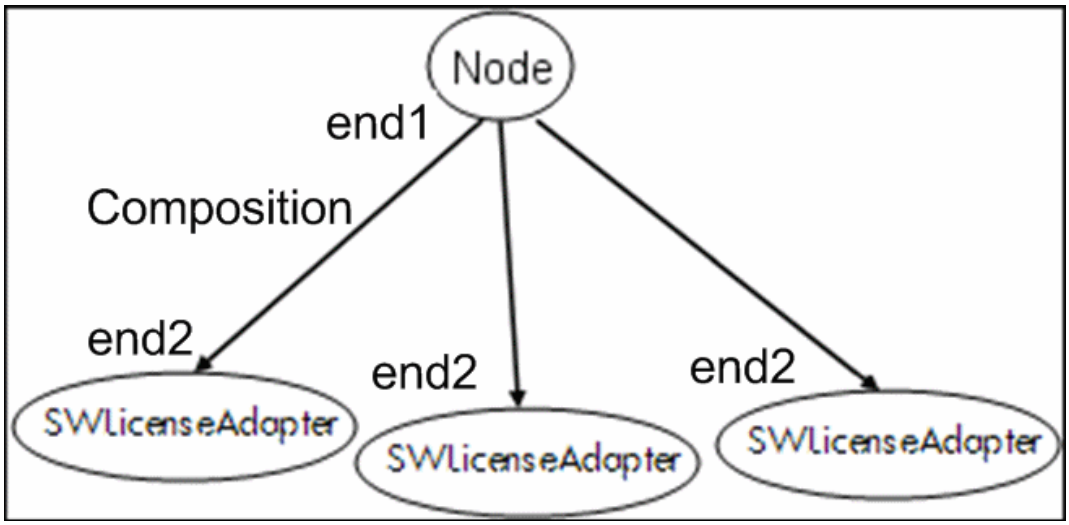
   - Uno a uno (One-to-one):



   El código para este tipo de relación es:

```
<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>
```

   - Varios a uno (Many to one):

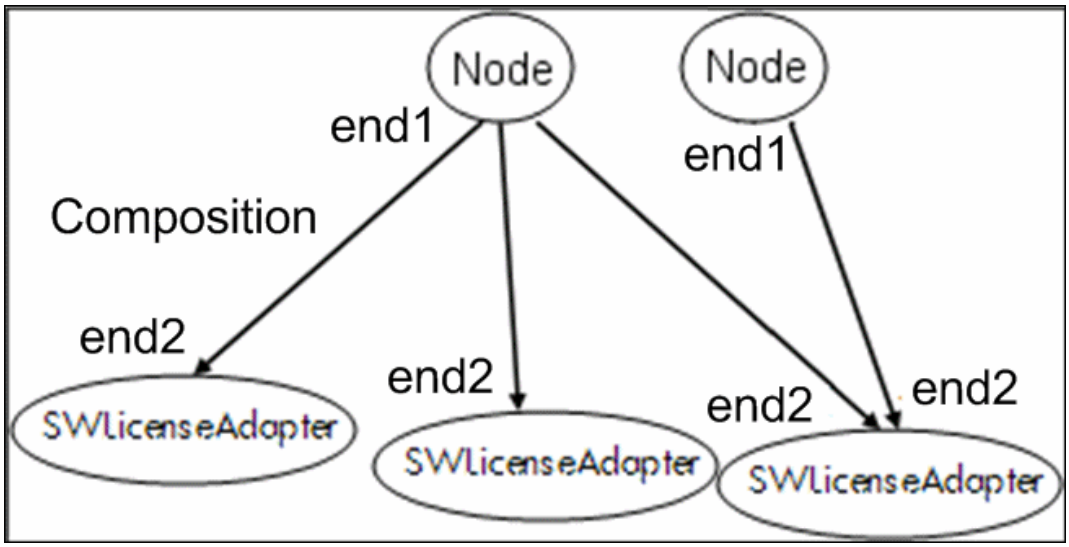


   El código para este tipo de relación es:

```
<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>
```

- Varios a varios (Many to many):



El código para este tipo de relación es:

```
<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>
```

Para obtener más información sobre las convenciones de nomenclatura, consulte "Convenciones de nomenclatura" en la página 124.

Ejemplo de asignación de entidades entre el modelo de datos y el RDBMS:

Nota: Los atributos que no tienen que ser configurados se omiten en los ejemplos siguientes.

- La clase del CIT de CMDB:

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

- El nombre de la tabla del RDBMS:

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

- El nombre de columna del identificador único de la tabla RDBMS:

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

- El nombre del atributo del CIT de CMDB:

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

- El nombre del campo de la tabla del origen de datos externo:

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

- El nombre del nuevo CIT que ha creado en "Crear un tipo de CI" en la página 90:

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

- El nombre de la tabla correspondiente del RDBMS:

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

- La identidad única del RDBMS:

- El nombre de atributo del CIT de CMDB y el nombre del atributo correspondiente del RDBMS:

Ejemplo de asignación de relaciones entre el modelo de datos y el RDBMS:

- La clase de la relación de CMDB:

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

- El nombre de la tabla de RDBMS donde se realiza la relación:

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

- El Id. único del 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>
```

- El tipo de relación y el CIT de CMDB:

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

- Los campos de clave principal y la clave externa del RDBMS:

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

Configurar el archivo reconciliation_types.txt

Abra el archivo **reconciliation_types.txt** en un editor de texto.

Para obtener más información, consulte "Archivo reconciliation_types.txt" en la página 133.

Configurar el archivo reconciliation_rules.txt

En este paso puede definir las reglas por las que el adaptador reconcilia el CMDB y el RDBMS (solo si se utiliza el motor de asignación, por compatibilidad con la versión 8.x):

1. Abra **META-INF\reconciliation_rules.txt** en un editor de texto.
2. Realice los cambios en el archivo de acuerdo con el CIT que está asignando. Por ejemplo, para asignar un CIT de nodo, utilice la siguiente expresión:

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

   Nota:

   - Si los datos de la base de datos distinguen mayúsculas de minúsculas, no elimine el carácter de control (**^**).
   - Compruebe que cada corchete de apertura tiene un corchete de cierre correspondiente.

Para obtener más información, consulte "Archivo reconciliation_rules.txt (para compatibilidad con versiones anteriores)" en la página 133.

Implementación de un complemento

Esta tarea describe cómo implementar y desplegar un adaptador de bases de datos genéricas con complementos.

Nota: Antes de escribir un complemento para un adaptador, asegúrese de que ha completado todos los pasos necesarios de "Preparación del paquete de adaptadores" en la página 92.

1. Copie los siguientes archivos jar desde el directorio de instalación del servidor UCMDB en la ruta de clase de desarrollo:
   - Copie el archivo **db-interfaces.jar** y el archivo **db-interfaces-javadoc.jar** de la carpeta **tools\adapter-dev-kit\db-adapter-framework**.
   - Copie el archivo **federation-api.jar** y el archivo **federation-api-javadoc.jar** de la carpeta

\tools\adapter-dev-kit\SampleAdapters\production-lib.

> **Nota:** En los archivos **db-interfaces-javadoc.jar** y **federation-api-javadoc.jar** y en la documentación en línea podrá encontrar más información sobre cómo desarrollar un complemento:
>
> - **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. Escriba una clase Java que implemente la interfaz Java del complemento. Las interfaces se definen en el archivo **db-interfaces.jar**. La tabla siguiente especifica la interfaz que debe implementarse para cada complemento:

Tipo de complemento	Nombre de interfaz	Método
Sincronizar la topología completa	FcmdbPluginForSyncGetFullTopology	getFullTopology
Sincronizar cambios	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Sincronizar diseño	FcmdbPluginForSyncGetLayout	getLayout
Recuperar consultas admitidas	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Modificar la definición de consulta TQL y los resultados	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Modificar peticiones de diseño para CI	FcmdbPluginGetCIsLayout	getCisLayout
Modificar peticiones de diseño para vínculos	FcmdbPluginGetRelationsLayout	getRelationsLayout
Id. de devolución	FcmdbPluginPushBackIds	getPushBackIdsSQL

La clase del complemento debe tener un constructor público predeterminado. Además, todas las interfaces muestran un método llamado initPlugin. Este método se llama antes de cualquier

otro método y se utiliza para inicializar al adaptador con el objeto de entorno del adaptador continente.

Si se implementa **FcmdbPluginForSyncGetChangesTopology**, hay dos maneras diferentes de informar de los cambios:

- **Informar de la topología raíz entera en todo momento.** Según esta topología, la función de eliminación automática encuentra los CI que deben quitarse. En este caso, la función de eliminación automática debe habilitarse utilizando lo siguiente:

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

- **Informar de cada instancia de CI que se ha quitado/actualizado.** En este caso, la función de eliminación automática debe habilitarse utilizando lo siguiente:

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

3. Asegúrese de que dispone de los archivos JAR del SDK de Federación y los JAR del adaptador de bases de datos genéricas antes de compilar el código Java. El SDK de Federación es el archivo **federation_api.jar**, que se puede encontrar en el directorio **C:\hp\UCMDB\UCMDBServer\lib**

4. Empaquete su clase en un archivo jar y colóquela en la carpeta adapterCode\<Nombre adaptador> en el paquete del adaptador, antes de desplegarla.

   Los complementos se configuran con el archivo **plugins.txt**, situado en la carpeta **\META-INF** del adaptador.

   A continuación se muestra un ejemplo del archivo desde el adaptador DDMi:

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

```
# internal not mandatory plugin to change layout request and
relations result
[getRelationsLayout]
# internal not mandatory plugin to change action on pushBackIds
[pushBackIds]
```

Leyenda:

#: línea de comentarios.

[<Tipo de adaptador>]: comienzo de la sección de definición para un tipo de adaptador específico.

Puede ser una línea vacía debajo de cada [<Tipo de adaptador>], lo que significa que no hay ninguna clase de complemento asociado, o se puede mostrar el nombre completo de la clase de complemento.

5. Empaquete el adaptador con el nuevo archivo jar y el archivo **plugins.xml** actualizado. El resto de los archivos del paquete debería ser el mismo que en cualquier adaptador basado en el adaptador de bases de datos genéricas.

Despliegue del adaptador

1. En UCMDB, acceda al Administrador de paquetes. Para obtener más información, consulte "Página Administrador de paquetes" en *HP Universal CMDB – Guía de administración*.
2. Pulse el icono **Desplegar paquetes en servidor (desde un disco local)** y busque el paquete del adaptador. Seleccione el paquete y haga clic en **Abrir**; a continuación, haga clic en **Desplegar** para mostrar el paquete en el Administrador de paquetes.
3. Seleccione el paquete de la lista y haga clic en el icono **Ver recursos del paquete** para comprobar que el contenido del paquete lo reconoce el Administrador de paquetes.

Edición del adaptador

Cuando haya creado y desplegado el adaptador, puede editarlo en UCMDB. Para obtener más información, consulte "Adapter Management" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Crear un punto de integración

En este paso, comprobará que la federación funciona. Es decir, que la conexión es válida y el archivo XML es válido. Sin embargo, esta comprobación no verifica que el XML se está asignando a los campos correctos del RDBMS.

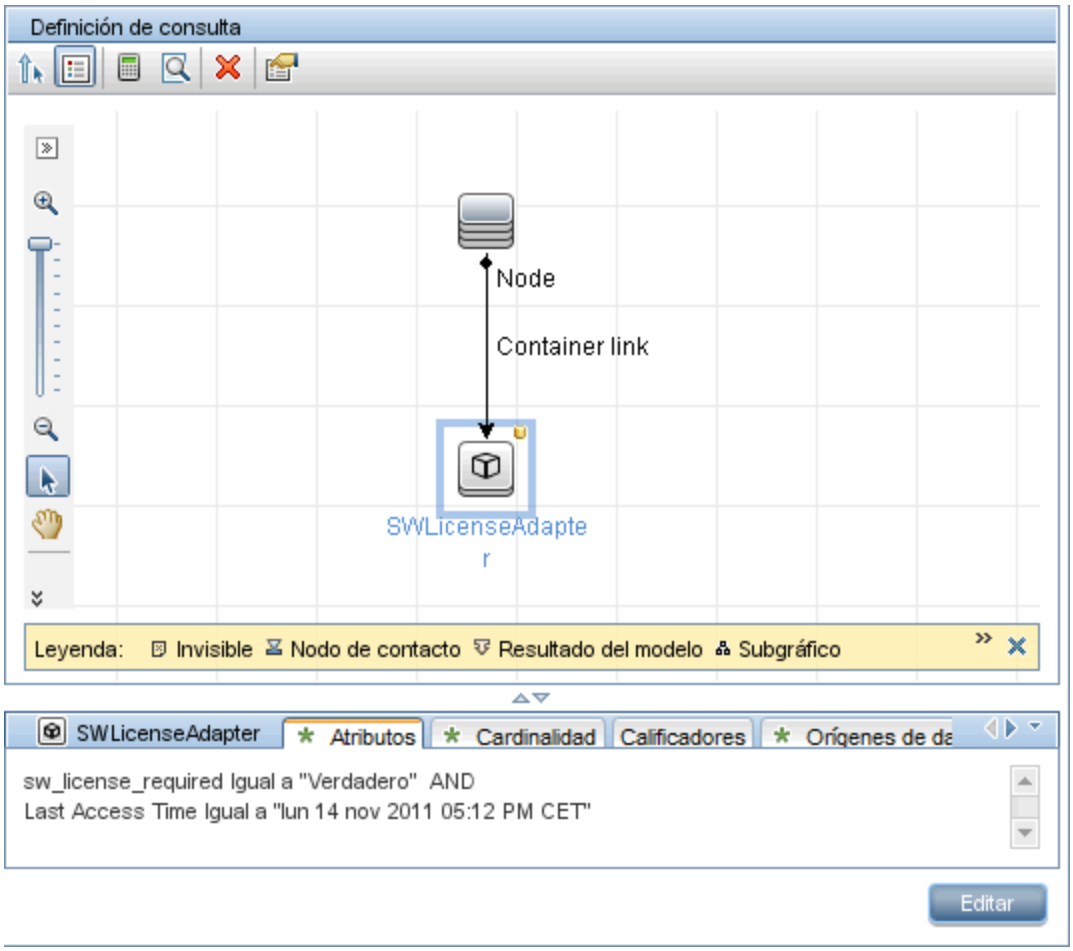
1. En UCMDB, acceda a Integration Studio (**Administración de Data Flow > Integration Studio**).
2. Cree un punto de integración. Para obtener más información, consulte "Cuadro de diálogo Nuevo punto de integración/Editar punto de integración" en *HP Universal CMDB – Guía de Administración de Data Flow*.

La pestaña Federación muestra todos los CIT que se pueden federar usando este punto de integración. Para obtener más información, consulte "Ficha Federación" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Creación de una vista

En este paso va a crear una vista que le permitirá ver instancias del CIT.

1. En UCMDB, acceda a Modeling Studio (**Modelado > Modeling Studio**).
2. Cree una vista. Para obtener más información, consulte "Crear una vista de patrón" en *HP Universal CMDB – Guía de modelado*.
3. Puede agregar condiciones a TQL, por ejemplo, la hora del último acceso es superior a seis meses:



Cálculo de los resultados

En este paso, comprobará los resultados.

1. En UCMDB, acceda a Modeling Studio (**Modelado > Modeling Studio**).
2. Abra una vista.

3. Calcule los resultados haciendo clic en el botón **Calcular número de resultados de la consulta**  .
4. Haga clic en el botón **Vista previa** para ver los CI de la vista.

Visualización de resultados

En este paso, verá los resultados y depurará los problemas del procedimiento. Por ejemplo, si no se muestra nada en la vista, compruebe las definiciones en el archivo **orm.xml**; suprima los atributos de la relación y vuelva a cambiar el adaptador.

1. En UCMDB, acceda a IT Universe Manager (**Modelado > IT Universe Manager**).
2. Seleccione un CI. La pestaña Propiedades muestra los resultados de la federación.

Ver informes

En este paso, visualizará los informes de topología. Para obtener más información, consulte "Información general de los informes de topología" en *HP Universal CMDB – Guía de modelado*.

Habilitación de archivos de registro

Consulte los archivos de registro para comprender los flujos de cálculo, el ciclo de vida del adaptador y para ver la información de depuración. Para obtener más información, consulte "Archivos de registro del adaptador" en la página 151.

Uso de Eclipse para asignar entre atributos CIT y tablas de bases de datos

Precaución: Este procedimiento está dirigido a usuarios con un conocimiento avanzado del desarrollo de contenido. Si tiene alguna duda, póngase en contacto con HP Software Support.

En esta tarea se describe cómo instalar y usar el complemento JPA, incluido con la edición J2EE de Eclipse, para:

- Habilite la asignación gráfica entre los atributos de clase de CMDB y las columnas de la tabla de la base de datos.
- Habilite la edición manual del archivo de asignación (`orm.xml`), mientras proporciona precisión. La comprobación de precisión incluye una comprobación de sintaxis así como verificación de que los atributos de la clase y columnas de la tabla de la base de datos asignada se indican correctamente.
- Habilite el despliegue del archivo de asignación en el servidor de CMDB y para ver los errores, como una comprobación de precisión adicional.
- Defina una consulta de ejemplo en el servidor de CMDB y ejecútelo directamente en Eclipse, para probar el archivo de asignación.

La versión 1.1 del complemento es compatible con UCMDB versión 9.01 o posterior y Eclipse IDE for Java EE Developers, versión 1.2.2.20100217-2310 o posterior.

Esta tarea incluye los siguientes pasos:

1. ## Requisitos previos

   Instale la última actualización de **Java Runtime Environment (JRE) 6** en el equipo en donde se ejecutará Eclipse desde el sitio siguiente: **http://java.sun.com/javase/downloads/index.jsp**.

2. ## Instalación

   a. Descargue y extraiga **Eclipse IDE for Java EE Developers** de **http://www.eclipse.org/downloads**en una carpeta local, por ejemplo, **C:\Archivos de programa\eclipse**.

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

   c. Inicie **C:\Archivos de programa\Eclipse\eclipse.exe**. Si aparece un mensaje relativo a

que no se encuentra la máquina virtual Java, inicie **eclipse.exe** con la siguiente línea de comandos:

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

3. ## Preparación del entorno de trabajo

En este paso, configurará el espacio de trabajo, base de datos, conexiones y propiedades del controlador.

a. Extraiga el archivo **workspaces_gdb.zip** de **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** en **C:\Documents and Settings\All Users**.

> **Nota:** Debe usar la ruta exacta de la carpeta. Si descomprime el archivo en la ruta errónea o deja el archivo sin descomprimir, el procedimiento no funcionará.

b. En Eclipse, elija **File > Switch Workspace > Other**:

Si está trabajando con:

- SQL Server, seleccione la carpeta siguiente: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.
- MySQL, seleccione la carpeta siguiente: **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
- Oracle, seleccione la carpeta siguiente: **C:\Documents and Settings\All Users\workspace_gdb_oracle**.

c. Haga clic en **Aceptar**.

d. En Eclipse, muestre la vista Project Explorer y seleccione **<Active project> > JPA Content > persistence.xml > <nombre proyecto activo> > orm.xml**.

e. En la vista Data Source Explorer (panel inferior izquierdo), haga clic con el botón derecho en la conexión de la base de datos y seleccione el menú **Properties**.

f. En el cuadro de diálogo **Properties for <Connection name>**, seleccione **Common** y seleccione la casilla**Connect every time the workbench is started**. Seleccione **Driver Properties** y rellene las propiedades de conexión. Haga clic en **Test Connection** y compruebe que la conexión funciona. Haga clic en **Aceptar**.

g. En la vista Data Source Explorer, haga clic con el botón derecho la conexión de la base de datos y haga clic en **Connect**. Se muestra un árbol que contiene los esquemas y tablas de la base de datos debajo del icono de la conexión de la base de datos.

4. ## Creación de un adaptador

Cree un adaptador usando las directrices indicadas en "Paso 1: Creación de un adaptador" en la página 29.

5. ## Configuración del complemento CMDB

a. En Eclipse, haga clic en **UCMDB > Settings** para abrir el cuadro de diálogo **CMDB Settings**.

b. Si no está seleccionado, seleccione el proyecto JPA recientemente creado como proyecto activo.

c. Introduzca el nombre de host de CMDB, por ejemplo, **localhost** o **labm1.itdep1**. No es preciso incluir el número de puerto o el prefijo `http://` en la dirección.

d. Rellene el nombre de usuario y la contraseña para acceder a la API de CMDB, por lo general **admin**/**admin**.

e. Asegúrese de que la carpeta **C:\hp** en el servidor de CMDB se asigna como una unidad de red.

f. Seleccione la carpeta base del adaptador relevante en **C:\hp**. La carpeta base es la única que contiene el archivo **dbAdapter.jar** y la subcarpeta **META-INF**. Su ruta debería ser **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<nombre adaptador>**. Compruebe que no hay ninguna barra invertida (\) al final.

6. Importación del modelo de clase de UCMDB

En este paso, seleccione los CIT que se van a asignar como entidades JPA.

a. Haga clic en **UCMDB > Import CMDB Class Model** para abrir el cuadro de diálogo **CI Type Selection**.

b. Seleccione los tipos de CI que pretende asignar como entidades de JPA. Haga clic en **Aceptar**. Los tipos de CI se importan como clases Java. Compruebe que aparecen en la carpeta **src** del proyecto activo.

7. Construcción del archivo ORM: Asignación de clases de UCMDB a tablas de la base de datos

En este paso, va a asignar las clases Java (las que ha importado en el paso anterior) a las tablas de la base de datos.

a. Asegúrese de que la conexión de la base de datos está conectada. Haga clic con con el botón derecho el proyecto activo (llamado myProject de manera predeterminada) en Project Explorer. Seleccione la vista JPA, seleccione la casilla **Override default schema from connection** y seleccione el esquema de base de datos relevante. Haga clic en **Aceptar**.

b. Asigne un CIT: En la vista JPA Structure, haga clic con el botón derecho la rama **Entity Mappings** y seleccione **Add Class**. Se abre el cuadro de diálogo **Add Persistent Class**. No cambie el campo **Map as** (**Entity**).

c. Haga clic en **Browse** y seleccione la clase UCMDB que se va a asignar (todas las clases de UCMDB pertenecen al paquete **generic_db_adapter**).

d. Haga clic en **OK** en ambos cuadros de diálogo. La clase seleccionada se muestra bajo la rama **Entity Mappings** en la vista JPA Structure.

> **Nota:** Si la entidad aparece sin un árbol de atributos, haga clic con el botón derecho en el proyecto activo de la vista Project Explorer. Elija **Close** y después **Open**.

e. En la vista JPA Details, seleccione la tabla de la base de datos principal a la que deberá asignarse la clase de UCMDB. Deje los demás campos sin modificar.

8. ## Asignación de Id.

De acuerdo con los estándares de JPA, cada clase persistente debe tener al menos un atributo de Id. Para las clases de UCMDB, puede asignar hasta tres atributos como ID. Los posibles atributos de ID se llaman **id1**, **id2** y **id3**.

Para asignar un atributo de Id.:

a. Expanda la clase correspondiente debajo de la rama **Entity Mappings** en la vista JPA Structure, haga clic con el botón derecho en el atributo relevante (por ejemplo, **id1**) y seleccione **Add Attribute to XML and Map…**.

b. Se abre el cuadro de diálogo **Add Persistent Attribute**. Seleccione **Id** en el campo **Map as** y haga clic en **OK**.

c. En la vista JPA Details, seleccione la columna de la tabla de la base de datos a la que deberá asignarse el campo Id.

9. ## Asignación de atributos

En este paso, puede asignar atributos a las columnas de la base de datos.

a. Expanda la clase correspondiente debajo de la rama **Entity Mappings** en la vista JPA Structure, haga clic con el botón derecho en el atributo relevante (por ejemplo, **host_ hostname**) y seleccione **Add Attribute to XML and Map…**.

b. Se abre el cuadro de diálogo **Add Persistent Attribute**. Seleccione **Basic** en el campo **Map as** y haga clic en **OK**.

c. En la vista JPA Details, seleccione la columna de la tabla de la base de datos a la que deberá asignarse el campo de atributo.

10. ## Asignación de un vínculo válido

Realice los pasos descritos en el paso"Construcción del archivo ORM: Asignación de clases de UCMDB a tablas de la base de datos" en la página precedente para asignar una clase de UCMDB que denote un vínculo válido. El nombre de cada una de estas clases tiene la estructura siguiente: **<nombre entidad end1>_<nombre vínculo>_<nombre entidad end 2>**. Por ejemplo, un vínculo **Contains** entre un host y una ubicación se indica por una clase Java cuyo nombre es **generic_db_adapter.host_contains_location**. Para obtener más información, consulte "Archivo reconciliation_rules.txt (para compatibilidad con versiones anteriores)" en la página 133.

a. Asigne los atributos de Id. de la clase de vínculo, tal como se describe en "Asignación de Id." arriba. Para cada atributo de Id., expanda el grupo de casillas de verificación **Details** en la vista JPA Details y borre las casillas **Insertable** y **Updateable**.

b. Asigne los atributos **end1** y **end2** de la clase de vínculo de la forma siguiente: Para cada uno de los atributos **end1** y **end2** de la clase de vínculo:

- Expanda la clase correspondiente debajo de la rama **Entity Mappings** en la vista JPA Structure, haga clic con el botón derecho en el atributo relevante (por ejemplo, **end1**) y seleccione **Add Attribute to XML and Map…**.

- En el cuadro de diálogo **Add Persistent Attribute**, seleccione **Many to One** o **One to One** en el campo **Map as**.
- Seleccione **Many to One** si el CI **end1** o **end2** especificado puede tener varios vínculos de este tipo. En caso contrario, seleccione **One to One**. Por ejemplo, para un vínculo **host_contains_ip** el extremo **host** debería asignarse como **Many to One**, ya que un host puede tener varios IP y el extremo **ip** debería asignarse como **One to One**, ya que un IP puede tener solo un único host.
- En la vista JPA Details, seleccione **Target entity**, por ejemplo, **generic_db_adapter.host**.
- En la sección **Join Columns** de la vista JPA Details, marque **Override Default**. Haga clic en **Edit**. En el cuadro de diálogo **Edit Join Column**, seleccione la columna de la clave externa de la tabla de la base de datos de vínculos a una entrada en la tabla de la entidad de destino **end1**/**end2**. Si el nombre de columna referenciado en la tabla de la entidad de destino **end1**/**end2** está asignado a su atributo de Id., deje **Referenced Column Name** sin cambiar. En caso contrario, seleccione el nombre de la columna a la que apunta la columna de la clave externa. Borre las casillas **Insertable** y **Updatable** y haga clic en **OK**.
- Si la entidad de destino **end1**/**end2** tiene más de un Id., haga clic en el botón **Add** para agregar columnas de unión adicionales y asignarlas de la misma forma que se describió en el paso anterior.

11. ### Construcción del archivo ORM: uso de tablas secundarias

JPA permite que una clase Java se asigne a más de una tabla de la base de datos. Por ejemplo, **Host** se puede asignar a la tabla **Device** para habilitar la persistencia de la mayoría de sus atributos y para la tabla **NetworkNames** para habilitar la persistencia de **host_hostName**. En este caso, **Device** es la tabla principal y **NetworkNames** es la tabla secundaria. Se puede definir cualquier número de tablas secundarias. La única condición es que tiene que haber una relación uno-a-uno entre las entradas de las tablas principal y secundaria.

12. ### Definición de una tabla secundaria

Seleccione la clase apropiada en la vista JPA Structure. En la vista **JPA Details**, acceda a la sección **Secondary Tables** y haga clic en **Add**. En el cuadro de diálogo **Add Secondary Table**, seleccione la tabla secundaria apropiada. Deje los otros campos sin modificar.

Si la tabla principal y la secundaria no tienen las mismas claves principales, configure las columnas de unión en la sección **Primary Key Join Columns** de la vista **JPA Details**.

13. ### Asignación de un atributo a una tabla secundaria

Puede asignar un atributo de clase a un campo de una tabla secundaria de la forma siguiente:

a. Asigne el atributo, tal como se describe en "Asignación de atributos" en la página precedente.

b. En la sección **Column** de la vista JPA Details, seleccione el nombre de la tabla secundaria en el campo **Table** para reemplazar el valor predeterminado.

14. ### Utilización de un archivo ORM existente como base

Para usar un archivo **orm.xml** existente como base para el que está desarrollando, ejecute los

pasos siguientes:

a. Compruebe que todos los CIT asignados en el archivo **orm.xml** se importan en el proyecto Eclipse activo.

b. Seleccione y copie todo o parte de las asignaciones de entidades del archivo existente.

c. Seleccione la pestaña **Source** del archivo **orm.xml** en la perspectiva JPA de Eclipse.

d. Pegue las asignaciones de entidades copiadas en la etiqueta **<entity-mappings>** del archivo **orm.xml**, debajo de la etiqueta **<schema>**. Asegúrese de que la etiqueta schema se configura tal como se describe en el paso "Construcción del archivo ORM: Asignación de clases de UCMDB a tablas de la base de datos" en la página 112. Todas las entidades pegadas ahora aparecen en la vista JPA Structure. De ahora en adelante, las asignaciones se pueden editar tanto gráfica como manualmente a través del código xml del archivo **orm.xml**.

e. Haga clic en **Guardar**.

15. Importación de un archivo ORM existente desde un adaptador

Si ya existe un adaptador, se puede usar el complemento de Eclipse para editar gráficamente su archivo ORM. Importe el archivo **orm.xml** en Eclipse, edítelo usando el complemento y, a continuación, vuelva a implementarlo en el equipo UCMDB. Para importar el archivo ORM, haga clic en el botón en la barra de herramientas de Eclipse. Se muestra un cuadro de diálogo de confirmación. Haga clic en **Aceptar**. El archivo ORM se copia desde el equipo de UCMDB al proyecto Eclipse activo y todas las clases relevantes se importan desde el modelo de clase de UCMDB.

Si las clases relevantes no aparecen en la vista JPA Structure, haga clic con el botón derecho en el proyecto activo de la vista Project Explorer, elija **Close** y, a continuación, **Open**.

De ahora en adelante, el archivo ORM se puede editar gráficamente usando Eclipse y, a continuación, implementarlo de nuevo en el equipo de UCMDB , tal como se describe en "Despliegue el archivo ORM en el CMDB" en la página siguiente.

16. Comprobación de la precisión del archivo orm.xml - Comprobación de precisión incorporada

El complemento JPA de Eclipse comprueba si hay algún error y lo marca en el archivo **orm.xml**. Se comprueban tanto los errores de sintaxis (por ejemplo, nombre de etiqueta erróneo, etiqueta sin cerrar, Id. que falta) como los errores de asignación (por ejemplo, nombre del atributo o nombre de cambio de la tabla de base de datos erróneos). Si hay errores, su descripción aparece en la vista **Problems**.

17. Creación de un nuevo punto de integración

Si no hay ningún punto de integración en el CMDB para este adaptador, puede crearla en Integration Studio. Para obtener más información, consulte "Integration Studio" en *HP Universal CMDB – Guía de Administración de Data Flow*.

Rellene el nombre del punto de integración en el cuadro de diálogo que se muestra. El archivo **orm.xml** se copia en la carpeta del adaptador. Se crea un punto de integración con todos los tipos de CI importados como sus clases admitidas, excepto por los CIT multinodo, si están configurados en el archivo **reconciliation_rules.txt**. Para obtener más información, consulte

"Archivo reconciliation_rules.txt (para compatibilidad con versiones anteriores)" en la página 133.

18. **Despliegue el archivo ORM en el CMDB**

    Guarde el archivo **orm.xml** y despliéguelo en el servidor UCMDB pulsando **UCMDB > Deploy ORM**. El archivo **orm.xml** se copia en la carpeta del adaptador y este se recarga. El resultado de la operación se muestra en el cuadro de diálogo **Operation Result**. Si ocurre cualquier error durante el proceso de recarga, se muestra el seguimiento de la pila de excepción Java en el cuadro de diálogo. Si no se ha definido todavía un punto de integración usando el adaptador, no se detecta ningún error de asignación después del despliegue.

19. **Ejecución de una consulta TQL de ejemplo**

    a. Defina una consulta (no una vista) en Modeling Studio Para obtener más información, consulte "Modeling Studio" en *HP Universal CMDB – Guía de Administración de Data Flow*.

    b. Cree un punto de integración mediante el adaptador que creó en el paso "Creación de un nuevo punto de integración". Para obtener más información, consulte "Cuadro de diálogo Nuevo punto de integración/Editar punto de integración" en *HP Universal CMDB – Guía de Administración de Data Flow*.

    c. Durante la creación del adaptador, compruebe que los tipos de CI que deben participar en la consulta se admiten en este punto de integración.

    d. Al configurar el complemento CMDB, utilice este nombre de consulta de ejemplo en el cuadro de diálogo Configuración. Para obtener más información, consulte el paso "Configuración del complemento CMDB" en la página 111.

    e. Haga clic en el botón **Ejecutar consulta** para ejecutar una consulta de ejemplo y comprobar si devuelve los resultados requeridos usando el archivo **orm.xml** recientemente creado.

Archivos de configuración de adaptadores

Los archivos que se tratan en esta sección están ubicados en el paquete **db-adapter.zip** en la carpeta **C:\hp\UCMDB\UCMDBServer\content\adapters**.

En esta sección se describen los siguientes archivos de configuración:

- "Archivo fixed_values.txt" en la página 137
- "Archivo persistence.conf " en la página 137

Configuración general

- **adapter.conf.** El archivo de configuración del adaptador. Para obtener más información, consulte "Archivo adapter.conf " abajo.

Configuración sencilla

- **simplifiedConfiguration.xml**. Archivo de configuración que reemplaza **orm.xml**, **transformations.txt** y **reconciliation_rules.txt** con menos capacidades. Para obtener más información, consulte "Archivo simplifiedConfiguration.xml" en la página siguiente.

Configuración avanzada

- **orm.xml**. El archivo de asignación relacional de objetos en el que asigna entre los CIT de CMDB y las tablas de base de datos. Para obtener más información, consulte "Archivo orm.xml" en la página 120.
- **reconciliation_types.txt**. Contiene las reglas que se utilizan para configurar los tipos de reconciliación. Para obtener más información, consulte "Archivo reconciliation_types.txt" en la página 133.
- **reconciliation_rules.txt**. Contiene las reglas de reconciliación. Para obtener más información, consulte "Archivo reconciliation_rules.txt (para compatibilidad con versiones anteriores)" en la página 133.
- **transformations.txt.** Archivo de transformaciones en el que se especifican los convertidores que se van a aplicar para convertir el valor de CMDB al valor de la base de datos, y viceversa. Para obtener más información, consulte "Archivo transformations.txt" en la página 135.
- **Discriminator.properties.** Este archivo asigna cada tipo de CI admitido a una lista separada por comas de posibles valores correspondientes. Para obtener más información, consulte "Archivo discriminator.properties" en la página 135.
- **Replication_config.txt.**Este archivo contiene una lista separada por comas de tipos de CI y relaciones, cuyas condiciones de propiedades se admiten en el complemento de replicación. Para obtener más información, consulte "Archivo replication_config.txt" en la página 136.
- **Fixed_values.txt.**Este archivo le permite configurar valores fijos para atributos específicos de ciertos CIT. Para obtener más información, consulte "Archivo fixed_values.txt" en la página 137.

Configuración de Hibernate

- **persistence.xml.**Se utiliza para reemplazar las configuraciones de Hibernate de serie. Para obtener más información, consulte "Archivo persistence.conf " en la página 137.

Archivo adapter.conf

Este archivo contiene la configuración siguiente:

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

  Nota: El uso de este archivo significa que `orm.xml, transformations.txt` y `reconciliation_rules.txt` se reemplazan con menos capacidades.

- **dal.ids.chunk.size=300.** No cambie este valor.
- **dal.use.persistence.xml=false**. **true**: el adaptador lee la configuración de Hibernate de `persistence.xml`.

  Nota: No se recomienda reemplazar la configuración de Hibernate.

- **performance.memory.id.filtering=true**. Cuando GDBA ejecuta TQLS, en algunos casos puede recuperarse un gran número de ID y devolverlos a la base de datos mediante SQL. Para evitar este trabajo excesivo y mejorar el rendimiento, GDBA intenta leer toda la vista/tabla y filtra los resultados que hay en la memoria.
- **id.reconciliation.cmdb.id.type=string/bytes**. Cuando se asigna el adaptador de BD genérica mediante la reconciliación de ID (para obtener información, vea el paso "Configure el archivo reconciliation_types.txt (para el motor de asignación predeterminado UCMDB 9.0x)" in "Implementación del motor de asignación" en la página 179, puede asignar la cadena **cmdb_id** a un tipo de columna **string** o **bytes/raw** cambiando la propiedad **META-INF/ adapter.conf**.
- **performance.enable.single.sql=true**. Este parámetro es opcional. Si no aparece en el archivo, su valor predeterminado es **true**. Si es **true**, el adaptador de bases de datos genéricas intenta generar una única sentencia SQL para cada consulta que se ejecute (sea para rellenado o una consulta federada). El uso de una sola sentencia SQL mejora el rendimiento y el consumo de memoria del adaptador de bases de datos genéricas. Si es **false**, el adaptador de bases de datos genéricas genera varias sentencias SQL, que pueden tardar más tiempo y consumir más memoria que una sola. Aunque este atributo se establezca como **true**, el adaptador no genera una única sentencia SQL en las situaciones siguientes:
  - La base de datos a la que se conecta el adaptador no es un servidor Oracle o SQL.
  - La TQL que se ejecuta contiene una condición de cardinalidad distinta de 0..* y 1..* (por ejemplo, si existe una condición de cardinalidad como 2..* o 0..2).
- **in.expression.size.limit=950** (predeterminado). Este parámetro divide la expresión 'IN' del SQL ejecutado, cuando se alcance el límite de tamaño de la lista de argumentos.

Archivo simplifiedConfiguration.xml

Este archivo se usa para la asignación simple de clases de UCMDB a tablas de bases de datos. Para acceder a la plantilla para editar el archivo, desplácese a **Administración de adaptador > db-adapter > Archivos de configuración**.

Esta sección incluye los siguientes temas:

Plantilla del archivo simplifiedConfiguration.xml

La propiedad **CMDB-class-name** es el tipo de multinodo (el nodo al que los CIT federados se conectan en el 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. La reconciliación se puede realizar usando uno o dos nodos. En este ejemplo de caso, la reconciliación usa dos nodos.

connected-node-CMDB-class-name. El segundo tipo de clase que se necesita en el TQL de reconciliación.

CMDB-link-type. El tipo de relación que se necesita en el TQL de reconciliación.

link-direction. La dirección de la relación del TQL de reconciliación (desde `node` a `ip_address` o desde `ip_address` a `node`):

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

La expresión de reconciliación se realiza en la forma de OR y cada OR incluye AND.

is-ordered. Determina si la reconciliación se realiza en forma ordenada o con una comparación `OR`.

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

Si la propiedad de reconciliación se recupera desde la clase principal (el multinodo), utilice la etiqueta **attribute**, en caso contrario, utilice la etiqueta **connected-node-attribute**.

ignore-case.true: cuando los datos del modelo de clase de UCMDB se comparan con los datos del RDBMS, no se distingue entre mayúsculas y minúsculas:

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

El nombre de columna es el nombre de la columna de la clave externa (la columna con valores que apuntan a la columna de la clave principal multinodo).

Si la columna de la clave principal está compuesta por varias columnas, es preciso que haya varias columnas de claves externas, una para cada columna clave principal.

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

Si hay pocas columnas de claves principales, duplique esta columna.

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

Las propiedades **from-CMDB-converter** y **to-CMDB-converter** son clases Java que implementan las interfaces siguientes:

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

Utilice estos convertidores si el valor de CMDB y de la base de datos no son los mismos.

En este ejemplo, `GenericEnumTransformer` se usa para convertir el enumerador de acuerdo con el archivo XML que se escribe dentro del paréntesis (**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>
```

Limitaciones

- Se puede usar para asignar solo consultas TQL que contengan un nodo (en el origen de la base de datos). Por ejemplo, puede ejecutar un `node > ticket` y una consulta TQL `ticket`. Para llevar la jerarquía de nodos desde la base de datos, debe usar el archivo **orm.xml** avanzado.
- Solo se admiten las relaciones uno-a-varios. Por ejemplo, puede llevar uno o más tickets en cada nodo. No puede llevar tickets que pertenecen a más de un nodo.
- No puede conectar la misma clase a tipos diferentes de CIT de CMDB. Por ejemplo, si define que `ticket` está conectado a `node`, no puede conectarse también a `application`.

Archivo orm.xml

Este archivo se usa para asignar los CTI de CMDB a las tablas de bases de datos.

En el directorio **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF** se encuentra una plantilla que se puede usar para crear un archivo nuevo.

Para editar el archivo XML en un adaptador implementado, vaya a **Administración de adaptador > db-adapter > Archivos de configuración**.

Esta sección incluye los siguientes temas:

- "Plantilla del archivo orm.xml" en la página siguiente
- "Varios archivos ORM" en la página 124

Plantilla del archivo 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>
```

No cambie el nombre del paquete.

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

entity. El nombre del CTI de CMDB. Esta es la entidad multinodo.

Asegúrese de que **class** incluye un prefijo **generic_db_adapter.**

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

Utilice una tabla secundaria si la entidad está asignada a más de una tabla.

```
        <secondary-table name="" />
        <attributes>
```

Para la herencia de una única tabla con discriminador, utilice el código siguiente:

```
        <inheritance strategy="SINGLE_TABLE" />
        <discriminator-value>node</discriminator-value>
        <discriminator-column name="[column_name]" />
```

Los atributos con la etiqueta **id** son las columnas de claves principales. Asegúrese de que la convención de nomenclatura para estas columnas de claves principales son **idX** (id1, id2, etc.) donde **X** es el índice de columnas de la clave principal.

```
            <id name="id1">
```

Cambie solo el nombre de columna de la clave principal.

```
                <column updatable="false" insertable="false" name="
```

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

basic. Se utiliza para declarar los atributos de CMDB. Asegúrese de editar solo las propiedades **name** y **column_name**.

```
            <basic name="name">
                <column updatable="false" insertable="false" name="
[column_name]" />
            </basic>
```

Para la herencia de una única tabla con discriminador, asigne las clases ampliadas de la forma siguiente:

```
    <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>
```

En el ejemplo siguiente se muestra un nombre de atributo de CMDB sin prefijo:

```
            <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>
```

Esta es la entidad de relación. La convención de nomenclatura es **end1Type_linkType_ end2Type**. En este ejemplo **end1Type** es **node** y el **linkType** es **composition**.

```
    <entity name="node_composition_[CMDB_class_name]"
class="generic_db_adapter.node_composition_[CMDB_class_name]">
        <table name="[default_table_name]" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="
[column_name]" />
                <generated-value strategy="TABLE" />
            </id>
```

La entidad de destino es la entidad a la que está apuntando esta propiedad. En este ejemplo, **end1** se asigna a la entidad **node**.

many-to-one. Varias relaciones se pueden conectar a un nodo.

join-column. La columna que contiene los Id. **end1** (los Id. de entidades de destino).

referenced-column-name. El nombre de columna en la entidad de destino (**node**) que contiene los Id. que se usan en la columna de unión.

```
            <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. Una relación se puede conectar a una **[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. Esto es un ejemplo se cómo agregar un atributo de nodo.

```
<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>
```

Varios archivos ORM

Se admiten varios archivos de asignación. Cada archivo de asignación debe finalizar con **orm.xml**. Todos los archivos de asignación deberían colocarse debajo de la carpeta META-INF del adaptador.

Convenciones de nomenclatura

- En cada entidad, la propiedad de clase debe coincidir con la propiedad de nombre con el prefijo `generic_db_adapter`.
- Las columnas de claves principales deben tomar los nombres del formulario **idX** donde **X = 1**, **2**, ..., de acuerdo con el número de claves principales de la tabla.
- Los nombres de atributos deben coincidir con los nombres de atributos de la clase, distinguiendo mayúsculas y minúsculas.
- El nombre de la relación toma la forma de `end1Type_linkType_end2Type`.
- Los CIT de CMDB, que también son palabras reservadas de Java, deben llevar el prefijo **gdba_**. Por ejemplo, para el CIT de CMDB**goto**, la entidad ORM debe llamarse **gdba_goto**.

Uso de instrucciones SQL integradas en lugar de nombres de tablas

Puede asignar entidades en cláusulas `select` integradas en lugar de en tablas de bases de datos.

Es el equivalente a definir una vista en la base de datos y a asignar una entidad en esta vista. Por ejemplo:

```
    <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)" />
```

En este ejemplo, los atributos de nodo deberían asignarse a las columnas id1, name y host_os, en lugar de id, name y os.

Se aplican las siguientes limitaciones:

- La instrucción SQL integrada está disponible solo cuando se usa Hibernate como proveedor de JPA.
- Es obligatorio el uso de corchetes redondos en torno a la cláusula select de SQL integrada.
- El elemento **<schema>** no debería estar presente en el archivo **orm.xml**. En el caso de Microsoft SQL Server 2005, significa que todos los nombres de tablas deberían llevar el prefijo `dbo.`, en lugar de definirlas globalmente con `<schema>dbo</schema>`.

Esquema orm.xml

En la tabla siguiente se explican los elementos comunes del archivo **orm.xml**. El esquema completo se puede encontrar en http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. La lista no es completa y explica principalmente el comportamiento específico del API de persistencia Java estándar para el adaptador de bases de datos genéricos.

Nombre y ruta del elemento	Descripción	Atributos
entity-mappings	El elemento raíz del documento de asignación de la entidad. Este elemento debe ser exactamente el mismo que el dado en los archivos de ejemplo GDBA.	
descripción (entity-mappings)	Una descripción libre del documento de asignación de la entidad. (Opcional)	
paquete (entity-mappings)	El nombre del paquete Java que contendrá las clases de asignación. Siempre debería contener el texto `generic_db_adapter`.	1. **Nombre:** name **Descripción:** El nombre del tipo de CI de UCMDB al que está asignada la entidad. Si esta entidad se asigna a un vínculo de CMDB, el nombre de la entidad debería tener el formato end_1>_link_name>_end_2>. `<end_1>_<link_name>_`

Nombre y ruta del elemento	Descripción	Atributos
		<end_2>. Por ejemplo, node_composition_cpu define una entidad que se asignará al vínculo de composición entre un nodo y un CPU. Si el nombre del tipo de CI es el mismo que el nombre de la clase Java sin el prefijo del paquete, este campo se puede omitir. **¿Es necesario?:** Opcional **Tipo:** Cadena 2. **Nombre:** class **Descripción:** El nombre completo de la clase Java que se creará para esta entidad de la base de datos. El nombre del paquete de la clase Java debería ser el mismo que el nombre dado en el elemento package. Puede no utilizar palabras reservadas Java, como interface o switch, como nombre de clase. En ese caso, agregue el prefijo gdba_ al nombre (por tanto la interfaz será generic_db_adapter.gdba_interface. **¿Es necesario?:** Requerido **Tipo:** Cadena
table (entity-mappings>entity)	Este elemento define la tabla principal de la entidad de la base de datos. Solo puede aparecer una vez. Requerido.	**Nombre:** name **Descripción:** El nombre de la tabla principal. Si el nombre de la tabla no contiene el esquema al que pertenece, la tabla se buscará solo en el esquema del usuario que se usó para crear el punto de integración. También puede ser cualquier instrucción SELECT válida. Si es una instrucción SELECT, debe ir entre paréntesis. **¿Es necesario?:** Requerido **Tipo:** Cadena
secondary-table (entity-mappings >	Este elemento se puede usar para definir una tabla secundaria para la entidad de la base de	**Nombre:** name **Descripción:** El nombre de la tabla

Nombre y ruta del elemento	Descripción	Atributos
entity)	datos. Esta tabla debe estar conectada a la tabla principal con una relación uno-a-uno. Puede definir más de una tabla secundaria. Opcional.	secundaria. Si el nombre de la tabla no contiene el esquema al que pertenece, la tabla se buscará solo en el esquema del usuario que se usó para crear el punto de integración. También puede ser cualquier instrucción SELECT válida. Si es una instrucción SELECT, debe ir entre paréntesis. **¿Es necesario?:** Requerido **Tipo:** Cadena
primary-key-join-column (entity-mappings > entity > secondary-table)	Si la tabla secundaria y la tabla principal no están conectadas usando campos con el mismo nombre, este elemento define el nombre del campo de la clave principal en la tabla secundaria que necesita conectarse al campo de clave principal de la tabla principal.	**Nombre:** name **Descripción:** El nombre del campo de la tabla principal en la tabla secundaria. Si este elemento no existe, se asume que el campo de clave principal tiene el mismo nombre que el campo de la clave principal de la tabla principal. **¿Es necesario?:** Opcional **Tipo:** Cadena
inheritance (entity-mappings > entity)	Si la entidad actual es la entidad principal de una familia de entidades de bases de datos, utilice este elemento para marcarlo como tal. Opcional.	**Nombre:** strategy **Descripción:** Define la forma en que se implementa la herencia en la base de datos. **¿Es necesario?:** Requerido **Tipo:** Uno de los valores siguientes: • SINGLE_TABLE: Esta entidad y todas las entidades secundarias existen en la misma tabla. • JOINED: Las entidades secundarias son tablas unidas. • TABLE_PER_CLASS: Cada entidad está completamente definida en una tabla separada.
discriminator-column (entity-mappings > entity)	Si la herencia es de tipo SINGLE_TABLE, este elemento se usa para definir el nombre del campo usado para determinar el tipo de entidad para cada fila.	**Nombre:** name **Descripción:** El nombre de la columna discriminador. **¿Es necesario?:** Requerido **Tipo:** Cadena
discriminator-value	Este elemento define el tipo de	

Nombre y ruta del elemento	Descripción	Atributos
(entity-mappings > entity)	entidad específica en el árbol de herencia. Este nombre necesita ser el mismo que el nombre definido en el archivo **discriminator.properties** para el grupo de valor de este tipo de entidad específica.	
attributes (entity-mappings > entity)	El elemento raíz de todas las asignaciones de atributos para una entidad.	
id (entity-mappings > entity attributes)	Este elemento define el campo de clave para la entidad. Debe haber por lo menos un campo de Id. definido. Si existe más de un elemento de id, sus campos crean una clave compuesta para la entidad. Debería intentar y evitar claves compuestas para las entidades de CI (no para vínculos).	**Nombre:** name **Descripción:** Una cadena de tipo idX, donde X es un número entre 1 y 9. El primer Id. debería marcarse como id1, el segundo como id2 y así sucesivamente. NO es el nombre del atributo de clave de UCMDB. **¿Es necesario?:** Requerido **Tipo:** Cadena
basic (entity-mappings > entity attributes)	Este elemento define una asignación entre un campo de la tabla, que no forma parte de la clave principal de la tabla, y un atributo UCMDB.	**Nombre:** name **Descripción:** El nombre del atributo de UCMDB para el que se asigna el campo. Este atributo debe existir en el CI de UCMDB al que está asignada la entidad actual. **¿Es necesario?:** Requerido **Tipo:** Cadena
column (entity-mappings > entity > attributes >id -O BIEN- (entity-mappings > entity > attributes > basic)	Define el nombre de la columna de la tabla para la asignación básica o un campo de Id.	1. **Nombre:** name **Descripción:** El nombre del campo. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** table **Descripción:** El nombre de la tabla al que pertenece el campo. Esto debe ser la tabla principal o bien una de las tablas secundarias definidas para la entidad. Si se omite este atributo, se asume que el campo pertenece a la tabla principal.

Nombre y ruta del elemento	Descripción	Atributos
		¿Es necesario?: Opcional **Tipo:** Cadena
one-to-one (entity-mappings > entity > attributes)	Define una columna cuyo valor está en otra tabla, y las dos tablas se conectan usando una relación uno-a-uno. Este elemento solo se admite para asignaciones de entidades de vínculo y no para los otros tipos de CI. Esta es la única forma de definir una asignación entre una tabla y un vínculo de UCMDB.	1. **Nombre:** name **Descripción:** Cuál de los dos extremos representa este campo. **¿Es necesario?:** Requerido **Tipo:** Bien `end1` o `end2` 2. **Nombre:** target-entity **Descripción:** El nombre de la entidad a la que se refiere el extremo. **¿Es necesario?:** Requerido **Tipo:** Uno de los nombres de entidad definidos en el documento de asignación de entidades.
join-column (entity-mappings > entity attributes > one-to-one)	Define la forma de unir la entidad de destino definida en el elemento uno-a-uno de primer nivel y la entidad actual.	1. **Nombre:** name **Descripción:** El nombre del campo de la tabla actual que se usará para realizar la unión uno-a-uno. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** name **Descripción:** El nombre de un campo en la entidad conjunta mediante el cual se realiza la unión. Si este atributo se omite, asume que la tabla unida tiene una columna con el mismo nombre que el campo definido en el atributo de nombre. **¿Es necesario?:** Opcional **Tipo:** Cadena

Ejemplo de creación del archivo orm.xml

El ejemplo aquí presentado muestra cómo crear el **archivo orm.xml**. En este ejemplo, las tablas de SQL de una base de datos remota se asignan a tipos de CI del UCMDB.

En las tablas que se encuentran en la base de datos remota con el formato que se muestra a continuación, rellene la tabla **Hosts** con nodos, la tabla **IP_Addresses** con direcciones IP, y después cree vínculos entre los nodos y las direcciones IP como se muestra a continuación:

Tabla de hosts

host_name	host_id
Test1	1
Test2	2
Test3	3

IP_Addresses Table

ip_address	ip_id
10.1.1.1	1
10.2.2.2	2
10.3.3.2	3
10.4.4.4	4

Tabla Host_IP_Link (vínculos entre nodos y direcciones IP)

host_id	ip_id
1	1
2	2
2	3
3	4

La clave principal de la tabla de **Hosts** es el campo **host_id** y la clave principal de la tabla **IP_ Addresses** es el campo **ip_id**. En la tabla **Host_IP_Link**, **host_id** e **ip_id** son claves externas de la tabla **Hosts** y **IP_Addresses**.

De acuerdo con las tablas anteriores, cree el archivo **orm.xml** como se indica en los pasos siguientes. Las entidades utilizadas en este ejemplo son **node**, **ip_address** y **node_containment_ ip_address**

1. Cree la entidad **node** asignando **host_id** de la tabla **Hosts** de la forma siguiente:

```
<entity-mappings
xmlns="http://java.sun.com/xml/ns/persistence/orm"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
version="1.0"

        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm_
1_0.xsd">

    <description>test_integration</description>

    <package>generic_db_adapter</package>

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

```
      <table name="Hosts"/>
      <attributes>
        <id name="id1">
                <column updatable="false" insertable="false" name="host_id"/>
                <generated-value strategy="TABLE"/>
        </id>
        <basic name="name">
                <column updatable="false" insertable="false" name="host_
name"/>
        </basic>
      </attributes>
    </entity>
```

La entidad **class** debe ser un tipo de CI que ya existe en UCMDB. La tabla **name** es la tabla contenida en la base de datos que incluye un Id. y la información del host. El atributo de Id. se requiere para identificar hosts específicos y se usará más tarde en la asignación. En este ejemplo, el atributo **name** de esta entidad se rellena con la columna **host_name** en la tabla Hosts.

2. En la siguiente entidad, asigne direcciones IP de la tabla de interfaces:

```
<entity name="ip_address" class="generic_db_adapter.ip_address">
    <table name="IP_Addresses"/>
    <attributes>
      <id name="id1">
        <column insertable="false" updatable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
      </id>
      <basic name="name">
        <column updatable="false" insertable="false" name="ip_
address"/>
      </basic>
    </attributes>
</entity>
```

3. A continuación, el vínculo entre el nodo y la dirección IP debe crearse mediante la tabla de asignación y debe hacer referencia al campo **ip_id** (aunque podría hacer referencia, si se desea, a los campos **host_id** e **ip_id**).

```
<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>
```

El nombre de entidad para el contenedor tiene el formato: [end1 CIT]_[link CIT]_[end2 CIT]. Así, para este ejemplo, dado que el tipo de CI de vínculo es **containment**, el nombre de entidad para el contenedor es: **node_containment_ip_address** y la clase de entidad es **generic_db_adapter.node_containment_ip_address**. El Id. se requiere en este bloque de código y, aunque este ejemplo funciona con un único Id. de la interfaz, las dos columnas podrían referirse a id1 e id2. En ese caso el código sería:

```
<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>
```

Los dos extremos de este vínculo son ‘many-to-one’ y ‘one-to-one’, lo que significa que cada dirección IP estará vinculada a un nodo, pero un nodo podrá vincularse a varias direcciones IP.

Las columnas incluidas proceden de la tabla de vínculos y hacen referencia a las tablas de hosts e interfaces.

Archivo reconciliation_types.txt

Este archivo se utiliza para configurar los tipos de reconciliación.

Cada fila del archivo representa un CIT de CMDB que se conecta a un CIT de base de datos federado en la consulta TQL.

Archivo reconciliation_rules.txt (para compatibilidad con versiones anteriores)

Este archivo se usa para configurar las reglas de reconciliación si desea ejecutar la reconciliación cuando se configura DBMappingEngine en el adaptador. Si no usa DBMappingEngine, se utiliza el mecanismo de reconciliación UCMDB genérico y no es preciso configurar este archivo.

Cada fila del archivo representa una regla. Por ejemplo:

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

El multinodo se rellena con el nombre del multinodo (el CIT de CMDB que está conectado al CIT de la base de datos federada en la consulta TQL).

Esta expresión incluye la lógica que decide si dos multinodos son iguales (un multinodo en el CMDB y el otro en el origen de la base de datos).

La expresión está compuesta de `OR` o `AND`.

La convención concerniente a los nombres de atributos en la parte de expresión es `[className].[attributeName]`. Por ejemplo, `attributeName` en la clase `ip_address` se escribe `ip_address.name`.

Para una correspondencia ordenada (si la primera subexpresión `OR` devuelve una respuesta indicando que los multinodos no son iguales, no se compara la segunda subexpresión OR), utilice `expresión ordenada` en lugar de `expresión`.

Para ignorar la distinción de mayúsculas y minúsculas durante una comparación, utilice el signo de control (**^**).

Los parámetros `end1_type`, `end2_type` y `link_type` solo se usan si la consulta TQL de reconciliación contiene dos nodos y no solo un multinodo. En este caso, la consulta TQL de reconciliación es `end1_type > (link_type) > end2_type`.

No es preciso agregar el diseño relevante cuando se toma de la expresión.

Tipos de reglas de reconciliación

Las reglas de reconciliación toman la forma de condiciones OR y AND. Puede definir estas reglas en varios nodos diferentes (por ejemplo, el nodo se identifica por `name from nodeAND/ORname from ip_address`).

La opciones siguientes encuentran una correspondencia:

- **Correspondencia ordenada.** La expresión de reconciliación se lee de izquierda a derecha. Dos subexpresiones `OR` se consideran iguales si tienen valores y son iguales. Dos subexpresiones `OR` se consideran no iguales si ambos tienen valores y no son iguales. Para cualquier otro caso no hay decisión y se busca la igualdad en la siguiente subexpresión `OR`.

  name from node OR from ip_address. Si tanto CMDB como el origen de datos incluyen `name` y son iguales, los nodos se consideran iguales. Si ambos incluyen `name` pero no son iguales, los nodos se consideran que no son iguales sin probar el `name` de `ip_address`. Si el `name of node` falta en CMDB o en el origen de datos, se comprueba el `name of ip_address`.

- **Correspondencia regular.** Si hay igualdad en una de las expresiones `OR`, tanto el CMDB y el origen de datos se consideran iguales.

  name from node OR from ip_address. Si no hay correspondencia en `name of node`, `name of ip_address` se comprueba por igualdad.

En las reconciliaciones complejas, donde se modela la entidad de reconciliación en el modelo de clase con varios CIT con relaciones (como `nodo`), la asignación de un nodo de superconjunto incluye todos los atributos relevantes de todos los CIT modelados.

Nota: Como resultado, existe una limitación de que todos los atributos de reconciliación en el origen de datos deberían residir en tablas que comparten la misma clave principal.

Otra limitación indica que la consulta TQL de reconciliación no debería tener más de dos nodos. Por ejemplo, la consulta TQL `node > ticket` tiene un nodo en el CMDB y un ticket en el origen de datos.

Para reconciliar los resultados, `name` debe recuperarse desde el nodo o de `ip_address`.

Si el `name` en CMDB está en el formato `*.m.com`, puede utilizarse un convertidor desde CMDB a la base de datos federada, y viceversa, para convertir estos valores.

Se usa la columna `node_id` de la tabla del ticket de la base de datos para conectar entre las entidades (la asociación definida también se puede realizar en una tabla de nodo):

Nodo de BD	
PK	node_id
	nombre

BD de IP_Address	
PK	ip_id
	nombre

Ticket de BD	
PK	ticket_id
	node_id

Nota: Las tres tablas deben formar parte del origen del RDBMS federado y no de la base de datos de CMDB.

Archivo transformations.txt

Este archivo contiene todas las definiciones de convertidor.

El formato es que cada línea contiene una nueva definición.

Plantilla del archivo transformations.xml

```
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. El nombre de entidad tal como aparece en el archivo `orm.xml`.

attribute. El nombre de atributo tal como aparece en el archivo `orm.xml`.

to_DB_class. El nombre completo de una clase que implementa la interfaz **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB**. Los elementos entre paréntesis se incluyen en este constructor de clase. Utilice este convertidor para transformar valores de CMDB en valores de base de datos, por ejemplo, para anexar el sufijo **.com** a cada nombre de nodo.

from_DB_class. El nombre completo de una clase que implementa la interfaz **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB**. Los elementos entre paréntesis se incluyen en este constructor de clase. Utilice este convertidor para transformar valores de base de datos en valores de CMDB, por ejemplo, para anexar el sufijo **.com** a cada nombre de nodo.

Para obtener más información, consulte "Convertidores de serie" en la página 138.

Archivo discriminator.properties

Este archivo asigna cada tipo de CI admitido (que también se usa como un valor de discriminador en orm.xml) a una lista separada por comas de posibles valores correspondientes de la columna discriminador, o una condición que coincida con valores posibles de la columna discriminador.

Si se utiliza una condición, utilice la sintaxis: `like(condición)`, donde `condición` es una cadena que puede contener los comodines siguientes:

- **%** (signo de porcentaje): permite que coincida con cualquier cadena de cualquier longitud (incluso una cadena de longitud cero)
- **_** (subrayado): permite que coincida con un solo carácter

Por ejemplo, `like(%unix%)` coincidirá con unix, linux, unix-aix, etc. Las condiciones de like sólo pueden aplicarse a columnas de tipo cadena.

También puede tener un solo valor discriminador asignado a cualquier valor que no pertenezca a otro discriminador indicando `'all-other'`.

Si el adaptador que está creando usa capacidades de discriminador, debe definir todos los valores de discriminador en el archivo **discriminator.properties**.

Ejemplo de asignación del discriminador:

Por ejemplo, el adaptador admite los tipos de CI `node`, `nt` y `unix`, y la base de datos contiene una sola tabla denominada `t_nodes` que contiene una columna denominada **type**. Si el tipo es 10001, la fila representa un nodo; si el tipo es 10004, representa una máquina unix, etc. El archivo **discriminator.properties** podría tener el aspecto siguiente:

```
node=10001, 10005 nt=10002,10003 unix=2% mainframe=all-other
```

El archivo **orm.xml** incluye el código siguiente:

```
<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>
```

El atributo discriminator_column se calcula de la forma siguiente:

- Si **type** contiene `10002` o `10003` para una entrada determinada, la entrada se asigna al CIT **nt**.
- Si **type** contiene `10001` o `10005` para una entrada determinada, la entrada se asigna al CIT **node**.
- Si **type** empieza por `2` o para una entrada determinada, la entrada se asigna al CIT **unix**.
- Cualquier otro valor en la columna **type** se asigna al CIT **mainframe**.

> **Nota:** El CIT **node** también es el elemento de primer nivel de **nt** y **unix**.

Archivo replication_config.txt

Este archivo contiene una lista separada por comas de tipos de CI y relaciones, cuyas condiciones de propiedades se admiten en el complemento de replicación. Para obtener más información, consulte "Complementos" en la página 142.

Archivo fixed_values.txt

Este archivo le permite configurar valores fijos para atributos específicos de ciertos CIT. De esta forma, a cada uno de estos atributos se le puede asignar un valor fijo que no está almacenado en la base de datos.

El archivo debería contener cero o más entradas del formato siguiente:

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

Por ejemplo:

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

El archivo también admite una lista de constantes. Para definir una lista de constantes, utilice la sintaxis siguiente:

```
entity[<entityName>] attribute[<attributeName>] value[{<Val1>, <Val2>,
<Val3>, ... }]
```

Archivo persistence.conf

Este archivo se usa para reemplazar la configuración predeterminada de Hibernate y para agregar soporte para tipos de bases de datos que no son de serie (los tipos de bases de datos de serie son Oracle Server, Microsoft SQL Server y MySQL).

Si necesita admitir un nuevo tipo de base de datos, asegúrese de que proporciona un proveedor de grupos de conexión (el valor predeterminado es `c3p0`) y un controlador JDBC para su base de datos (ponga los archivos `*.jar` en la carpeta del adaptador).

Para ver todos los valores disponibles de Hibernate que se pueden cambiar, consulte la clase **org.hibernate.cfg.Environment** (para obtener más información, consulte http://www.hibernate.org.)

Ejemplo del archivo 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>
```

Convertidores de serie

Puede usar los siguientes convertidores (transformadores) para convertir consultas federadas y trabajos de replicación a y desde datos de base de datos.

Esta sección incluye los siguientes temas:

- "Convertidores de serie" arriba
- "Convertidor SuffixTransformer" en la página 141
- "Convertidor PrefixTransformer" en la página 141
- "Convertidor BytesToStringTransformer" en la página 142
- "Convertidor StringDelimitedListTransformer " en la página 142

Convertidor enum-transformer

Este convertidor usa un archivo XML que se le incluye como parámetro de entrada.

El archivo XML asigna entre valores de CMDB codificados de forma rígida y valores de base de datos (enums). Si uno de los valores no existe, puede optar por devolver el mismo valor, devolver un valor nulo o lanzar una excepción.

El transformador realiza una comparación entre dos cadenas mediante un método que distinga o que no distinga entre mayúsculas y minúsculas. El comportamiento predeterminado es distinguir mayúsculas de minúsculas Para definir que el uso no distinga entre mayúsculas y minúsculas: `case-sensitive="false"` en el elemento enum-transformer.

Utilice un archivo de asignación XML para cada atributo de entidad.

Nota: Este convertidor se puede usar para los campos `to_DB_class` y `from_DB_class` del archivo **transformations.txt**.

XSD de archivo de entrada:

```
<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>
```

Ejemplo de conversión de un valor 'sys' en un valor 'System':

En este ejemplo, el valor `sys` de CMDB se transforma en un valor `System` en la base de datos federada y el valor `System` de la base de datos federada se transforma en el valor `sys` en CMDB.

Si el valor no existe en el archivo XML (por ejemplo, la cadena `demo`), el convertidor devuelve el mismo valor de entrada que recibe.

```
<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>
```

Ejemplo de conversión de un valor externo o de CMDB en un valor nulo:

En este ejemplo, el valor de **NNN** en la base de datos remota se transforma en un valor nulo en la base de datos de CMDB.

```
<value cmdb-value="null" is-cmdb-value-null="true" external-db-
value="NNN"/>
```

En este ejemplo, el valor de **OOO** en CMDB se transforma en un valor nulo en la base de datos remota.

```
<value cmdb-value="OOO" external-db-value="null" is-db-value-
null="true"/>
```

Convertidor SuffixTransformer

Este convertidor se usa para agregar o suprimir sufijos desde el valor de CMDB o el de origen de la base de datos federada.

Hay dos implementaciones:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddSuffixTransformer**. Agrega el sufijo (dado como entrada) al convertir un valor de base de datos federada en un valor de CMDB y suprime el sufijo al convertir del valor de CMDB en un valor de la base de datos federada.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemoveSuffixTransformer**. Quita el sufijo (dado como entrada) al convertir un valor de base de datos federada en un valor de CMDB y agrega el sufijo al convertir del valor de CMDB en un valor de la base de datos federada.

Convertidor PrefixTransformer

Este convertidor se usa para agregar o suprimir un prefijo desde el valor de CMDB o el de origen de

la base de datos federada.

Hay dos implementaciones:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddPrefixTransformer**. Agrega el prefijo (dado como entrada) al convertir un valor de base de datos federada en un valor de CMDB y suprime el prefijo al convertir del valor de CMDB en un valor de la base de datos federada.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemovePrefixTransformer**. Quita el prefijo (dado como entrada) al convertir un valor de base de datos federada en un valor de CMDB y agrega el prefijo al convertir del valor de CMDB en un valor de la base de datos federada.

Convertidor BytesToStringTransformer

Este convertidor se utiliza para convertir matrices de bytes en el CMDB en su representación de cadena del origen de la base de datos federada.

El convertidor es: **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer**.

Convertidor StringDelimitedListTransformer

Este convertidor se utiliza para transformar una sola lista de cadenas a una lista de enteros/cadenas en CMDB.

El convertidor es: **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.StringDelimitedListTransformer.**

Complementos

El adaptador de bases de datos genéricas admite los complementos siguientes:

- Un complemento opcional para la completa sincronización de topología.
- Un complemento opcional para sincronizar los campos en la topología. Si no se implementa ningún complemento para sincronizar los cambios, es posible realizar una sincronización diferencial, pero dicha sincronización será en realidad una sincronización completa.
- Un complemento opcional para sincronizar el diseño.
- Se requiere un complemento opcional para recuperar las consultas admitidas para sincronización. Si este complemento no se define, se devuelven todos los nombres TQL.
- Un complemento interno, opcional para cambiar la definición TQL y el resultado TQL.
- Un complemento interno, opcional para cambiar la petición de diseño y el resultado de los CI.
- Un complemento interno, opcional para cambiar la petición de diseño y el resultado de las relaciones.
- Un complemento interno opcional para cambiar la acción de devolver los Id.

Para obtener más información acerca de implementar y desplegar complementos, consulte "Implementación de un complemento" en la página 104.

Ejemplos de configuración

En esta sección se dan ejemplos de configuraciones.

Esta sección incluye los siguientes temas:

Caso de uso

Caso de uso. Una consulta TQL es:

node > (composition) > card

donde:

- **node** es la entidad de CMDB
- **card** es la entidad de origen de la base de datos federada
- **composition** es la relación entre ellas

El ejemplo se ejecuta en la base de datos `ED`. Los `ED nodes` se almacenan en la tabla `Device` y `card` se almacena en la tabla `hwCards`. En los ejemplos siguientes, `card` siempre se asigna de la misma forma.

Reconciliación de un único nodo

En este ejemplo la reconciliación se ejecuta en la propiedad `name`.

Definición simplificada

La reconciliación se realizada por `node` y se resalta con la etiqueta especial **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>
```

Definición avanzada

Archivo orm.xml

Preste atención a la adición de la asignación de relaciones. Para obtener información detallada, consulte la sección de definición en "Archivo orm.xml" en la página 120.

Ejemplo del archivo 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>
```

Archivo reconciliation_types.txt

Para obtener más información, consulte "Archivo reconciliation_types.txt" en la página 133.

```
node
```

Archivo reconciliation_rules.txt

Para obtener más información, consulte "Archivo reconciliation_rules.txt (para compatibilidad con versiones anteriores)" en la página 133.

```
multinode[node] expression[node.name]
```

Archivo transformation.txt

Este archivo sigue vacío, ya que no es necesario convertir valores en este ejemplo.

Reconciliación de dos nodos

En este ejemplo, la reconciliación se calcula de acuerdo con la propiedad `name` de `node` y de `ip_`

`address` con distintas variantes.

La consulta TQL de reconciliación es **node > (containment) > ip_address**.

Definición simplificada

La reconciliación se realiza por `name` de `node` OR de `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

La reconciliación se realiza por `name` de `node` AND de `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
            <and>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
```

```
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress" />
            </and>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

La reconciliación se realiza por `name` de `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-
CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID" />
        <reconciliation-by-two-nodes connected-node-CMDB-class-
name="ip_address" CMDB-link-type="containment">
            <or>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

Definición avanzada

Archivo orm.xml

Como la expresión de la reconciliación no se define en este archivo, se debe usar la misma versión para cualquier expresión de reconciliación.

Archivo reconciliation_types.txt

Para obtener más información, consulte "Archivo reconciliation_types.txt" en la página 133.

```
node
```

Archivo reconciliation_rules.txt

Para obtener más información, consulte "Archivo reconciliation_rules.txt (para compatibilidad con versiones anteriores)" en la página 133.

```
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]
```

Archivo transformation.txt

Este archivo sigue vacío, ya que no es necesario convertir valores en este ejemplo.

Uso de una clave principal que contiene más de una columna

Si la clave principal está compuesta por más de una columna, se añade el código siguiente a las definiciones XML:

Definición simplificada

Hay más de una etiqueta de clave principal y para cada columna hay una etiqueta.

```
    <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>
```

Definición avanzada

Archivo orm.xml

Se añade una nueva entidad `id` que se asigna a las columnas de las claves principales. Las entidades que usan esta entidad `id` deben agregar una etiqueta especial.

Si utiliza una clave externa (etiqueta join-column) para esta clave principal, debe asignar entre cada columna de la clave externa a una columna de la clave principal.

Para obtener más información, consulte "Archivo orm.xml" en la página 120.

Ejemplo del archivo 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>
```

Uso de transformaciones

En el ejemplo siguiente, el transformador **enum** genérico se convierte de valores `1`, `2`, `3` a valores `a`, `b`, `c` respectivamente en la columna `name`.

El archivo de asignación es `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>
```

Definición simplificada

```
    <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>
```

Definición avanzada

Solo hay un cambio en el archivo **transformation.txt**.

Archivo transformation.txt

Asegúrese de que los nombres de atributos y los nombres de entidad son iguales en el archivo `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)]
```

Archivos de registro del adaptador

Consulte los archivos de registro siguientes para comprender los flujos de cálculo y el ciclo de vida del adaptador, y para ver la información de depuración.

Esta sección incluye los siguientes temas:

- "Niveles de registro." abajo
- "Ubicaciones de registro." abajo

Niveles de registro.

Puede configurar el nivel de registro para cada uno de los registros.

En un editor de texto, abra el archivo **C:\hp\UCMDB\UCMDBServer\conf\log\ fcmdb.gdba.properties**
.

El nivel de registro predeterminado es **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL loglevel=ERROR
```

- Para aumentar el nivel de registro para todos los archivos de registro, cambie **loglevel=ERROR** a **loglevel=DEBUG** o **loglevel=INFO**.
- Para cambiar el nivel de registro para un archivo específico, cambie la línea de categoría **log4j** específica en consecuencia. Por ejemplo, para cambiar el nivel de registro de fcmdb.gdba.dal.sql.log a **INFO**, cambie

  ```
  log4j.category.fcmdb.gdba.dal.SQL=${loglevel},
  fcmdb.gdba.dal.SQL.appender
  ```

  por:

  ```
  log4j.category.fcmdb.gdba.dal.SQL=INFO,fcmdb.gdba.dal.SQL.appender
  ```

Ubicaciones de registro.

Los archivos de registro se encuentran en el directorio **C:\hp\UCMDB\UCMDBServer\runtime\log**.

- **Fcmdb.gdba.log**

  El registro del ciclo de vida del adaptador. Da detalles sobre cuándo se ha iniciado o detenido el adaptador, y cuáles son los CIT que admite este adaptador.

  Consulte errores de iniciación (carga/descarga del adaptador).

- **fcmdb.log**

  Consulte las excepciones.

- **cmdb.log**

  Consulte las excepciones.

- **Fcmdb.gdba.mapping.engine.log**

  El registro del motor de asignación. Da detalles sobre la consulta TQL de reconciliación que usa el motor de asignación, así como las topologías de reconciliación que se comparan durante la fase de conexión.

  Consulte este registro cuando una consulta TQL no da resultados aunque sepa que hay CI relevantes en la base de datos, o los resultados no sean los previstos (compruebe la reconciliación).

- **Fcmdb.gdba.TQL.log**

  El registro de TQL. Da detalles sobre las consultas TQL y sus resultados.

  Consulte este registro cuando una consulta TQL no devuelve resultados y el registro del motor de asignación muestra que no hay resultados en el origen de datos federado.

- **Fcmdb.gdba.dal.log**

  El registro del ciclo de vida de DAL. Da detalles sobre la generación de CIT y detalles de la conexión de la base de datos.

  Consulte este registro cuando no puede conectarse a la base de datos o cuando hay CIT o atributos no admitidos por la consulta.

- **Fcmdb.gdba.dal.command.log**

  El registro de operaciones de DAL. Da detalles sobre las operaciones internas de DAL que se han llamado. (Este registro es similar a `cmdb.dal.command.log`).

- **Fcmdb.gdba.dal.SQL.log**

  El registro de consultas SQL de DAL. Da detalles sobre las JPAQL (consultas SQL orientadas a objetos) llamadas y sus resultados.

  Consulte este registro cuando no puede conectarse a la base de datos o cuando hay CIT o atributos no admitidos por la consulta.

- **Fcmdb.gdba.hibrnate.log**

  El registro de Hibernate. Da detalles sobre las consultas SQL que se ejecutan, el análisis de cada JPAQL en SQL, los resultados de las consultas, los datos concernientes a la caché de Hibernate, etc. Para más información sobre Hibernate, consulte "Hibernate como proveedor de JPA " en la página 85.

Referencias externas

Para obtener más información sobre la especificación JavaBeans 3.0, consulte http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html..

Solución de problemas y limitaciones

En esta sección se describe la solución de problemas y las limitaciones del adaptador de bases de datos genéricas.

Limitaciones generales

- No se admite la autenticación NTLM de SQL Server.
- Al actualizar un paquete de adaptador, para editar los archivos de plantillas, utilice Notepad++, UltraEdit, o cualquier otro editor de texto de otros fabricantes que no sea Bloc de notas (cualquier versión) de Microsoft Corporation. De esta forma se evitará el uso de símbolos especiales, que pueden hacer que falle el despliegue del paquete preparado.

Limitaciones JPA

- Todas las tablas deben tener una columna de claves principales.
- Los nombres de atributo de la clase CMDB deben seguir la convención de nomenclatura de JavaBeans (por ejemplo, los nombres deben comenzar con minúscula).
- Dos CI que están conectados con una relación en el modelo de clase deben tener una asociación directa en la base de datos (por ejemplo, si `node` está conectado a `ticket` debe haber una clave externa o tabla de unión que los conecte).
- Varias tablas que están asignadas al mismo CIT deben compartir la misma tabla de claves principales.

Limitaciones funcionales

- No puede crear una relación manual entre el CMDB y los CIT federados. Para poder definir relaciones virtuales, se debe definir una lógica de relación especial (puede estar basada en las propiedades de la clase federada).
- Los CIT federados no pueden activar CIT en una regla de impacto, pero se pueden incluir en una consulta TQL de análisis de impacto.
- Un CIT federado puede formar parte de un TQL de enriquecimiento, pero no se puede usar en el nodo en el que se realiza el enriquecimiento (no puede agregar, actualizar o eliminar el CIT federado).
- No se admite el uso de un calificador de clase en una condición.
- No se admiten los subgráficos.
- No se admiten las relaciones compuestas.
- El `id` de CMDB de CI externo está compuesto de su clave principal y no de sus atributos de clave.
- Una columna de tipo `bytes` no se puede usar como columna de clave principal en Microsoft SQL Server.

- El cálculo de la consulta TQL falla si las condiciones de atributos que se definen en un nodo federado no tenían asignados sus nombres en el archivo **orm.xml**.
- El adaptador de bases de datos genéricas no admite la autenticación Windows para SQL Server.

Capítulo 5

Desarrollo de adaptadores Java

Este capítulo incluye:

Información general de Federation Framework

Nota:

- El término **relación** es equivalente al término **vínculo**.
- El término **CI** es equivalente al término **objeto**.
- Un gráfico es una colección de nodos y vínculos.

La funcionalidad Federation Framework utilice una API para recuperar la información de orígenes federados. Federation Framework proporciona tres capacidades principales:

- **Federación** sobre la marcha. Todas las consultas se ejecutan en los repositorios de datos originales y los resultados se generan sobre la marcha en CMDB.
- **Rellenado.** Rellena los datos (datos topológicos y propiedades de CI) en CMDB desde un origen de datos externo.
- **Inserción de datos.** Inserta los datos (datos topológicos y propiedades de CI) desde el CMDB local a un origen de datos remoto.

Todos los tipos de acciones requieren un adaptador para cada repositorio de datos, que puede proporcionar las capacidades específicas del repositorio de datos y recuperar o actualizar los datos requeridos. Cada petición del repositorio de datos se realiza a través de este adaptador.

Esta sección incluye también los siguientes temas:

- "Federación sobre la marcha" abajo
- "Inserción de datos" en la página siguiente
- "Rellenado" en la página 158

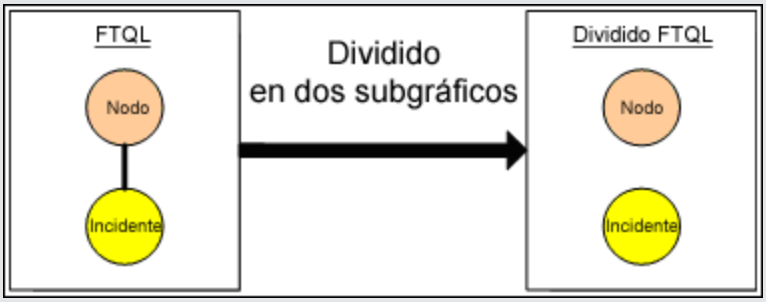
Federación sobre la marcha

Las consultas TQL federadas permiten la recuperación de los datos desde un repositorio de datos externo sin replicar sus datos.

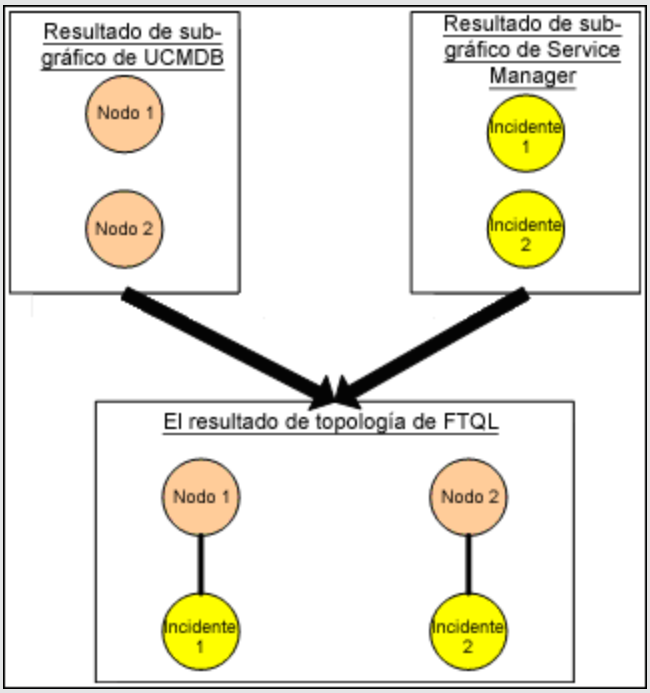
Una consulta TQL federada usa adaptadores que representan repositorios de datos externos para crear relaciones externas entre CI de diferentes repositorios de datos externos y los CI de UCMDB.

Ejemplo de flujo de federación sobre la marcha:

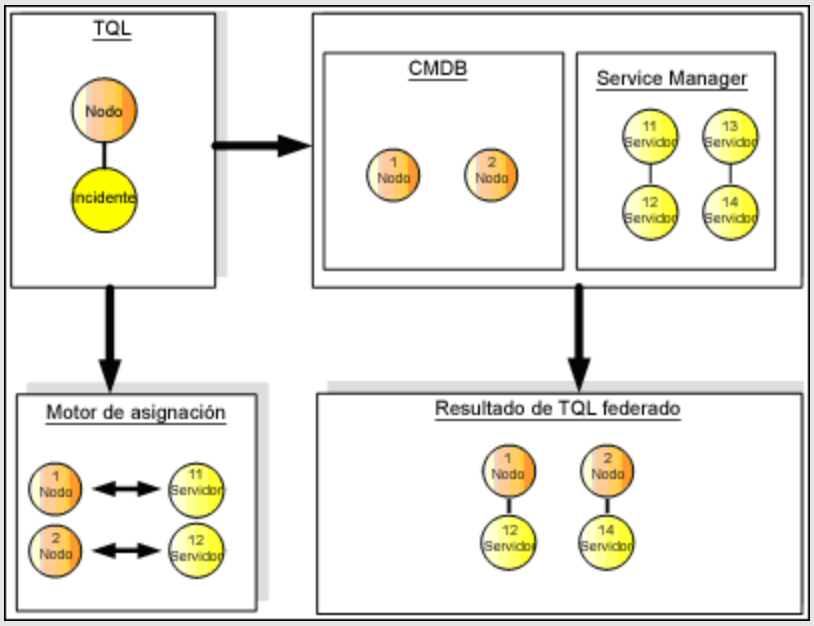
1. Federation Framework divide una consulta TQL federada en varios subgráficos, donde todos los nodos de un subgráfico se refieren al mismo repositorio de datos. Cada subgráfico está conectado a los otros subgráficos por una relación virtual (pero él mismo no contiene relaciones virtuales).



2. Después de que la consulta TQL federada se divida en subgráficos, Federation Framework calcula cada topología de subgráfico y conecta dos subgráficos apropiados al crear relaciones virtuales entre los nodos apropiados.



3. Después de que se calcule la topología TQL federada, Federation Framework recupera un diseño para el resultado de la topología.



Inserción de datos

Puede usar el flujo de inserción de datos para sincronizar los datos desde el CMDB local actual a un servicio remoto o un repositorio de datos de destino.

En la inserción de datos, los repositorios de datos se dividen en dos categorías: origen (CMDB local) y destino. Los datos se recuperan desde el repositorio de datos de origen y se actualizan en

el repositorio de datos de destino. El proceso de inserción de datos se basa en los nombres de consulta, lo que significa que los datos se sincronizan entre los repositorios de datos de origen (CMDB local) y destino y se recupera un nombre de consulta TQL desde el CMDB local.

El flujo del proceso de inserción de datos incluye los pasos siguientes:

1. Recuperación del resultado de la topología con firmas desde el repositorio de datos de origen.
2. Comparación de los nuevos resultados con los anteriores.
3. Recuperación de un diseño completo (es decir, todas las propiedades de CI) de los CI y relaciones, sólo para resultados modificados.
4. Actualización del repositorio de datos de destino con el diseño completo recibido de CI y relaciones. Si se elimina cualquier CI o relación en el repositorio de datos de origen y la consulta es exclusiva, el proceso de replicación suprime también los CI o relaciones en el repositorio de datos de destino.

CMDB tiene dos orígenes de datos ocultos (**hiddenRMIDataSource** y **hiddenChangesDataSource**), que siempre son el origen de datos "de origen" en los flujos de inserción de datos. Para implementar un nuevo adaptador para los flujos de inserción de datos, solo hay que implementar el adaptador "destino".

Rellenado

Puede usar el flujo de rellenado para rellenar CMDB con datos de orígenes externos.

El flujo siempre usa un origen de datos "de origen" para recuperar los datos e inserta los datos recuperados a la sonda en un proceso similar al flujo de un trabajo de detección.

Para implementar un nuevo adaptador para flujos de rellenado, solo tiene que implementar el adaptador de origen, ya que la sonda de Data Flow actúa como el destino.

El adaptador del flujo de rellenado se ejecuta en la sonda. La depuración y el registro debería realizarse en la sonda y no en CMDB.

El flujo de rellenado se basa en los nombres de consultas, es decir, los datos se sincronizan entre el repositorio de datos de origen y la sonda de Data Flow y se recupera por un nombre de consulta en el repositorio de datos de origen. Por ejemplo, en UCMDB, el nombre de la consulta es el nombre de la consulta TQL. Sin embargo, en otro repositorio de datos, el nombre de la consulta puede ser un nombre de código que devuelve datos. El adaptador está diseñado para manejar correctamente el nombre de consulta.

Cada trabajo se puede definir como un trabajo exclusivo. Esto significa que los CI y la relaciones en los resultados del trabajo son únicos en el CMDB local y que ninguna otra consulta puede llevarlos al destino. El adaptador del repositorio de datos de origen admite consultas específicas y puede recuperar los datos desde este repositorio de datos. El adaptador del repositorio de datos de destino permite la actualización de datos recuperados en este repositorio de datos.

Flujo SourceDataAdapter

- Recupera el resultado de la topología con firmas desde el repositorio de datos de origen.
- Compara los nuevos resultados con los anteriores.
- Recupera un diseño completo (es decir, todas las propiedades de CI) de los CI y relaciones,

sólo para resultados modificados.

- Actualiza el repositorio de datos de destino con el diseño completo recibido de CI y relaciones. Si se elimina cualquier CI o relación en el repositorio de datos de origen y la consulta es exclusiva, el proceso de replicación suprime también los CI o relaciones en el repositorio de datos de destino.

Flujo SourceChangesDataAdapter

- Recupera el resultado de topología que tuvo lugar desde la última fecha dada.
- Recupera un diseño completo (es decir, todas las propiedades de CI) de los CI y relaciones, sólo para resultados modificados.
- Actualiza el repositorio de datos de destino con el diseño completo recibido de CI y relaciones. Si se elimina cualquier CI o relación en el repositorio de datos de origen y la consulta es exclusiva, el proceso de replicación suprime también los CI o relaciones en el repositorio de datos de destino.

Flujo PopulateDataAdapter

- Recupera la topología completa con el resultado del diseño solicitado.
- Usa el mecanismo de fragmentos de topología para recuperar los datos en fragmentos.
- La sonda quita los datos que ya se trajeron en ejecuciones anteriores.
- Actualiza el repositorio de datos de destino con el diseño recibido de CI y relaciones. Si se elimina cualquier CI o relación en el repositorio de datos de origen y la consulta es exclusiva, el proceso de replicación suprime también los CI o relaciones en el repositorio de datos de destino.

Flujo PopulateChangesDataAdapter

- Recupera la topología con el resultado del diseño solicitado que tiene cambios desde la última ejecución.
- Usa el mecanismo de fragmentos de topología para recuperar los datos en fragmentos.
- La sonda quita los datos que ya se trajeron en ejecuciones anteriores (incluyendo este flujo).
- Actualiza el repositorio de datos de destino con el diseño recibido de CI y relaciones. Si se elimina cualquier CI o relación en el repositorio de datos de origen y la consulta es exclusiva, el proceso de replicación suprime también los CI o relaciones en el repositorio de datos de destino.

Adaptador y asignación de interacción con Federation Framework

Un adaptador es una entidad en UCMDB que representa datos externos (datos no guardados en UCMDB). En los flujos federados, todas las interacciones con los orígenes de datos externos se realizan mediante adaptadores. Las interfaces del adaptador y el flujo de interacción Federation Framework son diferentes para la réplica y las consultas TQL federadas.

Esta sección incluye también los siguientes temas:

- "Ciclo de vida del adaptador" abajo
- "Métodos de ayuda del adaptador" abajo

Ciclo de vida del adaptador

Se crea una instancia del adaptador para cada repositorio de datos externos. El adaptador inicia su ciclo de vida con la primera acción aplicada a él (como `calcular TQL` o `recuperar/actualizar datos`). Cuando se llama al método **start**, el adaptador recibe información medioambiental, como la configuración del repositorio de datos, el registro, etc. El ciclo de vida del adaptador termina cuando el repositorio de datos se elimina de la configuración y se llama al método **shutdown**. Esto significa que el adaptador tiene control del estado y puede contener la conexión con el repositorio de datos externo si es necesario.

Métodos de ayuda del adaptador

El adaptador tiene varios métodos `de ayuda` que pueden agregar configuraciones del repositorio de datos externo. Estos métodos no forman parte del ciclo de vida del adaptador y crean un adaptador cada vez que se les llama.

- El primer método comprueba la conexión al repositorio de datos externo para una determinada configuración. testConnection puede ejecutarse en el servidor UCMDB o la sonda de Data Flow, según el tipo de adaptador.
- El segundo método sólo es relevante para el adaptador de origen y devuelve las consultas admitidas para réplica. (Este método sólo se ejecuta en la sonda).
- El tercer método solo es relevante para los flujos de federación y rellenado, y devuelve clases externas admitidas por el repositorio de datos externo. (Este método se ejecuta en el servidor de UCMDB).

Todos estos métodos se usan al crear o ver configuraciones de integración.

Federation Framework para consultas TQL federadas

Esta sección incluye los siguientes temas:

- "Definiciones y términos" abajo
- "Motor de asignación" en la página siguiente
- "Adaptador federado" en la página siguiente

Consulte "Interacciones entre Federation Framework, servidor, adaptador y motor de asignación" en la página siguiente para ver los diagramas que ilustran las interacciones entre Federation Framework, UCMDB, el adaptador y el motor de asignación.

Definiciones y términos

Datos de reconciliación. La regla para comparar los CI del tipo especificado que se reciben desde CMDB y el repositorio de datos externo. La regla de reconciliación puede ser de tres tipos:

- **Reconciliación de ID.** Solo puede usarse si el repositorio de datos externo contiene el Id. de CMDB de los objetos de reconciliación.
- **Reconciliación de la propiedad.** Se usa cuando la comparación puede realizarse mediante

propiedades sólo del tipo CI de reconciliación.

- **Reconciliación de topologías.** Se usa cuando se necesitan las propiedades de CIT adicionales (no sólo de CIT de reconciliación) para realizar una comparación de los CI de reconciliación. Por ejemplo, puede realizar una reconciliación del tipo de nodo por la propiedad `name`, que pertenece al CIT `ip_address` .

Objeto de reconciliación. El adaptador crea el objeto de acuerdo con los datos de reconciliación recibidos. Este objeto debe hacer referencia a un CI externo y lo usa el motor de asignación para conectar entre los CI externos y los CI de CMDB.

Tipo de CI de reconciliación. El tipo de CI que representa objetos de reconciliación. Estos CI deben almacenarse en los repositorios de datos externos y CMDB.

Motor de asignación. Un componente que identifica relaciones entre CI de diferentes repositorios de datos con una relación virtual entre ellos. La identificación se realiza reconciliando los objetos de reconciliación de CMDB y los objetos de reconciliación de CI externos.

Motor de asignación

Federation Framework usa el motor de asignación para calcular la consulta TQL federada. El motor de asignación conecta entre los CI que se reciben de diferentes repositorios de datos y se conectan mediante relaciones virtuales. El motor de asignación también proporciona datos de reconciliación para la relación virtual. Un extremo de la relación virtual debe hacer referencia al CMDB. Este extremo es un tipo de `reconciliación`. Para el cálculo de los dos subgráficos, puede iniciarse una relación virtual desde cualquier nodo del extremo.

Adaptador federado

El adaptador federado proporciona dos tipos de datos de repositorios de datos externos: Datos de CI externos y objetos de reconciliación que pertenecen a unos CI externos.

- **Datos de CI externos.** Datos externos que no existen en CMDB. Son los datos del destino del repositorio de datos externo.
- **Datos de objetos de reconciliación.** Datos auxiliares usados por Federation Framework para conectar los CI de CMDB y los datos externos. Cada objeto de reconciliación debe hacer referencia a un CI externo. El tipo de objetos de reconciliación es el tipo (o subtipo) de uno de los extremos de la relación virtual del que se recuperan los datos. Los objetos de reconciliación deben ajustar el adaptador recibido a los datos de reconciliación. El objeto de reconciliación puede ser de uno de los tres tipos siguientes: `IdReconciliationObject`, `PropertyReconciliationObject` o `TopologyReconciliationObject`.

En las interfaces basadas en DataAdapter (DataAdapter, PopulateDataAdapter y PopulateChangesDataAdapter), se solicita la reconciliación como parte de la definición de la consulta.

Interacciones entre Federation Framework, servidor, adaptador y motor de asignación

Los siguientes diagramas ilustran las interacciones entre Federation Framework, el servidor UCMDB, el adaptador y el motor de asignación. La consulta TQL federada en los diagramas de ejemplo sólo tiene una relación virtual, por lo que sólo UCMDB y un repositorio de datos externo están implicados en la consulta TQL federada.

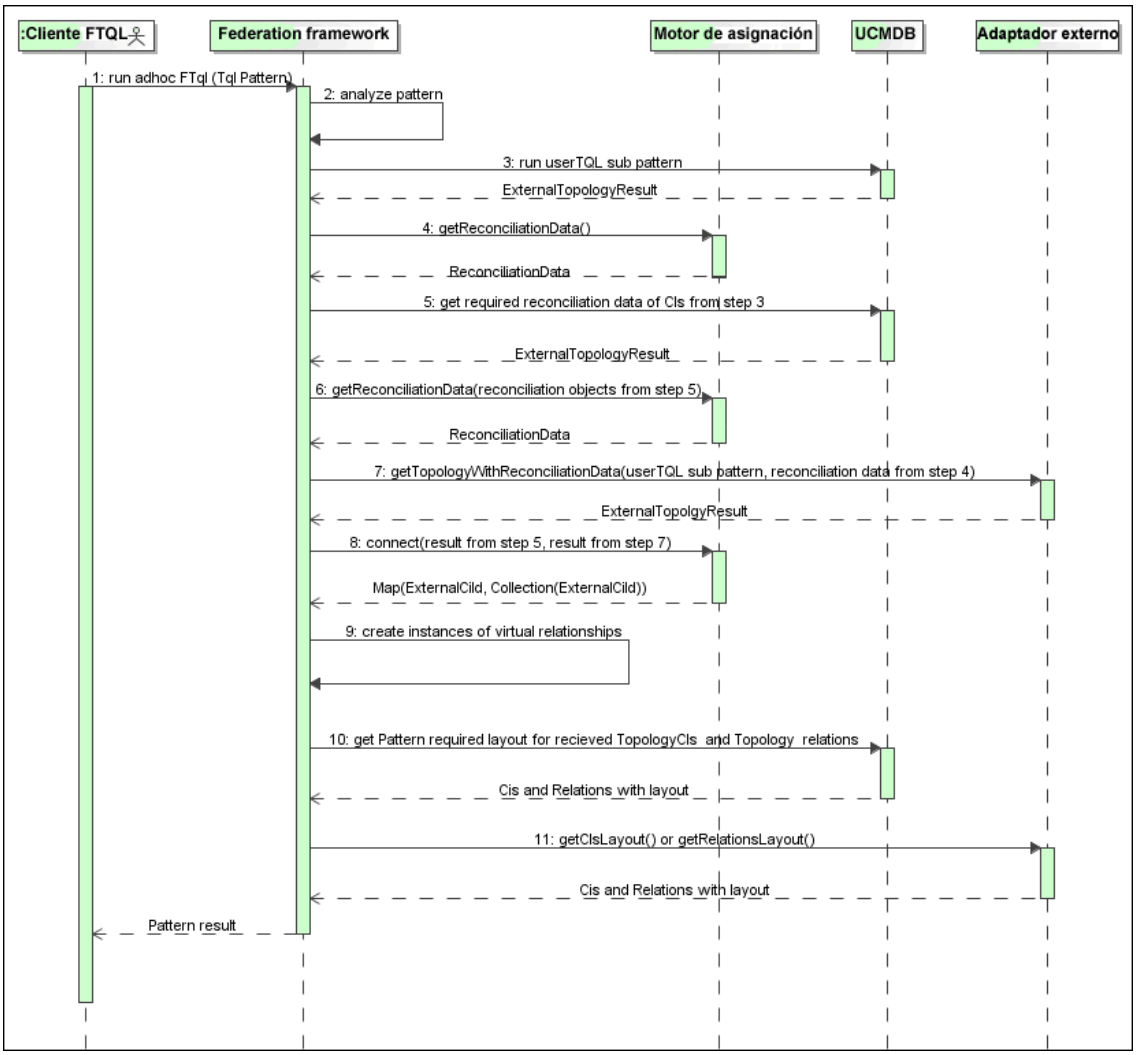
Esta sección incluye los siguientes temas:

- "El cálculo empieza en el extremo del servidor" abajo
- "El cálculo empieza en el extremo del adaptador externo" en la página 164
- "Ejemplo del flujo de Federation Framework para consultas TQL federadas" en la página 165

En el primer diagrama, el cálculo empieza en UCMDB y en el segundo diagrama, en el adaptador externo. Cada paso del diagrama incluye referencias a la llamada al método adecuado del adaptador o la interfaz del motor de asignación.

El cálculo empieza en el extremo del servidor

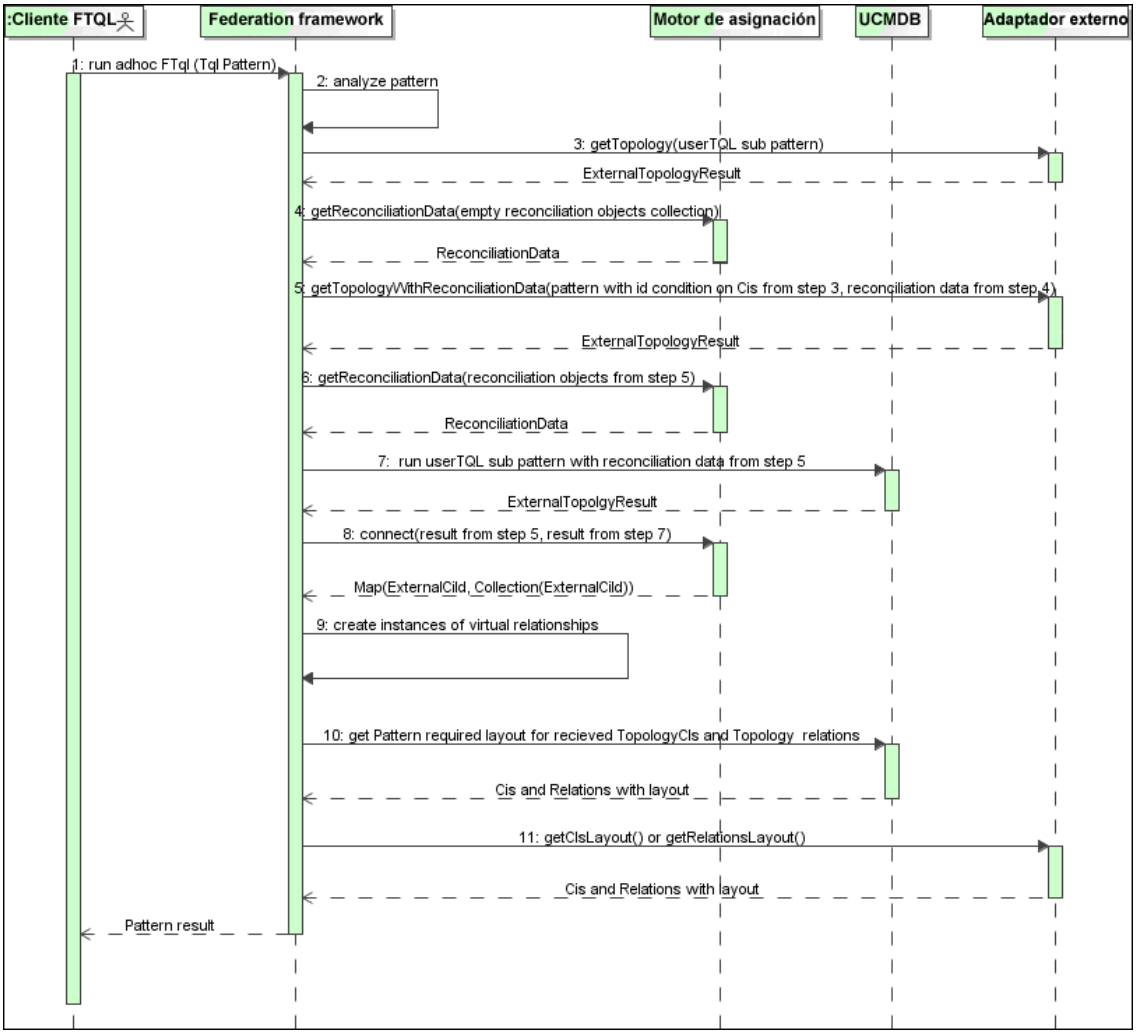
El siguiente diagrama de secuencia ilustra las interacciones entre Federation Framework, UCMDB, el adaptador y el motor de asignación. La consulta TQL federada en el diagrama de ejemplo sólo tiene una relación virtual, por lo que sólo UCMDB y un repositorio de datos externo están implicados en la consulta TQL federada.



Los números de esta imagen se explican a continuación:

Número	Explicación
1	Federation Framework recibe una llamada para un cálculo TQL federado.
2	Federation Framework analiza el adaptador, encuentra la relación virtual y divide el TQL original en dos subadaptadores; uno para UCMDB y otro para el repositorio de datos externo.
3	Federation Framework solicita la topología del subTQL desde UCMDB.
4	Después de recibir el resultado de la topología, Federation Framework llama al motor de asignación adecuado para la relación virtual actual y solicita los datos de reconciliación. El parámetro `reconciliationObject` está vacío en esta fase; es decir que no se añadido ninguna condición a los datos de reconciliación en esta llamada. Los datos de reconciliación devueltos definen los datos necesarios para comparar los CI de reconciliación en UCMDB con el repositorio de datos externo. Los datos de reconciliación pueden ser de uno de los siguientes tipos: • **IdReconciliationData.** Los CI se reconcilian de acuerdo a su Id. • **PropertyReconciliationData.** Los CI se reconcilian de acuerdo a las propiedades de uno de los CI. • **TopologyReconciliationData.** Los CI se reconcilian de acuerdo con la topología (por ejemplo, para reconciliar los CI de nodo, también es necesaria la dirección IP de **IP**).
5	Federation Framework solicita datos de reconciliación para los CI de los extremos de la relación virtual que se recibieron en el paso "3" arriba de UCMDB.
6	Federation Framework llama al motor de asignación para recuperar los datos de reconciliación. En este estado (por contraste con el paso "3" arriba), el motor de asignación recibe los objetos de reconciliación desde el paso "5" arriba como parámetros. El motor de asignación convierte el objeto de reconciliación recibido en la condición de los datos de reconciliación.
7	Federation Framework solicita la topología del subTQL desde el repositorio de datos externo. El adaptador externo recibe los datos de reconciliación desde el paso "6" arriba como un parámetro.
8	Federation Framework llama al motor de asignación para conectar los resultados recibidos. El parámetro `firstResult` es el resultado de topología externa recibido de UCMDB en el paso "5" arriba y el parámetro `secondResult` es el resultado de topología externa recibido del adaptador externo en el paso "7" arriba. El motor de asignación devuelve un mapa en que el Id. de CI externo del primer repositorio de datos (UCMDB en este caso) se ha asignado a los Id. de CI externos del segundo repositorio de datos (externo).
9	Para cada asignación, Federation Framework crea una relación virtual.
10	Después del cálculo de los resultados de la consulta TQL federada (sólo en la fase de topología), Federation Framework recupera el diseño de TQL original para los CI resultantes y las relaciones desde los repositorios de datos adecuados.

El cálculo empieza en el extremo del adaptador externo



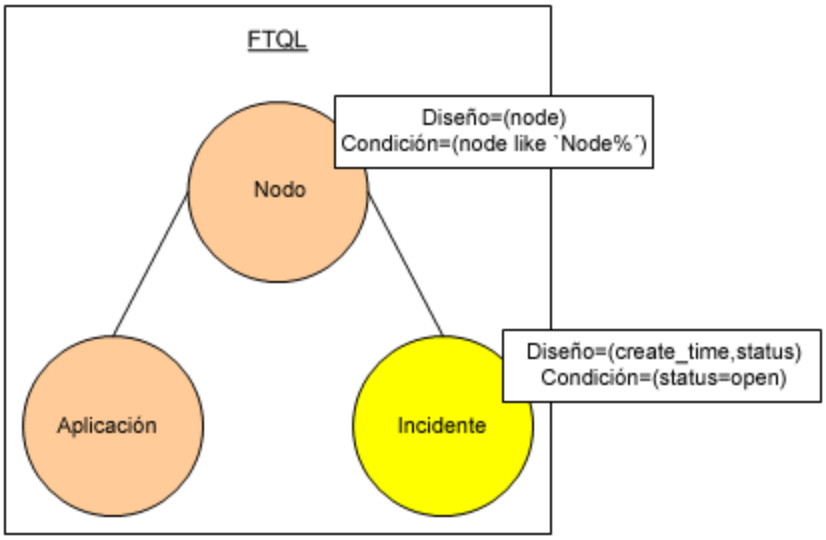
Los números de esta imagen se explican a continuación:

Número	Explicación
1	Federation Framework recibe una llamada de un cálculo de TQL federado.
2	Federation Framework analiza el adaptador, encuentra la relación virtual y divide el TQL original en dos subadaptadores; uno para UCMDB y otro para el repositorio de datos externo.
3	Federation Framework solicita la topología del subTQL del adaptador externo. Se supone que el `ExternalTopologyResult` devuelto no contiene ningún objeto de reconciliación, ya que los datos de reconciliación no forman parte de la solicitud.
4	Después de recibir los resultados de la topología Federation Framework llama al motor de asignación adecuado con la relación virtual actual y solicita los datos de reconciliación. El parámetro `reconciliationObjects` está vacío en este estado; es decir, que no se añade ninguna condición a los datos de reconciliación

Número	Explicación
	en esta llamada. Los datos de reconciliación devueltos definen los datos necesarios para comparar los CI de reconciliación en UCMDB con el repositorio de datos externo. Los datos de reconciliación pueden ser de uno de los tres tipos siguientes: • **IdReconciliationData.** Los CI se reconcilian de acuerdo a su Id. • **PropertyReconciliationData.** Los CI se reconcilian de acuerdo a las propiedades de uno de los CI. • **TopologyReconciliationData.** Los CI se reconcilian de acuerdo con la topología (por ejemplo, para reconciliar los CI de nodo, también es necesaria la dirección IP de **IP**).
5	Federation Framework solicita objetos de reconciliación de los CI que se recibieron en el paso 3 desde el repositorio de datos externo. Federation Framework llama al método **getTopologyWithReconciliationData()** en el adaptador externo, donde la topología solicitada es una topología de un nodo con los CI recibidos en el paso 3 como la condición ID y los datos de reconciliación del paso 4.
6	Federation Framework llama al motor de asignación para recuperar los datos de reconciliación. En este estado (por contraste con el paso 3), el motor de asignación recibe los objetos de reconciliación desde el paso 5 como parámetros. El motor de asignación convierte el objeto de reconciliación recibido en la condición de los datos de reconciliación.
7	Federation Framework solicita la topología del subTQL con datos de reconciliación del paso 6 de UCMDB.
8	Federation Framework llama al motor de asignación para conectar los resultados recibidos. El parámetro `firstResult` es el resultado de la topología externa recibido del adaptador externo en el paso 5 y el parámetro `secondResult` es el resultado de la topología externa recibido de UCMDB7. El motor de asignación devuelve un mapa en el que el Id. de CI externo del primer repositorio de datos (en este caso el repositorio de datos externo) se asigna a los Id. de CI externo desde el segundo repositorio de datos (UCMDB).
9	Para cada asignación, Federation Framework crea una relación virtual.
10	Después del cálculo de los resultados de la consulta TQL federada (sólo en la fase de topología), Federation Framework recupera el diseño de TQL original para los CI resultantes y las relaciones desde los repositorios de datos adecuados.

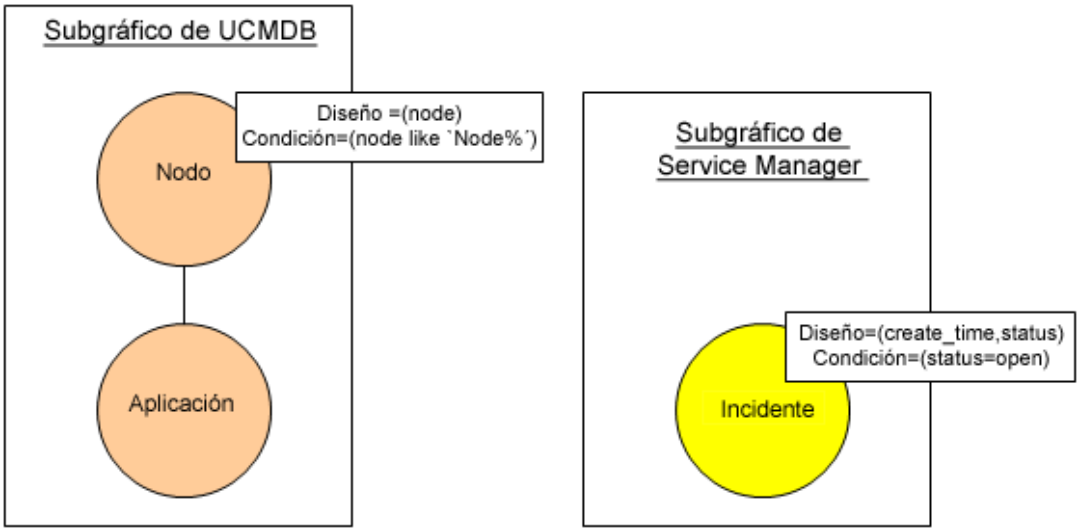
Ejemplo del flujo de Federation Framework para consultas TQL federadas

En este ejemplo se explica el modo de ver todos los incidentes abiertos en nodos específicos. El repositorio de datos ServiceCenter es el repositorio de datos externo. Las instancias del nodo se almacenan en UCMDB y las instancias del incidente se almacenan en ServiceCenter. Se considera que para conectar las instancias del incidente al nodo adecuado, son necesarias las propiedades `node` y `ip_address` del host y la dirección IP. Estas son las propiedades de reconciliación que identifican los nodos de ServiceCenter en UCMDB.

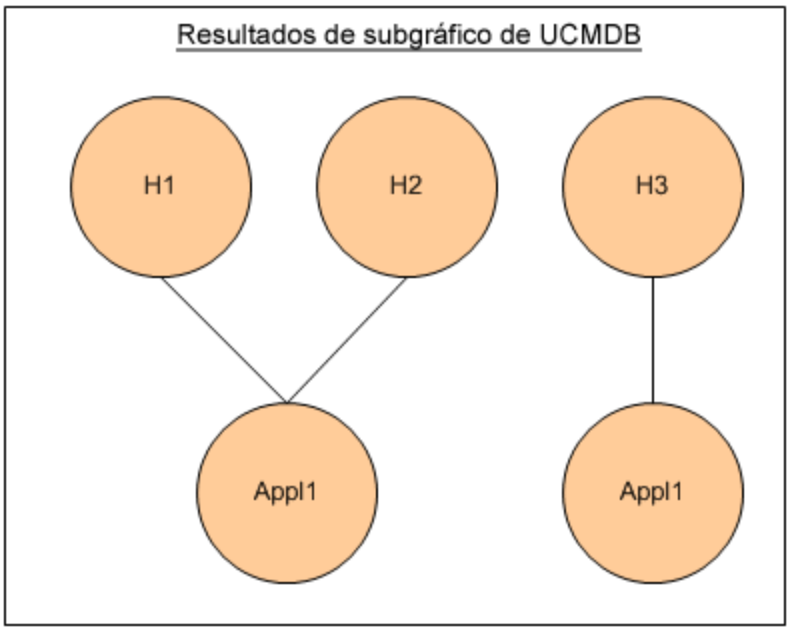


Nota: Para la federación de atributos, se llama al método **getTopology** del adaptador. Los datos de reconciliación se adaptan en el TQL del usuario (en este caso, el elemento CI).

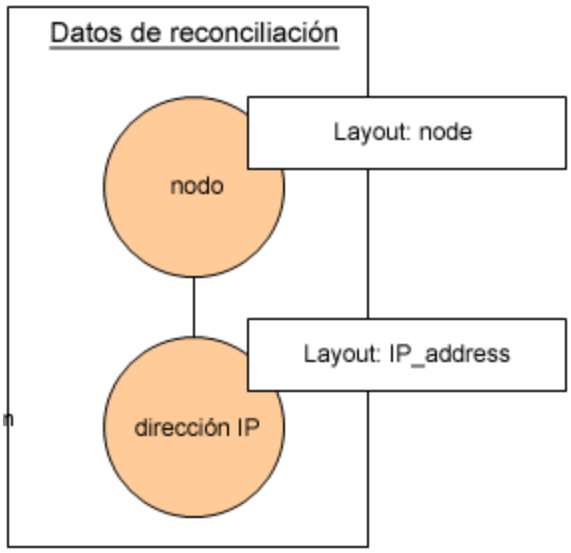
1. Después de analizar el adaptador, Federation Framework reconoce la relación virtual entre `Node` e `Incident` y divide la consulta TQL federada en dos subgráficos:



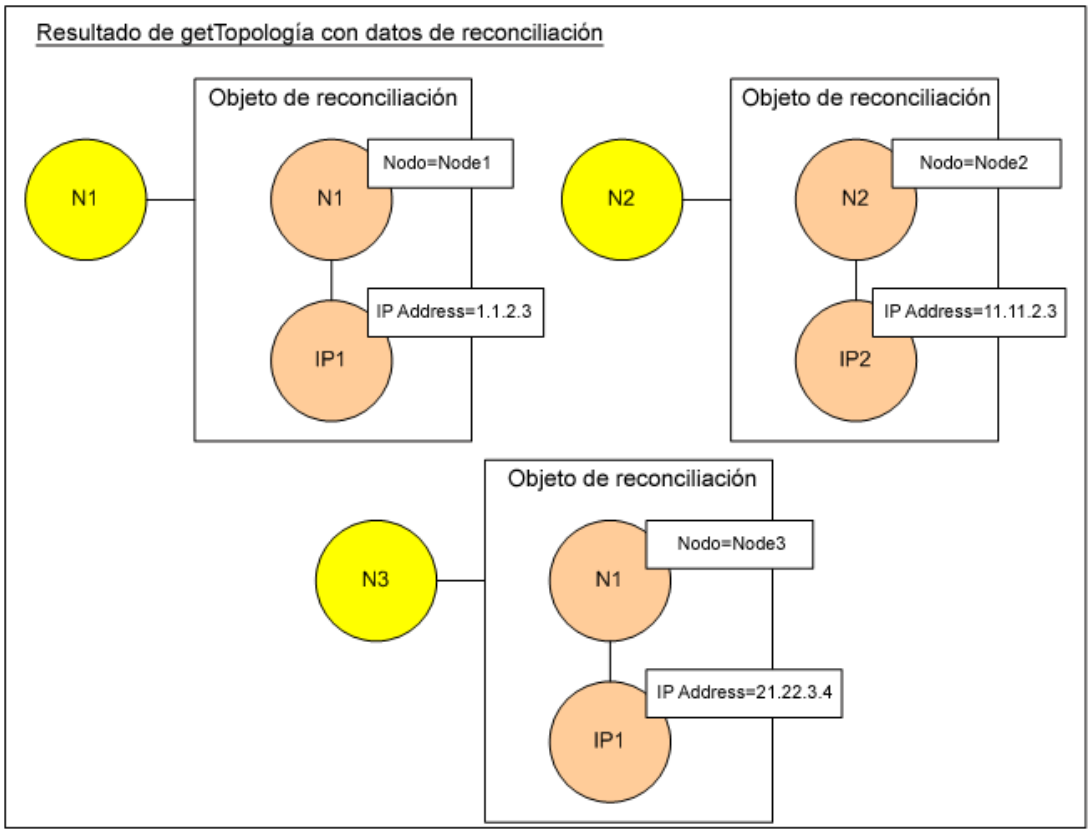
2. Federation Framework ejecuta el subgráfico UCMDB para solicitar la topología y recibe los siguientes resultados:



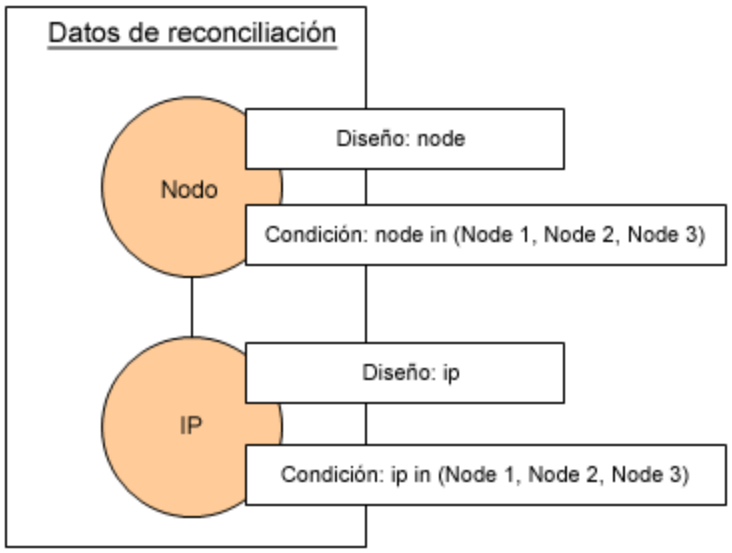
3. Federation Framework solicita, desde el motor de asignación adecuado, los datos de reconciliación del primer repositorio de datos (UCMDB) que contiene la información para conectar entre los datos recibidos de dos repositorios de datos. En este caso, los datos de reconciliación son:



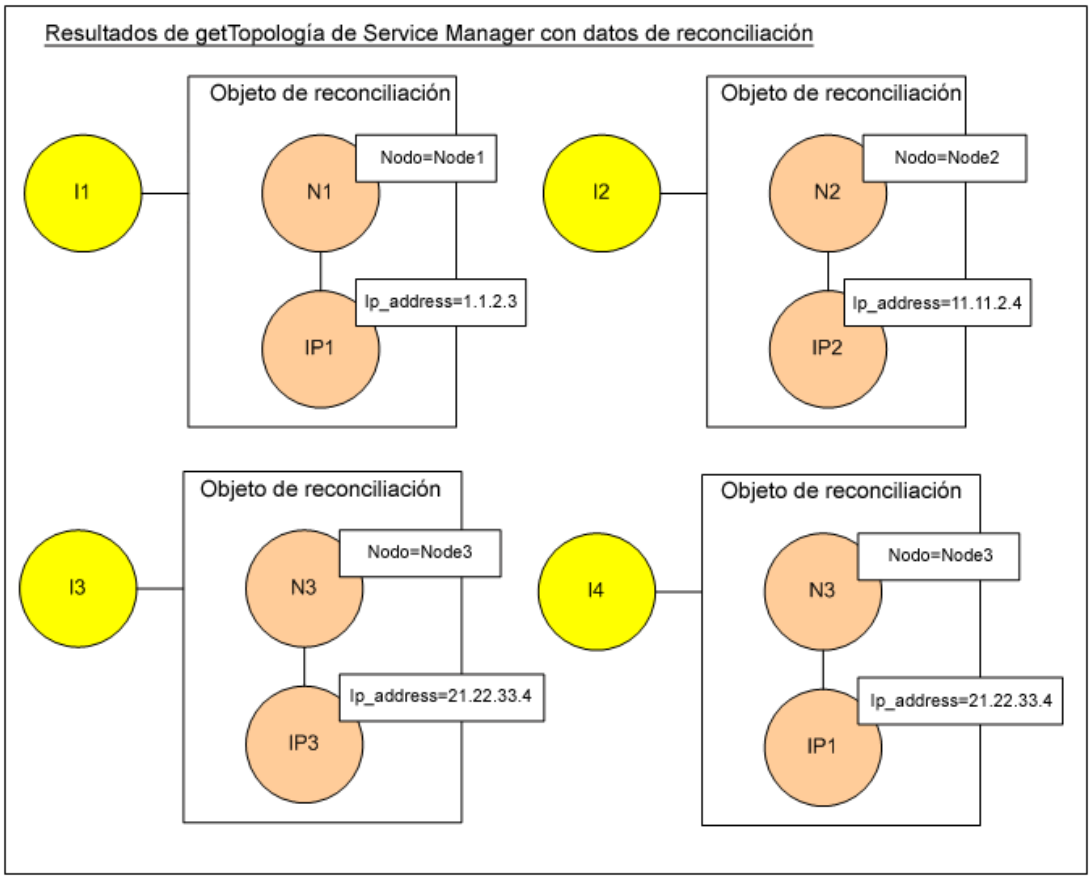
4. Federation Framework crea una consulta de topología de un nodo con las condiciones Node e ID en ella del resultado anterior (`node` en H1, H2, H3) y ejecute esta consulta con los datos de reconciliación necesarios en UCMDB. El resultado incluye los CI Node relevantes para la condición ID y el objeto de reconciliación adecuado para cada CI:



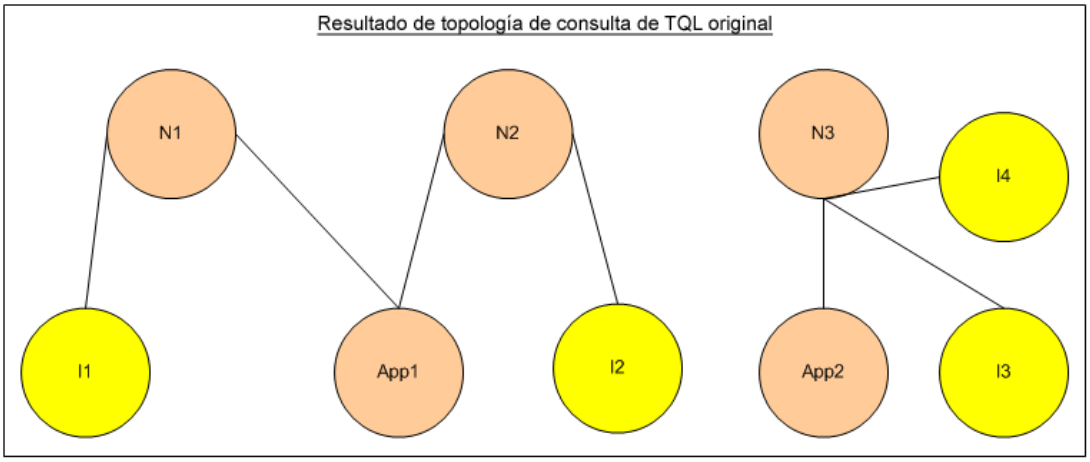
5. Los datos de reconciliación de ServiceCenter deben contener una condición para `node` e `ip` derivada de los objetos de reconciliación recibidos de UCMDB:



6. Federation Framework ejecuta el subgráfico ServiceCenter con los datos de reconciliación para solicitar la topología y los objetos de reconciliación adecuados, y recibe los siguientes resultados:



7. El resultado después de la conexión en el motor de asignación y de crear relaciones virtuales es:



8. Federation Framework solicita el diseño del TQL original para las instancias recibidas de UCMDB y ServiceCenter.

Flujo de Federation Framework para Rellenado

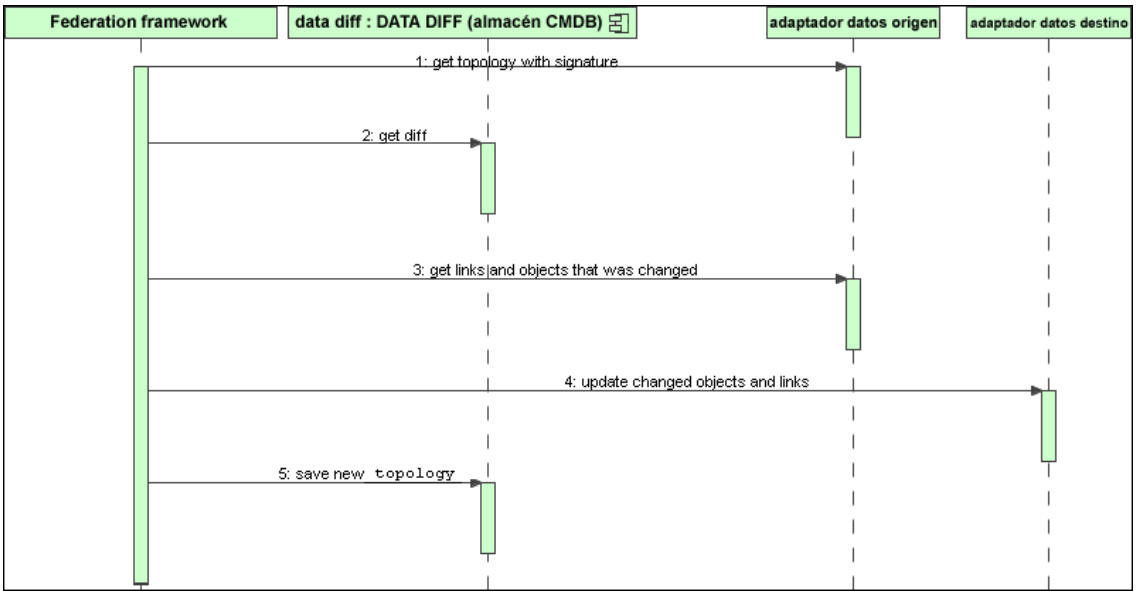
Esta sección incluye los siguientes temas:

Definiciones y términos

Firma. Denota el estado de las propiedades en el CI. Si se realizan cambios en los valores de la propiedad en un CI, la firma del CI también debe cambiarse. La firma del CI ayuda a detectar si un CI ha cambiado sin recuperar y comparar toda las propiedades del CI. El CI y la firma del CI se proporcionan mediante el adaptador adecuado. El adaptador responsable de cambiar la firma del CI cuando se modifican las propiedades del CI.

Diagrama de flujos

En el siguiente diagrama de secuencias se ilustra la interacción entre Federation Framework y los adaptadores de origen y destino en un flujo de rellenado:



1. Federation Framework recibe la topología para el resultado de la consulta del adaptador de origen. El adaptador reconoce la consulta por su nombre y la ejecuta en el repositorio de datos externo. El resultado de la topología contiene el ID y la firma de cada CI y la relación en el resultado. El ID es el ID lógico que define el CI como exclusivo en el repositorio de datos externo. La firma debe modificarse si se modifica el CI o la relación.
2. Federation Framework usa las firmas para comparar los resultados de las consultas de topología recién recibidos con los guardados, y determinar los CI que han cambiado.
3. Cuando Federation Framework encuentra los CI y las relaciones que han cambiado, llama al adaptador de origen con los Id. de los CI cambiados y las relaciones como un parámetro para recuperar su diseño completo.
4. Federation Framework envía la actualización al adaptador de destino. El adaptador de destino actualiza el origen de datos externo con los datos recibidos.
5. Después de la actualización, Federation Framework guarda el resultado de la última consulta.

Interfaces del adaptador

Esta sección incluye los siguientes temas:

Definiciones y términos

Relación externa. La relación entre dos tipos de CI externos admitidos por el mismo adaptador.

Interfaces del adaptador para consultas TQL federadas

Use la interfaz del adaptador adecuada para cada adaptador, como se indica a continuación.

- Una interfaz de topología de **un nodo** se usa cuando el adaptador no admite ninguna relación externa; es decir, el adaptador no va a recibir ninguna solicitud con más de un CI externo. Todas las interfaces OneNode se crean para simplificar el flujo de trabajo; para los casos en los que necesite usar una consulta más amplia, use la interfaz `DataAdapter`.
- Una **interfaz DataAdapter** se usa para definir los adaptadores que admiten consultas federadas complejas. La solicitud de reconciliación en estos adaptadores forma parte del único parámetro QueryDefinition. Estos adaptadores también pueden usarse para Rellenado.

Interfaces OneNode

Las siguientes interfaces tienen distintos tipos de datos de reconciliación:

- **OneNodeTopologyIdReconciliationDataAdapter.** Se usa si el adaptador admite un **TQL de un solo nodo** y el ID calcula la reconciliación entre los repositorios de datos.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Se usa si el adaptador admite un **TQL de un solo nodo** y la reconciliación entre los repositorios de datos se realiza mediante las propiedades de un CI.
- **OneNodeTopologyDataAdapter.** Se usa si el adaptador admite un **TQL de un solo nodo** y la reconciliación entre los repositorios de datos se realiza mediante la topología.

Interfaces del adaptador de datos

- **DataAdapter.** Use este adaptador para admitir consultas TQL federadas complejas. Permite la mayor diversidad.
- **PopulateDataAdapter.** Use este adaptador para admitir consultas TQL federadas complejas y los flujos de rellenado. En un flujo de rellenado, este adaptador recupera todo el conjunto de datos y permite que la sonda filtre la diferencia desde la última ejecución del trabajo.
- **PopulateChangesDataAdapter.** Use este adaptador para admitir consultas TQL federadas complejas y los flujos de rellenado. En un flujo de rellenado, este adaptador admite la recuperación sólo de los cambios que se produjeron desde la última ejecución del trabajo.

Nota: Al desarrollar un adaptador que pueda devolver grandes conjuntos de datos, es

importante permitir la fragmentación implementando la interfaz ChunkGetter. Consulte el documento Java del adaptador específico para obtener más información.

Interfaces adicionales

- **SortResultDataAdapter.** Se usa si se pueden ordenar los CI en el repositorio de datos externo.
- **FunctionalLayoutDataAdapter.** Se usa si se puede calcular el diseño funcional del repositorio de datos externo.

Interfaces del adaptador para la sincronización

- **SourceDataAdapter.** Se usa para los adaptadores de origen en flujos de rellenado.
- **TargetDataAdapter.** Se usa para los adaptadores de destino en los flujos de inserción de datos.

Depuración de recursos del adaptador

Esta tarea describe cómo utilizar la consola JMX para crear, ver y suprimir recursos de estado del adaptador (cualquier recurso creando mediante los métodos de manipulación de recursos en la interfaz DataAdapterEnvironment, que se guardan en la base de datos de UCMDB o la base de datos de la sonda) para fines de depuración y desarrollo.

1. Inicie el explorador web y especifique la dirección del servidor, del siguiente modo:
   - En el servidor UCMDB: `http://localhost:8080/jmx-console`
   - Para la sonda: `http://localhost:1977`

   Es posible que deba iniciar sesión con un nombre de usuario y una contraseña (los valores predeterminados son sysadmin/sysadmin).
2. Para usar la página JMX MBEAN View, realice una de las siguientes acciones:
   - En el servidor UCMDB: haga clic en **UCMDB:service=FCMDB Adapter State Resource Services**
   - En la sonda: haga clic en **type=AdapterStateResources**
3. Introduzca valores en las operaciones que desea utilizar y haga clic en **Invocar**.

Adición de un adaptador para un nuevo origen de datos externo

Esta tarea explica el modo de definir un adaptador para que admita un nuevo origen de datos externo.

Esta tarea incluye los siguientes pasos:

- "Requisitos previos" en la página siguiente
- "Definición de relaciones válidas para relaciones virtuales" en la página siguiente

1. ## Requisitos previos

   Clases de adaptador admitidas por el modelo para los CI y relaciones en el modelo de datos UCMDB. Como programador de adaptadores, debe:

   - tener conocimientos de la jerarquía de los tipos de CI de UCMDB para entender el modo en que los CIT externos están relacionados con los CIT de UCMDB.
   - modelar los CIT externos en el modelo de clase de UCMDB.
   - Añadir las definiciones de los nuevos tipos de CI y sus relaciones
   - definir relaciones válidas en el modelo de clase de UCMDB para las relaciones válidas entre clases internas del adaptador. (Los CIT pueden colocarse en cualquier nivel del árbol del modelo de clase de UCMDB).

   El modelo debe ser el mismo, independientemente del tipo de federación (sobre la marcha o réplica). Para ver información detallada acerca de la adición de nuevas definiciones de CIT al modelo de clase UCMDB, consulte "Cómo trabajar con el Selector de CI " en la publicación HP Universal CMDB – Guía de modelado.

   Para que el adaptador admita atributos federados en los CIT, añada este CIT a las clases admitidas con atributos admitidos y la regla de reconciliación para este CIT.

2. ## Definición de relaciones válidas para relaciones virtuales

   Nota: Esta sección sólo es relevante para la federación.

   Para recuperar los CIT federados que están conectados a los CIT del CMDB local, debe haber una definición de vínculo valida entre los dos CIT del CMDB.

   a. Cree un archivo XML de vínculos válidos que contiene estos vínculos (si aún no existen).

   b. Añada el archivo XML de vínculos al paquete del adaptador en la carpeta **\validlinks**. Para obtener más información, consulte "Administrador de paquetes" en *HP Universal CMDB – Guía de administración*.

   Ejemplo de definición de relaciones válidas:

   En el siguiente ejemplo, la relación de tipo `containment` entre instancias de tipo `node` e instancias de tipo `myclass1` es una definición de relaciones válida.

```
<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. **Definición de la configuración de un adaptador**

   a. Vaya a **Administración de adaptador**.

   b. Haga clic en el botón **Crear recurso nuevo**  .

   c. En el cuadro de diálogo Nuevo adaptador, seleccione **Integración** y **Adaptador Java**.

   d. Haga clic con el botón derecho en el adaptador que ha creado y seleccione **Editar adaptador** en el menú contextual.

   e. Edite las siguientes etiquetas XML:

```
<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="newAdapterIdName"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
description="Adapter Description" schemaVersion="9.0"
displayName="New Adapter Display Name">

<deletable>true</deletable>

<discoveredClasses>

<discoveredClass>link</discoveredClass>

<discoveredClass>object</discoveredClass>

</discoveredClasses>

<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
 AdapterService">

<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
 AdapterServiceParams" enableAging="true"
enableDebugging="false" enableRecording=
"false" autoDeleteOnErrors="success" recordResult="false"
maxThreads="1" patternType="java_adapter"
maxThreadRuntime="25200000">

<className>com.yourCompany.adapter.MyAdapter.MyAdapterClass
</className>

</params>
```

```
<destinationInfo
className="com.hp.ucmdb.discovery.probe.tasks.BaseDestinationDat-
a">

<!-- check -->

<destinationData name="adapterId"
description="">${ADAPTER.adapter_id}</destinationData>

<destinationData name="attributeValues"
description="">${SOURCE.attribute_values}</destinationData>

<destinationData name="credentialsId"
description="">${SOURCE.credentials_id}</destinationData>

<destinationData name="destinationId"
description="">${SOURCE.destination_id}</destinationData>

</destinationInfo>

<resultMechanism isEnabled="true">

<autoDeleteCITs isEnabled="true">

<CIT>link</CIT>

<CIT>object</CIT>

</autoDeleteCITs>

</resultMechanism>

</taskInfo>

<adapterInfo>

<adapter-capabilities>

<support-federated-query>

<!--<supported-classes/> <!—see the section about supported
classes-->

<topology>

<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 />

<parámetros>

<!--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>
```

Para obtener información detallada acerca de las etiquetas XML, consulte "Propiedades y etiquetas de configuración XML " en la página 182.

4. ## Definición de las clases admitidas

Defina las clases admitidas del código del adaptador implementando el método *getSupportedClasses()* o usando el archivo XML de patrón.

```
<supported-classes>
    <supported-class name="HistoryChange" is-derived="false" is-
reconciliation-supported="false" federation-not-supported="false"
is-id-reconciliation-supported="false">
    <supported-conditions>
        <attribute-operators attribute-name="change_create_time">
            <operator>GREATER</operator>
            <operator>LESS</operator>
            <operator>GREATER_OR_EQUAL</operator>
            <operator>LESS_OR_EQUAL</operator>
            <operator>CHANGED_DURING</operator>
        </attribute-operators>
    </supported-conditions>
 </supported-class>
```

name	Nombre del tipo de CI
is-derived	Especifica si esta definición incluye todos los secundarios heredados
is-reconciliation-supported	Especifica si esta clase se usa para la reconciliación

is-id-reconciliation-supported	Especifica si esta clase se usa para id-reconciliation
federation-not-supported	Especifica si este CIT no debe permitirse para la federación (bloqueando algunos CIT; por ejemplo, un CIT definido únicamente para la federación)
<supported-conditions>	Especifica las condiciones admitidas para cada atributo

5. ## Implementación del adaptador

   Seleccione la clase de implementación del adaptador correcta de acuerdo con sus capacidades definidas. La clase de implementación del adaptador implementa las interfaces adecuadas según las capacidades definidas.

   La compatibilidad de la reconciliación de adaptadores puede definirse según **global_id**, en cuyo caso **global_id** debe definirse como parte de los atributos de reconciliación en las clases admitidas por el adaptador. Si la compatibilidad de la reconciliación de adaptadores se define según **global_id**, **getTopologyWithReconciliationData()** debe devolver **global_id** como parte de las propiedades de los objetos de reconciliación. UCMDB utiliza **global_id** para la reconciliación de los resultados de federación para un CIT en lugar de la regla de identificación.

6. ## Definición de las reglas de reconciliación o implementación del motor de asignación

   Si el adaptador admite consultas TQL federadas, tiene tres opciones para definir el motor de asignación:

   - Use el motor de asignación predeterminado de CMDB 9.0x, que usa las reglas de reconciliación internas de CMDB para la asignación. Para usarlo, deje la etiqueta XML **<default-mapping-engine>** vacía.

     Para obtener más información, consulte "Archivo reconciliation_types.txt" en la página 133.

   - Use el motor de asignación de CMDB 8.0x. Para ello, utilice la siguiente etiqueta XML: **<default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>**

     Para obtener más información, consulte "Archivo reconciliation_rules.txt (para compatibilidad con versiones anteriores)" en la página 133.

   - Escriba su propio motor de asignación implementando la interfaz del motor de asignación y colocando el archivo JAR con el resto del código del adaptador. Para ello, utilice la siguiente etiqueta XML: **<default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>**

7. ## Adición de archivos Jar necesarios para la implementación en la ruta de clase

   Para implementar las clases, añada el archivo **federation_api.jar** a la ruta de clase de su editor de código.

8. ## Despliegue del adaptador

Implante el paquete del adaptador. Para obtener información general acerca de la implementación de un paquete, consulte "Administrador de paquetes" en la *HP Universal CMDB – Guía de administración*.

El paquete debe contener las siguientes entidades:

- Nueva definición de CIT (opcional):
- Usado sólo si el adaptador admite nuevos tipos de CI que aún no existen en UCMDB.
- Las nuevas definiciones de CIT se encuentran en la carpeta `class` del paquete.
- Nueva definición del tipo de datos (opcional):
- Usado sólo si los nuevos CIT requieren nuevos tipos de datos.
- Las nuevas definiciones de tipos de datos se encuentran en la carpeta `typedef` del paquete.
- Nueva definición de relaciones válidas (opcional):
- Usado sólo si el adaptador admite el TQL federado.
- Las nuevas definiciones de relaciones válidas se encuentran en la carpeta `validlinks` del paquete.
- El archivo XML de configuración del patrón debe encontrarse en la carpeta `discoveryPatterns` del paquete.
- **Descriptor.** Define las definiciones del paquete.
- Coloque sus clases compiladas (normalmente un archivo jar) en el paquete bajo la carpeta **adapterCode\<id_adaptador>**.

  Nota: El nombre de la carpeta `id_adaptador` tiene el mismo valor que en la configuración del adaptador.

- Si crea su propio archivo de configuración, debe colocar el archivo en el paquete bajo la carpeta **adapterCode\<id_adaptador>**.

9. ### Actualización del adaptador

   Se pueden realizar cambios en cualquiera de los archivos no binarios del adaptador en el módulo Administración de adaptador. Al realizar cambios en los archivos de configuración del módulo Administración de adaptador, el adaptador se vuelve a cargar con las nuevas configuraciones.

   También se pueden realizar actualizaciones editando los archivos en el paquete (archivos binarios y no binarios) y volviendo a implantar el paquete mediante el Administrador de paquetes. Para obtener más información, consulte "Despliegue de un paquete" en la *HP Universal CMDB – Guía de administración*.

Implementación del motor de asignación

La configuración del motor de asignación depende del motor de asignación que esté usando.

Esta tarea incluye los siguientes pasos:

- "Configure el archivo reconciliation_types.txt (para el motor de asignación predeterminado UCMDB 9.0x)" abajo
- "Configure el archivo reconciliation_rules.txt (para el motor de asignación de UCMDB 8.0x)" abajo

1. ## Configure el archivo reconciliation_types.txt (para el motor de asignación predeterminado UCMDB 9.0x)

   El archivo se usa para definir los tipos de CI usados para la reconciliación en el adaptador.

   Escriba cada uno de los tipos de CI usados para la reconciliación en una sola línea, como se muestra a continuación:

   ```
   node business_application
   ```

   Coloque el archivo en el paquete del adaptador de la carpeta **adapterCode\<AdapterID>\META-INF**. Para admitir la reconciliación de ID (reconciliación basada en la asignación de ID entre ID de CMDB en CMDB a un valor en la base de datos remota), debe asignar un atributo especial de CMDB denominado **cmdb_id** a una columna en la base de datos de tipo string (char, varchar) o byte[] (raw/bytes).

2. ## Configure el archivo reconciliation_rules.txt (para el motor de asignación de UCMDB 8.0x)

   Este archivo se usa para configurar las reglas de reconciliación. Cada fila del archivo representa una regla. Por ejemplo:

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

   El parámetro **reconcilition_type** se rellena con el tipo de CI en el que se realiza la reconciliación (el nombre de la clase de UCMDB que está conectado a la clase federada en el TQL).

   El parámetro **expression** es la lógica que decide si dos objetos de reconciliación son iguales (un objeto de reconciliación de UCMDB y el otro del adaptador federado).

   la expresión está compuesta de varios OR y AND.

   La convención relativa a los nombres de atributos en la parte de la expresión es **[className].[attributeName]**. Por ejemplo, el atributo **ip_address** de la clase **ip** se escribe **ip.ip_address**.

   Puede definir coincidencias ordenadas. La coincidencia ordenada comprueba la primera subexpresión OR. Si los objetos de reconciliación tienen el valor en los atributos de la subexpresión y el valor devuelto es falso (los objetos de reconciliación no son iguales) la segunda subexpresión OR no se compara.

   Para una comparación ordenada, use **ordered expression** en lugar de **expression**.

   El acento circunflejo (**^**) se usa para ignorar las mayúsculas y minúsculas durante la comparación.

   Los demás parámetros (**end1_type**, **end2_type** y **link_type**) sólo se usan si los datos de reconciliación contienen dos nodos, no sólo el nodo del tipo de reconciliación (los datos de reconciliación topológicos). En este caso, los datos de reconciliación son **end1_type -(link_type)> end2_type**.

   No hay necesidad de añadir el diseño relevante, ya que se recupera de la expresión.

Para realizar la reconciliación mediante el Id. de UCMDB, use **cmdb_id** como nombre del atributo en la expresión.

Coloque el archivo en el paquete del adaptador de la carpeta **adapterCode\<AdapterID>\ META-INF**.

Ejemplos:

- Sólo puede añadir una regla de reconciliación para un CIT de nodo. Esto se debe a que sólo los CIT de nodo tienen relaciones válidas con CIT externos. Por ejemplo, un CI de nodo en CMDB se compara con un CI de nodo en ServiceCenter mediante el atributo `node.name` o mediante el atributo `ip_address.name`.
- En este caso, la regla de reconciliación es una regla de topología y la expresión está ordenada. La regla realiza las siguientes comprobaciones en los CI que se están comparando:
  - Si el atributo `node.name` es igual, la regla compara los nodos.
  - Si el atributo `node.name` no es igual, la regla no compara los modos.
  - Si el atributo `node.name` es nulo en uno de los CI comparados, la regla compara el atributo `ip_address.name`. Si el atributo `ip_address.name` es igual, la regla compara los nodos.

Creación de un adaptador de ejemplo

En este ejemplo se ilustra el modo de crear un adaptador de ejemplo. Esta tarea incluye los siguientes pasos:

- "Selección de la lógica del adaptador" abajo
- "Carga del proyecto" abajo

1. ## Selección de la lógica del adaptador

   Al implementar un adaptador, es necesario elegir el modo de gestionar la lógica de la condición en la implementación (condiciones de la propiedad, condiciones de ID, condiciones de reconciliación y condiciones de vínculos).

   a. Recupere todos los datos en la memoria del adaptador y deje que seleccione o filtre las instancias de CI necesarias.

   b. Convierta todas las condiciones en el idioma de origen de los datos y deje que filtre y seleccione los datos. Por ejemplo:
      - Convierta la condición en una consulta SQL.
      - Convierta la condición en un objeto de filtro API de Java.

   c. Filtre algunos de los datos del servicio remoto y que el adaptador seleccione y filtre el resto.

   En el ejemplo MyAdapter, se usa la lógica de la opción *a*.

2. ## Carga del proyecto

   Copie los archivos en la carpeta **C:\hp\UCMDB\UCMDBServer\tools\ adapter-dev-kit\SampleAdapters** y siga las instrucciones que aparecen en los archivos

Léame.

Nota: Si usa un adaptador con conjuntos de datos grandes, puede ser necesario usar la caché y la indexación para mejorar el rendimiento para la federación.

La documentación de javadocs en línea está disponible en:

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

Propiedades y etiquetas de configuración XML

id="newAdapterIdName"		Define el nombre real del adaptador. Usado para búsquedas en carpetas y registros
displayName="New Adapter Display Name"		Define el nombre de la pantalla del adaptador, como aparece en la UI.
<className>...</className>		Define la interfaz del adaptador que implementa la clase Java.
<category >My Category</category>		Define la categoría del adaptador.
<parámetros>		Define las propiedades de la configuración que están disponibles en la UI al configurar un nuevo punto de integración.
	name	Nombre de la propiedad (usado principalmente por el código)
	description	Pista de visualización de la propiedad
	type	Cadena o número entero (use los valores válidos con las cadenas para Booleano).
	display-name	Nombre de la propiedad en la UI.
	mandatory	Especifica si esta propiedad de configuración es obligatoria para el usuario.
	order-index	Orden de colocación de la propiedad (pequeño = arriba)
	valid-values	Lista de los posibles valores inválidos separados por caracteres ';' (por ejemplo, valid-values="Oracle;SQLServer;MySQL" or valid-values="True;False").
<adapterInfo>		Contiene la definición de las capacidades y la configuración estática del adaptador.
	<support-federated-query>	Define este adaptador como capaz de federación.

	<one-node-topology>	Capacidad de las consultas federadas con un nodo de consulta federado.
	<pattern-topology>	Posibilidad de federar consultas complejas.
	<support-replication-data>	Define la capacidad de ejecutar inserción de datos y flujos de rellenado.
	<source>	Este adaptador puede usarse para flujos de rellenado.
	<push-back-ids>	Devolver el Id. global del CI a la columna global_id de la tabla (debe definirse en orm.xml). Este comportamiento puede sustituirse implementando el complemento **FcmdbPluginPushBackIds**.
	<changes-source>	Este adaptador puede usarse para flujos de cambios en el rellenado.
	<target>	Este adaptador puede usarse para los flujos de inserción de datos.
	<default-mapping-engine>	Permite definir un motor de asignación para el adaptador (de forma predeterminada, el adaptador usa el motor de asignación predeterminado). Para cualquier otro motor de asignación, introduzca el nombre de la clase de implementación del motor de asignación (para el motor de asignación UCMDB 8.0x usa: **com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine**)
	<removedAttributes>	Fuerza la eliminación de atributos específicos del resultado.
	<full-population-days-interval>	Especifica cuándo se ejecutará un trabajo de rellenado completo, en lugar de un trabajo diferencial (cada 'x' días). Usa el mecanismo de envejecimiento junto con el flujo de cambios.
	<adapter-settings>	Lista de ajustes del adaptador.
	<list.attributes.for.set>	Determina qué atributos sustituyen al valor anterior (si existe).

Capítulo 6

Desarrollo de los adaptadores de inserción

Este capítulo incluye:

Información general del desarrollo de los adaptadores de inserción

El adaptador genérico de inserción ofrece una plataforma que permite el desarrollo rápido de aplicaciones que insertan datos de UCMDB 9.0x en repositorios de datos externos (bases de datos y aplicaciones de terceros). El desarrollo de una integración personalizada basada en el adaptador genérico de inserción requiere:

- un archivo de asignación XML entre los tipos de vínculos de CI de UCMDB y los elementos de datos externos.
- una secuencia de comandos Jython para insertar los elementos de datos en el repositorio de datos externo.

Sincronización diferencial

Para que el adaptador de inserción admita la sincronización diferencial, la función **DiscoveryMain** debe devolver un objeto que implemente la interfaz **DataPushResults**, que contiene las asignaciones entre los Id. que recibe la secuencia de comandos Jython desde el XML y los Id. que crea la secuencia de comandos Jython en el equipo remoto. Los últimos Id. son de tipo **ExternalId**.

El comando **ExternalIdUtil.restoreExternal**, que recibe el Id. del CI en CMDB como parámetro, restaura el Id. externo del Id. del CI en CMDB. Por ejemplo, este comando puede utilizarse al realizar una sincronización diferencial y se recibe un vínculo en que uno de sus extremos no está en la operación masiva (ya se ha sincronizado).

Si el método **DiscoveryMain** de la secuencia de comandos Jython en la que se base el adaptador de inserción devuelve una instancia **ObjectStateHolderVector** vacía, el adaptador no admitirá la sincronización diferencial. Esto significa que aunque se esté ejecutando un trabajo de

sincronización diferencial, en realidad se está realizando una sincronización completa. Por tanto, no se puede actualizar ni suprimir ningún dato en el sistema remoto, ya que todos los datos se añaden a CMDB durante cada sincronización.

Preparación de los archivos de asignación

Nota: Para poder recuperar todos los CI y relaciones tal como son en CMDB sin asignarlos, no debe crear el archivo **mappings.xml**. Esta acción devolverá todos los CI y relaciones con todos sus atributos.

Hay dos formas diferentes de preparar los archivos de asignación:

- Puede preparar un único archivo de asignación global.

  Todas las asignaciones se colocan en un único archivo llamado **mappings.xml**.

- Puede preparar un archivo independiente para cada consulta de inserción.

  Cada archivo de asignación se denomina **<nombre de consulta>.xml**.

Para obtener más información, consulte "Esquema del archivo de asignación" en la página 193.

Esta tarea incluye los siguientes pasos:

- "Creación del archivo de asignación" abajo
- "Asignación de CI" abajo
- "Asignación de vínculos " en la página siguiente

1. ## Creación del archivo de asignación

   La estructura del archivo de asignación se crea de la forma siguiente:

```
<?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. ## Asignación de CI

   Hay dos maneras de asignar tipos de CI de CMDB:

   - Asigne un tipo de CI de manera que los CI de ese tipo y todos los tipos heredados se asignen de la misma manera:

```
<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>
```

- Asigne un tipo de CI de manera que solo se procesen los CI de ese tipo. Los CI de tipos heredados no se procesarán, a menos que también se asigne su tipo (de una de las dos maneras):

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

Un tipo de CI que se asigne indirectamente (uno de sus antecesores se asigna mediante **source_ci_type_tree**), también puede sustituir la asignación del principal haciendo que aparezca en su propio **source_ci_type_tree** o **source_ci_type**.

Se recomienda utilizar **source_ci_type_tree** siempre que sea posible. De lo contrario, los CI resultantes de un tipo de CI que no aparece en los archivos de asignación no se transferirán a la secuencia de comandos de Jython.

3. ## Asignación de vínculos

Hay dos maneras de asignar vínculos:

- Asigne un vínculo que también asigne todos los tipos de vínculo que heredan de ese vínculo específico:

```
<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>
```

- Asigne un vínculo que también asigne solamente ese tipo de vínculo específico y no todos los tipos de vínculo que heredan del mismo:

```
<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>
```

Escritura de secuencias de comandos Jython

La secuencia de comandos de asignación es una secuencia de comandos Jython regular y debería seguir las reglas de las secuencias de comandos Jython. Para obtener más información, consulte "Desarrollo de los adaptadores Jython" en la página 39.

La secuencia de comandos debería contener la función **DiscoveryMain**, que puede devolver un **OSHVResult** vacío o una instancia **DataPushResults** en caso de resultado satisfactorio.

Para informar de algún fallo, la secuencia de comandos debería generar una excepción, por ejemplo:

```
raise Exception('Failed to insert to remote UCMDB using
TopologyUpdateService. See log of the remote UCMDB')
```

En la función DiscoveryMain, los elementos de datos que se van a insertar o a eliminar de la aplicación externa se pueden obtener de la forma siguiente:

```
# 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()
```

El objeto cliente para la aplicación externa se puede obtener de la forma siguiente:

```
oracleClient = Framework.createClient()
```

Este objeto cliente usa automáticamente el ID de credenciales, el nombre de host y el número de puerto que pasó el adaptador a través de Framework.

Si tiene que usar los parámetros de conexión definidos para el adaptador (para obtener más información, consulte el paso "Edite el archivo discoveryPatterns\push_adapter.xml." en "Construcción de un paquete de adaptadores " en la página 192), utilice el código siguiente:

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

Por ejemplo:

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

Esta sección también incluye:

- "Trabajo con los resultados de la asignación" abajo
- "Manejo de la conexión de prueba en la secuencia de comandos" en la página 190

Trabajo con los resultados de la asignación

El adaptador genérico de inserción crea cadenas XML que describen los datos que se van a agregar, actualizar o eliminar del sistema de destino. La secuencia de comandos Jython tiene que analizar este XML y, a continuación, realizar la operación de agregar, actualizar o eliminar en el destino.

En el XML de la operación de adición que recibe la secuencia de comandos Jython, el atributo `mamId` para los objetos y vínculos siempre es el identificador UCMDB del objeto o vínculo original antes de que su tipo, atributo o cualquier otra información se modifique al esquema del sistema remoto.

En el XML de las operaciones de actualización o supresión, el atributo `mamId` de cada objeto o vínculo contiene la representación de cadena del mismo `ExternalId` que se devolvió de la secuencia de comandos Jython de la anterior sincronización.

En el XML, el atributo `id` de un CI contiene `cmdbId` como Id. externo o el `ExternalId` de ese CI si el CI obtuvo un `ExternalId` cuando se envió el CI a la secuencia de comandos. Los campos `end1Id` y `end2Id` del vínculo contienen para cada extremo del vínculo el `cmdbId` como Id. externo o el `ExternalId` de ese extremo del vínculo si el CI que está en el extremo del vínculo obtuvo un `ExternalId` cuando se envió a la secuencia de comandos.

Al procesar los CI en la secuencia de comandos de Jython, el valor devuelto de la secuencia de comandos es una asignación del Id. de CMDB del CI y el Id. proporcionado (el Id. dado a cada CI de la secuencia de comandos). Si se inserta un CI por primera vez, el Id. que está en el XML de ese CI es el Id. de CMDB. Si no se inserta el CI por primera vez, el Id. del CI es el mismo Id. que se dio a ese CI en la secuencia de comandos cuando se insertó por primera vez.

El Id. se recupera de la secuencia de comandos XML de CI de la manera siguiente:

1. En el CI Element del XML, recupere el Id. del atributo id. Por ejemplo: `id = objectElement.getAttributeValue('id')`.
2. Tras recuperar el Id. del XML, restaure el Id. del atributo (cadena). Por ejemplo: `objectId = CmdbObjectID.Factory.restoreObjectID(id)`.
3. Compruebe si el `objectId` recibido en el paso anterior es el Id. de CMDB. Para hacer esto, puede comprobar si el `objectId` tiene el nuevo Id. que le proporciona la secuencia de comandos. En caso afirmativo, el Id. devuelto no es el Id. de CMDB. Por ejemplo: `newId = objectId.getPropertyValue(<el nombre del atributo id proporcionado por la secuencia de comandos>)`.

   Si `newId` es nulo, el Id. devuelto en el XML es un Id. de CMDB.
4. Si el Id. es un Id. de CMDB (es decir, `newId` es nulo), realice lo siguiente (si el Id. no es un Id. de CMDB, vaya al paso 5):

   a. Cree una propiedad para ese CI que contiene el nuevo Id. Por ejemplo: `propArray = [TypesFactory.createProperty('<el nombre del atributo id`

```
proporcionado por la secuencia de comandos>', '<nuevo Id.>')].
```

b. Cree un `externaId` para ese CI. Por ejemplo:
```
cmdbId = extI.getPropertyValue('internal_id')
className = extI.getType()
externalId = ExternalIdFactory.createExternalCiId(className,
propArray)
```

c. Asigne el Id. de CMDB al externalId recién creado (y en el próximo paso, devuelva esa asignación al adaptador). Por ejemplo: `objectMappings.put(cmdbId, externalId)`

d. Cuando se asignan todos los CI y los vínculos:
```
updateResult = DataPushResultsFactory.createDataPushResults
(objectMappings, linkMappings);
return updateResult
```

5. Si el Id. es el nuevo Id. (es decir, `newId` no es nulo), el `externalId` es el `newId`.

Ejemplo de resultado XML

```
<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>
```

Nota: En caso de que `datatype="BYTE"`, el valor del resultado devuelto es una **cadena** que se genera como: `new String([the byte array attribute])`. El `objeto byte[]` puede reconstruirse mediante: `<la cadena recibida>.getBytes()`. Puede haber diferencias en la configuración regional predeterminada entre el servidor y la sonda. En este caso, la reconstrucción se hará según la configuración regional predeterminada del servidor.

Manejo de la conexión de prueba en la secuencia de comandos

Se puede invocar una secuencia de comandos Jython para probar la conexión con una aplicación externa. En este caso, el atributo de destino `testConnection` será `true`. Este atributo se puede obtener de Framework de la forma siguiente:

```
testConnection = Framework.getTriggerCIData('testConnection')
```

Al ejecutarse en modo de conexión de prueba, la secuencia de comandos generaría una excepción si no se puede establecer una conexión a la aplicación externa. En caso contrario, si la conexión se realiza correctamente, la función **DiscoveryMain** debería devolver un **OSHVResult** vacío.

Admisión de la sincronización diferencial

Importante: Si est implementando la sincronización diferencial en un adaptador existente que se creó en la versión 9.00 o 0.01, debe usar el archivo push-adapter.zip de la versión 9.02 o posterior para recrear su paquete de adaptador. Para obtener más información, consulte "Construcción de un paquete de adaptadores " en la página 192.

Esta tarea permite que el adaptador de inserción realice la sincronización diferencial. Para obtener más información, consulte "Sincronización diferencial " en la página 184.

La secuencia de comandos Jython devuelve el objeto **DataPushResults** que contiene dos asignaciones Java: una para las asignaciones de ID de objetos (las claves y valores son objetos de tipo ExternalCiId) y otra para los ID de vínculos (las claves y valores son objetos de tipos ExternalRelationId).

- Añada las siguientes instrucciones **from** a la secuencia de comandos Jython:

```
from com.hp.ucmdb.federationspi.data.query.types import
ExternalIdFactory

from com.hp.ucmdb.adapters.push import DataPushResults

from com.hp.ucmdb.adapters.push import DataPushResultsFactory

from com.mercury.topaz.cmdb.server.fcmdb.spi.data.query.types import
ExternalIdUtil
```

- Utilice la clase de fábrica **DataPushResultsFactory** para obtener el objeto **DataPushResults** de la función **DiscoveryMain**.

```
# Create the UpdateResult object

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

- Utilice los comandos siguientes para crear asignaciones Java para el objeto **DataPushResults**:

```
# Prepare the maps to store the mappings if IDs

objectMappings = HashMap()

linkMappings = HashMap()
```

- Utilice la clase **ExternalIdFactory** para crear los siguientes ID de ExternalId:
  - ExternalId para objetos o vínculos que se originan en un CMDB (por ejemplo, todos los CI de una operación de adición proceden de CMDB):

```
externaCIlId = ExternalIdFactory.createExternalCmdbCiId(ciType,
ciIDAsString)

externalRelationId =
ExternalIdFactory.createExternalCmdbRelationId(linkType,
end1ExternalCIId, end2ExternalCIId, linkIDAsString)
```

  - ExternalId para objetos o vínculos que no se originan en un CMDB (por lo general, cada operación de actualización o supresión contiene dichos objetos):

```
myIDField = TypesFactory.createProperty("systemID", "1")

myExternalId = ExternalIdFactory.createExternalCiId(type,
myIDField)
```

> **Nota:** Si la secuencia de comandos Jython ha actualizado la información existente y el ID del objeto (o vínculo) cambia, debe devolver una asignación entre el ID externo anterior y el nuevo.

- Utilice los métodos **restoreCmdbCiIDString** o **restoreCmdbRelationIDString** de la clase **ExternalIdFactory** para recuperar la cadena UCMDB ID de un ID externo de un objeto o vínculo

que se originó en UCMDB.

- Utilice los métodos **restoreExternalCiId** y **restoreExternalRelationId** de la clase **ExternalIdUtil** para restaurar el objeto **ExternalId** desde el valor del atributo mamId del XML de las operaciones de actualización o supresión.

> **Nota:** Los objetos **ExternalId** son en realidad una matriz de propiedades. Esto significa que puede usar un objeto **ExternalId** para almacenar toda la información que puede necesitar que identificará los datos en el sistema remoto.

Construcción de un paquete de adaptadores

1. Extraiga el contenido de **C:\hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip** en una carpeta temporal. En el paquete de adaptador, el archivo **sql_queries** ubicado en **adapterCode > PushAdapter > sqlTablesCreation,** contiene las consultas necesarias para crear tablas en un esquema nuevo en Oracle para probar el adaptador. Las tablas corresponden al archivo **adapterCode\<ID de adaptador>\mappings\mappings.xml**.

   > **Nota:** El archivo **sql_queries** no es necesario para el adaptador. Solo es un ejemplo.

2. Edite el archivo **discoveryPatterns\push_adapter.xml**.
   a. Modifique la etiqueta <pattern> con un nuevo ID y etiqueta de visualización. Sustituya:

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

   con:

   ```
   <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. Actualice la lista de parámetros, para que esta lista refleje los atributos de conexión requeridos. No suprima el atributo **probeName**.

3. Cambie el nombre de la carpeta **adapterCode\PushAdapter** con el ID del adaptador usado en el paso anterior (por ejemplo, **adapterCode\MyPushAdapter**).

4. En el archivo **discoveryScript** hay una secuencia de comandos **script pushScript.py** que inserta los CI y los vínculos en una base de datos Oracle externa. Sustituya **discoveryScripts\pushScript.py** con la secuencia de comandos que ha escrito (para obtener más información, consulte "Escritura de secuencias de comandos Jython " en la página 187). Si cambia el nombre de la secuencia de comandos, la propiedad `jythonScript.name` de **adapterCode\<adapter ID>\push.properties** debería actualizarse en consecuencia.

5. El archivo **adapterCode\<ID de adaptador>\mappings\mappings.xml**, que se encuentra en **adapterCode\<ID de adaptador>\mappings**, es un archivo de asignación de ejemplo que

contiene una asignación de:

- Tipo de CI de nodo con todos los tipos de CI que heredan del mismo.
- Tipo de CI UNIX con todos los tipos de CI que heredan del mismo.
- Tipo de vínculo de dependencia con todos los tipos de vínculo que heredan del mismo.
- Tipo de vínculo de conversación sin los tipos de vínculo heredados que heredan del mismo.

Este ejemplo de asignación corresponde al ejemplo de las tablas creadas en ORACLE en el archivo **sql_queries** (vea el paso 1).

Sustituya el archivo **adapterCode\<adapter ID>\mappings\mappings.xml** con los archivos de asignación que haya preparado (para obtener más información, consulte "Preparación de los archivos de asignación" en la página 185.

Si desea usar un archivo de asignación para cada método de TQL, asigne el nombre del TQL correspondiente a cada archivo XML, seguido por **.xml**. En este caso, el archivo **mappings.xml** se utilizará de forma predeterminado, si no se encuentra un archivo de asignación específico para el nombre de TQL actual. El nombre del archivo de asignación predeterminado se puede modificar cambiando la propiedad `mappingFile.default` en **adapterCode\<adapter ID>\push.properties**.

Esquema del archivo de asignación

Nombre y ruta del elemento	Descripción	Atributos
integración	Define el contenido de asignación del archivo. Debe ser el bloque más exterior del archivo excepto para la línea del comienzo y cualquier comentario.	
info (integration)	Define la información sobre los repositorios de datos que se están integrando.	
source (integration > info)	Define la información sobre el repositorio de datos de origen.	1. **Nombre:** type **Descripción:** Nombre del repositorio de datos de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** versions **Descripción:** Versiones de los repositorios de datos de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena

Nombre y ruta del elemento	Descripción	Atributos
		3. **Nombre:** vendor **Descripción:** Proveedor del repositorio de datos de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena
target (integration > info)	Define la información sobre el repositorio de datos de destino.	1. **Nombre:** type **Descripción:** Nombre del repositorio de datos de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** versions **Descripción:** Versiones del repositorio de datos de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena 3. **Nombre:** vendor **Descripción:** Proveedor del repositorio de datos de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena
targetcis (integration)	Elemento contenedor para todas las asignaciones de CIT.	
source_ci_type_tree (integration > targetcis)	Define un CIT de origen y todos los tipos de CI que heredan del mismo.	1. **Nombre:** name **Descripción:** Nombre del CIT de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** mode **Descripción:** El tipo de actualización requerido para el tipo de CI actual. **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes: a. **insert:** Utilícela solo si el CI todavía no existe. b. **update:** Utilícela solo si se sabe que el CI existe. c. **update_else_insert:** Si el CI existe, actualícelo; en caso contrario, cree un nuevo CI. d. **ignore:** No haga nada con este tipo de CI.
source_ci_type	Define un CIT de origen sin los tipos de	1. **Nombre:** name **Descripción:** Nombre del CIT de origen.

Nombre y ruta del elemento	Descripción	Atributos
(integration > targetcis)	CI que heredan del mismo.	**¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** mode **Descripción:** El tipo de actualización requerido para el tipo de CI actual. **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes: a. **insert:** Utilícela solo si el CI todavía no existe. b. **update:** Utilícela solo si se sabe que el CI existe. c. **update_else_insert:** Si el CI existe, actualícelo; en caso contrario, cree un nuevo CI. d. **ignore:** No haga nada con este tipo de CI.
target_ci_type (integration > targetcis > source_ci_type -O BIEN- integration > targetcis > source_ci_type_ tree)	Define un CIT de destino.	1. **Nombre:** name **Descripción:** Nombre del tipo de CI de destino. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** schema **Descripción:** El nombre del esquema que se usará para almacenar este tipo de CI en el destino. **¿Es necesario?:**No requerido **Tipo:** Cadena 3. **Nombre:** namespace **Descripción:** Indica el espacio de nombres de este tipo de CI en el destino. **¿Es necesario?:**No requerido **Tipo:** Cadena
targetprimarykey (integration > targetcis > source_ci_type) -O BIEN- (integration > targetcis > source_ci_type_tree -O BIEN- (integration > targetrelations > link)	Identifica los atributos de clave principal del CIT de destino.	

Nombre y ruta del elemento	Descripción	Atributos
-O BIEN- (integration > targetrelations > source_link_type_tree)		
pkey (integration > targetcis> source_ci_type > targetprimarykey -O BIEN- integration > targetcis > source_ci_type_tree > targetprimarykey -O BIEN- (integration > targetrelations > link > targetprimarykey) -O BIEN- integration > targetrelations > source_link_type_tree > targetprimarykey)	Identifica un atributo de clave principal. Necesario solo si el modo es **update** o **insert_else_update** **insert_else_update**.	
target_attribute (integration > targetcis> source_ci_type -O BIEN- integration > targetcis > source_ci_type_tree -O BIEN- integration > targetrelations > link -O BIEN- integration > targetrelations > source_link_type_tree)	Define el atributo del CIT de destino.	1. **Nombre:** name **Descripción:** Nombre del atributo del CIT de destino. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** datatype **Descripción:** Tipo de datos del atributo del CIT de destino. **¿Es necesario?:** Requerido **Tipo:** Cadena 3. **Nombre:** length **Descripción:** Para tipos de datos de cadena/carácter, tamaño de entero del atributo de destino. **¿Es necesario?:**No requerido **Tipo:** Entero 4. **Nombre**: option

Nombre y ruta del elemento	Descripción	Atributos
		Descripción: La función de conversión que se va a aplicar al valor. **Se requiere:** Falso **Tipo:** Una de las cadenas siguientes: a. **uppercase**: convierte a mayúsculas b. **lowercase**: convierte a minúsculas Si este atributo está vacío, no se aplicará ninguna función de conversión.
map (integration > targetcis > source_ci_type > target_attribute -O BIEN- integration > targetcis > source_ci_type_tree > target_attribute) -O BIEN- (integration > targetrelations > link > target_attribute -O BIEN- integration > targetrelations > source_link_type_tree > target_attribute)	Determina cómo obtener el valor del atributo del CIT de origen	1. **Nombre**: type **Descripción:** El tipo de asignación entre los valores de origen y destino. **Se requiere:** Requerido **Tipo:** Una de las cadenas siguientes: a.**direct**: especifica una asignación 1-a-1 del valor del atributo de origen al valor del atributo de destino. b. **compoundstring**: los elementos secundarios se unen en una única cadena y se establece el valor del atributo de destino. c. **childattr**: los elementos secundarios son uno o más atributos de los CIT secundarios. Los CIT secundarios se definen como los de la relación **composition** o **containment**. d. **constant**: cadena estática. 2. **Nombre**: value **Descripción:** Cadena constante para type=**constant** **Se requiere:** Sólo se requiere cuando type=**constant** **Tipo:** Cadena 3. **Nombre**: attr **Descripción:** Nombre de atributo de origen para type=**direct** **Se requiere:** Sólo se requiere cuando type=**direct** **Tipo:** Cadena
aggregation (integration > targetcis > source_ci_type > target_attribute > map	Determina cómo los valores de atributo de los CI secundarios de los CI de origen se combinan en un valor	**Nombre:** type **Descripción:** El tipo de la función de agregación **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes:

Nombre y ruta del elemento	Descripción	Atributos
-O BIEN- integration > targetcis > source_ci_type_tree > target_attribute > map -O BIEN- (integration > targetrelations > link > target_attribute > map -O BIEN- integration > targetrelations > source_link_type_tree > target_attribute > map) Sólo es válido cuando el tipo de asignación es **childattr**)	único que se asignará al atributo del CI de destino. Opcional.	• **csv**: concatena todos los valores incluidos en una lista (numérica o cadena/carácter) separada por comas. • **count**: devuelve un recuento numérico de todos los valores incluidos. • **sum**: devuelve un recuento numérico de todos los valores incluidos. • **average**: devuelve un promedio numérico de todos los valores incluidos. • **min**: devuelve el valor numérico/carácter más bajo incluido. • **max**: devuelve el valor numérico/carácter más alto incluido.
source_child_ci_type (integration > targetcis> source_ci_type > target_attribute > map -O BIEN- integration > targetcis > source_ci_type_tree > target_attribute > map -O BIEN- (integration > targetrelations > link > target_attribute > map -O BIEN- integration > targetrelations > source_link_type_tree > target_attribute >	Determina de qué CI conectado se toma el atributo secundario.	1. **Nombre**: name **Descripción:** El tipo de CI secundario **Se requiere:** Requerido **Tipo:** Cadena 2. **Nombre.** source_attribute **Descripción:** El atributo del CI secundario que se asigna. **Se requiere:** Se requiere solo si el tipo de agregación **childAttr** (que está en la misma ruta) no es =count. **Tipo:** Cadena

Nombre y ruta del elemento	Descripción	Atributos
map) Sólo es válido cuando el tipo de asignación es **childattr.**		
validation (integration > targetcis > source_ci_type > target_attribute > map -O BIEN- integration > targetcis > source_ci_type_tree > target_attribute > map -O BIEN- (integration > targetrelations > link > target_attribute > map -O BIEN- integration > targetrelations > source_link_type_tree > target_attribute > map) Sólo es válido cuando el tipo de asignación es **childattr**.	Permite la exclusión del filtrado de los CI secundarios del CI de origen basándose en valores de atributos. Se utiliza con el subelemento de agregación para conseguir granularidad de los atributos secundarios que se asignan al valor de atributo de los CIT de destino. Opcional.	1. **Nombre**: minlength **Descripción:** Excluye cadenas más cortas que el valor dado. **¿Es necesario?:**No requerido **Tipo:** Entero 2. **Nombre**: maxlength **Descripción:** Excluye cadenas más largas que el valor dado. **¿Es necesario?:**No requerido **Tipo:** Entero 3. **Nombre**: minvalue **Descripción:** Excluye números más pequeños que el valor especificado. **¿Es necesario?:**No requerido **Tipo:**Numérico 4. **Nombre**: maxvalue **Descripción:** Excluye números más grandes que el valor especificado. **¿Es necesario?:**No requerido **Tipo:**Numérico
targetrelations (integration)	Elemento contenedor para todas las asignaciones de relaciones. Opcional.	
source_link_type_tree (integration > targetrelations)	Asigna a una relación de destino un tipo de relación de origen sin los tipos que heredan del mismo. Solo obligatorio si **targetrelation** está presente.	1. **Nombre:** name **Descripción:** Nombre de la relación de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** target_link_type **Descripción:** Nombre de la relación de destino.

Nombre y ruta del elemento	Descripción	Atributos
		¿Es necesario?: Requerido **Tipo:** Cadena 3. **Nombre:** nameSpace **Descripción:** El espacio de nombres para el vínculo se creará en el destino. **¿Es necesario?:**No requerido **Tipo:** Cadena 4. **Nombre:** mode **Descripción:** El tipo de actualización requerido para el vínculo actual. **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes: ▪ **insert**: utilícela solo si el CI todavía no existe. ▪ **update**: utilícela solo si el CI se sabe que existe. ▪ **update_else_insert**: si el CI existe, actualícelo; en caso contrario, cree un nuevo CI. ▪ **ignore**: no hacer nada con este tipo de CI. 5. **Nombre:** source_ci_type_end1 **Descripción:** Tipo de CI End1 de la relación de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena 6. **Nombre:** source_ci_type_end2 **Descripción:** Tipo de CI End2 de la relación de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena
link (integration > targetrelations)	Asigna una relación de origen a una relación de destino. Solo obligatorio si **targetrelation** está presente.	1. **Nombre:** source_link_type **Descripción:** Nombre de la relación de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** target_link_type **Descripción:** Nombre de la relación de destino. **¿Es necesario?:** Requerido **Tipo:** Cadena

Nombre y ruta del elemento	Descripción	Atributos
		3. **Nombre:** nameSpace **Descripción:** El espacio de nombres para el vínculo se creará en el destino. **¿Es necesario?:**No requerido **Tipo:** Cadena 4. **Nombre:** mode **Descripción:** El tipo de actualización requerido para el vínculo actual. **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes: ▪ **insert**: utilícela solo si el CI todavía no existe. ▪ **update**: utilícela solo si el CI se sabe que existe. ▪ **update_else_insert**: si el CI existe, actualícelo; en caso contrario, cree un nuevo CI. ▪ **ignore**: no hacer nada con este tipo de CI. 5. **Nombre:** source_ci_type_end1 **Descripción:** Tipo de CI End1 de la relación de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena 6. **Nombre:** source_ci_type_end2 **Descripción:** Tipo de CI End2 de la relación de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena
target_ci_type_end1 (integration > targetrelations > link -O BIEN- integration > targetrelations > source_link_type_tree)	Tipo de CI End1 de la relación de destino.	1. **Nombre:** name **Descripción:**Nombre del tipo de CI End1 de la relación de destino. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** superclass **Descripción:** Nombre de la super clase del tipo de CI End1. **¿Es necesario?:**No requerido **Tipo:** Cadena
target_ci_type_end2 (integration >	Tipo de CI End2 de la relación de destino.	1. **Nombre:** name **Descripción:** Nombre del tipo de CI

Nombre y ruta del elemento	Descripción	Atributos
targetrelations > link -O BIEN- integration > targetrelations > source_link_type_tree)		End2 de la relación de destino. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** superclass **Descripción:** Nombre de la super clase del tipo de CI End2. **¿Es necesario?:**No requerido **Tipo:** Cadena

Esquema de resultados de asignación

Nombre y ruta del elemento	Descripción	Atributos
root	La raíz del documento de resultados.	
data (root)	La raíz de los datos propiamente dichos.	
objects (root > data)	El elemento raíz de los objetos que se van a actualizar.	
Objeto (root > data > objects)	Describe la operación de actualización para un único objeto y todos sus atributos.	1. **Nombre:** name **Descripción:** Nombre del tipo de CI **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** mode **Descripción:** El tipo de actualización requerido para el tipo de CI actual. **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes: a. **insert**: utilícela solo si el CI todavía no existe. b. **update**: utilícela solo si se sabe que el CI existe. c. **update_else_insert**: si el CI existe, actualícelo; en caso contrario, cree un nuevo CI. d. **ignore**: no haga nada con este tipo de CI. 3. **Nombre:** operation **Descripción:** La operación que se va a

Nombre y ruta del elemento	Descripción	Atributos
		realizar este CI. **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes: a. **add**: el CI debe agregarse b. **update**: el CI debe actualizarse c. **delete**: el CI debe eliminarse Si no se establece ningún valor, se usa el valor predeterminado **add**. 4. **Nombre:** mamId **Descripción:** El Id. del objeto en el CMDB de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena
field (root > data > objects> Object -O BIEN- root > data > links > link)	Describe el valor de un único campo de un objeto. El texto del campo es el nuevo valor del campo, y si el campo contiene un vínculo, el valor es el Id. de uno de los extremos. Cada Id. de extremo aparece como un objeto (bajo <objetos>).	1. **Nombre:** name **Descripción:** Nombre del campo. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** key **Descripción:** Especifica si este campo es una clave para el objeto. **¿Es necesario?:** Requerido **Tipo:**Booleano 3. **Nombre:** datatype **Descripción:** El tipo del campo. **¿Es necesario?:** Requerido **Tipo:** Cadena 4. **Nombre:** length **Descripción:** Para tipos de datos de cadena/carácter, es el tamaño de entero del atributo de destino. **¿Es necesario?:**No requerido **Tipo:** Entero
links (root > data)	El elemento raíz de los vínculos que se van a actualizar.	1. **Nombre:** targetRelationshipClass **Descripción:** El nombre de la relación (vínculo) del sistema destino. **¿Es necesario?:** Requerido **Tipo:** Cadena 2. **Nombre:** targetParent **Descripción:** El tipo del primer extremo del vínculo (elemento de primer nivel). **¿Es necesario?:** Requerido **Tipo:** Cadena

Nombre y ruta del elemento	Descripción	Atributos
		3. **Nombre:** targetChild **Descripción:** El tipo del segundo extremo del vínculo (elemento de segundo nivel). **¿Es necesario?:** Requerido **Tipo:** Cadena 4. **Nombre:** mode **Descripción:** El tipo de actualización requerido para el tipo de CI actual. **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes: a. **insert**: utilícela solo si el CI todavía no existe. b. **update**: utilícela solo si se sabe que el CI existe. c. **update_else_insert**: si el CI existe, actualícelo; en caso contrario, cree un nuevo CI. d. **ignore**: no haga nada con este tipo de CI. 5. **Nombre:** operation **Descripción:** La operación que se va a realizar este CI. **¿Es necesario?:** Requerido **Tipo:** Una de las cadenas siguientes: a. add: el CI debe agregarse b. update: el CI debe actualizarse c. delete: el CI debe eliminarse Si no se establece ningún valor, se usa el valor predeterminado **add**. 6. **Nombre:** mamId **Descripción:** El Id. del objeto en el CMDB de origen. **¿Es necesario?:** Requerido **Tipo:** Cadena

Capítulo 7

Desarrollo de adaptadores de inserción genéricos mejorados

Este capítulo incluye:

Información general del desarrollo de los adaptadores de inserción mejorados

Un adaptador de inserción mejorado es una estructura de datos que representa el resultado de la consulta TQL. Cada adaptador creado sobre el adaptador de inserción mejorado gestionará esta estructura de datos y lo insertará en su destino requerido.

La estructura de datos se denomina **ResultTreeNode (RTN)**. El RTN se crea según el archivo de asignación del adaptador y los resultados de la consulta TQL. Las consultas utilizadas para el adaptador de inserción mejorado deben estar basadas en la raíz, es decir, la consulta debe contener un nodo de consulta con el nombre de elemento **root**, o uno o más elementos de relación que empiezan por el prefijo **root**. Este CI o relación sirve como elemento raíz de la consulta. Para obtener más información, consulte "Inserción de datos" en *HP Universal CMDB – Guía de Administración de Data Flow*.

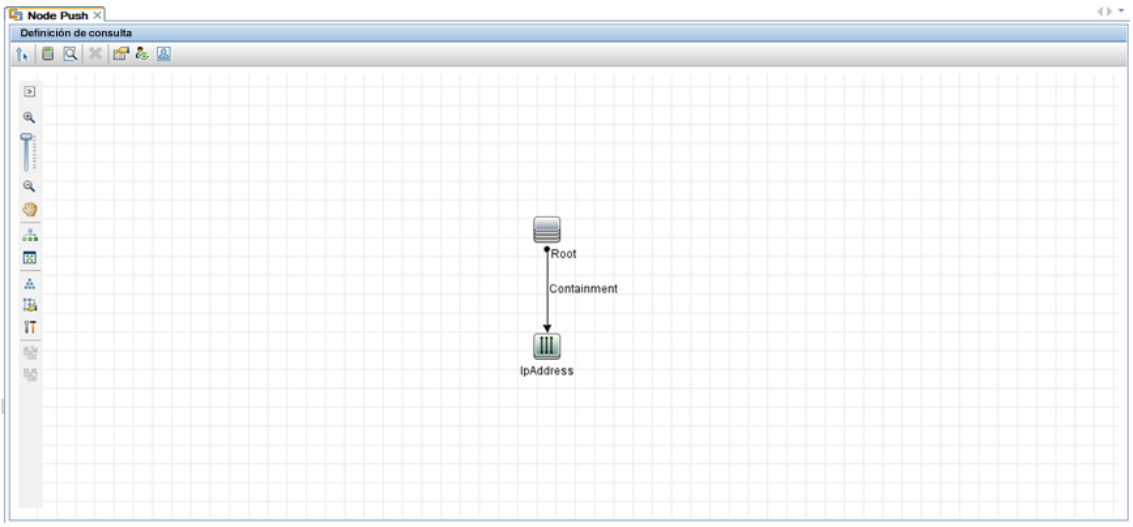
Hay dos pasos básicos implicados en el desarrollo del adaptador de inserción mejorado:

1. Implementación de la interfaz de PushConnector – esta interfaz recibe los datos que se deben agregar, actualizar y eliminar como una Lista de RTN y realizar la inserción en el destino.
2. Creación del archivo de asignación: el archivo de asignación determina la creación de la estructura del RTN, al asignar los CI y los atributos del resultado del TQL.

El archivo de asignación

El ejemplo siguiente muestra cómo crear el archivo de asignación.

En este ejemplo, simularemos una inserción de nodo y dirección IP. Crearemos una consulta TQL denominada: **Node Push**, de la manera siguiente:



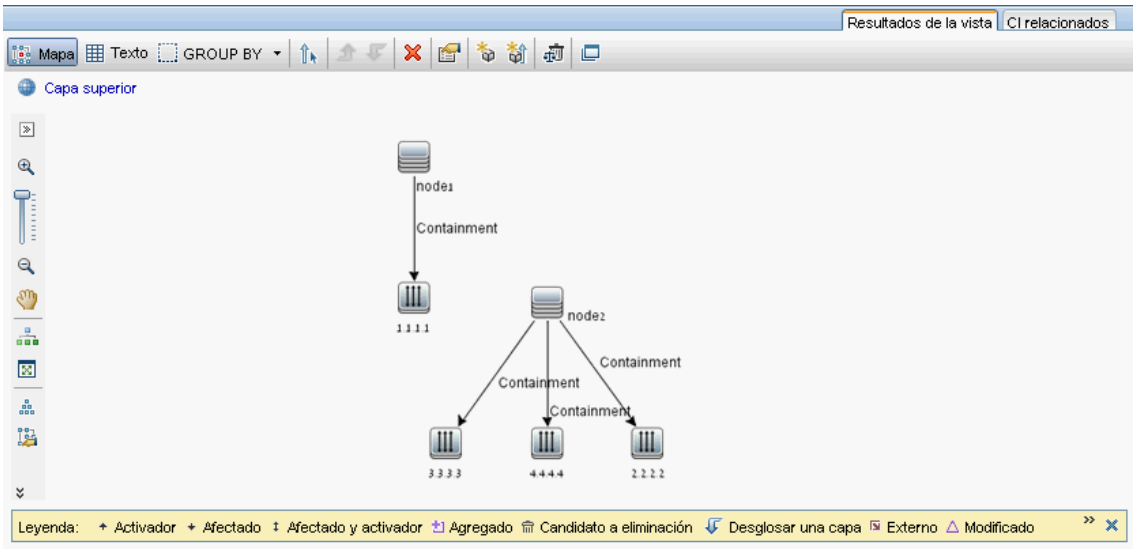
En el archivo de asignación creamos dos tipos de CI de destino: **Computer** e **IP**. Computer tiene una variable y dos atributos. IP tiene un atributo.

El siguiente es el archivo XML de asignación:

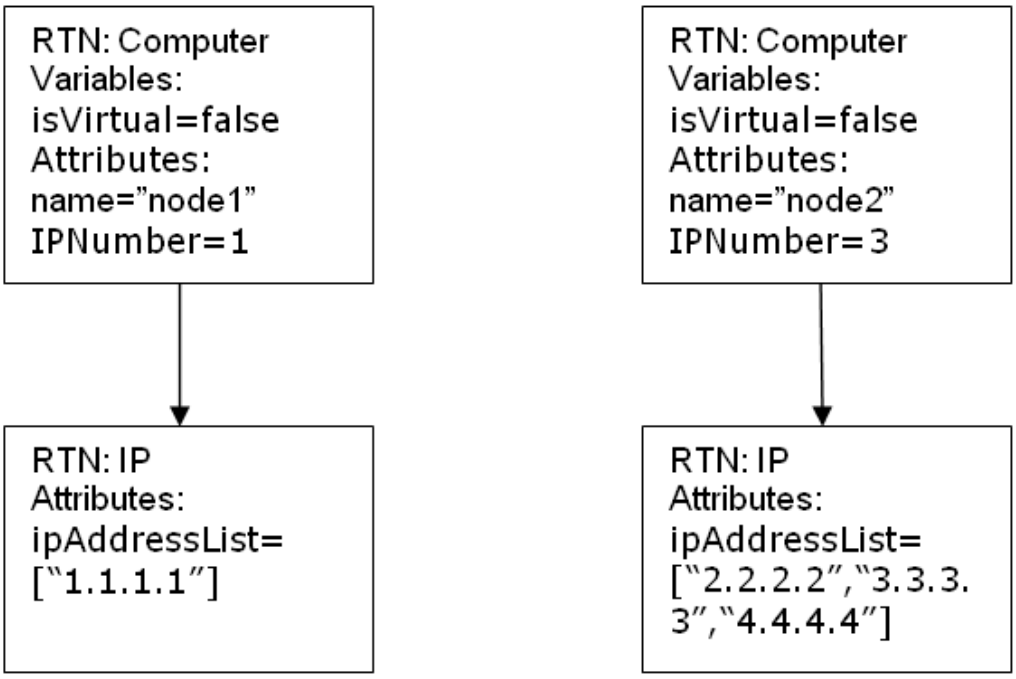
```
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://schemas.hp.com/ucmdb/1/pushAdap
ter">
        <info>
            <source name="UCMDB" versions="10.0" vendor="HP" />
            <target name="PushProduct" versions="9.3" vendor="HP" />
        </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>
```

Los resultados de la consulta aparecen de la manera siguiente:



Esta es la lista de RTN creada según este archivo de asignación:



Cada instancia raíz se asigna por separado mediante el archivo de asignación. Por lo tanto, en este ejemplo, PushConnector recibe una lista de dos raíces RTN.

Nota: El adaptador de inserción anterior tenía la capacidad de crear una asignación general para un tipo de CI. La nueva asignación de adaptador de inserción se realiza por consulta TQL. Mientras se ejecuta un trabajo de inserción que utiliza una consulta denominada **x**, el adaptador busca el archivo de asignación relevante (el que tiene el atributo: query-name=x).

Puede calcular los valores en el archivo de asignación mediante el lenguaje de secuencia de comandos groovy. Para obtener más información, consulte "Groovy Traveler" abajo.

Groovy Traveler

Podemos acceder a los resultados de la consulta de TQL de la manera siguiente:

- **Root[attr]** devuelve el atributo **attr** del elemento Root.
- **Root.Query_Element_Name** devuelve una lista de instancias de CI indicadas en Query_Element_Name del TQL y están vinculadas al CI raíz actual.
- **Root.Query_Element_Name[2][attr]** devuelve el atributo **attr** del tercer Query_Element_Name que está vinculado al CI raíz actual.
- **Root.Query_Element_Name*.getAt(attr)** devuelve una lista de los atributos **attr** de las instancias de CI indicadas como Query_Element_Name en el TQL y están vinculadas al CI raíz actual.

Hay atributos adicionales a los que Groovy Traveler puede acceder:

- **cmdb_id** – devuelve el Id. de UCMDB del CI o la relación como una cadena.
- **external_cmdb_id** – devuelve el Id. externo de UCMDB del CI o la relación como una cadena.
- **Element_type** – devuelve el tipo de elemento del CI o la relación como una cadena.

La etiqueta import:

```
<import>

<scriptFile path="mappings.scripts.PushFunctions"/>

</import>
```

Significa que se declara una importación para todas las secuencias de comandos groovy en el archivo de asignación. En este ejemplo, **PushFunctions** es un archivo de secuencia de comandos groovy que contiene algunas funciones estáticas y a las que podemos acceder durante la asignación (por ejemplo, value=" PushFunctions.foo()")

source_instance_type

La asignación se realiza por TQL y el valor de query-name es el TQL relacionado de la asignación actual. '*' significa que este archivo de asignación está asociado con todas las consultas de TQL que empiezan por el prefijo: **Node Push**.

```
<source_instance_type query-name="Node Push*" root-element-
name="Root">
```

La etiqueta source_instance_type designa el elemento Root que estamos asignando.

root-element-name debe ser exactamente el mismo que el nombre del elemento raíz en el TQL.

target_ci_type

Esta etiqueta se utiliza para la creación del RTN.

El atributo name representa el nombre de target_ci_type: name=Computer

El atributo **is-valid** es un valor booleano que se calcula durante la asignación y determina si el target_ci actual es válido. Los target_ci_types no válidos no se agregan al RTN. En este ejemplo, no queremos crear una instancia de target_ci_type para la que el atributo **root_ iscandidatefordeletion** de UCMDB sea true.

El target_ci_type puede tener variables que se calculan durante la asignación:

```
<variable name="vSerialNo" datatype="STRING" value="Root['serial_
number']"/>
```

La variable **vSerialNo** obtiene el valor de **serial_number** de la raíz actual.

La etiqueta **target_mapping** crea el atributo del RTN El resultado de la ejecución de la secuencia de comandos groovy en el campo **value** se asigna al atributo RTN.

```
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
```

SerialNo asigna el valor de la variable **vSerialNo**.

Es posible definir target_ci_type como subordinado de otro target_ci_type de la forma siguiente:

```
<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>
```

El RTN **Portfolio** tendrá el RTN subordinado denominado **Asset**.

for-each-source-ci

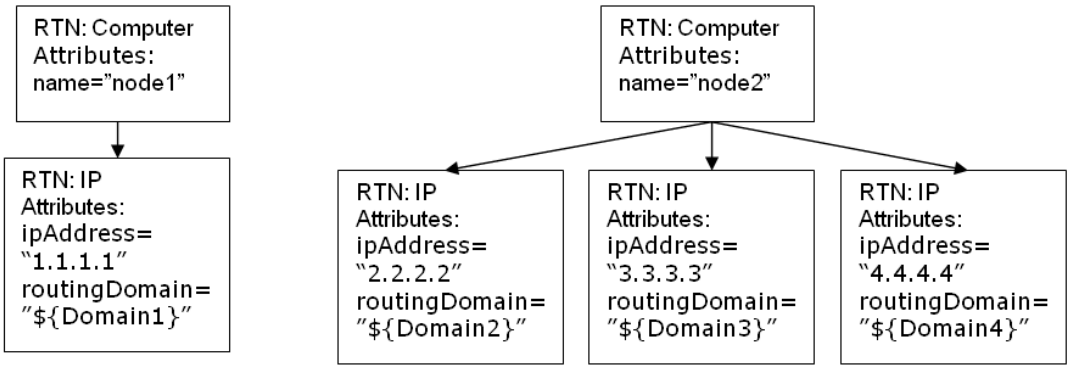
Esta etiqueta genera una lista de los CI específicos de la instancia raíz. Tiene los campos siguientes:

- source-cis=" " – la lista de los CI para los que se crea un CI de destino. Groovy Traveler define esta lista en el campo **Root.IpAddress**.
- count-index=" " – variable que contiene el índice del CI en la iteración actual del bucle.
- var-name=" " – nombre del CI en la iteración actual del bucle.

Vamos a modificar nuestro archivo de asignación de ejemplo:

```
<target_ci_type name="Computer">
  <target_mapping name="name" datatype="String" value="Root[`name'] " />
   <for-each-source-ci count-index="i" var-name="currIP" source-cis="Root.IpAddress" >
     <target_ci_type name="IP">
         <target_mapping name="ipAddress" datatype="STRING"
     value="Root.IpAddress[i][`name']"/>
         <target_mapping name="routingDomain" datatype="STRING"
     value="currIP['routing_domain']"/>
   </target_ci_type>
 </for-each-source-ci>
 </target_ci_type>
```

La lista de RTN que construirá según este archivo de asignación tendrá un aspecto como el siguiente:



dynamic_mapping

Esta etiqueta agrega la capacidad de crear una asignación de datos del almacén de datos de destino durante la creación de la estructura de RTN.

Ejemplo: Suponga que el destino es una base de datos con una tabla denominada **Computer** que tiene una columna **id** y una columna **name** que está correlacionada con **Node.name** en UCMDB. Ambas columnas son únicas. Además, la base de datos tiene una tabla denominada **IP** que tiene una clave referenciada con **parentID** en la tabla Computer. 'dynamic_mapping' puede crear una asignación que almacene el nombre y el Id. como <name,id>. Según este mapa, el adaptador puede emparejar los Id. con equipos y puede insertar el valor correcto en el atributo parentID de la tabla de IP. Puede utilizar este mapa para asignar un valor al atributo parentID mientras se crea el RTN.

map_property determina la asignación. dynamic_mapping se ejecuta una sola vez por cada fragmento.

```
<dynamic_mapping name="IdByName " keys-unique="true">
```

El atributo **name** representa el nombre del mapa. El atributo **keys-unique** indica si las claves son únicas (cada clave se asigna a un valor o a un conjunto de valores).

El nombre del mapa en este ejemplo es **IdByName** y tiene unas claves exclusivas. Para acceder al mapa en la secuencia de comandos, ejecute el comando siguiente:

```
DynamicMapHolder.getMap('IdByName')
```

Devuelve una referencia a ese mapa.

La etiqueta **map_property** crea la propiedad en la que se basa la asignación.

Ejemplo:

```
<map_property property-name="SQLQuery" datatype="STRING"
property-value="SELECT name, id FROM Computer"/>
```

En este ejemplo, el nombre de la propiedad es **SQLQuery** y su valor es una sentencia SQL que crea el mapa. La implementación de los métodos **retrieveUniqueMapping** y **retrieveNonUniqueMapping** para que la interfaz de PushConnector determine el contenido real del mapa devuelto.

Variables globales

Las variables globales siguientes son accesibles para la secuencia de comandos groovy del archivo de asignación:

- **Topology** – Tipo: Topología. Una instancia de la topología del fragmento actual.
- **QueryDefinition** - Tipo: QueryDefinition. Una instancia de la definición de consulta del TQL actual.
- **OutputCI** – Tipo: ResultTreeNode. El RTN del elemento raíz de la asignación de árbol actual.
- **ClassModel** – Tipo: ClassModel. Una instancia del modelo de clase.
- **CustomerInformation** – Tipo: CustomerInformation. Información sobre el cliente que ejecuta el trabajo.
- **Logger** – Tipo: DataAdapterLogger. Este registrador está disponible en el adaptador para escribir registros en el marco de registros de UCMDB.

Escritura de secuencias de comandos Groovy

En esta sección crearemos el archivo **PushFunctions.groovy**. Este archivo contendrá funciones estáticas que se utilicen durante la asignación de la instancia raíz.

```
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.
        }
}
```

Implementar la interfaz de PushConnector

La implementación debe admitir los pasos básicos siguientes:

```
public class PushExampleAdapter implements PushConnector
```

```
public class PushExampleAdapter implements PushConnector
```

```
{

public UpdateResult pushTreeNodes(ResultTreeNodeActionData
resultTreeNodes, QueryDefinition queryDefinition) throws
DataAccessException{

// 1. build an UpdateResult instance - the UpdateResult is used to
return mappings between the sent ids to the actual ids that entered
the data store.
// Also has an update status which allows to pass errors to the
statistics in the UI.
// 2. handle the data:
// a. handle data to add. Can be retrieved by:
resultTreeNodeActionData.getDataToAdd();
//  b. handle data to update.
//  c. handle data to delete.
// 3. Return the Update result.
        }

public void start(PushDataAdapterEnvironment env) throws
DataAccessException{
        // this method is called when the integration point created, or when
the adapter is reloaded
        //(i.e after changing one of the mapping files
        // and pressing 'save').

}



public void testConnection(PushDataAdapterEnvironment env) throws
DataAccessException {
                // this method is called when pressing the 'test connection' button
in the
                //creation of the integration point.
                // For example if we push data to RDBMS this method can create a
connection
```

```
                //to the database and will run a dummy SQL statement.
                // If it fails it writes an error message to the log and throws an
exception.
        }



Map<Object, Object> retrieveUniqueMapping(MappingQuery mappingQuery){
//This method will create the map according to the given mappingQuery.
It will be called in the
// mapping stage of the adapter execution, before the 'UpdateResult
pushTreeNodes' method.
// This method is called when the 'keys-unique' attribute of the
'dynamic_mapping' tag is true.
}

Map<Object, Set<Object>> retrieveNonUniqueMapping(MappingQuery
mappingQuery){
 // This method is called when the 'keys-unique' attribute of the
'dynamic_mapping' tag is false.
//  In this case a key can be mapped to several values.
}
}
```

Construcción de un paquete de adaptadores

El paquete de adaptador debe contener las siguientes carpetas:

- **adapterCode.** En esta carpeta, cree una carpeta llamada **PushExampleAdapter**, que contendrá el archivo .jar que creamos a partir de PushExampleAdapter.java. También contendrá una carpeta llamada **mappings**, donde podrá colocar el archivo de asignación creado anteriormente, **computerIPMapping.xml**. También debe contener otra carpeta llamada **scripts** que contiene el archivo **PushFunctions.groovy**.
- **discoveryConfigFiles.** Para contener archivos de configuración como, por ejemplo, los códigos de error utilizados al notificar un error mediante UpdateResult. En este ejemplo, la carpeta está vacía.
- **discoveryPatterns.** Para contener **push_example_adapter.xml**.
- **tql.** Para contener la consulta TQL creada para el ejemplo. Esta carpeta es opcional, pero cuando se despliega el paquete, el TQL se crea automáticamente.

Esquema del archivo de asignación

Nombre y ruta del elemento	Descripción	Atributos
Integración	Define el contenido de asignación del archivo. Debe ser el bloque más exterior del archivo excepto para la línea del	

Nombre y ruta del elemento	Descripción	Atributos
	comienzo y cualquier comentario.	
info (integration)	Define la información sobre los repositorios de datos que se están integrando.	
source (info)	El producto de origen	**Nombre:** name **Descripción:** el nombre del producto **¿Es necesario?:** necesario **Tipo:** Cadena
		Nombre: vendor **Descripción:** el proveedor del producto **¿Es necesario?:** necesario **Tipo:** Cadena
		Nombre: versions **Descripción:** la versión del producto **¿Es necesario?:** necesario **Tipo:** decimal
target (info)	El producto de destino	**Nombre:** name **Descripción:** el nombre del producto **¿Es necesario?:** necesario **Tipo:** Cadena
		Nombre: vendor **Descripción:** el proveedor del producto **¿Es necesario?:** necesario

Nombre y ruta del elemento	Descripción	Atributos
		Tipo: Cadena
		Nombre: versions **Descripción:** la versión del producto **¿Es necesario?:** necesario **Tipo:** decimal
Import (integration)	Un elemento contenedor para archivos de secuencia de comandos importados.	
scriptFile (integration>import)	Define el archivo de secuencia de comandos groovy que se van a importar.	Nombre: path Descripción: la ruta de acceso del archivo de secuencia de comandos ¿Es necesario?: necesario Tipo: cadena
Targetcis (integration)	Un elemento contenedor para tipos de CI de destino.	
Source_instance_type (integration>targetcis)	Define el tipo de instancia de CI de origen. Debe ser el elemento raíz tal como está definido en el TQL.	**Nombre:** query-name. **Descripción:** El nombre del TQL relevante **¿Es necesario?:** necesario **Tipo:** Cadena
		Nombre: name **Descripción:** El elemento raíz tal como está definido en el TQL. **¿Es necesario?:** necesario **Tipo:** Cadena
dynamic_mapping (integration>targetcis> source_instance_type)	Define un mapa que se crea una vez por fragmento.	**Nombre:** name **Descripción:** el

Nombre y ruta del elemento	Descripción	Atributos
		nombre del mapa **¿Es necesario?:** necesario **Tipo:** Cadena
		Nombre: keys-unique **Descripción:** Indica si las claves son únicas o no. **¿Es necesario?:** necesario **Tipo:** boolean
target_ci_type (integration>targetcis> source_instance_type) -OR- (integration>targetcis> source_instance_type> for-each-source-ci)	Define el tipo de CI de destino que se agregará al RTN.	**Nombre:** name **Descripción:** el nombre del tipo de CI de destino **¿Es necesario?:** necesario **Tipo:** Cadena
		Nombre: is-valid **Descripción:** comprueba si el CI de destino actual según la secuencia de comandos dada. **¿Es necesario?:** no requerido **Tipo:** Cadena (secuencia de comandos groovy)
Variable (target_ci_type)	Define una variable para el tipo de CI de destino.	**Nombre:** name **Descripción:** el nombre de la variable **¿Es necesario?:** necesario **Tipo:** Cadena

Nombre y ruta del elemento	Descripción	Atributos
		Nombre: datatype **Descripción:** el tipo de datos de la variable. **¿Es necesario?:** necesario **Tipo:** type-enum (puede ser uno de los siguientes: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Nombre: value **Descripción:** El valor que se asignará a la variable **¿Es necesario?:** necesario **Tipo:** secuencia de comandos groovy
target_mapping (target_ci_type)	Define un atributo para el tipo de CI de destino.	**Nombre:** name **Descripción:** el nombre del atributo **¿Es necesario?:** necesario **Tipo:** Cadena
		Nombre: datatype **Descripción:** el tipo de datos de la variable. **¿Es necesario?:** necesario **Tipo:** type-enum (puede ser uno de los siguientes: INTEGER,

Nombre y ruta del elemento	Descripción	Atributos
		LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Nombre: value **Descripción:** El valor que se asignará a la variable **¿Es necesario?:** necesario **Tipo:** secuencia de comandos groovy
		Nombre: ignore-on-null **Descripción:** Si el valor de asignación de destino es nulo y este atributo es TRUE, omita el atributo **¿Es necesario?:** no requerido **Tipo:** Booleano (secuencia de comandos groovy)
before-mapping (target_ci_type)	Una secuencia de comandos groovy que se ejecuta antes de la asignación del tipo de CI de destino.	
after-mapping (target_ci_type)	Una secuencia de comandos groovy que se ejecuta después de la asignación del tipo de CI de destino.	
for-each-source-ci (target_ci_type)	Define una iteración de los CI específicos de la instancia raíz.	**Nombre:** count-index **Descripción:** el índice del CI iterado actual **¿Es necesario?:** necesario **Tipo:** Cadena

Nombre y ruta del elemento	Descripción	Atributos
		Nombre: var-name **Descripción:** la variable que hace referencia al CI iterado actual. **¿Es necesario?:** no requerido **Tipo:** Cadena
		Nombre: source-cis **Descripción:** El nombre de CI de la consulta que debe iterarse **¿Es necesario?:** necesario **Tipo:** Cadena (secuencia de comandos groovy)

Utilización de las API

Capítulo 8

Introducción a las API

Este capítulo incluye:

Información general de las API

Las API se incluyen con HP Universal CMDB:

- **API de Java de UCMDB.** Explica cómo las herramientas de terceros o personalizadas pueden usar la API de Java para extraer datos y cálculos y para escribir datos en UCMDB (Universal Configuration Management database). Para obtener más información, consulte "API de HP Universal CMDB " en la página 222.
- **API del servicio web de UCMDB.** Permite escribir definiciones de elementos de configuración y relaciones topológicas a UCMDB (Universal Configuration Management database) y consultar la información con TQL y consultas ad hoc. Para obtener más información, consulte " API de servicio web de HP Universal CMDB" en la página 229.
- **API de servicio web de Data Flow Management.** Permite la administración de sondas, trabajos, activadores y credenciales para Data Flow Management. Para obtener más información, consulte "API de Data Flow Management " en la página 286.

Nota: Para obtener todo el valor de la documentación de API, es recomendable acceder a la documentación en línea. La versión PDF no tiene los vínculos a la documentación de API que se ha generado en formato HTML.

Capítulo 9

API de HP Universal CMDB

Este capítulo incluye:

Convenciones

Este capítulo utiliza las convenciones siguientes:

- **UCMDB** se refiere a la misma base de datos de Universal Configuration Management. HP Universal CMDB hace referencia a la aplicación.
- Los elementos de UCMDB y argumentos del método se deletrean en caso de que sean específicos en las interfaces.

Para consultar la documentación completa de las API disponibles, consulte HP UCMDB API Reference.

Estos archivos están ubicados en la carpeta siguiente:

\\<directorio raíz de UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Utilización de las API de HP Universal CMDB

Nota: Utilice este capítulo junto con el documento de las API Java, disponible en la biblioteca de documentación en línea.

La API de HP Universal CMDB se utiliza para integrar aplicaciones con Universal CMDB (CMDB). La API proporciona métodos para:

- agregar, eliminar y actualizar CI y relaciones de CMDB
- recuperar información sobre el modelo de clase

- recuperar información sobre el historial de UCMDB
- ejecutar escenarios what-if
- recuperar información sobre elementos de configuración y relaciones

Los métodos para recuperar información sobre elementos de configuración y relaciones suelen usar el Lenguaje de consulta de topología (TQL). Para obtener más información, consulte "Topology Query Language" en *HP Universal CMDB – Guía de modelado*.

Los usuarios de la API HP Universal CMDB deberían estar familiarizados con:

- El lenguaje de programación Java
- HP Universal CMDB

Esta sección incluye los siguientes temas:

- "Usos de la API" abajo
- "Permisos" abajo

Usos de la API

La API se usa para cumplir un número de requisitos empresariales. Por ejemplo, un sistema de terceros puede consultar el modelo de clase en busca de información sobre los elementos de configuración disponibles (CI). Para obtener información sobre los casos de uso, consulte "Casos de uso" en la página 227.

Permisos

El administrador proporciona credenciales de inicio de sesión para conectarse con la API. El cliente de la API necesita el nombre de usuario y la contraseña de un usuario de integración definido en CMDB. Estos usuarios no representan usuarios humanos de CMDB, sino más bien aplicaciones que se conectan a CMDB.

Precaución: El cliente de la API puede trabajar también con usuarios normales mientras estos tengan permiso de autenticación de API. Sin embargo, no se recomienda esta opción.

Para obtener más información, consulte "Creación de un usuario de integración" en la página 225.

Estructura general de una aplicación

Sólo hay una fábrica estática, UcmdbServiceFactory. Esta fábrica es el punto de entrada para una aplicación. UcmdbServiceFactory expone los métodos getServiceProvider. Estos métodos devuelven una instancia de la interfaz **UcmdbServiceProvider**.

El cliente crea otros objetos usando métodos de interfaz. Por ejemplo, para crear una nueva definición de consulta, el cliente:

1. obtiene el servicio de consulta del objeto de servicio de CMDB principal
2. Obtiene un objeto de fábrica de consulta del objeto de servicio
3. Obtiene una nueva definición de consulta de la fábrica

```
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");
```

Los servicios disponibles de **UcmdbService** son:

Métodos de servicio	Uso
getClassModelService	Información sobre tipos de CI y relaciones
getConfigurationService	Administración de configuración de infraestructura, para la configuración de servidor
getDDMConfigurationService	Configurar el sistema de Data Flow Management
getDDMManagementService	Analizar y ver el progreso, resultados y errores del sistema Data Flow Management
getHistoryService	Información sobre el historial de los CI supervisados (cambios, eliminaciones, etc.)
getImpactAnalysisService	Ejecutar un escenario de análisis de impacto (también conocido como **correlación**).
getQueryManagementService	Gestionar el acceso a las consultas: guardar, eliminar, salir de la lista. También proporciona una validación de consultas y detección de dependencias de consultas.
getResourceBundleManagementService	Servicios de "agrupación" de etiquetado de recursos. Permite la creación explícita de nuevas etiquetas y la eliminación de etiquetas de todos los recursos etiquetados.
getStateService	Proporciona servicios para administrar estados (listar, agregar, quitar, etc.)
getSoftwareSignatureService	Definir los elementos de software que va a detectar el sistema Discovery and Dependency Management.
getSnapshotService	Proporciona servicios para administrar instantáneas (obtener, guardar, comparar, etc.)
getTopologyQueryService	Obtener información sobre el universo de TI

Métodos de servicio	Uso
getTopologyUpdateService	Cambiar la información en el universo de TI
getViewService	Ver el servicio de ejecución (definición de ejecución, ejecución guardada) y el servicio de gestión (guardar, eliminar, lista existente). También proporciona una validación de vistas y detección de dependencias.
getViewArchiveService	Ver servicios de archivado de resultados. Permite el guardado del resultado de la vista actual y la recuperación de los resultados previamente guardados.
SystemHealthService	Proporciona servicios de estado del sistema (indicadores de rendimiento básico del sistema, medidas de capacidad y disponibilidad)

El cliente se comunica con el servidor sobre HTTP.

Colocar el archivo Jar de la API en la ruta de clase

El uso de este conjunto de API requiere el archivo **ucmdb-api.jar**. Puede descargar el archivo introduciendo `http://<localhost>:8080` en un navegador web, donde `localhost` es el equipo donde se ha instalado UCMDB, y haciendo clic en el vínculo **API Client Download** link.

Coloque el archivo jar en la ruta de clase antes de compilar o ejecutar la aplicación.

Nota: El uso del archivo jar de la API de Java de UCMDB requiere tener instalada la versión 6 o posterior de JRE.

Creación de un usuario de integración

Puede crear un usuario dedicado para integraciones entre otros productos y UCMDB. Este usuario habilita un producto que usa el SDK del cliente UCMDB para que se autentique en el SDK del servidor y ejecute las API. Las aplicaciones escritas con este conjunto de API deben iniciar sesión con las credenciales del usuario de integración.

Precaución: También es posible conectar con un usuario normal de UCMDB (por ejemplo, admin); sin embargo, no se recomienda esta opción. Para conectar con un usuario de UCMDB, debe concederle el permiso de autenticación de API.

Para crear un usuario de integración:

1. Inicie el navegador web y especifique la dirección del servidor, del siguiente modo:

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

   Es posible que deba iniciar sesión con un nombre de usuario y una contraseña (los valores predeterminados son sysadmin/sysadmin).

2. En UCMDB, haga clic en **service=UCMDB Authorization Services**.
3. Localice la operación **createUser**. Este método acepta los parámetros siguientes:
   - **customerId**. El Id. del cliente.
   - **username**. El nombre de usuario de integración.
   - **userDisplayName**. El nombre de visualización del usuario de integración.
   - **userLoginName**. El nombre de inicio de sesión del usuario de integración.
   - **password**. La contraseña de usuario de integración.
4. Haga clic en **Invoke**.
5. En un entorno de un solo arrendatario, localice el método **setRolesForUser** e introduzca los parámetros siguientes:
   - **userName**. El nombre de usuario de integración.
   - **roles.** SuperAdmin.

   Haga clic en **Invoke**.
6. En un entorno de múltiples arrendatarios, localice el método **grantRolesToUserForAllTenants** e introduzca los parámetros siguientes para asignar la función en conexión con todos los arrendatarios:
   - **userName**. El nombre de usuario de integración.
   - **roles.** SuperAdmin.

   Haga clic en **Invoke**.

   Como método alternativo, para asignar la función en conexión con arrendatarios específicos, invoque el método **grantRolesToUserForTenants** utilizando los mismos valores de los parámetros userName y roles. En el parámetro **tenantNames**, escriba los arrendatarios requeridos.
7. Cree más usuarios o cierre la consola JMX.
8. Inicie sesión en UCMDB como administrador.
9. En la pestaña **Administración**, ejecute **Administrador de paquetes**.
10. Haga clic en el icono **Crear paquete personalizado**.
11. Introduzca un nombre para el nuevo paquete y haga clic en **Siguiente**.
12. En la pestaña Selección de un recurso, en **Configuración**, haga clic en **Usuarios**.
13. Seleccione a un usuario o usuarios que haya creado usando la consola JMX.
14. Haga clic en **Siguiente** y, a continuación, en **Finalizar**. Su nuevo paquete aparece en la lista Nombre del paquete en Administrador de paquetes.
15. Implante el paquete a los usuarios que ejecutarán las aplicaciones API.

    Para obtener más información, consulte "Despliegue de un paquete" en la *HP Universal CMDB – Guía de administración*.

Nota:

El usuario de integración es por cliente. Para crear un usuario de integración más fuerte para un uso entre clientes, utilice un **systemUser** con el indicador **isSuperIntegrationUser** establecido en **true**. Utilice los métodos **systemUser** (**removeUser**, **resetPassword**, **UserAuthenticate**, etc.).

Hay dos usuarios del sistema listos para usar; se recomienda cambiar sus contraseñas después de la instalación usando el método **resetPassword**.

- **sysadmin**/**sysadmin**.
- **UISysadmin**/**UISysadmin** (Este usuario es también el súper usuario de integración **SuperIntegrationUser**).

  Si cambia la contraseña de UISysadmin mediante **resetPassword**, debe ejecutar el método siguiente: en JMX Console, localice el servicio **UCMDB-UI:name=UCMDB Integration**. Ejecute **setCMDBSuperIntegrationUser** con el nombre de usuario y la nueva contraseña del usuario de integración.

Referencia de la API de HP Universal CMDB

Estos archivos están ubicados en la carpeta siguiente:

\\<directorio raíz de UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Casos de uso

Los casos de uso siguientes asumen dos sistemas:

- Servidor de HP Universal CMDB
- Un sistema de terceros que contiene un repositorio de elementos de configuración

Esta sección incluye los siguientes temas:

Rellenado de CMDB

Casos de uso:

- Una gestión de activos de terceros actualiza CMDB con información disponible solo en la gestión de activos
- Un número de sistemas de terceros rellenan CMDB para crear un CMDB central que puede realizar el seguimiento de los cambios y realizar análisis de impacto

- Un sistema de terceros crea elementos de configuración y relaciones de acuerdo con la lógica empresarial de terceros, para aprovechar las capacidades de consulta de UCMDB

Consulta de CMDB

Casos de uso:

- Un sistema de terceros obtiene los elementos de configuración y las relaciones que representan el sistema SAP recuperando los resultados de SAP TQL
- Un sistema de terceros obtiene la lista de servidores Oracle que se han agregado o modificado en las últimas cinco horas
- Un sistema de terceros obtiene la lista de servidores cuyo nombre de host contiene la subcadena `lab`
- Un sistema de terceros encuentra los elementos relacionados con un CI determinado obteniendo sus vecinos

Consulta del modelo de clase

Casos de uso:

- Un sistema de terceros permite a los usuarios especificar el conjunto de datos que se va a recuperar desde CMDB. Una interfaz de usuario se puede construir en el modelo de clase para mostrar a los usuarios las posibles propiedades y solicitarles los datos requeridos. El usuario puede elegir después la información que se va a recuperar.
- Un sistema de terceros explora el modelo de clase cuando el usuario no puede acceder a la interfaz de usuario de UCMDB.

Análisis del impacto de cambio

Caso de uso:

Un sistema de terceros produce una lista de los servicios empresariales que podrían verse impactados por un cambio en un host especificado.

Ejemplos

Vea los ejemplos de código siguientes:

- 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

Estos archivos están ubicados en el siguiente directorio:

\\<directorio raíz de UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples

Capítulo 10

API de servicio web de HP Universal CMDB

Este capítulo incluye:

Convenciones

Este capítulo utiliza las convenciones siguientes:

- **UCMDB** se refiere a la misma base de datos de Universal Configuration Management. HP Universal CMDB hace referencia a la aplicación.
- Los elementos de UCMDB y argumentos del método se deletrean en caso de que se especifiquen en el esquema. Los elementos o argumentos a un método no se escriben en mayúsculas. Por ejemplo, `relation` es un elemento del tipo `Relation` pasado a un método.

Para obtener la documentación completa en las estructuras de solicitud y respuesta, consulte HP UCMDB Web Service API Reference. Estos archivos están ubicados en la carpeta siguiente:

<directorio raíz de UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

Información general de la API de servicio web de HP Universal CMDB

Nota: Utilice este capítulo junto con la documentación del esquema de UCMDB, disponible en la biblioteca de documentación en línea.

HP Universal CMDBLa API del servicio web de HP Universal CMDB (UCMDB).. La API proporciona métodos para:

- agregar, eliminar y actualizar CI y relaciones de CMDB
- recuperar información sobre el modelo de clase
- recuperar los análisis de impacto
- recuperar información sobre elementos de configuración y relaciones
- administrar credenciales: ver, agregar, actualizar y suprimir
- administrar trabajos: ver estado, activar y desactivar
- administrar intervalos de sondas: ver, agregar y actualizar
- administrar activadores: agregar o suprimir un CI de activación, y agregar, suprimir o deshabilitar un TQL de activación
- ver datos generales en dominios y sondas

Los métodos para recuperar información sobre elementos de configuración y relaciones suelen usar el Lenguaje de consulta de topología (TQL). Para obtener más información, consulte "Topology Query Language" en *HP Universal CMDB – Guía de modelado*.

Los usuarios de la API del servicio web de HP Universal CMDB deberían estar familiarizados con:

- La especificación SOAP
- Un lenguaje de programación orientado a objetos, como C++, C# o Java
- HP Universal CMDB
- Administración de Data Flow

Esta sección incluye los siguientes temas:

- "Usos de la API" abajo
- "Permisos" en la página siguiente

Usos de la API

La API se usa para cumplir un número de requisitos empresariales. Por ejemplo:

- Un sistema de terceros puede consultar el modelo de clase en busca de información sobre los elementos de configuración disponibles (CI).
- Una herramienta de gestión de activos de terceros puede actualizar CMDB con información disponible solo a dicha herramienta, unificando por tanto sus datos con los recopilados por las aplicaciones de HP.

- Un número de sistemas de terceros pueden rellenar CMDB para crear un CMDB central que puede realizar el seguimiento de los cambios y realizar el análisis de impacto.
- Un sistema de terceros puede crear entidades y relaciones de acuerdo con su lógica empresarial y, después, escribir los datos en CMDB para sacar partido a las capacidades de consulta de CMDB.
- Otros sistemas, como Release Control (CCM), puede usar los métodos de análisis de impacto para el análisis de cambios.

Permisos

Para acceder al archivo WSDL para el servicio web, visite: **http://localhost:8080/axis2/services/UcmdbService?wsdl**. Tendrá que proporcionar credenciales de usuario administrador de servidor para ver el archivo WSDL.

El usuario debe tener el permiso de acción general **Ejecutar API heredada** para poder iniciar la sesión.

La tabla siguiente muestra los permisos necesarios adicionales para cada comando de la API de servicio web:

Comando de la API de servicio web	Permisos obligatorios
addCIsAndRelations deleteCIsAndRelations updateCIsAndRelations	Acción general: **Actualización de datos**
executeTopologyQueryByName(AdHoc) executeTopologyQueryByNameWithParameters(AdHoc) executeTopologyQueryWithParameters(AdHoc)	Acción general: **Ejecutar consulta por definición** Para cada consulta: **Ver** permiso
getTopologyQueryExistingResultByName getTopologyQueryResultCountByName releaseChunks pullTopologyMapChunks getCINeighbours getFilteredCIsByType getCIsById getCIsByType getRelationsById	Acción general: **Ver los CI** Para cada consulta: **Ver** permiso
getQueryNameOfView	Acción general: **Ver los CI** Para cada vista: **Ver** permiso
getChangedCIs	Acción general: **Ver historial**, **Ver los CI**
calculateImpact getImpactPath	Acción general: **Ejecutar análisis de impacto**

Comando de la API de servicio web	Permisos obligatorios
getImpactRulesByGroupName getImpactRulesByNamePrefix	
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Ninguno

HP Universal CMDB Web Service API Reference

Para obtener la documentación completa en las estructuras de solicitud y respuesta, consulte HP UCMDB Web Service API Reference. Estos archivos están ubicados en la carpeta siguiente:

<directorio raíz de UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

Llamada al servicio web

Se utilizan las técnicas de programación SOAP estándar en el servicio web de HP Universal CMDB para habilitar la llamada a los métodos del servidor. Si la instrucción no se puede analizar o si hay un problema al invocar al método, los métodos de la API lanzan una excepción `SoapFault`. Cuando se lanza una excepción `SoapFault`, UCMDB rellena uno o varios de los campos de mensaje de error, código de error y mensaje de excepción. Si no hay ningún error, se devuelven los resultados de la invocación.

Los programadores de SOAP pueden acceder a WSDL en:

http://<servidor>[:port]/axis2/services/UcmdbService?wsdl

La especificación del puerto solo es necesaria para las instalaciones no estándares. Pregunte al administrador del sistema el número de puerto correcto.

La dirección URL para llamar al servicio es:

http://<servidor>[:port]/axis2/services/UcmdbService

Para obtener ejemplos de conexión a CMDB, consulte "Casos de uso" en la página 259.

Consulta de CMDB

CMDB es consultado mediante las API descritas en "Métodos de consulta de UCMDB " en la página 242.

Las consultas y los elementos CMDB devueltos siempre contienen ID de UMDB reales.

Para ver ejemplos del uso de los métodos de consulta, consulte "Ejemplo de consulta" en la página 262.

Esta sección incluye los siguientes temas:

Cálculo de la respuesta JIT (Justo a tiempo)

Para todos los métodos de consulta, el servidor de UMDB calcula los valores solicitados por el método de consulta cuando se recibe la petición, y devuelve los resultados basados en los datos más recientes. El resultados siempre se calcula en el momento en que se recibe la petición, aunque la consulta TQL esté activa y exista un resultado anteriormente calculado. Por tanto, los resultados de ejecutar una consulta devuelta a la aplicación cliente pueden ser diferentes de los resultados de la misma consulta mostrada en la interfaz de usuario.

Tip: Si la aplicación usa los resultados de una consulta dada más de una vez y no está previsto que los datos cambien de manera significativa entre los usos de los datos del resultado, puede mejorar el rendimiento haciendo que la aplicación cliente almacene los datos en lugar de ejecutar repetidamente la consulta.

Procesamiento de respuestas largas

La respuesta a una consulta siempre incluye las estructuras de los datos solicitados por el método de consulta, aunque no se esté transmitiendo ningún dato real. Para los métodos cuyos datos sean una colección o asignación, la respuesta también incluye la estructura `ChunkInfo`, compuesta de `chunksKey` y `numberOfChunks`. El campo `numberOfChunks` indica el número de fragmentos que contienen datos que deben recuperarse.

El administrador del sistema establece el tamaño máximo de transmisión de datos. Si los datos devueltos de la consulta son mayores que el tamaño máximo, las estructuras de datos de la primera respuesta no contendrán información significativa y el valor del campo `numberOfChunks` será `2` o mayor. Si los datos no son mayores que el tamaño máximo, el campo `numberOfChunks` será 0 (cero) y los datos se transmitirán en la primera respuesta. Por consiguiente, cuando procese una respuesta, compruebe primero el valor de `numberOfChunks`. Si es mayor que `1`, descarte los datos de la transmisión y solicite los fragmentos de datos. En caso contrario, utilice los datos en la respuesta.

Para obtener información sobre cómo gestionar datos fragmentados, consulte "pullTopologyMapChunks" en la página 251 y "releaseChunks" en la página 253.

Especificación de propiedades para devolver

Los CI y las relaciones por lo general tienen varias propiedades. Algunos métodos que devuelven colecciones o gráficos de estos elementos aceptan parámetros de entrada que especifican qué valores de propiedades se devuelven con cada elemento que coincida con la consulta. CMDB no devuelve propiedades vacías. Por tanto, la respuesta a una consulta puede tener menos propiedades que las solicitadas en la consulta.

En esta sección se describen los tipos de conjuntos usados para especificar las propiedades que se van a devolver.

Las propiedades se pueden hacer referencia en dos formas:

- Por sus nombres
- Usando nombres de reglas de propiedades predefinidas. Las reglas de propiedades predefinidas las usa CMDB para crear una lista de nombres de propiedades reales.

Cuando una aplicación hace referencia a las propiedades por el nombre, pasa un elemento `PropertiesList`.

Tip: Siempre que sea posible, utilice `PropertiesList` para especificar los nombres de las propiedades en las que está interesado, en lugar de un conjunto basado en reglas. El uso de reglas de propiedades predefinidas casi siempre da como resultado la devolución de más propiedades que las necesarias, lo que implica una carga en el rendimiento.

Hay dos tipos de propiedades predefinidas: propiedades de calificador y propiedades simples

- **Propiedades de calificador.** Utilícelas cuando la aplicación cliente debe pasar un elemento `QualifierProperties` (una lista de calificadores que se pueden aplicar a las propiedades). CMDB convierte la lista de calificadores pasados por la aplicación cliente a la lista de propiedades a la que se aplica al menos uno de los calificadores. Los valores de estas propiedades se devuelven con los elementos `CI` o `Relation`.
- **Propiedades simples.** Para usar propiedades basadas en reglas simples, la aplicación cliente pasa un elemento `SimplePredefinedProperty` o `SimpleTypedPredefinedProperty`. Estos elementos contienen el nombre de la regla por la cual CMDB genera la lista de propiedades que se van a devolver. Las reglas que se pueden especificar en un elemento `SimplePredefinedProperty` o `SimpleTypedPredefinedProperty` son `CONCRETE`, `DERIVED` y `NAMING`.

Propiedades concretas

Las propiedades concretas son el conjunto de propiedades definidas por el CIT especificado. Las propiedades agregadas por las clases derivadas no se devuelven por instancias de esas clases derivadas.

Una colección de instancias devueltas por un método puede consistir en instancias de un CIT especificado en la invocación del método e instancias de los CIT que heredan de dicho CIT. Los CIT derivados heredan las propiedades del CIT especificado. Además, los CIT derivados amplían el CIT principal agregando propiedades.

Ejemplo de propiedades concretas:

El CIT `T1` tiene las propiedades `P1` y `P2`. El CIT `T11` hereda de `T1` y amplía `T1` con las propiedades `P21` y `P22`.

La colección de los CI de tipo `T1` incluye las instancias de `T1` y `T11`. Las propiedades concretas de todas las instancias de esta colección son `P1` y `P2`.

Propiedades derivadas

Las propiedades derivadas son el conjunto de propiedades definidas para el CIT especificado y, para cada CIT derivado, las propiedades añadidas por el CIT derivado.

Ejemplo de propiedades derivadas:

Siguiendo con el ejemplo de las propiedades concretas, las propiedades derivadas de las instancias de `T1` son `P1` y `P2`. Las propiedades derivadas de las instancias de `T11` son `P1`, `P2`, `P21` y `P22`.

Propiedades de nomenclatura

Las propiedades de nomenclatura son `display_label` y `data_name`.

Otros elementos de especificación de propiedades

- **PredefinedProperties**

  `PredefinedProperties` puede contener un elemento `QualifierProperties` y un elemento `SimplePredefinedProperty` para cada una de las otras reglas posibles. Un conjunto de `PredefinedProperties` no contiene necesariamente todos los tipos de listas.

- **PredefinedTypedProperties**

  `PredefinedTypedProperties` se usa para aplicar un conjunto diferente de propiedades para cada CIT. `PredefinedTypedProperties` puede contener un elemento `QualifierProperties` y un elemento `SimpleTypedPredefinedProperty` para cada una de las otras reglas aplicables. Como `PredefinedTypedProperties` se aplica a cada CIT individualmente, no son relevantes las propiedades derivadas. Un conjunto de `PredefinedProperties` no contiene necesariamente todos los tipos de listas aplicables.

- **CustomProperties**

  `CustomProperties` puede contener cualquier combinación de la `PropertiesList` básica y las listas de propiedades basadas en reglas. El filtro de propiedades es la unión de todas las propiedades devueltas por todas las listas.

- **CustomTypedProperties**

  `CustomTypedProperties` puede contener cualquier combinación de la `PropertiesList` básica y las listas de propiedades basadas en reglas aplicables. El filtro de propiedades es la unión de todas las propiedades devueltas por todas las listas.

- **TypedProperties**

  `TypedProperties` se usa para pasar un conjunto diferente de propiedades para cada CIT. `TypedProperties` es una colección de pares compuestos de nombres de tipo y conjuntos de propiedades de todos los tipos. Cada conjunto de propiedades se aplica solo al tipo correspondiente.

Actualización de UCMDB

Actualice CMDB con las API de actualización. Para obtener más información sobre los métodos de API, consulte "Métodos de actualización de UCMDB " en la página 253. Para ver ejemplos del uso de los métodos de actualización, consulte "Ejemplo de actualización" en la página 274.

Esta tarea incluye los siguientes pasos:

- "Actualización de UCMDB" arriba
- "Utilización de tipos de Id. con métodos de actualización" en la página siguiente

Parámetros de actualización de UCMDB

En este tema se describen los parámetros usados solo por los métodos de actualización del servicio. Para más información, consulte schema documentation..

- **CIsAndRelationsUpdates**

  El tipo `CIsAndRelationsUpdates` consta de `CIsForUpdate`, `relationsForUpdate`, `referencedRelations` y `referencedCIs`. Una instancia de `CIsAndRelationsUpdates` no incluye necesariamente los tres elementos.

  `CIsForUpdate` es una colección de CI. `relationsForUpdate` es una colección de elementos `Relation`. Los elementos `CI` y `relation` en las colecciones tienen un elemento `props`. Al crear un CI o una relación, las propiedades que tienen el atributo `required` o el atributo `key` en la definición del tipo de CI deben estar rellenados con valores. Los elementos de estas colecciones se actualizan o se crean por el método.

  `referencedCIs` y `referencedRelations` son colecciones de CI que ya están definidas en CMDB. Los elementos de la colección se identifican con un Id. temporal junto con todas las propiedades clave. Estos elementos se usan para resolver las identidades de los CI y las relaciones para su actualización. Nunca se crean ni se actualizan por el método.

  Cada uno de los elementos `CI` y `relation` de estas colecciones tienen una colección de propiedades. Los nuevos elementos se crean con los valores de las propiedades de estas colecciones.

Utilización de tipos de Id. con métodos de actualización

Lo siguiente describe los CIT de Id., así como los CI y relaciones. Cuando el Id. no es un Id. de CMDB auténtico, se requieren el tipo y los atributos clave.

- **Eliminación o actualización de elementos de configuración**

  El cliente puede usar un Id. temporal o vacío al llamar a un método para eliminar o actualizar un elemento. En este caso, deben establecerse el tipo de CI y los "Atributos clave" que identifican el CI.

- **Eliminación o actualización de relaciones**

  Al eliminar o actualizar relaciones, el Id. de relación puede estar vacío, ser temporal o real.

  Si el Id. de un CI es temporal, el CI debe pasarse en la colección `referencedCIs` y se deben especificar sus atributos clave. Para más información, consulte referencedCIs en "CIsAndRelationsUpdates" arriba.

- **Inserción de nuevos elementos de configuración en CMDB**

  Es posible usar un Id. vacío o un Id. temporal para insertar un nuevo CI. Sin embargo, si el Id. está vacío, el servidor no puede devolver el Id. de CMDB en la estructura `createIDsMap` porque no hay ningún `clientID`. Para obtener más información, consulte "addCIsAndRelations" en la página 253 y "Métodos de consulta de UCMDB " en la página 242.

- **Inserción de nuevas relaciones en CMDB**

  El Id. de relación puede ser temporal o estar vacío. Sin embargo, si la relación es nueva pero los elementos de configuración de cada extremo de la relación ya están definidos en CMDB, en ese caso los CI que ya existen deben ser identificados por un Id. de CMDB real o especificarse en una colección de `referencedCIs`.

Importación del modelo de clase de UCMDB

Los métodos del modelo de clase devuelven información sobre los CIT y relaciones. El modelo de clase se configura usando el Administrador de tipos de CI. Para obtener más información, consulte "Administrador de tipos de CI" en la página 1 en *HP Universal CMDB – Guía de modelado*.

Para ver ejemplos del uso de los métodos de modelo de clase, consulte "Ejemplo de modelo de clase" en la página 278.

En esta sección se proporciona información sobre los siguientes me´todos que devuelven información sobre CIT y relaciones:

- "getClassAncestors" abajo
- "getAllClassesHierarchy" abajo
- "getCmdbClassDefinition" en la página siguiente

getClassAncestors

El método `getClassAncestors` recupera la ruta entre el CIT dado y su raíz, incluyendo la raíz.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
className	El nombre del tipo. Para obtener más información, consulte "Nombre de tipo" en la página 240.

Salida

Parámetro	Comentario
classHierarchy	Una colección de pares de nombres de clase y nombre de clase principal.
comments	Sólo para uso interno.

getAllClassesHierarchy

El método `getAllClassesHierarchy` recupera todo el árbol del modelo de clase.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.

Salida

Parámetro	Comentario
classesHierarchy	Una colección de pares de nombre de clase y nombre de clase principal.
comments	Sólo para uso interno.

getCmdbClassDefinition

El método `getCmdbClassDefinition` recupera información sobre la clase especificada.

Si usa getCmdbClassDefinition para recuperar los atributos clave, también debe consultar las clases principales hasta la clase base. `getCmdbClassDefinition` identifica como atributos clave solo aquellos atributos con el ID_ATTRIBUTE establecido en la definición de clase especificada por `className`. Los atributos clave heredados no se reconocen como atributos clave de la clase especificada. Por consiguiente, la lista completa de atributos clave para la clase especificada es la unión de todas las claves de la clase y de todos sus elementos principales, hasta la raíz.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página siguiente.
className	El nombre del tipo. Para obtener más información, consulte "Parámetros generales de UCMDB " abajo.

Salida

Parámetro	Comentario
cmdbclass	La definición de la clase, que consiste en `name`, `classType`, `displayLabel`, `description`, `parentName`, calificadores y atributos.
comments	Sólo para uso interno.

Consulta de análisis de impacto

El `identificador` en los métodos de análisis de impacto apunta a los datos de respuesta del servicio. Es único para la respuesta actual y se descarta de la caché de la memoria del servidor si pasan 10 minutos sin utilizarse.

Para ver ejemplos del uso de los métodos de análisis de impacto, consulte "Ejemplo de análisis de impacto" en la página 279.

Parámetros generales de UCMDB

Esta sección describe los parámetros más comunes de los métodos del servicio. Para obtener información detallada, consulte la schema documentation.

Esta sección incluye los siguientes temas:

CmdbContext

Todas las invocaciones a los servicios de la API del servicio web de UCMDB requieren un argumento `CmdbContext`. `CmdbContext` es una cadena de `callerApplication` que identifica la aplicación que invoca al servicio. `CmdbContext` se utiliza para el registro y la solución de problemas.

ID

Cada `CI` y Relation tiene un campo `ID`. Consiste en una cadena de Id. que distingue mayúsculas de minúsculas y un indicador `temp` opcional, que indica si el Id. es temporal.

Atributos clave

Para identificar un `CI` o `Relation` en algunos contextos, se pueden usar los atributos clave en lugar de un Id. de CMDB. Los atributos clave son aquellos atributos con el conjunto ID_ ATTRIBUTE en la definición de clase.

En la interfaz de usuario, los atributos clave tiene un icono de clave junto a ellos en la lista de atributos de tipos de elementos de configuración en la interfaz de usuario. Para obtener más información, consulte "Cuadro de diálogo Agregar/Editar atributo" en la página 1 en HP Universal CMDB – Guía de modelado. Para obtener información sobre cómo identificar los atributos clave desde dentro de la aplicación cliente de API, consulte "getCmdbClassDefinition" en la página precedente.

Tipos de Id.

Un elemento `ID` puede contener un Id. real o un Id. temporal.

Un Id. real es una cadena asignada por CMDB que identifica una entidad en la base de datos. Un Id. temporal puede ser cualquier cadena que sea única en la petición actual.

El cliente puede asignar un Id. temporal y con frecuencia representa el Id. del CI tal como fue almacenado por el cliente. No representa necesariamente una entidad ya creada en CMDB. Cuando el cliente pasa un Id. temporal, si CMDB puede identificar un elemento de configuración de datos existente usando las propiedades clave del CI, dicho CI se usa tal como resulte apropiado por el contexto como si hubiera sido identificado con un Id. real.

CIProperties

El elemento `CIProperties` está compuesto de colecciones, conteniendo cada una de ellas una secuencia de elementos nombre-valor que especifican propiedades del tipo indicado por el nombre de la colección. No se requiere ninguna de las colecciones, por tanto el elemento `CIProperties`

puede contener cualquier combinación de colecciones.

`CIProperties` los usan los elementos `CI` y `Relation`. Para obtener más información, consulte "Elemento de configuración (CI)" abajo y "Relation" abajo.

Las colecciones de propiedades son las siguientes:

- `dateProps`: colección de elementos `DateProp`
- `doubleProps`: colección de elementos `DoubleProp`
- `floatProps`: colección de elementos `FloatProp`
- `intListProps`: colección de elementos `intListProp`
- `intProps`: colección de elementos `IntProp`
- `strProps`: colección de elementos `StrProp`
- `strListProps`: colección de elementos `StrListProp`
- `longProps`: colección de elementos `LongProp`
- `bytesProps`: colección de elementos `BytesProp`
- `xmlProps`: colección de elementos `XmlProp`

Nombre de tipo

El nombre de tipo es el nombre de la clase de un tipo de elemento de configuración o tipo de relación. El nombre de tipo se usa en el código para referirse a la clase. No debería confundirse con el nombre de visualización, que se ve en la interfaz de usuario donde se menciona la clase, pero que carece de sentido en el código.

Elemento de configuración (CI)

Un elemento `CI` está compuesto de un `ID`, `type` y una colección de `props`.

Al utilizar "Métodos de actualización de UCMDB " para actualizar un `CI`, el elemento `ID` puede contener un Id. de CMDB real o un Id. temporal asignado por el cliente. Si se usa un Id. temporal, establezca el indicador `temp` en true. Al eliminar un elemento, el `ID` puede estar vacío. Los "Métodos de consulta de UCMDB " toman `ID` reales como parámetros de entrada y devuelven `ID` reales en los resultados de consultas.

`type` puede ser cualquier nombre de tipo definido en el Administrador de tipos de CI. Para obtener más información, consulte "Administrador de tipos de CI" en la página 1 en HP Universal CMDB – Guía de modelado.

El elemento `props` es una colección de CIProperties. Para más información, consulte "Parámetros generales de UCMDB " en la página 238.

Relation

`Relation` es una entidad que enlaza dos elementos de configuración. El elemento `Relation` está compuesto de un `ID`, un `type`, los identificadores de los dos elementos que se están enlazando `(end1ID` y `end2ID)` y una colección de `props`.

Al utilizar los "Métodos de actualización de UCMDB " para actualizar un elemento `Relation`, el valor del Id. de `Relation` puede ser un Id. de CMDB real o un Id. temporal. Al eliminar un

elemento, el `ID` puede estar vacío. Los "Métodos de consulta de UCMDB " toman `ID` reales como parámetros de entrada y devuelven `ID` reales en los resultados de la búsqueda.

El tipo de relación es el `nombre de tipo` de la clase de UCMDB desde la que se instancia la relación. El tipo puede ser cualquier tipo de relación definido en CMDB. Para obtener más información sobre las clases o tipos, consulte "Importación del modelo de clase de UCMDB" en la página 237.

Para obtener más información, consulte "Administrador de tipos de CI" en la página 1 en *HP Universal CMDB – Guía de modelado*.

Los dos Id. de extremo de la relación no deben ser ID vacíos porque se utilizan para crear el Id. de la relación actual. Sin embargo, ambos pueden tener unos Id. temporales asignados a ellos por el cliente.

El elemento `props` es una colección de CIProperties. Para más información, consulte "CIProperties" en la página 239.

Parámetros de salida de UCMDB

Esta sección describe los parámetros de salida más comunes de los métodos del servicio. Para obtener información detallada, consulte la schema documentation.

Esta sección incluye los siguientes temas:

- "CIs" abajo
- "ShallowRelation" abajo
- "Topology" abajo
- "CINode" abajo
- "RelationNode" en la página siguiente
- "TopologyMap" en la página siguiente
- "ChunkInfo" en la página siguiente

CIs

`CIs` es una colección de elementos de CI.

ShallowRelation

`ShallowRelation` es una entidad que enlaza dos elementos de configuración, compuestos de un `ID`, un `tipo` y de los identificadores de los dos elementos que se están enlazando `(end1ID` y `end2ID)`. El tipo de relación es el `nombre de tipo` de la clase de CMDB desde la que se instancia la relación. El tipo puede ser cualquier tipo de relación definido en CMDB.

Topology

`Topology` es un grafo de elementos y relaciones de `CI`. Un elemento `Topology` consta de una colección de `CIs` y una colección de `Relations` que contienen uno o varios elementos `Relation`.

CINode

`CINode` está compuesto de una colección de `CIs` con un elemento `label`. El elemento `label`

en `CINode` es la etiqueta definida en el nodo del TQL usado en la consulta.

RelationNode

`RelationNode` es un conjunto de colecciones de elementos `Relation` con un elemento `label`. El elemento `label` en `RelationNode` es la etiqueta definida en el nodo del TQL usado en la consulta.

TopologyMap

`TopologyMap` es la salida de un cálculo de consulta que coincide con una consulta TQL. El elemento `label` en `TopologyMap` son las etiquetas de nodo definidas en el TQL usado en la consulta.

Los datos de `TopologyMap` se devuelven de la forma siguiente:

- `CINodes`. Es uno o varios `CINode` (consulte "CINode" en la página precedente).
- `relationNodes`. Es uno o varios `RelationNode` (consulte "RelationNode" arriba).

Los elementos `label` de estas dos estructuras ordenan las listas de elementos de configuración y relaciones.

ChunkInfo

Cuando una consulta devuelve una gran cantidad de datos, el servidor almacena los datos, divididos en segmentos llamados fragmentos. La información que usa el cliente para recuperar los datos fragmentados se localiza en la estructura `ChunkInfo` devuelta por la consulta. `ChunkInfo` está compuesta de `numberOfChunks` que debe recuperarse y por `chunksKey`. `chunksKey` es un identificador único de los datos en el servidor para esta invocación de consulta específica.

Para obtener más información, consulte "Procesamiento de respuestas largas" en la página 233.

Métodos de consulta de UCMDB

En esta sección se proporciona información sobre los siguientes métodos:

executeTopologyQueryByNameWithParameters

El método `executeTopologyQueryByNameWithParameters` recupera un elemento `topologyMap` que coincide con la consulta parametrizada especificada.

Los valores de los parámetros de la consulta se pasan en el argumento `parameterizedNodes`. El TQL especificado debe tener etiquetas únicas definidas para cada `CINode` y para cada `relationNode` o la invocación del método producirá un error.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
queryName	El nombre del TQL parametrizado en CMDB para el que se obtiene el mapa.
parameterizedNodes	Las condiciones que cada nodo debe cumplir para ser incluido en los resultados de la consulta.
queryTypedProperties	Una colección de conjuntos de propiedades para recuperar elementos de un tipo de elemento de configuración específico.

Salida

Parámetro	Comentario
topologyMap	Para obtener más información, consulte "TopologyMap" en la página precedente.
chunkInfo	Para obtener más información, consulte: "ChunkInfo" en la página precedente y "Procesamiento de respuestas largas" en la página 233.

executeTopologyQueryWithParameters

El método `executeTopologyQueryWithParameters` recupera un elemento `topologyMap` que coincide con la consulta parametrizada.

La consulta se pasa en el argumento `queryXML`. Los valores de los parámetros de la consulta se pasan en el argumento `parameterizedNodes`. El TQL debe tener etiquetas únicas definidas para cada `CINode` y para cada `relationNode`.

El método `executeTopologyQueryWithParameters` se usa para pasar consultas ad hoc, en lugar de acceder a una consulta definida en CMDB. Puede usar este método cuando no tiene acceso a la interfaz de usuario de UCMDB para definir una consulta, o cuando no desea guardar la consulta en la base de datos.

Para utilizar un TQL exportado como entrada a este método, realice lo siguiente:

1. Inicie el navegador web e introduzca la siguiente dirección: **http://localhost:8080/jmx-console**.

Es posible que tenga que iniciar sesión con un nombre de usuario y una contraseña. El valor predeterminado es **sysadmin**/**sysadmin**

2. Haga clic en **UCMDB:service=TQL Services**.
3. Localice la operación **exportTql**.
   - En el cuadro del parámetro **customerId**, escriba **1** (el valor predeterminado).
   - En el cuadro de parámetros **patternName**, introduzca un nombre de TQL válido.
4. Haga clic en **Invoke**.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
queryXML	Una cadena XML que representa un TQL sin etiquetas de recursos.
parameterizedNodes	Las condiciones que cada nodo debe cumplir para ser incluido en los resultados de la consulta.

Salida

Parámetro	Comentario
topologyMap	Para obtener más información, consulte "TopologyMap" en la página 242.
chunkInfo	Para obtener más información, consulte "ChunkInfo" en la página 242 y "Procesamiento de respuestas largas" en la página 233.

getChangedCIs

El método `getChangedCIs` devuelve los datos de cambio para todos los CI relacionados con los CI especificados.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
ids	La lista de ID de los CI raíz en cuyos CI relacionados se están comprobando los cambios. Solo los Id. reales de CMDB son válidos en esta colección
fromDate	El comienzo del periodo en donde comprobar si los CI han cambiado.
toDate	El fin del periodo en donde comprobar si los CI han cambiado.

Salida

Parámetro	Comentario
changeDataInfo	Cero o más colecciones de elementos `ChangedDataInfo`.

getCINeighbours

El método `getCINeighbours` devuelve los vecinos inmediatos del CI especificado.

Por ejemplo, si la consulta es sobre los vecinos de CI `A` y el CI `A` contiene el CI `B` que usa el CI `C`, se devuelve el CI `B`, pero no el CI `C`. Es decir, solo se devuelven los vecinos del tipo especificado.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
ID	El Id. del CI con el que recuperar los vecinos. Debe ser un Id. de CMDB real.
neighbourType	El nombre de CIT de los vecinos que se van a recuperar. Se devuelven los vecinos del tipo especificado y de los tipos procedentes de ese tipo. Para obtener más información, consulte "Nombre de tipo" en la página 240.
CIProperties	Los datos que se van a devolver en cada elemento de configuración, conocidos como diseño de consulta en la interfaz de usuario. Para obtener más información, consulte "TypedProperties" en la página 235.
relationProperties	Los datos que se van a devolver en cada relación (conocidos como diseño de consulta en la interfaz de usuario). Para obtener más información, consulte "TypedProperties" en la página 235

Salida

Parámetro	Comentario
topology	Para obtener más información, consulte "Topology" en la página 241.
comments	Sólo para uso interno.

getCIsByID

El método `getCIsByID` recupera los elementos de configuración por sus Id. de CMDB.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
CIsTypedProperties	Una colección de propiedades con tipo. Para obtener más información,

Parámetro	Comentario
	consulte "Otros elementos de especificación de propiedades" en la página 235.
IDs	Solo los Id. reales de CMDB son válidos en esta colección

Salida

Parámetro	Comentario
CIs	Colección de elementos de CI.
chunkInfo	Para obtener más información, consulte: "ChunkInfo" en la página 242 y "Procesamiento de respuestas largas" en la página 233.

getClsByType

El método `getCIsByType` devuelve la colección de elementos de configuración del tipo especificado y de todos los tipo que heredan del tipo especificado.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
type	El nombre de la clase. Para obtener más información, consulte "Nombre de tipo" en la página 240.
properties	Los datos que se van a devolver en cada elemento de configuración. Para obtener más información, consulte "CustomProperties" en la página 235.

Salida

Parámetro	Comentario
CIs	Colección de elementos de CI.
chunkInfo	Para obtener más información, consulte: "ChunkInfo" en la página 242 y "Procesamiento de respuestas largas" en la página 233.

getFilteredClsByType

El método `getFilteredCIsByType` recupera los CI del tipo especificado que cumplen las condiciones usadas por el método. Una condición consta de:

- un campo de nombre que contiene el nombre de una propiedad
- un campo de operador que contiene un operador de comparación
- un campo de valor opcional que contiene un valor o lista de valores

Juntos, forman una expresión booleana:

`<elemento>.property.value [operator] <condición>.value`

Por ejemplo, si el nombre de la condición es `root_actualdeletionperiod`, el valor de la condición es `40` y el operador es `Igual`, la instrucción booleana es:

`<elemento>.root_actualdeletionperiod.value = = 40`

La consulta devuelve todos los elementos cuyo `root_actualdeletionperiod`is `40`, asumiendo que no hay otras condiciones.

Si el argumento `conditionsLogicalOperator` es `AND`, la consulta devuelve los elementos que cumplen todas las condiciones de la colección `conditions`. Si `conditionsLogicalOperator` es `OR`, la consulta devuelve los elementos que cumplen al menos una de las condiciones de la colección `conditions`.

La siguiente tabla muestra los operadores de comparación:

Operador	Tipo de condición/comentarios
ChangedDuring	Fecha Esta es una comprobación del intervalo. El valor de la condición se especifica en horas. Si el valor de la propiedad de fecha se encuentra en el intervalo del tiempo en que se invoca el método más o menos el valor de la condición, la condición es true. Por ejemplo, si el valor de la condición es `24`, la condición es true si el valor de la propiedad de fecha se encuentra entre ayer a esta hora y mañana a esta hora. **Nota:** el nombre `ChangedDuring` se mantiene para conservar la compatibilidad con versiones anteriores. En versiones anteriores, el operador se ha usado solo con las propiedades de tiempo de creación y modificación.
Igual	Cadena y numérica
EqualIgnoreCase	Cadena
Mayor que	Numérica
GreaterEqual	Numérica
En	Cadena, numérica y lista El valor de la condición es una lista. La condición es true si el valor de la propiedad es uno de los valores de la lista.
InList	Lista El valor de la condición y el valor de la propiedad son listas. La condición es true si todos los valores de la lista de condiciones también aparece en la lista de propiedades del elemento. Puede haber más valores de propiedad que los especificados en la condición sin que afecte a la verdad de la condición.

Operador	Tipo de condición/comentarios
IsNull	Cadena, numérica y lista La propiedad del elemento no tiene ningún valor. Cuando se usa el operador `IsNull`, el valor de la condición se ignora y en algunos casos puede ser nulo.
Menos	Numérica
LessEqual	Numérica
Como	Cadena El valor de la condición es una subcadena del valor del valor de la propiedad. El valor de la condición debe ir rodeado con signos de porcentaje (`%`). Por ejemplo, `%Bi%` coincide con `Bismark` y `Bay of Biscay`, pero no con `biscuit`.
LikeIgnoreCase	Cadena Utilice el operador `LikeIgnoreCase` de la misma forma que usa el operador `Like`. Sin embargo, la coincidencia no distingue entre mayúsculas y minúsculas. Por tanto, `%Bi%` coincide con `biscuit`.
NotEqual	Cadena y numérica
UnchangedDuring	Fecha Esta es una comprobación del intervalo. El valor de la condición se especifica en horas. Si el valor de la propiedad de fecha se encuentra en el intervalo del tiempo en que se invoca el método más o menos el valor de la condición, la condición es false. Se se encuentra fuera de ese intervalo, la condición es true. Por ejemplo, si el valor de la condición es `24`, la condición es true si el valor de la propiedad de fecha se encuentra antes de ayer a esta hora y después de mañana a esta hora. **Nota:** el nombre `UnchangedDuring` se mantiene para conservar la compatibilidad con versiones anteriores. En versiones anteriores, el operador se ha usado solo con las propiedades de tiempo de creación y modificación.

Ejemplo de configuración de una condición:

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.Equal);
```

Ejemplo de consulta para propiedades heredadas:

El CI de destino es `sample` que tiene dos atributos, `name` y `size`. `sampleII` amplía el CI con dos atributos, `level` y `grade`. Este ejemplo configura una consulta para las propiedades de `sampleII` que se heredaron de `sample` especificándolas por nombre.

```
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)
```

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
type	El nombre de la clase. Para obtener más información, consulte "Nombre de tipo" en la página 240. El tipo puede ser cualquiera de los tipos definidos usando el Administrador de tipos de CI. Para obtener más información, consulte "Administrador de tipos de CI" en *HP Universal CMDB – Guía de modelado*.
properties	Los datos que se van a devolver en cada CI (conocidos como diseño de consulta en la interfaz de usuario). Para obtener más información, consulte "CustomProperties" en la página 235.
conditions	Una colección de pares nombre-valor y los operadores que están relacionados entre sí. Por ejemplo, `host_hostname like QA`.
conditionsLogicalOperator	• **AND.** Todas las condiciones deben cumplirse. • **OR**. Al menos una de las condiciones debe cumplirse.

Salida

Parámetro	Comentario
CIs	Colección de elementos de CI.
chunkInfo	Para obtener más información, consulte "ChunkInfo" en la página 242 y "Procesamiento de respuestas largas" en la página 233.

getQueryNameOfView

El método `getQueryNameOfView` recupera el nombre del TQL en el que está basada la vista especificada.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
viewName	El nombre de una vista, es decir, un subconjunto del modelo de clase de CMDB.

Salida

Parámetro	Comentario
queryName	El nombre de TQL en CMDB en el que está basada la vista.

getTopologyQueryExistingResultByName

El método `getTopologyQueryExistingResultByName` recupera el resultado más reciente de la ejecución del TQL especificado. La llamada no ejecuta el TQL. Si no hay resultados de una ejecución anterior, no se devuelve nada.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
queryName	El nombre de un TQL.
queryTypedProperties	Una colección de conjuntos de propiedades para recuperar elementos de un tipo de elemento de configuración específico.

Salida

Parámetro	Comentario
queryName	El nombre de TQL en CMDB en el que está basada la vista.

getTopologyQueryResultCountByName

El método `getTopologyQueryResultCountByName` recupera el número de instancias de cada nodo que coincida con la consulta especificada.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
queryName	El nombre de un TQL.
countInvisible	Si es true, la salida incluye los CI definidos como invisible en la consulta.

Salida

Parámetro	Comentario
queryName	El nombre de TQL en CMDB en el que está basada la vista.

pullTopologyMapChunks

El método `pullTopologyMapChunks` recupera uno de los fragmentos que contienen la respuesta a un método.

Cada fragmento contiene un elemento `topologyMap` que forma parte de la respuesta. El primer fragmento se numera en `1`, por tanto el contador del bucle de recuperación se repite de 1 a <*objeto de respuesta*>`.getChunkInfo().getNumberOfChunks()`.

Para obtener más información, consulte "ChunkInfo" en la página 242 y "Consulta de CMDB" en la página 232.

La aplicación cliente debe ser capaz de gestionar los mapas parciales. Consulte el ejemplo siguiente de manipulación de una colección de CI y el ejemplo de fusionar fragmentos en un mapa de "Ejemplo de consulta" en la página 262.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
ChunkRequest	El número del fragmento que se va a recuperar y el `ChunkInfo` devuelto por el método de consulta.

Salida

Parámetro	Comentario
topologyMap	Para obtener más información, consulte "TopologyMap" en la página 242.
comments	Sólo para uso interno.

Ejemplo del manejo de fragmentos:

```
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

El método `releaseChunks` libera la memoria de los fragmentos que contienen datos de la consulta.

> **Tip:** El servidor descarta los datos tras diez minutos. Llamar a este método para descartar los datos tan pronto como se han leído conserva los recursos del servidor.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
chunksKey	El identificador de los datos en el servidor que se ha fragmentado. La clave es un elemento de `ChunkInfo`.

Métodos de actualización de UCMDB

En esta sección se proporciona información sobre los siguientes métodos:

- "addCIsAndRelations" abajo
- "addCustomer" en la página siguiente
- "deleteCIsAndRelations" en la página siguiente
- "removeCustomer" en la página 255
- "updateCIsAndRelations" en la página 255

addCIsAndRelations

El método `addCIsAndRelations` agrega o actualiza los CI y las relaciones.

En el caso de CI o relaciones que no existen en CMDB, se agregan y sus propiedades se establecen de acuerdo con el contenido del argumento `CIsAndRelationsUpdates`.

Si los CI o relaciones existen en CMDB, se actualizan con los nuevos datos, si `updateExisting` es **true**.

Si `updateExisting` es **false**, `CIsAndRelationsUpdates` no puede hacer referencia a elementos de configuración o relaciones existentes. Cualquier intento de hacer referencia a elementos existentes cuando `updateExisting` es falso da como resultado una excepción.

Si `updateExisting` es **true**, la operación de adición o de actualización se realiza sin validar los CI, con independencia del valor de `ignoreValidation`.

Si `updateExisiting` es **false** e `ignoreValidation` es **true**, la operación de adición se realiza sin validar los CI.

Si `updateExisiting` es **false** e `ignoreValidation` es **true**, la operación de adición se realiza sin validar los CI.

Las relaciones nunca se validan.

`CreatedIDsMap` es un mapa o diccionario de tipo `ClientIDToCmdbID` que conecta los Id. temporales del cliente con los Id. de CMDB reales correspondientes.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
updateExisting	Establézcalo en *true* para actualizar los elementos que ya existen en CMDB. Establézcalo en *false* para lanzar una excepción si ya existe un elemento
CIsAndRelationsUpdates	Los elementos que se van a actualizar o crear. Para obtener más información, consulte "CIsAndRelationsUpdates" en la página 236.
ignoreValidation	Si es true, no se realiza ninguna comprobación antes de actualizar CMDB.

Salida

Parámetro	Comentario
CreatedIDsMap	La asignación de los Id. de cliente a los Id. de CMDB. Para obtener más información, consulte la descripción anterior.
comments	Sólo para uso interno.

addCustomer

El método `addCustomer` agrega un cliente.

Entrada

Parámetro	Comentario
CustomerID	El Id. numérico del cliente.

deleteCIsAndRelations

El método `deleteCIsAndRelations` elimina los elementos de configuración y relaciones específicos de CMDB.

Cuando se elimina un CI y este es un extremo de uno o varios elementos `Relation`, estos elementos `Relation` también se eliminan.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
CIsAndRelationsUpdates	Los elementos que se van a eliminar. Para obtener más información, consulte "CIsAndRelationsUpdates" en la página 236.

removeCustomer

El método `removeCustomer` elimina un registro del cliente.

Entrada

Parámetro	Comentario
CustomerID	El Id. numérico del cliente.

updateCIsAndRelations

El método `updateCIsAndRelations` actualiza los CI y relaciones especificados.

La actualización usa los valores de propiedad desde el argumento `CIsAndRelationsUpdates`. Si alguna de los CI o relaciones no existe en CMDB, se lanza una excepción.

`CreatedIDsMap` es un mapa o diccionario de tipo `ClientIDToCmdbID` que conecta los Id. temporales del cliente con los Id. de CMDB reales correspondientes.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
CIsAndRelationsUpdates	Los elementos que se van a actualizar. Para obtener más información, consulte "CIsAndRelationsUpdates" en la página 236.
ignoreValidation	Si es true, no se realiza ninguna comprobación antes de actualizar CMDB.

Salida

Parámetro	Comentario
CreatedIDsMap	La asignación de los Id. de cliente a los Id. de CMDB. Para obtener más información, consulte "addCIsAndRelations" en la página 253.

Métodos de análisis de impacto de UCMDB

En esta sección se proporciona información sobre los siguientes métodos:

calculateImpact

El método `calculateImpact` calcula los CI que están afectados por un CI determinado de acuerdo con las reglas definidas en CMDB.

Esto muestra el efecto de un evento que activa la regla. La salida `identifier` de `calculateImpact` se usa como entrada para "getImpactPath" abajo.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
impactCategory	El tipo de evento que podría activar la regla que se está simulando.
IDs	Una colección de elementos de Id.
impactRulesNames	Una colección de elementos de `ImpactRuleName`.
severity	La gravedad del evento de activación.

Salida

Parámetro	Comentario
impactTopology	Para obtener más información, consulte "Topology" en la página 241.
identifier	La clave de la respuesta del servidor.

getImpactPath

El método `getImpactPath` recupera el gráfico de topología de la ruta entre el CI afectado y el CI que lo afecta.

La salida `identifier` de "calculateImpact" arriba se usa como argumento de entrada de `identifier` de `getImpactPath`.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
identifier	La clave a la respuesta del servidor que devolvió calculateImpact.
relation	Una relación basada en uno de los "ShallowRelation" devueltos por calculateImpact en el elemento `impactTopology`.

Salida

Parámetro	Comentario
impactPathTopology	Una colección de `CIs` y una colección de `ImpactRelations`.
comments	Sólo para uso interno.

Un elemento `ImpactRelations` consta de un `ID`, `type`, `end1ID`, `end2ID`, `rule` y `action`.

getImpactRulesByNamePrefix

El método `getImpactRulesByNamePrefix` recupera las reglas usando un filtro de prefijo.

Este método se aplica a las reglas de impacto que se denominan con un prefijo que indica el contexto al que se aplican, por ejemplo, `SAP_myrule`, `ORA_myrule`, etc. Este método filtra todos los nombres de las reglas de impacto para los que comienzan con el prefijo especificado por el argumento `ruleNamePrefixFilter`.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
ruleNamePrefixFilter	Una cadena que contiene las primeras letras de los nombres de reglas que van a coincidir.

Salida

Parámetro	Comentario
impactRules	`impactRules` está compuesta por cero o varias `impactRule`. Una `impactRule`, que especifica el efecto de un cambio, está compuesta de `ruleName`, `description`, `queryName` e `isActive`.

API del servicio web de estado real

La API del servicio web de estado real la utiliza principalmente el Administrador de servicio para recuperar información de estado real para un Id. de CMDB o Id. global y un Id. de cliente específico. La API busca una consulta coincidente en la carpeta **Integration/SM Query** y ejecuta el TQL con el Id. de CMDB o el Id. global como condición, y devuelve la salida de la consulta.

URL de servicio web: http://[nombre_equipo]:8080/axis2/services/ucmdbSMService

Esquema de servicio web: http://[nombre_equipo]:8080/axis2/services/ucmdbSMService?xsd=xsd0

Flujo

Cuando se llama al método API, este intenta encontrar una consulta adecuada en la carpeta **Integration/SM Query**. Intenta primero que coincida el tipo del CMDBID/GlobalID con una de las consultas de la última carpeta; para ello, busque un **QueryElement** con el nombre **Root**, y si no lo encuentra, intenta utilizar cualquier **QueryNode** del mismo tipo que el CMDBID/GlobalID solicitado. Una vez que se han encontrado un Query y QueryNode adecuados, pone CMDBID/GlobalID como condición en QueryNode y ejecuta la consulta. A continuación, el resultado es devuelto al llamante de la API.

Manipulación del resultado mediante transformaciones

En algunos casos, quizá quiera aplicar transformaciones adicionales al XML resultante (por ejemplo, para sumar los tamaños de todos los discos y agregar esa suma como un atributo adicional al CI). Para agregar transformaciones adicionales en los resultados de TQL, ponga un recurso denominado **[tql_name].xslt** en la configuración del adaptador como se indica a continuación: **Administración de adaptador > ServiceDeskAdapter7-1 > Archivos de configuración > [tql_name].xslt.**

Registros de la API del servicio web de estado real

La configuración de registro para UCMDB reside en: **UCMDBServer/Conf/log** en los diversos archivos ***.properties**.

Para ver registros del flujo de Estado actual de SM:

1. Abra el archivo **cmdb_soaapi.properties** y cambie el nivel de registro a DEBUG de la manera siguiente: **loglevel=DEBUG**.
2. Abra el archivo **fcmdb.properties** y cambie el nivel de registro a DEBUG de la manera siguiente: **loglevel=DEBUG**.
3. Espere 1 minuto a que el servidor recupere los cambios.
4. Ejecute el Estado actual de SM.
5. Vea los archivos de registro siguientes en **UCMDBServer/Runtime/log**:
   - cmdb.soaapi.log
   - fcmdb.log

Habilitación del estado real de los CI replicados después de cambiar el contexto raíz

Si ha cambiado el contexto raíz que se utiliza para acceder a UCMDB, debe realizar los cambios de configuración siguientes para habilitar el estado real de los CI replicados:

1. En **UCMDBServer\deploy\axis2\WEB-INF**, abra el archivo **web.xml**.
2. Agregue el parámetro **servlet init** siguiente a AxisServlet (pegue estas cuatro líneas después

de la línea 28):

```
<init-param>
<param-name>axis2.find.context</param-name>
<param-value>false</param-value>
</init-param>
```

Esta configuración impide que Axis2 intente calcular la raíz de contexto y le indica que la busque explícitamente en **axis2.xml**.

3. En **UCMDBServer\deploy\axis2\WEB-INF\conf**, abra el archivo **axis2.xml**.
4. En la línea 58, elimine comentarios del parámetro **contextRoot** y edítelo de la manera siguiente:

```
<parameter name="contextRoot" locked="false">test/axis2</parameter>
```

(donde **test** es el nuevo contexto raíz en **cmdb.xml**).

> **Nota:** No hay ninguna barra al inicio de **test/axis2**.

Casos de uso

Los casos de uso siguientes asumen dos sistemas:

- HP Universal CMDB Servidor de
- Un sistema de terceros que contiene un repositorio de elementos de configuración

Esta sección incluye los siguientes temas:

Rellenado de CMDB

Casos de uso:

- Una gestión de activos de terceros actualiza CMDB con información disponible solo en la gestión de activos
- Un número de sistemas de terceros pueden rellenar CMDB para crear un CMDB central que puede realizar el seguimiento de los cambios y realizar el análisis de impacto.
- Un sistema de terceros crea elementos de configuración y relaciones de acuerdo con la lógica empresarial de un tercero, para aprovechar las capacidades de consulta de CMDB

Consulta de CMDB

Casos de uso:

- Un sistema de terceros obtiene los elementos de configuración y las relaciones que representan el sistema SAP obteniendo los resultados de SAP TQL
- Un sistema de terceros obtiene la lista de servidores Oracle que se han agregado o modificado en las últimas cinco horas
- Un sistema de terceros obtiene la lista de servidores cuyo nombre de host contiene la subcadena *lab*
- Un sistema de terceros encuentra los elementos relacionados con un CI determinado obteniendo sus vecinos

Consulta del modelo de clase

Casos de uso:

- Un sistema de terceros permite a los usuarios especificar el conjunto de datos que se va a recuperar desde CMDB. Una interfaz de usuario se puede construir en el modelo de clase para mostrar a los usuarios las posibles propiedades y solicitarles los datos requeridos. El usuario puede elegir después la información que se va a recuperar.
- Un sistema de terceros explora el modelo de clase cuando el usuario no puede acceder a la interfaz de usuario de UCMDB.

Análisis del impacto de cambio

Caso de uso:

Un sistema de terceros produce una lista de los servicios empresariales que podrían verse impactados por un cambio en un host especificado.

Ejemplos

Esta sección incluye los siguientes temas:

Clase base de ejemplo

```
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;
 }
}
```

Ejemplo de consulta

```
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;
```

```
    }
}
```

Ejemplo de actualización

```
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 si = 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();
          }
      }
  }
```

Ejemplo de modelo de clase

```
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
        }
    }
}
```

Ejemplo de análisis de impacto

```
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
        }
}
}
```

Ejemplo de adición de credenciales

```
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
```

Capítulo 11

API de Data Flow Management

Este capítulo incluye:

Información general de la API de Administración de Data Flow

Este capítulo explica cómo las herramientas de terceros o personalizadas pueden utilizar el servicio web de Administración de Data Flow de HP para administrar Administración de Data Flow.

Para obtener toda la información sobre las API disponibles, consulte *HP Discovery and Dependency Mapping Schema Reference*. Estos archivos están ubicados en la carpeta siguiente:

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

Convenciones

Este capítulo utiliza las convenciones siguientes:

- Este estilo `Element` indica que un elemento es una entidad de la base de datos o un elemento definido en el esquema, incluidas las estructuras pasadas a, o devueltas por métodos. El texto sin formato indica que el elemento se analiza en un contexto general.
- Los elementos de Administración de Data Flow y los argumentos del método se deletrean en caso de que sean específicos en el esquema. Esto significa generalmente que un nombre de clase o una referencia genérica a una instancia de la clase se escribe con mayúscula inicial. Los elementos o argumentos a un método no se escriben en mayúsculas. Por ejemplo, `credential` es un elemento del tipo `Credential` pasado a un método.

Servicio web de Data Flow Management

El servicio web de HP Data Flow Management es una API que se utiliza para integrar aplicaciones con HP Universal CMDB. La API proporciona métodos para:

- **Administrar credenciales**. Ver, agregar, actualizar y quitar.
- **Administrar trabajos**. Ver estado, activar y desactivar.
- **Administrar intervalos de sondas**. Ver, agregar y actualizar.
- **Administrar activadores**. Agregar o quitar un CI de activación, y agregar, quitar o deshabilitar una consulta de activación
- **Datos generales de la vista**. Datos sobre dominios y sondas.

Los usuarios de la API del servicio web de HP Data Flow Management deberían estar familiarizados con:

- La especificación SOAP
- Un lenguaje de programación orientado a objetos, como C++, C# o Java
- HP Universal CMDB
- Administración de Data Flow

Permisos

El administrador proporciona credenciales de inicio de sesión para conectarse con el servicio web. Los niveles de permiso son Ver, Actualizar y Ejecutar. Para ver los permisos requeridos para cada operación, consulte la documentación de solicitud de cada operación en *HP Discovery and Dependency Mapping Schema Reference*.

Llamar al servicio web

El servicio web de Asignación de dependencias de detección de HP permite llamar a métodos del servidor mediante técnicas de programación SOAP estándar. Si la instrucción no se puede analizar o si hay un problema al invocar al método, los métodos de la API lanzan una excepción `SoapFault`. Cuando se lanza una excepción `SoapFault`, el servicio rellena uno o varios de los campos de mensaje de error, código de error y mensaje de excepción. Si no hay ningún error, se devuelven los resultados de la invocación.

Para llamar al servicio, utilice:

- `Protocol: http o https (en función de la configuración del servidor)`
- `Dirección URL: <Procesamiento de datos BSM>:8080/axis2/services/DiscoveryService`
- `Contraseña predeterminada: "admin"`
- `Nombre de usuario predeterminado: "admin"`

Los programadores de SOAP pueden acceder a WSDL en:

- `axis2/services/DiscoveryService?wsdl`

Métodos de Data Flow Management

En esta sección se incluye una lista de las operaciones de servicios web y un breve resumen de su uso. Para consultar la documentación completa de la petición y la respuesta a cada operación, consulte la publicación *HP Discovery and Dependency Mapping Schema Reference*.

Esta sección incluye los siguientes temas:

Estructuras de datos

Estas son algunas de las estructuras de datos que se utilizan en la API de servicio web de Data Flow Management.

CIProperties

`CIProperties` es una colección de colecciones. Cada colección contiene propiedades de un tipo de datos diferente. Por ejemplo, puede haber una colección `dateProps`, una colección `strListProps`, una colección `xmlProps`, etc.

Cada colección de tipos contiene propiedades individuales de ese tipo concreto. Los nombres de estos elementos de propiedades son los mismos que para el contenedor, pero en singular. Por ejemplo, `dateProps` contiene elementos de `dateProp`. Cada propiedad es un par de nombre-valor.

Consulte CIProperties en la publicación *HP Discovery and Dependency Mapping Schema Reference*.

IPList

Una lista de elementos de `IP`, cada uno de los cuales contiene una dirección IPv4.

Consulte IPList en la publicación *HP Discovery and Dependency Mapping Schema Reference*.

IPRange

Un `IPRange` tiene dos elementos, `Start` y `End`. Cada uno de ellos contiene un elemento `Address`, que es una dirección IPv4.

Consulte IPLRange en la publicación *HP Discovery and Dependency Mapping Schema Reference*.

Scope

Dos `IPRanges`. `Exclude` es una colección de `IPRanges` que se deben excluir del trabajo. `Include` es una colección de `IPRanges` que se deben incluir en el trabajo.

Consulte Scope en la publicación *HP Discovery and Dependency Mapping Schema Reference*.

Administración de los métodos de trabajo de detección

activateJob

Activa el trabajo especificado.

Consulte "Ejemplo de código" en la página 297.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo.

deactivateJob

Desactiva el trabajo especificado.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo.

dispatchAdHocJob

Distribuye un trabajo en la sonda ad hoc. El trabajo debe estar activo y contener el CI de activación especificado.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo.
CIID	El Id. del CI de activación.
ProbeName	El nombre de la sonda.
Tiempo de espera	En milisegundos

getDiscoveryJobsNames

Devuelve la lista de nombres de trabajos.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.

Salida

Parámetro	Comentario
strList	La lista de nombres de trabajos.

isJobActive

Comprueba si el trabajo está activo.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo que se va a comprobar.

Salida

Parámetro	Comentario
JobState	Es verdadero (true) si el trabajo está activo.

Gestión de métodos de activación

addTriggerCI

Agrega un nuevo CI de activación al trabajo especificado.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo.
CIID	El Id. del CI de activación.

addTriggerTQL

Agrega un nuevo TQL de activación al trabajo especificado.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo.
TqlName	El nombre del TQL que se va a agregar.

disableTriggerTQL

Evita que el TQL active el trabajo, pero no lo elimina de manera permanente de la lista de consultas que activan el trabajo.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo.

removeTriggerCI

Suprime el CI especificado de la lista de CI que activan el trabajo.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo.
CIID	El Id. del CI de activación.

removeTriggerTQL

Suprime el TQL especificado de la lista de consultas que activan el trabajo.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	Colección de nombres de trabajos que se van a comprobar.
CIID	El Id. del TQL que va a eliminarse.

setTriggerTQLProbesLimit

Restringe las sondas en las que TQL está activo en el trabajo a la lista especificada.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
JobName	El nombre del trabajo.
TqlName	El nombre del TQL.
probesLimit	La lista de las sondas para las que está activo el TQL.

Métodos de datos de sondas y dominios

getDomainType

Devuelve el tipo de dominio.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El nombre del dominio.

Salida

Parámetro	Comentario
domainType	El tipo de dominio.

getDomainsNames

Devuelve los nombres de los dominios actuales.

Consulte "Ejemplo de código" en la página 297.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.

Salida

Parámetro	Comentario
domainNames	La lista de nombres de dominios.

getProbeIPs

Devuelve las direcciones IP de la sonda especificada.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio que se va a comprobar.
probeName	El nombre de la sonda usada en ese dominio.

Salida

Parámetro	Comentario
probeIPs	La "IPList" de las direcciones de la sonda.

getProbesNames

Devuelve los nombres de las sondas en el dominio especificado.

Consulte "Ejemplo de código" en la página 297.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio que se va a comprobar.

Salida

Parámetro	Comentario
probesName	La lista de sondas del dominio.

getProbeScope

Devuelve la definición del ámbito de la sonda especificada.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio que se va a comprobar.
probeName	El nombre de la sonda.

Salida

Parámetro	Comentario
probeScope	El "Scope" de la sonda.

isProbeConnected

Comprueba si la sonda especificada está conectada.

Consulte "Ejemplo de código" en la página 297.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio que se va a comprobar.
probeName	La sonda que se va a comprobar.

Salida

Parámetro	Comentario
isConnected	Verdadero si la sonda está conectada.

updateProbeScope

Establece el ámbito de la sonda especificada, reemplazando el ámbito existente.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio.
probeName	La sonda que se va a actualizar.
newScope	El "Scope" que se establecerá para la sonda.

Métodos de datos de credenciales

addCredentialsEntry

Agrega una entrada de credenciales al protocolo especificado para el domino especificado.

Consulte "Ejemplo de código" en la página 297.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio que se va a actualizar.
protocolName	El nombre del protocolo.
credentialsEntryParameters	La colección de "CIProperties" de las nuevas credenciales.

Salida

Parámetro	Comentario
credentialsEntryID	El Id. de CI de la nueva entrada de credencial.

getCredentialsEntriesIDs

Devuelve los Id. de las credenciales definidas para el protocolo especificado.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio para el que se van a conseguir las credenciales.
protocolName	El nombre de un protocolo usado en ese dominio.

Salida

Parámetro	Comentario
credentialsEntryIDs	La lista de Id. de credenciales para el protocolo del dominio.

getCredentialsEntry

Devuelve las credenciales definidas para el protocolo especificado. Los atributos cifrados se devuelven vacíos.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio para el que se van a conseguir las credenciales.
protocolName	Un protocolo usado en ese dominio.
credentialsEntryID	El Id. de credencial que se va a conseguir.

Salida

Parámetro	Comentario
credentialsEntryParameters	La colección de "CIProperties" de las credenciales.

removeCredentialsEntry

Elimina las credenciales especificadas del protocolo.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio.
protocolName	Un protocolo usado en el dominio.
credentialsEntryID	El Id. de la credencial que va a eliminarse.

updateCredentialsEntry

Establece nuevos valores para las propiedades de la entrada de credenciales especificada.

Las propiedades existentes se eliminan y estas propiedades se establecen. Las propiedades cuyo valor no se ha establecido en esta llamada quedarán sin definir.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
domainName	El dominio en el que se van a actualizar credenciales.
protocolName	Un protocolo usado en el dominio.
credentialsEntryID	El Id. de las credenciales que van a actualizarse.
credentialsEntryParameters	La colección de "CIProperties" que se establecerán como propiedades de las credenciales.

Métodos de actualización de datos

rediscoverCIs

Localiza los activadores que han detectado los objetos de CI especificados y vuelve a ejecutar los activadores.

rediscoverCIs se ejecuta de manera asíncrona. Llame a **checkDiscoveryProgress** para determinar si la redetección está completa.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
CmdbIDs	Colección de los Id. de los objetos que se volverán a detectar.

Salida

Parámetro	Comentario
isSucceed	Es verdadero (true) si la nueva detección de los CI se ha realizado correctamente.

checkDiscoveryProgress

Devuelve el progreso de la llamada a **rediscoverCIs** más reciente en los Id. especificados. La respuesta es un valor de `0` a `1`. Cuando la respuesta es `1`, la llamada a **rediscoverCIs** se ha completado.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
CmdbIDs	Colección de los Id. de los objetos de la llamada para volver a detectar de los cuales se hará un seguimiento.

Salida

Parámetro	Comentario
progress	Un trabajo completado tiene un valor de progress igual a 1. Los trabajos que no se han completado tienen una fracción inferior a 1.

rediscoverViewCIs

Localiza los activadores que han creado los datos para rellenar la vista especificada y vuelve a ejecutar los activadores.

rediscoverViewCIs se ejecuta de manera asíncrona. Llame a **checkViewDiscoveryProgress** para determinar si la redetección está completa.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
viewName	Las vistas que se van a comprobar.

Salida

Parámetro	Comentario
isSucceed	Es verdadero (true) si la nueva detección de los CI se ha realizado correctamente.

checkViewDiscoveryProgress

Devuelve el progreso de la llamada a **rediscoverViewCIs** más reciente en la vista especificada. La respuesta es un valor de 0 a 1. Cuando la respuesta es 1, la llamada a **rediscoverCIs** se completa.

Entrada

Parámetro	Comentario
cmdbContext	Para obtener más información, consulte "CmdbContext" en la página 239.
viewName	La colección de vistas que se van a comprobar.

Salida

Parámetro	Comentario
progress	Un trabajo completado tiene un valor de progress igual a 1. Los trabajos que no se han completado tienen una fracción inferior a 1.

Ejemplo de código

```
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