HP Universal CMDB

para os sistemas operacionais Windows e Linux

Versão do software: 9.02

Guia de Referência do Desenvolvedor

Data de lançamento do documento: outubro de 2010
Data de lançamento do software: outubro de 2010

Avisos Legais

Garantia

As únicas garantias para produtos e serviços HP estão estipuladas nas declarações de garantia que acompanham tais produtos e serviços. Nada neste documento deve ser interpretado como constituindo garantia adicional. A HP não se responsabiliza por erros técnicos ou editoriais, nem omissões contidas neste documento.

As informações aqui contidas estão sujeitas a alteração sem prévio aviso.

Legenda de Direitos Restritos

Software de computador confidencial. Licença válida da HP necessária para posse, uso ou reprodução. Em conformidade com as cláusulas 12.211 e 12.212 da FAR, Software de Computação Comercial, Documentação de Software de Computador e Dados Técnicos para Itens Comerciais são licenciados ao Governo dos EUA sob uma licença comercial padrão do fabricante.

Avisos de Direitos Autorais

Avisos de Marcas Comerciais

Adobe® e Acrobat® são marcas comerciais da Adobe Systems Incorporated.

AMD e o símbolo da seta da AMD são marcas comerciais da Advanced Micro Devices, Inc.

Google™ e Google Maps™ são marcas comerciais da Google Inc.

Intel®, Itanium®, Pentium® e Intel® Xeon® são marcas comerciais da Intel Corporation nos EUA e em outros países.

Java™ é uma marca da Sun Microsystems, Inc. nos EUA.

Microsoft®, Windows®, Windows NT®, Windows® XP e Windows Vista® são marcas registradas da Microsoft Corporation nos EUA.

Oracle é uma marca registrada da Oracle Corporation e/ou de suas afiliadas.

UNIX® é uma marca registrada do The Open Group.

Reconhecimentos

- Este produto inclui software desenvolvido pela Apache Software Foundation (http://www.apache.org/licenses).
- Este produto inclui código OpenLDAP da OpenLDAP Foundation (http://www.openldap.org/foundation/).
- Este produto inclui código GNU da Free Software Foundation, Inc. (http://www.fsf.org/).
- Este produto inclui código JiBX de Dennis M. Sosnoski.
- Este produto inclui o analisador XPP3 XMLPull incluído na distribuição e usado em todo o JiBX, da Extreme! Lab, da Universidade de Indiana.
- Este produto inclui a licença do Office Look and Feels de Robert Futrell (http://sourceforge.net/projects/officelnfs).
- Este produto inclui código JEP - Java Expression Parser da Netaphor Software, Inc. (http://www.netaphor.com/home.asp).

Atualizações da Documentação

A página de título deste documento contém as seguintes informações de identificação:

- Número da versão do software.
- Data de lançamento do documento, que muda toda vez que o documento é atualizado.
- Data de lançamento do software, que indica a data de lançamento desta versão do software.

Para verificar as atualizações recentes ou confirmar se você está usando a edição mais recente de um documento, vá para:

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

Este site requer que você se cadastre para um HP Passport e faça logon. Para se cadastrar e receber um HP Passport ID, vá para:

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

Ou clique no link **New users - please register** (Novos usuários - cadastre-se) na página de logon do HP Passport.

Você também receberá edições novas ou atualizadas se assinar o serviço de suporte ao produto apropriado. Contate seu representante de vendas HP para obter detalhes.

Suporte

Visite o site de suporte da HP Software em:

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

Esse site fornece informações de contato e detalhes sobre os produtos, serviços e suporte oferecidos pela HP Software.

O suporte online da HP Software oferece recursos para o cliente resolver problemas por conta própria. Ele fornece uma maneira rápida e eficiente de acessar as ferramentas interativas de suporte técnico necessárias para você gerenciar seus negócios. Na qualidade de cliente de suporte, você pode se beneficiar usando o site de suporte para:

- Pesquisar documentos de conhecimento que sejam de seu interesse
- Enviar e acompanhar casos de suporte e solicitações de aprimoramento
- Baixar patches de software
- Gerenciar contratos de suporte
- Consultar contatos de suporte HP
- Examinar informações sobre os serviços disponíveis
- Participar de discussões com outros clientes de software
- Pesquisar e se inscrever para treinamento em software

A maioria das áreas de suporte exige que você se cadastre como um usuário do HP Passport e faça logon. Muitas também exigem um contrato de suporte. Para se cadastrar e obter um ID do HP Passport, vá para:

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

Para encontrar mais informações sobre níveis de acesso, vá para:

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

Sumário

Bem-vindo a este guia

Este guia explica como criar e gerenciar adaptadores que permitem enviar e receber dados de repositórios de dados externos e outros CMDBs.

Como este guia está organizado

O guia contém os seguintes capítulos:

Parte I Criando adaptadores de descoberta e integração

Descreve como criar adaptadores.

Parte II Usando APIs

Descreve como trabalhar com as APIs para extrair dados de configuração do HP Universal CMDB.

Quem deve ler este guia

Este guia destina-se aos seguintes usuários do HP Universal CMDB:

- Administradores do HP Universal CMDB
- Administradores de plataforma do HP Universal CMDB
- Administradores de aplicativos do HP Universal CMDB
- Administradores de gerenciamento de dados do HP Universal CMDB

Os leitores deste guia deverão ter um profundo conhecimento sobre administração de sistemas empresariais, familiaridade com conceitos de ITIL e conhecer bem o HP Universal CMDB.

Documentação online do HP Universal CMDB

O HP Universal CMDB inclui a seguinte documentação online:

Leiame. Fornece uma lista de limitações da versão e atualizações de última hora. No diretório raiz do DVD do HP Universal CMDB, clique duas vezes em **readme.html**. Você também pode acessar o arquivo leiame mais atualizado no site do Suporte da HP Software.

Novidades. Fornece uma lista de novos recursos e destaques da versão. No HP Universal CMDB, selecione **Ajuda** > **Novidades**.

Documentação para impressão. Selecione **Ajuda** > **Ajuda do UCMDB**. Os seguintes guias são publicados exclusivamente em formato PDF:

- o PDF do *Guia de Implantação do HP Universal CMDB*. Explica os requisitos de hardware e software necessários para configurar o HP Universal CMDB, como instalar ou atualizar o HP Universal CMDB, como proteger o sistema e como fazer logon no aplicativo.
- o PDF do *Guia do Banco de Dados do HP Universal CMDB*. Explica como configurar o banco de dados (MS SQL Server ou Oracle) necessário para o HP Universal CMDB.
- o PDF do *Guia de Conteúdo de Descoberta e Integração do HP Universal CMDB*. Explica como executar a descoberta para descobrir aplicativos, sistemas operacionais e componentes de rede em execução no seu sistema. Explica também como descobrir dados em outros repositórios através da integração.

A Ajuda Online do **HP Universal CMDB inclui**:

- **Modelagem**. Permite gerenciar o conteúdo do seu modelo de Universo de TI.
- **Gerenciamento de Fluxo de Dados**. Explica como integrar o HP Universal CMDB com outros repositórios de dados e como configurar o HP Universal CMDB para descobrir componentes de rede.
- **Administração do UCMDB**. Explica como trabalhar com o HP Universal CMDB.

- **Referência do Desenvolvedor**. Para usuários com conhecimento avançado do HP Universal CMDB. Explica como definir e usar adaptadores, e como usar APIs para acessar dados.

A Ajuda Online também está disponível em janelas específicas do HP Universal CMDB. Basta clicar na janela e clicar no botão **Ajuda**.

Os manuais online podem ser exibidos e impressos usando o Adobe Reader, que pode ser baixado do site da Adobe (www.adobe.com).

Tipos de tópico

Neste guia, cada assunto é organizado em tópicos. Um tópico contém um módulo distinto de informações de um assunto. Os tópicos são geralmente classificados de acordo com o tipo de informação que contêm.

Essa estrutura foi elaborada para proporcionar acesso mais fácil a informações específicas, dividindo a documentação nos diferentes tipos de informação de que você pode precisar em diferentes momentos.

Três tipos de tópico principais são usados: **Conceitos**, **Tarefas** e **Referência**. Os tipos de tópico são diferenciados visualmente usando ícones.

Tipo de tópico	Descrição	Uso
Conceitos 	Informações de apoio, descritivas ou conceituais.	Obter informações gerais sobre o que um recurso faz.
Tarefas 	**Tarefas instrutivas.** Orientação passo a passo para ajudá-lo a trabalhar com o aplicativo e atingir suas metas. Algumas etapas das tarefas incluem exemplos, usando dados de amostra. As etapas das tarefas podem ou não ser numeradas: ➤ **Etapas numeradas.** Tarefas que são executadas seguindo cada etapa em ordem consecutiva. ➤ **Etapas não numeradas.** Uma lista de operações autocontidas que podem ser executadas em qualquer ordem.	➤ Conhecer o fluxo de trabalho geral de uma tarefa. ➤ Seguir as etapas listadas em uma tarefa numerada para concluí-la. ➤ Executar operações independentes concluindo etapas em uma tarefa não numerada.
	Tarefas de cenários de casos de uso. Exemplos de como executar uma tarefa para uma situação específica.	Saber como uma tarefa poderia ser executada em um cenário realista.

Tipo de tópico	Descrição	Uso
Referência 	**Referência geral.**. Listas e explicações detalhadas de materiais voltados para referência.	Pesquisar uma informação de referência específica, relevante para um determinado contexto.
	Referência da interface do usuário. Tópicos de referência especializados que descrevem em detalhes uma determinada interface do usuário. Ao selecionar **Ajuda sobre esta página** no menu Ajuda do produto, geralmente abrem-se os tópicos da interface do usuário.	Pesquisar informações específicas sobre o que inserir ou como usar um ou mais elementos específicos da interface do usuário, como uma janela, caixa de diálogo ou assistente.
Solução de problemas e limitações 	**Solução de problemas e limitações**. Tópicos de referência especializados que descrevem problemas comumente encontrados e suas soluções, além de listar as limitações de um recurso ou área do produto.	Aumentar sua conscientização sobre problemas importantes antes de trabalhar com um recurso, ou caso você encontre problemas de uso no software.

Recursos online adicionais

Solução de Problemas e Base de Conhecimento acessa a página de solução de problemas (em inglês) do site de suporte da HP Software, onde você pode pesquisar a base de conhecimentos para autossolução. Acesse **Ajuda** > **Solução de Problemas e Base de Conhecimento**. A URL desse site é http://h20230.www2.hp.com/troubleshooting.jsp.

Suporte da HP Software acessa o site de suporte (em inglês), no qual você pode pesquisar a base de conhecimento para autossolução. Você também pode postar e pesquisar em fóruns de discussão de usuários, enviar solicitações de suporte, baixar patches e documentação atualizada etc. Acesse **Ajuda** > **Suporte da HP Software**. A URL desse site é www.hp.com/go/hpsoftwaresupport.

A maioria das áreas de suporte exige que você se cadastre como um usuário do HP Passport e faça logon. Muitas também exigem um contrato de suporte.

Para encontrar mais informações sobre níveis de acesso, vá para:

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

Para se cadastrar e obter uma ID de usuário do HP Passport, vá para:

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

O **site da HP Software** Esse site fornece a você as informações mais-atuais sobre os produtos da HP Software. Inclui lançamentos de softwares, seminários, feiras, suporte ao cliente e muito mais. Acesse **Ajuda** > **Site da HP Software**. A URL desse site é www.hp.com/go/software.

Atualizações da Documentação

A HP Software atualiza as documentações de seus produtos constantemente com novas informações.

Para verificar as atualizações recentes ou confirmar se você está usando a edição mais recente de um documento, vá para o site dos Manuais de Produto da HP Software (http://h20230.www2.hp.com/selfsolve/manuals).

Parte I

Criando adaptadores de descoberta e integração

1

Criação e desenvolvimento do adaptador

Este capítulo inclui:

Conceitos

Tarefas

Conceitos

Visão geral da criação e desenvolvimento de adaptador

Antes de começar o planejamento real do desenvolvimento de novos adaptadores, é importante compreender os processos e as interações normalmente associados a esse desenvolvimento.

As seguintes seções podem ajudá-lo a compreender o que você precisa de saber fazer para gerenciar e executar com êxito um projecto de desenvolvimento de descoberta.

Este capítulo:

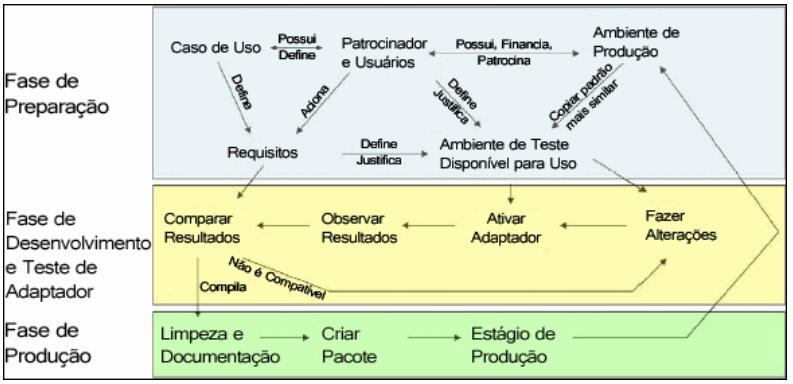
- Têm como pressuposto um conhecimento práticoHP Universal CMDB e familiaridade básica com os elementos do sistema. Seu propósito é ajudar no processo de aprendizagem e não proporciona um manual completo.
- São abordadas as fases de planejamento, pesquisa e implementação de novo conteúdo de descoberta do HP Universal CMDB, juntamente com as diretrizes e as considerações que devem ser levadas em conta.
- Além disso, informação sobre as principais APIs da estrutura do Gerenciamento de Fluxo de Dados são fornecidas. Para ver a documentação completa sobre as APIs disponíveis, consulte *Referência da API de Gerenciamento de Fluxo de Dados do HP Universal CMDB*. (Existem outras APIs informais, porém embora sejam usadas em adaptadores já prontos, podem estar sujeitas à mudança.)

Criação de conteúdo

Esta seção inclui:

O ciclo de desenvolvimento do adaptador

A seguinte ilustração mostra um fluxograma da escrita de adaptadores. A maior parte do tempo é dedicada à seção intermediária, que representa o loop iterativo do desenvolvimento e teste.



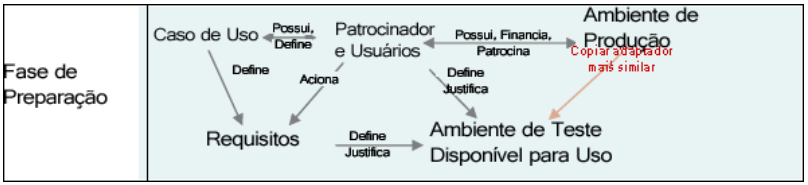
Cada fase do desenvolvimento do adaptador baseia-se na fase anterior.

Quando você estiver satisfeito com o adaptador e o modo como funciona, estará pronto para inseri-lo em um pacote. Usando o Gerenciador de Pacotes do UCMDB ou a exportação manual dos componentes, crie um arquivo *.zip do pacote. É prática recomendada distribuir e testar esse pacote em outro sistema com o UCMDB antes de sua instalação real, para assegurar-se de que todos os componentes estejam presentes e tenham sido inseridos no pacote com êxito. Para ver detalhes sobre a criação de pacotes, consulte "Gerenciador de Pacotes" no *Guia de Administração do HP Universal CMDB*.

As seções a seguir descrevem em detalhes cada um das fases e mostram as etapas e melhores práticas essenciais:

- Fase de pesquisa e preparação
- Desenvolvimento e teste do adaptador
- Empacotamento e produtização do adaptador

Fase de pesquisa e preparação

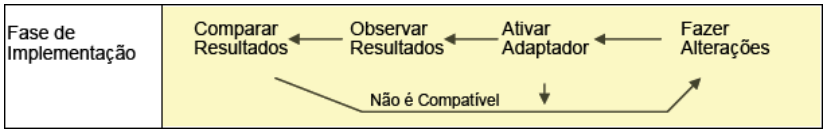


A fase de pesquisa e preparação abrange as razões comerciais e os casos de uso, além de esclarecer os requisitos necessários para desenvolver e testar o adaptador.

1. Se você planeja alterar um adaptador existente, a primeira etapa técnica é fazer backup desse adaptador e assegurar-se de que é possível retorná-lo ao seu estado original. Se você planeja criar um adaptador novo, copie o adaptador mais similar e salve-o usando um nome apropriado. Para ver detalhes, consulte "Painel Recursos" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
2. Pesquise como o adaptador deve coletar dados:
   - Use ferramentas/protocolos externos para obter os dados
   - Desenvolva como o adaptador deve criar ECs baseado nos dados
   - Você já sabe como deve ser um adaptador similar
3. Determine o adaptador com o maior número de características em comum de acordo com:
   - ECs criados
   - Protocolos usados (SNMP)
   - Tipo de destino (por tipo e versão do sistema operacional, e assim por diante)

4 Copie o pacote inteiro.

5 Descompacte-o no espaço de trabalho e renomeie os arquivos do adaptador (XML) e Jython (.py).



Desenvolvimento e teste do adaptador

A fase de desenvolvimento e teste do adaptador é um processo altamente iterativo. Quando o adaptador começa a tomar forma, você começa a testá-lo em relação aos casos de uso, faz mudanças, testa outra vez e repete esse processo até que o adaptador atenda aos requisitos.

Início e preparação da cópia

- Altere partes do XML do adaptador: Nome (id) na linha 1, tipos de EC criados e nome do script Jython chamado.
- Verifique se a cópia obtém resultados idênticos ao adaptador original.
- Insira comentários para deixar de fora a maior parte do código, especialmente o código que produz os resultados mais importantes

Desenvolvimento e teste

- Use outro código de amostra para desenvolver mudanças
- Execute-o para testar o adaptador
- Use uma exibição dedicada para validar resultados complexos, e uma pesquisa para validar resultados simples

Empacotamento e produtização do adaptador

Afase de empacotamento e produtização do adaptador é a última fase do desenvolvimento. Como uma melhor prática, deve-se realizar uma fase final de eliminação de elementos, documentos e comentários remanescentes da depuração, para verificar as considerações sobre segurança etc., antes de passar ao empacotamento. Você deve sempre ter pelo menos um documento leiame para explicar o funcionamento essencial do adaptador. Alguém (ou você mesmo) pode precisar usar esse adaptador posteriormente e qualquer documentação, mesmo que limitada, poderá ser muito útil.

Limpeza e documentação

- Remova a depuração
- Insira comentários para todas as funções e alguns comentários no início da seção principal
- Crie uma TQL e exibição de amostra para o test de usuários

Criação do pacote

- Exporte adaptadores, TQL etc. para o Gerenciador de Pacotes. Para ver detalhes, consulte "Gerenciador de Pacotes" no *Guia de Administração do HP Universal CMDB*.
- Verifique todas as dependências do seu pacote em relação a outros pacotes, por exemplo, se os ECs criados por aqueles pacotes são ECs de entrada do seu adaptador.
- Use o Gerenciador de Pacotes para criar um arquivo zip do pacote. Para ver detalhes, consulte "Gerenciador de Pacotes" no *Guia de Administração do HP Universal CMDB*.
- Teste a implementação removendo partes do novo conteúdo e reimplementando, ou implementando em outro sistema de teste.

Gerenciamento de Fluxo de Dados e integração

Os adaptadores do DFM são capazes de realizar a integração com outros produtos. Considere as seguintes definições:

- O DFM coleta conteúdo específico de muitos destinos.
- A integração coleta vários tipos de conteúdo de um sistema.

Observe que essas definições não distinguem os métodos de coleta. O DFM também não faz essa distinção. O processo de desenvolvimento de um novo adaptador é o mesmo usado para desenvolver uma nova integração. Você faz a mesma pesquisa e as mesmas escolhas em relação a adaptadores novos ou existentes, escreve os adaptadores da mesma maneira, e assim por diante. Somente algumas coisas são diferentes:

- A programação do adaptador final. Os adaptadores de integração podem ter de ser executados com mais frequência do que os de descoberta, mas isso depende dos casos de uso.
- ECs de entrada:
  - Integração: acionador que não seja um EC a ser executado sem dados de entrada: uma origem ou nome de arquivo é passado por meio do parâmetro do adaptador.
  - Descoberta: usa ECs CMDB comuns para a entrada de dados.

Em projetos de integração, você quase sempre reutilizará um adaptador existente. A direção da integração (do HP Universal CMDB a outro produto, ou de outro produto ao HP Universal CMDB) pode afetar sua abordagem do desenvolvimento. Há pacotes específicos disponíveis que você pode copiar para seus próprios usos, usando técnicas comprovadas.

Do HP Universal CMDB para outro projeto:

- Crie uma TQL que produza os ECs e as relações a serem exportados.
- Use um adaptador wrapper genérico para executar a TQL e insira os resultados num arquivo XML que possa ser lido pelo produto externo.

Observação: Para obter exemplos de pacotes do campo, contate Suporte da HP Software.

Para integrar outro produto ao HP Universal CMDB: O adaptador de integração funcionará de acordo com o modo como o outro produto disponibiliza seus dados:

Tipo de integração	Exemplo de referência a reutilizar
Acesso direto ao dados do produto	HP ED
Ler um arquivo .csv ou .xml produzido por uma exportação	HP ServiceCenter
Acesso à API de um produto	BMC Atrium/Remedy

Associando valor comercial ao desenvolvimento da descoberta

O caso de uso do desenvolvimento de novo conteúdo de descoberta deve ser guiado por um caso e um plano de negócios a fim de produzir valor comercial. Isso significa que o objetivo de mapear componentes do sistema a ECs e adicioná-los ao CMDB é criar valor comercial.

O conteúdo talvez não seja sempre usado para mapeamento de aplicativo, contudo essa é uma etapa intermediária em muitos casos de uso. Qualquer que seja o propósito do uso desse conteúdo, seu plano deve responder às seguinte perguntas nessa abordagem:

- Quem é o consumidor? O que o consumidor fará com a informação fornecida pelos ECs (e os relacionamentos entre eles)? Qual é o contexto de negócios em que os ECs e os relacionamentos deverão ser visualizados? Quem é o consumidor desses ECs: uma pessoa, um produto ou ambos?
- Uma vez obtida a combinação perfeita de ECs e relacionamentos no CMDB, como isso será usado para produzir valor comercial?
- Como deve ser o mapeamento perfeito?
  - Que termo descreveria de forma mais significativa os relacionamentos entre ECs?
  - Que tipos de EC seriam os mais importantes a incluir?
  - Que é propósito e usuário final do mapa?
- Como seria o layout perfeito do relatório?

Uma vez estabelecidas as razões comerciais, a próxima etapa é converter o valor comercial em um documento. Isto significa a representação do mapa perfeito usando uma ferramenta de desenho e a compreensão do impacto e das dependências entre ECs, relatórios, registro de mudanças, que mudanças são importantes, monitoração, conformidade e valor comercial adicional segundo as exigências dos casos de uso.

Esse desenho (ou modelo) é chamado de **plano gráfico**.

Por exemplo, caso seja vital que o aplicativo saiba que um determinado arquivo de configuração foi alterado, o arquivo deverá ser mapeado e vinculado ao EC apropriado (ao qual se relaciona) no mapa desenhado.

Trabalhe em conjunto com especialista da área que seja o usuário final do conteúdo desenvolvido. Esse especialista deve indicar as entidades essenciais (ECs com atributos e relacionamentos) que devem existir no CMDB para proporcionar valor comercial.

É possível usar a abordagem de fornecer um questionário ao proprietário do aplicativo (neste caso, o especialista). O proprietário deve poder especificar os objetivos e o plano gráfico descritos acima. O proprietário deve fornecer ao menos a arquitetura atual do aplicativo.

Você deve mapear apenas dados essenciais e nenhum dado desnecessário: você poderá aprimorar o adaptador mais tarde. O objetivo deve ser estabelecer uma descoberta limitada que funcione e proporcione valor. Mapear grandes quantidades de dados produz mapas mais impressionantes, porém seu desenvolvimento pode ser desconcertante e demorado.

Uma vez que o modelo e o valor comercial estejam claros, passe à etapa seguinte. Essa etapa pode ser revisitada à medida que informações mais concretas sejam fornecidas nas etapas seguintes.

Pesquisando os requisitos da integração

O pré-requisito dessa etapa é um **plano gráfico** dos ECs e relacionamentos que devem ser descobertos pelo DFM, o que deve incluir os atributos que deverão ser descobertos. Para ver detalhes, consulte "Visão geral da criação e desenvolvimento de adaptador" na página 20.

Esta seção inclui os seguintes tópicos:

Alterando um adaptador existente

Você pode alterar um adaptador existente quando encontrar um adaptador já disponível no mercado, porém:

- ele não descobrirá atributos específicos necessários
- um tipo específico de destino (sistema operacional) talvez não seja descoberto ou seja descoberto incorretamente
- um relacionamento específico talvez não seja descoberto ou criado

Se você acredita que um adaptador existente faça parte do trabalho (mas não todo o trabalho), sua abordagem deve ser avaliar os adaptadores existentes e verificar se um deles faz quase todo o trabalho necessário; se fizer, você poderá alterá-lo.

Você deve avaliar também se um adaptador de campo existente está disponível. Adaptadores de campo são adaptadores de descoberta que estão disponíveis mas que têm de ser personalizados. Contate o Suporte da HP Software para receber a lista atual de adaptadores de campo.

Escrevendo um novo adaptador

Um adaptador novo precisa ser desenvolvido:

- Quando for mais rápido escrever um adaptador do que introduzir manualmente informações no CMDB (geralmente, cerca de 50 a 100 ECs e relacionamentos) ou quando a tarefa tiver de ser repetida mais de uma vez.
- Quando a necessidade justificar o esforço.
- Se adaptadores predefinidos ou de campo não estiverem disponíveis.
- Se os resultados puderem ser reutilizados.
- Quando o ambiente de destino ou seus dados estiverem disponíveis (você não pode descobrir o que não pode ver).

Pesquisa de modelo

- Consulte o modelo da classe do UCMDB (Gerenciador de Tipo de EC) e faça a correspondência entre as entidades e as relações de seu **plano gráfico** e os TECs existentes. É altamente recomendável aderir ao modelo atual para evitar complicações possíveis durante a atualização de versão. Se você precisar estender o modelo, deverá criar novos TECs, já que uma atualização poderá substituir os TECs predefinidos.
- Se algumas entidades, relações ou atributos estiverem ausentes no modelo atual, você deverá criá-los. É preferível criar um pacote com esses TECs (que posteriormente vão conter toda a descoberta, exibições e outros artefatos relacionados a esse pacote), uma vez que você deverá ser capaz de implementar esses TECs em cada instalação do HP Universal CMDB.

Pesquisa da tecnologia

Após ter verificado que o CMDB contém os ECs relevantes, a etapa seguinte será decidir como recuperar esses dados dos sistemas relevantes.

Recuperar dados geralmente envolve usar um protocolo para acessar uma porção do gerenciamento do aplicativo, dados reais do aplicativo ou bancos de dados e arquivos de configuração relacionados ao aplicativo. Todas as fontes de dados que possam fornecer informações em um sistema são valiosas. A pesquisa da tecnologia exige o conhecimento profundo do sistema em questão e, às vezes, criatividade.

No caso de aplicativos criados internamente, pode ser útil fornecer um questionário ao proprietário do aplicativo. Nesse questionário, o proprietário deve listar todas as áreas no aplicativo que podem fornecer informações necessárias ao plano gráfico e valor comercial. Essas informações devem incluir (mas não se limitar a) bancos de dados de gerenciamento, arquivos de configuração, arquivos de log, interfaces de gerenciamento, programas da administração, serviços Web, mensagens ou eventos enviados, e assim por diante.

No caso de produtos obtidos no mercado, você deve concentrar-se na documentação, fóruns ou suporte do produto. Procure guias de administração, plug-ins e guias de integração, guias de gerenciamento e assim por diante. Se ainda assim não encontrar os dados das interfaces de gerenciamento, leia sobre os arquivos de configuração do aplicativo, entradas do registro, arquivos de log, logs de evento NT e quaisquer artefatos do aplicativo que controlem sua operação correta.

Diretrizes para escolher modos de acesso aos dados

Importância: Selecione fontes ou uma combinação de fontes que forneçam o maior volume de dados. Se uma única fonte fornecer a maioria das informações, enquanto o resto das informações estiver distribuído ou for inacessível, tente avaliar o valor das informações ausentes em relação ao esforço ou risco de obtê-las. Às vezes você pode decidir reduzir o plano gráfico se o valor ou custo não justificarem o esforço investido.

Reutilização: Se o HP Universal CMDB já incluir o suporte a um protocolo de conexão específico, essa é uma boa razão para reutilizá-lo. Isso significa que a estrutura do DFM é capaz de fornecer um cliente e configuração já prontos para a conexão. Do contrário, você pode precisar investir no desenvolvimento da infraestrutura. Você pode ver os protocolos de conexão do HP Universal CMDB com suporte atualmente: **Decoberta > Configurar Sonda de Descoberta > painel Domínios e Sondas**. Para ver detalhes, consulte "Painel Domínios e Sondas" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Você pode adicionar protocolos novos adicionando novos ECs ao modelo. Para obter detalhes, contate o Suporte da HP Software.

Observação: Para acessar dados do Registro do Windows, você pode usar o WMI ou NTCmd.

Segurança: O acesso à informação geralmente exige as credenciais (nome e senha de usuário), que são inseridas no CMDB e mantidas seguras no produto. Se possível, e se adicionar medidas de segurança não entrar em conflito com outros princípios definidos, escolha as credenciais ou protocolos que sejam menos sensíveis mas que atendam às necessidades de acesso. Por exemplo, se a informação estiver disponível tanto via a JMX (interface de administração padrão, limitada) como via Telnet, é preferível usar JMX, pois inerentemente ela fornece acesso limitado e (geralmente) nenhum acesso à plataforma subjacente.

Conveniência: Algumas interfaces de gerenciamento podem incluir recursos mais avançados. Por exemplo, pode ser mais fácil enviar consultas (SQL, WMI) do que navegar árvores de informação ou criar expressões regulares para a análise.

Perfil do desenvolvedor: Desenvolvedores de adaptadores tendem a ter preferência quanto à tecnologia usada. Isso pode ocorrer quando duas tecnologias fornecerem quase a mesma informação a um custo similar em outros fatores.

Resumo

O resultado dessa etapa é um documento que descreve os métodos de acesso e a informação relevante que pode ser extraída por cada método. O documento deve conter também um mapeamento entre as fontes a os dados relevantes do plano gráfico.

Cada método de acesso deve ser marcado de acordo com as instruções acima. Finalmente, você agora tem um plano sobre que fontes descobrir e que informação extrair de cada fonte no modelo de plano gráfico (que agora deve estar mapeado ao modelo do UCMDB correspondente).

Desenvolvendo conteúdo de integração

Antes de criar uma nova integração, você deve compreender as exigências da integração:

- A integração deve copiar dados no CMDB? Os dados devem ser controlados por seu histórico? A fonte de dados é confiável?

  A população de dados é necessária

- A integração deve federar dados imediatamente para exibições e consultas TQL? A precisão das mudanças feitas nos dados é um fator crucial? A quantidade de dados é grande demais para copiar para o CMDB, mas a quantidade de dados solicitados é geralmente pequena?

  A **federação** é necessária.

- A integração deve enviar dados por push a fontes de dados remotas?

  O **push de dados** é necessário.

Observação: Os fluxos de federação e população podem ser configurados para a mesma integração, para a flexibilidade máxima.

Para ver detalhes sobre os diferentes tipos de integração, veja "Integration Studio" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Quatro opções diferentes estão disponíveis para criar adaptadores de integração:

- Adaptador Jython
  - O padrão de descoberta clássico
  - Escrito em Jython
  - Usado para a população

  Para ver detalhes, consulte "Desenvolvendo adaptadores Jython" na página 63.

- Adaptador Java
  - Um adaptador que implementa uma das interfaces de adaptador na estrutura de SDK da federação.
  - Pode ser usado para umas ou várias tarefas de federação, população ou push de dados (de acordo com a implementação exigida).
  - Escrito desde o início em Java, o que permite escrever código para conexão a qualquer fonte ou destino possível.
  - Apropriado para os trabalhos que conectam uma única fonte de dados ou destino.

  Para ver detalhes, consulte "Desenvolvendo adaptadores Java" na página 219.
- Adaptador de banco de dados genérico
  - Um adaptador abstrato baseado no adaptador Java que utiliza a estrutura de SDK de federação.
  - Permite a criação de adaptadores que conectam a repositórios de dados externos.
  - Dá suporte à federação e à população (com um plugin Java implementado para o suporte às mudanças).
  - Relativamente fácil de definir, uma vez que é baseado principalmente em XML e arquivos de configuração de propriedades.
  - A configuração principal é baseada em um arquivo **orm.xml** que faz o mapemanto entre classes do UCMDB e colunas de banco de dados.
  - Apropriado para trabalhos que conectam uma única fonte de dados.

  Para ver detalhes, consulte "Desenvolvendo adaptadores de banco de dados genéricos" na página 125.
- Adaptador push genérico
  - Um adaptador abstrato baseado no adaptador Java (estrutura de SDK de federação) e no adaptador Jython.
  - Permite a criação de adaptadores que enviam dados por push para destinos remotos.

- Relativamente fácil de definir; você precisa definir somente o mapeamento entre classes do UCMDB e XML, e um script Jython que envie os dados por push para o destino.
- Apropriado para trabalhos que conectam um único destino de dados.
- Usado para o envio por push de dados.

Para ver detalhes, consulte "Desenvolvendo adaptadores push" na página 259.

A tabela a seguir mostra as características de cada adaptador:

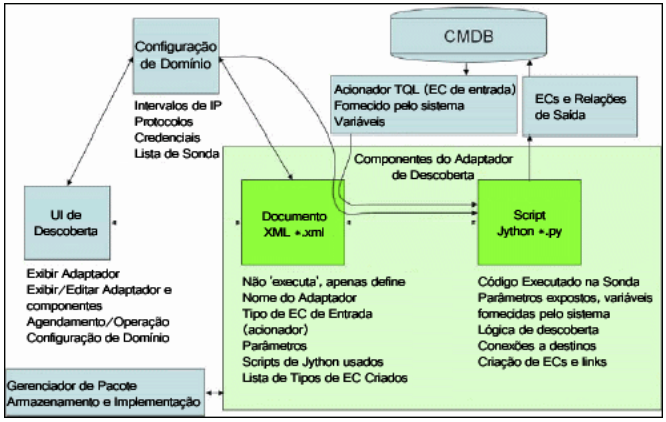
Fluxo/ adaptador	Adaptador Jython	Adaptador Java	Adaptador de BD genérico	Adaptador push
População	X	X	X	
Federação		X	X	
Push de dados		X		X

Desenvolvendo conteúdo de descoberta

Esta seção inclui:

Adaptadores de descoberta e componentes relacionados

O diagrama a seguir mostra os componentes de um adaptador e os componentes com os quais eles interagem para executar a descoberta. Os componentes em verde são os adaptadores em si, e os componentes em azul são os componentes que interagem com adaptadores.



Note que a composição mínima de um adaptador é dois arquivos: um documento XML e um script Jython. A estrutura da descoberta, incluindo ECs de entrada, credenciais e bibliotecas fornecidas pelo usuário, é exposta ao adaptador no tempo de execução. Ambos os componentes do adaptador de descoberta são administrados usando o Gerenciamento de Fluxo de Dados. Em termos de operação, eles são armazenados no próprio CMDB; embora o pacote externo permaneça, nenhuma referência é feita a ele durante a operação. O Gerenciador de Pacote permite preservar a nova capacidade de conteúdo de descoberta e integração.

ECs de entrada são fornecidos ao adaptador por uma TQL, e são expostos ao script do adaptador em variáveis fornecidas pelo sistema. Parâmetros do adaptador também são fornecidos como dados de destino, portanto você pode configurar a operação do adaptador de acordo com uma função específica do adaptador.

O aplicativo DFM é usado para criar e testar novos adaptadores. Você usa as páginas Painel de Controle de Descoberta, Gerenciamento do Adaptador e Configuração da Sonda de Fluxo de Dados durante a escrita do adaptador.

Os adaptadores são armazenados e transportados como pacotes. O aplicativo Gerenciador de Pacote e o console JMX são usados para criar pacotes com os adaptadores recém-criados, e para implementar adaptadores em sistemas novos.

Separando adaptadores

Tecnicamente, uma descoberta inteira pode ser definida em um único adaptador. Mas as boas práticas de design exigem que um sistema complexo seja separado em componentes mais simples e mais fáceis de serem gerenciados.

A seguir estão diretrizes e melhores práticas para dividir o processo do adaptador:

- A descoberta deve ser feita em etapas. Cada etapa deve ser representada por um adaptador que deve mapear uma área ou camada do sistema. Os adaptadores devem depender da descoberta feita na etapa ou camada precedente para continuar a descoberta do sistema. Por exemplo, o adaptador A é acionado pelo resultado de uma TQL de servidor de aplicativos e faz o mapemanto da camada de servidor de aplicativos. Como parte desse mapeamento, um componente de conexão JDBC é mapeado. O adaptador B registra um componente de conexão JDBC como uma TQL acionadora e usa os resultados do adaptador A para acessar a camada do banco de dados (por exemplo, usando o atributo URL do JDBC) e faz o mapeamento da camada de banco de dados.

- **O paradigma da conexão em duas etapas**: A maioria de sistemas exige credenciais para o acesso aos seus dados. Isso significa que uma combinação de usuário/senha precisa ser inserida nesses sistemas. O administrador do DFM fornece ao sistema as informação de credenciais de uma maneira segura, e pode dar diversas credenciais de logon priorizadas. Isso é chamado de **dicionário de protocolo**. Se o sistema não for acessível (seja qual for o motivo) não há razão para executar uma descoberta adicional. Se a conexão for bem-sucedida, é preciso que exista uma maneira de indicar que credenciaisl foram usadas com êxito, para garantir o acesso futuro da descoberta.

  Essas duas fases resultam na separação dos dois adaptadores nos seguintes casos:

  - **Adaptador de conexão**: Esse é um adaptador que aceita um acionador inicial e procura por um agente remoto naquele acionador. Ele faz isso testando todas as entradas no dicionário de protocolo que correspondam ao tipo desse agente. Se for bem-sucedido, esse adaptador fornece como resultado um EC de agente remoto (SNMP, WMI etc.), que também aponta para a entrada correta no dicionário de protocolo para garantir conexões futuras. Esse EC de agente torna-se então parte de um acionador do adaptador de conteúdo.
  - **Adaptador de conteúdo**: O pré-requisito desse adaptador é a conexão bem-sucedida do adaptador precedente (pré-requisito especificado pelas TQLs). Esses tipos de adaptador já não precisam mais pesquisar o dicionário de protocolo inteiro, pois têm uma maneira de obter as credenciais corretas do EC de agente remoto e usá-las para fazer logon no sistema descoberto.

- Considerações de programação diferentes também podem influenciar a divisão da descoberta. Por exemplo, um sistema pode ser consultado somente fora do horário comercial, portanto mesmo que faça sentido juntar esse adaptador a outro adaptador de descoberta de outro sistema, a diferença na programação significa que você deve criar dois adaptadores.
- A descoberta de interfaces de gerenciamento diferentes ou tecnologias diferentes de descoberta do mesmo sistema devem ser colocadas em adaptadores separados. Isso permite ativar o método de acesso apropriado para cada sistema ou organização. Por exemplo, algumas organizações têm o acesso WMI às máquinas mas não têm agentes SNMP instalados nelas.

Tarefas

Implementando um adaptador de descoberta

Uma tarefa do DFM tem como objetivo acessar sistemas remotos (ou locais), modelando dados extraídos como ECs, e salvar os ECs no CMDB. A tarefa consiste nas seguintes etapas:

1 **Adaptador do DFM.**

Você configura um arquivo de adaptador que contém o contexto, os parâmetros e os tipos de resultado selecionando os scripts que deverão fazer parte do adaptador. Para ver detalhes, consulte a seguinte seção.

2 **Trabalho de descoberta.**

Você configura um trabalho com informações de programação e uma TQL acionadora. Para ver detalhes, consulte "Etapa 2: Atribuir um trabalho ao adaptador" na página 50.

3 **Código da descoberta.**

Você pode editar o Jython ou o código Java contido nos arquivos do adaptador e que se refere à estrutura do DFM. Para ver detalhes, consulte "Etapa 3: Criar código Jython" na página 51.

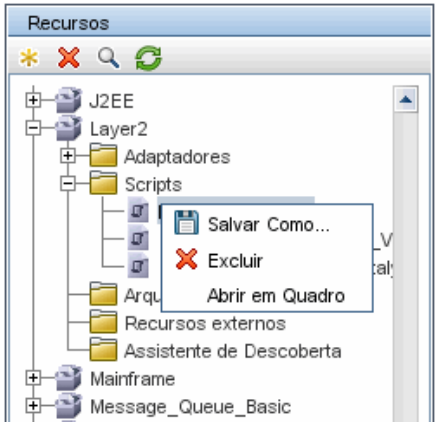
Para escrever novos adaptadores, você cria os componentes acima, e cada um deles é automaticamente associado ao componente da etapa precedente. Por exemplo, após você criar um trabalho e selecionar o adaptador relevante, o arquivo do adaptador será associado ao trabalho.

Código do adaptador

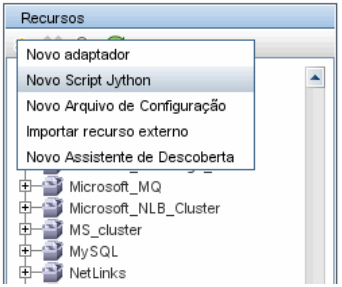
A implementação da conexão ao sistema remoto, a consulta aos seus dados e o seu mapeamento como dados do CMDB são executados pelo código Jython. Por exemplo, o código contem a lógica para a conexão ao banco de dados e a extração de seus dados. Nesse caso, o código espera receber uma URL do JDBC, nome de usuário, senha, portar e assim por diante. Esses parâmetros são específicos para cada instância do banco de dados que responde à consulta TQL. Você define essas variáveis no adaptador (nos dados do EC acionador) e quando o trabalho é executado, esses detalhes específicos são passados ao código para a execução.

O adaptador pode fazer referência a esse código por um nome de classe Java ou um nome de script Jython. Nesta seção, discutiremos como escrever código do DFM na forma de scripts Jython.

Um adaptador pode conter uma lista de scripts a serem usados ao executar uma descoberta. Ao criar um novo adaptador, você geralmente cria um novo script e o atribui ao adaptador. Um novo script inclui modelos básicos, mas você pode usar um dos outros scripts como modelos clicando com o botão direito do mouse e selecionando **Salvar como**:



Para ver detalhes sobre como escrever novos scripts Jython, consulte "Etapa 3: Criar código Jython" na página 51. Você adiciona scripts no painel Recursos:



Os scripts na lista são executados um após o outro, na ordem em que são definidos no adaptador:

Observação: Um script deve ser especificado mesmo que esteja sendo usado unicamente como uma biblioteca por outro script. Nesse caso, o script de biblioteca deve ser definido antes do script que vai utilizá-lo. Nesse exemplo, o script processdbutils.py é uma biblioteca usada pelo último script host_processes.py. As bibliotecas se diferenciam de scripts executáveis comuns pela ausência da função DiscoveryMain().

Etapa 1: Criar um adaptador

Um adaptador pode ser considerado como a definição de uma função. Essa função determina uma definição da entrada, executa a lógica na entrada, define a saída e fornece um resultado.

Cada adaptador especifica a entrada e a saída: A entrada e a saída são ECs acionadores definidos especificamente no adaptador. O adaptador extrai dados do EC acionador de entrada e passa esses dados como parâmetros ao código. (Às vezes, os dados de ECs relacionados também são passados ao código. Para ver detalhes, consulte "Caixa de diálogo ECs Relacionados" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.) Com a exceção desses parâmetros de entrada específicos do EC acionador que são passados a ele, o código de um adaptador é genérico.

Para ver detalhes sobre componentes de entrada, consulte "ECs acionadores e consultas de acionador" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Esta seção inclui os seguintes tópicos:

Definir a entrada do adaptador (TEC acionador e consulta de entrada)

Você usa os componentes TEC acionador e consulta de entrada para definir ECs específicos como entrada de dados do adaptador:

- O TEC acionador define que TEC será usado como entrada do adaptador. Por exemplo, no caso de um adaptador de descoberta de IPs, o TEC de entrada é Rede.
- A consulta de entrada é uma consulta comum, editável e que define a consulta a ser executada no CMDB. A consulta de entrada define limitações adicionais no TEC (por exemplo, se a tarefa exige um hostID ou um atributo application_ip) e pode definir mais dados de EC se necessário ao adaptador.

  Se o adaptador exigir informações adicionais dos ECs relacionados ao EC acionador, você poderá acrescentar nós adicionais à TQL de entrada. Para ver detalhes, consulte "Exemplo de definição de consulta de entrada:" na página 44 e "Adicionar nós de consulta e relacionamentos a uma consulta TQL" no *Guia de Modelagem do HP Universal CMDB*.
- Os dados do EC acionador contêm todas as informações necessárias no EC acionador, assim como informações sobre outros nós na TQL de entrada, caso tenham sido definidos. O DFM usa variáveis para recuperar dados dos ECs. Quando a tarefa é baixada na Sonda, as variáveis de dados do EC acionador são substituídas pelos valores reais que existem nos atributos de instâncias reais de EC.

Exemplo da definição de TEC acionador:

Neste exemplo, um TEC acionador define que ECs de IP são permitidos no adaptador.

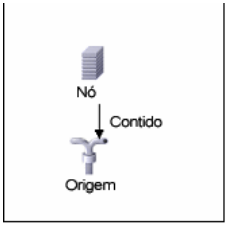
1. Acesse **Gerenciamento de Fluxo de Dados** > **Gerenciamento do Adaptador**. Selecione o adaptador HostProcesses (**Pacotes** > **Host_Resources_Basic** > **Adaptadores** > **HostProcesses**).
2. Encontre a caixa Tipo de EC de Entrada. Para ver detalhes, consulte"Dados do EC Acionado" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
3. Clique no botão para abrir a caixa de diálogo Escolher Classe Descoberta. Para ver detalhes, consulte "Caixa de diálogo Escolher Classe Descoberta" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
4. Selecione o TEC.

   Neste exemplo, o EC IP (Host) é permitido no adaptador:

Exemplo de definição de consulta de entrada:

Neste exemplo, a consulta TQL de entrada define que o EC IP (configurado no exemplo anterior como o TEC acionador) deve ser conectado a um EC Host.

1. Acesse **Gerenciamento de Fluxo de Dados** > **Gerenciamento do Adaptador**. Encontre a caixa TQL de Entrada. Clique no botão **Editar** para abrir o Editor de TQL de Entrada. Para ver detalhes, consulte"Janela Editor de Consulta de Entrada" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.
2. No Editor de TQL de Entrada, nomeie o nó **ORIGEM** do EC acionador: clique com o botão direito do mouse no nó e escolha **Propriedades do Nó**. Na caixa **Nome de Elemento**, mude o nome para **ORIGEM**.
3. Adicione um EC Host e um relacionamento Contém ao EC IP. Para ver detalhes sobre como trabalhar com o Editor de TQL de Entrada, consulte "Janela Editor de Consulta de Entrada" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.



O EC **IP** é conectado a um EC **HOST**. A TQL de entrada consiste em dois nós, **HOST** e **IP**, os quais estão vinculados. O EC **IP** é chamado de **ORIGEM**.

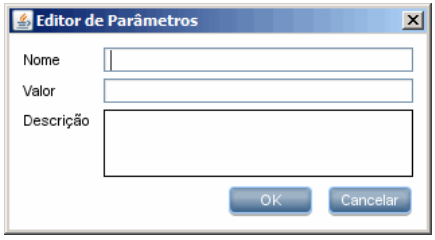
Exemplo da adição de variáveis à consulta TQL de entrada:

Neste exemplo, você adiciona as variáveis DIRECTORY e CONFIGURATION_FILE à consulta TQL de entrada criada no exemplo anterior. Essas variáveis ajudam a definir o que deve ser descoberto: neste caso, os arquivos de configuração que residem nos hosts vinculados ao IPs que você precisa descobrir.

1 Exiba a TQL de entrada criada no exemplo anterior.

Acesse **Gerenciamento de Fluxo de Dados** > **Gerenciamento do Adaptador**. Encontre o painel Dados de EC Acionado. Para ver detalhes, consulte"Dados do EC Acionado" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

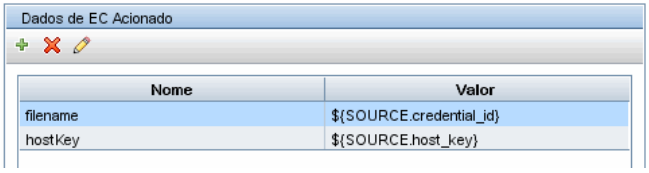
2 Adicione variáveis à TQL de entrada. Para ver detalhes, acesse **Gerenciamento de Fluxo de Dados** > **Gerenciamento do Adaptador**. Encontre o painel Dados de EC Acionado. Para ver detalhes, consulte"Dados do EC Acionado" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.



Exemplo da substituição de variáveis por dados reais:

Neste exemplo, as variáveis substituem os dados do EC IP pelos valores verdadeiros das instâncias reais de EC IP em seu sistema.

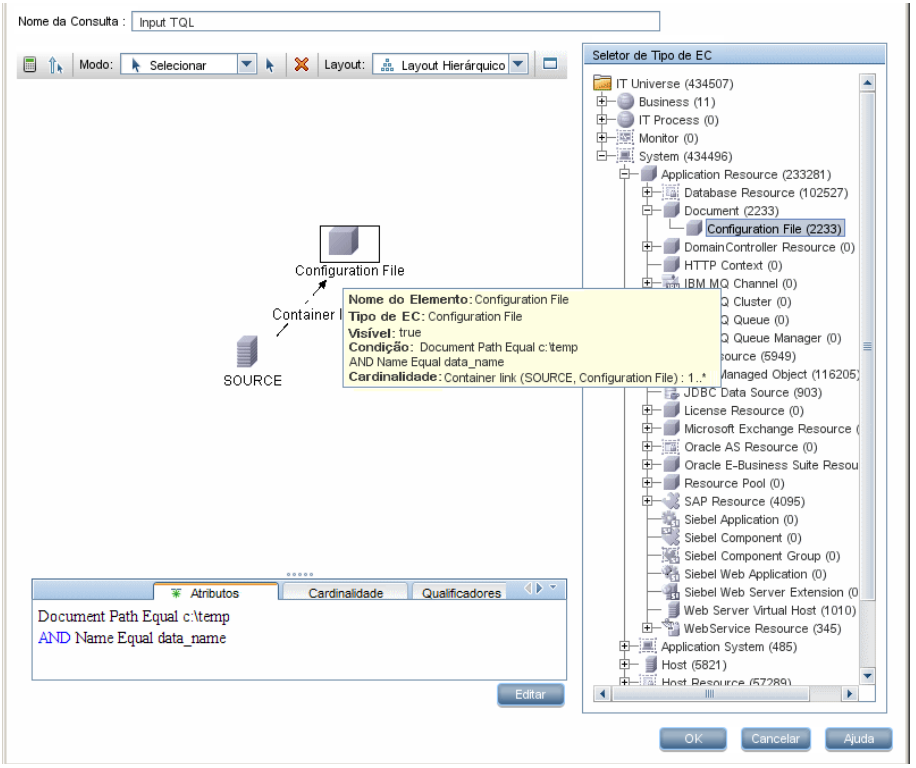
Os dados de EC acionado do EC **IP** incluem uma variável fileName. Essa variável permite a substituição do nó CONFIGURATION_FILE na TQL de entrada pelos valores reais do arquivo de configuração situado em um host:



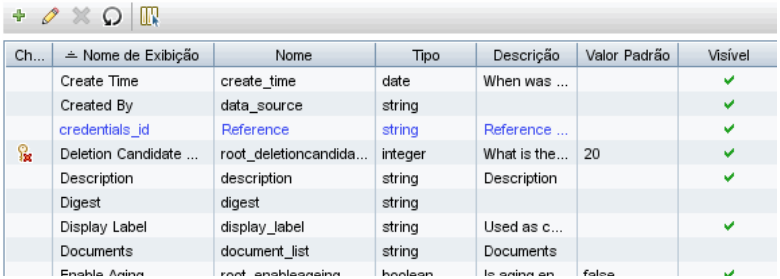
Os dados do EC acionador são carregados na Sonda com todas as variáveis substituídas por valores reais. O script do adaptador inclui um comando para usar a estrutura do DFM para recuperar os valores reais das variáveis definidas:

```
Framework.getTriggerCIData (“ip_address ")
```

As variáveis fileName e path usam os atributos data_name e document_path do nó Arquivo de Configuração (definido na TQL de entrada – consulte o exemplo anterior).



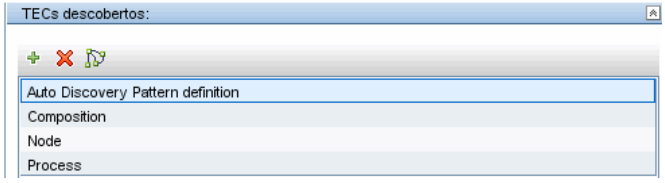
As variáveis Protocol, credentialsId e ip_address usam os atributos root_class, credentials_id e application_ip:



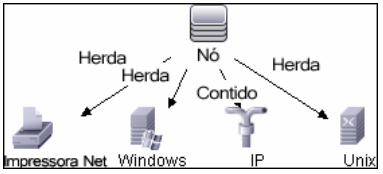
Ch...	Nome de Exibição	Nome	Tipo	Descrição	Valor Padrão	Visível
	Create Time	create_time	date	When was ...		✔
	Created By	data_source	string			✔
	credentials_id	Reference	string	Reference ...		✔
	Deletion Candidate ...	root_deletioncandida...	integer	What is the...	20	✔
	Description	description	string	Description		✔
	Digest	digest	string			
	Display Label	display_label	string	Used as c...		✔
	Documents	document_list	string	Documents		
	Enable Aging	root_enableageing	boolean	Is aging en...	false	✔

Definir a saída do adaptador

A saída do adaptador é uma lista de ECs descobertos (**Gerenciamento de Fluxo de Dados** > **Gerenciamento do Adaptador** > **Definição do Adaptador** > **guia TECs descobertos**) e os vínculos entre eles:



Você pode ver o TECs também como um mapa da topologia, isto é, os componentes e a maneira em que estão vinculados junto (clique no botão **Exibir TECs Descobertos como um Mapa**):



O ECs descobertos são retornados pelo código do DFM (isto é, o script Jython) no formato ObjectStateHolderVector do UCMDB. Para ver detalhes, consulte "Geração de resultados pelo script Jython" na página 72.

Exemplo da saída do adaptador:

Neste exemplo, você define que TECs farão parte da saída do EC IP.

1. Acesse **Gerenciamento de Fluxo de Dados** > **Gerenciamento do Adaptador**.
2. No painel Recursos, selecione **Rede** > **Adaptadores** > **NSLOOKUP_on_Probe**.
3. Na guia Definição do Adaptador, localize o painel TECs Descobertos.
4. Os TECs que devem fazer parte da saída do adaptador estarão listados. Adicione ou remova TECs da lista. Para ver detalhes, consulte "Painel de TECs Descobertos" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

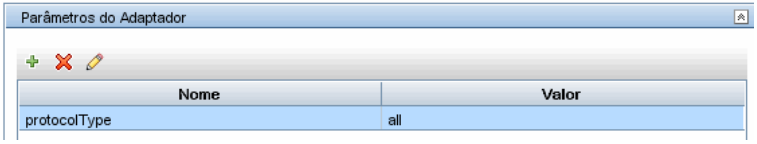
Substituir parâmetros do adaptador

Para configurar um adaptador para mais de um trabalho, você pode substituir parâmetros de adaptador. Por exemplo, o adaptador SQL_NET_Dis_Connection é usado pelos trabalhos MSSQL Connection by SQL e Oracle Connection by SQL

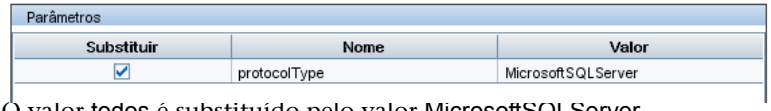
Exemplo da substituição de um parâmetro do adaptador:

Este exemplo ilustra a substituição de um parâmetro do adaptador de modo que um adaptador possa ser usado para descobrir bancos de dados do Microsoft SQL Server e Oracle.

1. Acesse **Gerenciamento de Fluxo de Dados** > **Gerenciamento do Adaptador**.
2. No painel Recursos, selecione **Básico de Banco de Dados** > **Adaptadores** > **SQL_NET_Dis_Connection**
3. Na guia Definição do Adaptador, localize o painel **Parâmetros de Padrão da Descoberta**. O parâmetro protocolType tem o valor **todos**:



4. Clique com botão direito do mouse no adaptador **SQL_NET_Dis_Connection_MsSql** e escolha **Ir para o Trabalho de Descoberta** > **MSSQL Connection by SQL**.
5. Exiba a guia Propriedades. Localize o painel Parâmetros:



O valor todos é substituído pelo valor MicrosoftSQLServer.

Observação: O trabalho **Oracle Connection by SQL** inclui o mesmo parâmetro porém o valor é substituído por um valor Oracle.

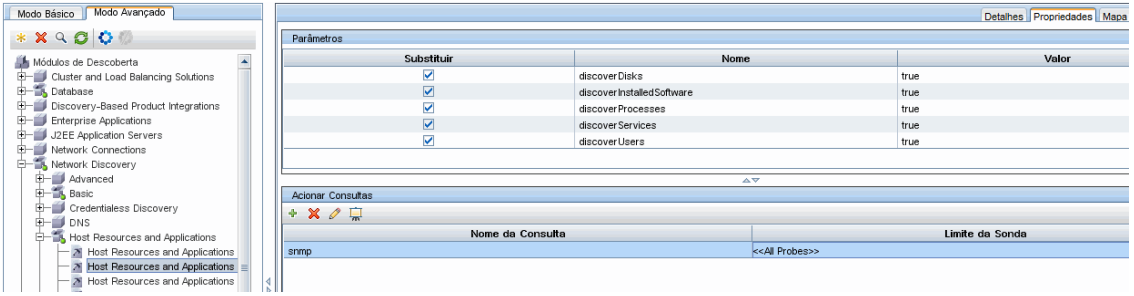
Para ver detalhes sobre a adição, exclusão ou edição de parâmetros, consulte "Painel Parâmetros do Adaptador" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

O DFM começa a procurar por instâncias do Microsoft SQL Server de acordo com esse parâmetro.

Etapa 2: Atribuir um trabalho ao adaptador

Cada adaptador tem um ou mais trabalhos associados que definem a política de execução. Os trabalhos permitem o agendamento do mesmo adaptador de modo diferente de acordo com conjuntos diferentes de ECs acionados, e também permitem passar parâmetros diferentes a cada conjunto.

Os trabalhos aparecem na árvore Módulos de Descoberta, e essa é a entidade que o usuário ativa.



TQL acionadora

Cada trabalho é associado a TQLs acionadoras. Essas TQLs acionadoras publicam os resultados que são usados como ECs acionadores de entrada para o adaptador desse trabalho.

Uma TQL acionadora pode adicionar restrições a uma TQL de entrada. Por exemplo, se os resultados de uma TQL de entrada forem IPs conectados ao SNMP, os resultados de uma TQL acionadora podem ser IPs conectados ao SNMP no intervalo 195.0.0.0-195.0.0.10.

Observação: Uma TQL acionadora deve fazer referência aos mesmos objetos aos quais a TQL de entrada se refere. Por exemplo, se uma TQL de entrada fizer uma consulta quanto aos IPs que executam SNMP, você não poderá definir uma TQL acionadora (para o mesmo trabalho) para fazer uma consulta quanto a IPs conectados a um host, porque alguns dos IPs podem não estar conectados a um objeto SNMP, conforme necessário à TQL de entrada.

Programação

A informação de programação da Sonda especifica quando executar o código em ECs acionadores. Se a caixa de seleção **Invocar Novos ECs Acionados Imediatamente** estiver marcada, o código também será executado uma vez em cada EC acionador quando alcançar a Sonda, qualquer que sejam as configurações de programações futuras.

Para cada ocorrência programada de cada trabalho, a Sonda executa o código em todos os ECs acionadores acumulados do trabalho. Para ver detalhes, consulte "Caixa de diálogo Programador de Descoberta" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Parâmetros

Ao configurar um trabalho, você pode substituir os parâmetros do adaptador. Para ver detalhes, consulte "Substituir parâmetros do adaptador" na página 49.

Etapa 3: Criar código Jython

HP Universal CMDB usa scripts Jython para a escrita de adaptadores. Por exemplo, o script SNMP_Connection.py é usado pelo adaptador SNMP_NET_Dis_Connection para tentar conectar a máquinas usando SNMP. Jython é uma linguagem baseada em Python e desenvolvida em Java.

Para ver detalhes sobre como trabalhar em Jython, consulte estes sites:

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

Para ver detalhes, consulte "Criar código Jython" na página 65.

2

Diretrizes da migração de conteúdo da descoberta

Este capítulo inclui:

Conceitos

Tarefas

Referência

Conceitos

Visão geral das diretrizes da migração de conteúdo da descoberta

No HP Universal CMDB versão 9.0x, o modelo de dados evoluiu significativamente, forçando mudanças relacionadas no código de conteúdo do antigo Mapeamento de Descoberta e Dependência (DDM). Consequentemente, alguns dos mecanismos principais do conteúdo do DDM também foram alterados. Sendo assim, conteúdo desenvolvido para o UCMDB antes da versão 9.0x deve ser atualizado para que corresponda ao modelo de dados 9.0x (Modelo de Dados BDM: BTO). Esta seção descreve o processo de adoção do conteúdo do DDM e seu alinhamento ao BDM.

Para ver detalhes da atualização do HP Universal CMDB, consulte"Atualizando o HP Universal CMDB da versão 8.0x para a versão 9.0x" no o PDF do *Guia de Implantação do HP Universal CMDB*.

Novos recursos de infraestrutura da versão 9.0x

Observação: Para ver detalhes sobre como acessar a documentação online do BDM, consulte"Acessar a documentação online do modelo de dados BTO" na página 60.

Esta seção inclui:

O modelo de dados BTO (BDM)

- Para obter detalhes sobre o modelo de dados BTO (BDM), consulte o documento sobre modelo de dados conceitual Conceptual Data Model. Este documento é um mapa dos conceitos sendo modelados, bem como o escopo do modelo. Este modelo de dados conceitual fornece um ponto de partida para compreender a semântica do domínio modelado.
- Para obter detalhes sobre classes do BDM, consulte o documento sobre referência de modelos de dados BTO HP Software BTO Data Model Reference. Este documento abrange todas as classes do BDM, incluindo informações sobre descrição de classe, atributo, qualificador e hierarquia.

Diferenças entre o modelo de classe do UCMDB 8.0x e o modelo de dados do UCMDB 9.0x

Mudanças feitas desde o modelo de classe do UCMDB versão 8.0x e o BDM são baixadas para a sonda no seguinte arquivo de configuração da Descoberta: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles\flat-class-model-changes.xml**.

bdm_changes.xml. Esse arquivo XML contém informações sobre mudanças feitas em nomes de classes, nomes de atributos, além de remoção de classes, atributos e qualificadores e assim por diante.

- Para obter detalhes sobre o mapeamento entre o modelo de classe do UCMDB versão 8.0x e o BDM, consulte o documento sobre mapeamento Mapping of UCMDB 9.0x (BTO Data Model) to UCMDB 8.0x Class Model.
- Para obter detalhes sobre mudanças no modelo de classe ocorridas entre a versão 8.0x e 9.0x, consulte o documento UCMDB Class Model Changes Report.

Novo mecanismo de identificação de TEC

Em versões do UCMDB anteriores à versão 9.0x, atributos-chave são usados para identificar ECs. No UCMDB versão 9.0x, esse conceito foi generalizado e a identificação agora é feita em um componente do servidor chamado Mecanismo de Reconciliação. O Mecanismo de Reconciliação é capaz de identificar ECs por regras lógicas chamadas regras DDA (algoritmos de definição de dados).

Esse novo mecanismos é muito útil para TECs em que a topologia relacionada é importante para sua identificação, por exemplo, o TEC Nó (Host nas versões anteriores) é identificado por seu nome e topologia relacionada, como endereço IP e TECs de interface. Alguns TECs ainda são identificados por atributos-chave; para esses TECs, uma regra DDA não é definida.

Para ver detalhes sobre o Mecanismo de Reconciliação, consulte "Visão geral da reconciliação" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Mecanismo de software em execução

O EC **Elemento de Software** da versão 8.0x é chamado **Executando Software** no BDM da versão 9.0x. Esse TEC é identificado na versão 9.0x por uma regra de DDA e não por atributos-chave.

Digamos que você tenha adicionado um TEC personalizado derivado do TEC **Executando Software**. Em versões anteriores, esse TEC personalizado era identificado por seus atributos-chave. Contudo, na versão 9.0x, ele é identificado por uma regra de DDA herdada, portanto atributos-chave são ignorados.

Se você adicionar um TEC derivado, considere o seguinte:

- Para identificar o novo TEC pela mesma regra de DDA de todos os TECs "Executando Software", você deverá manter a configuração atual.
- Para identificar o novo TEC por atributos-chave, você deve criar uma nova regra de DDA, definindo a identificação por atributos-chave. A seguir está um exemplo de uma regra de DDA, definida para o TEC **object**:

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

Identificação do lado da sonda

DDM_ID_ATTRIBUTE. A Sonda de Fluxo de Dados versão 9.0x identifica ECs apenas por seus atributos-chave (ou seja, **ID_ATTRIBUTE**). Se um TEC incluir uma regra de DDA (uma regra de reconciliação), o TEC talvez não inclua um atributo-chave. Nesse caso, os principais atributos do TEC são marcados com um qualificador **DDM_ID_ATTRIBUTE**. Sendo assim, para a identificação de um EC, a Sonda leva em conta todos os **DDM_ID_ATTRIBUTE**, bem como os qualificadores **ID_ATTRIBUTE**.

DDM_REQUIRED_TOPOLOGY. Uma regra de DDA para um TEC específico pode depender dos diferentes ECs relatados no mesmo volume que o EC examinado. Por exemplo, a identificação do TEC **Domínio J2EE** é feita não só pelo atributo de nome de domínio, mas também pelo TEC **Servidor de Aplicativo J2EE** conectado a ele com um vínculo membro.

Para assegurar que todos os ECs necessários sejam relatados com o EC examinado, marque cada um dos ECs examinados com o qualificador **DDM_REQUIRED_TOPOLOGY** que contém um item de dados especificando o tipo de vínculo necessário. No exemplo acima, o TEC **Domínio J2EE** está marcado com o qualificador **DDM_REQUIRED_TOPOLOGY** e com um item de dados de vínculo **membro**, portanto quando a Descoberta relatar um domínio J2EE, os servidores também serão relatados.

Para ver detalhes sobre qualificadores, consulte "Página Qualificadores" no *Guia de Modelagem do HP Universal CMDB*.

Camada de transformação

Para garantir a compatibilidade com versões antigas, um novo mecanismo de transformação foi introduzido na versão 9.0x da Sonda. O novo mecanismo é capaz de converter topologias da versão 8.0x para topologias 9.0x em tempo de execução. Ele permite que a Sonda continue a executar tarefas, como scripts Jython, que relata topologias compatíveis com a versão 8.0x.

O novo mecanismo de transformação usa os dados mantidos no arquivo **bdm_changes.xml** e realiza as mudanças necessárias (mudanças de nomes de classe e atributo, remoção de atributo, alteração de hierarquia etc.) para tornar as topologias 8.0x compatíveis com o BDM. Ao mesmo tempo (e independetemente das topologias relatadas pelas tarefas executadas pela Sonda), o UCMDB recebe topologias compatíveis com o BDM.

Utilitário de migração de pacote

A instalação do UCMDB 9.0x inclui um Utilitário de Migração de Pacote que permite que desenvolvedores de conteúdo convertam um pacote de conteúdo do modelo de classe da versão 8.0x para o modelo de dados da versão 9.0x. O Utilitário de Migração de Pacote converte recursos de pacote, incluindo cada subsistema, de modo que eles fiquem compatíveis com o novo modelo de classe. Definições de EC, consultas, trabalhos, adaptadores e módulos são transformados de acordo com os dados que contêm no arquivo **bdm_changes.xml**. Como resultado, eles podem ser implementados e usados por um Servidor do UCMDB 9.0x.

Para ver detalhes, consulte "Atualizando pacotes da versão 8.04 para a 9.02" no o PDF do *Guia de Implantação do HP Universal CMDB*.

Limitações do Utilitário de Migração de Pacote

- Scripts Jython não são atualizados pelo Utilitário de Migração de Pacote. Para dar suporte a scripts criados para corresponder ao modelo de classe da versão 8.0x do UCMDB, foi introduzido um novo módulo de **camada de transformação** no UCMDB 9.0x. Para ver detalhes, consulte "Camada de transformação" na página 57.
- Adaptadores da Descoberta de integração de tipo não são atualizados pelo Utilitário de Migração de Pacote e portanto devem ser atualizados manualmente.
- Em vez de ser atualizado, o trabalho de descoberta Topologia de Camada 2 (e seus recursos correspondentes, como Adaptador de Descoberta, TQL etc.) foi alterado significativamente e removido do Utilitário de Migração de Pacote.

Diretrizes de desenvolvimento de scripts de modelo de dados diversos

As diretrizes a seguir se aplicam tanto à versão 8.0x como 9.0x.

Biblioteca de APIs de scripts de Descoberta

A biblioteca de APIs de scripts de Descoberta é compatível com versões anteriores, portanto todas as bibliotecas e APIs da versão 8.0x têm suporte. Para ver detalhes, consulte "Bibliotecas e utilitários Jython" na página 113.

A API da versão 9.0x API inclui mais elementos e métodos. Por exemplo, um script Jython relata agora um código de erro (inteiro) em vez de uma mensagem de erro (cadeia de caracteres), permitindo assim a exibição de mensagens de erro de descoberta em vários idiomas. Para ver detalhes, consulte "Convenções da criação de mensagens" na página 119.

Dicas de implementação

- Use o módulo **modelagem** para criar um TEC **Executando Software** ou qualquer descendente para o qual o método relevante esteja presente.
- Use **HostBuilder** para criar um TEC do tipo **Nó**.
- Use **modeling.createOshByCmdbIdString** para restaurar o OSH por seu ID.
- Use a instância **ShellUtils** do módulo **shellutils** para todas as conexões baseadas em shell.
- Usa o mecanismo interno para recuperar a versão do UCMDB: logger.Version().getVersion(framework). Por exemplo, se um atributo adicional application_ip for adicionado somente para o UCMDB versão 9.0x ou posterior:

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

- Use **wmiutils** para criar uma descoberta baseada no WMI.
- Use **snmputils** para criar uma descoberta baseada em SNMP.

Tarefas

Acessar a documentação online do modelo de dados BTO

Para acessar a documentação do BDM:

1. Faça logon no HP Universal CMDB.
2. Clique em **Ajuda** > **Ajuda do UCMDB**.
3. Na página inicial, clique no link **Modelagem** sob **Aplicativos** para acessar o portal **Modelagem**.
4. Clique na guia **Modelo de Dados**.

Referência

Solução de problemas e limitações

- O valor de **ip_address** não é automaticamente passado ao padrão. Ele deve ser adicionado explicitamente ao padrão como Dados de EC Acionador.
- Se um script Jython predefinido exigir um recurso ou jar externo no caminho da classe, ele deve estar localizado no pacote relevante em uma subpasta chamada **discoveryResources**.
- Ao trabalhar com atributos do tipo **List**, como **StringVector** e **IntegerVector** (herdados de **BaseVector**), você não pode usar ambas as operações **add element** e **remove element** no mesmo objeto de lista.

3

Desenvolvendo adaptadores Jython

Este capítulo inclui:

Conceitos

Tarefas

Referência

Conceitos

Referência da API do HP Gerenciamento de Fluxo de Dados

Para ver a documentação completa sobre as APIs disponíveis, consulte *Referência da API de Gerenciamento de Fluxo de Dados do HP Universal CMDB*. Esses arquivos estão localizados na seguinte pasta:

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

Tarefas

Criar código Jython

HP Universal CMDB usa scripts Jython para escrita no adaptador. Por exemplo, os scripts SNMP_Connection.py são usados pelo adaptador SNMP_NET_Dis_Connection para testar e conectar a máquinas usando SNMP. Jython é uma linguagem baseada em Python e desenvolvida em Java.

Para ver detalhes sobre como trabalhar em Jython, consulte estes sites:

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

A seção a seguir descreve a escrita real de código Jython no DFM Framework. Esta seção trata especificamente dos pontos de contato entre o script Jython e o Framework que ele chama, além de descrever também as bibliotecas e utilitários Jython que devem ser usados sempre que possível.

Observação:

- Os scripts escritos para o DFM devem ser compatíveis com Jython versão 2.1.
- Para ver a documentação completa sobre as APIs disponíveis, consulte *Referência da API de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Esta seção inclui os seguintes tópicos:

Usar arquivos JAR Java externos em Jython

Ao implantar novos scripts Jython, as bibliotecas Java externas (arquivos JAR) ou os arquivos executáveis de terceiros às vezes são necessários como arquivos de utilitários Java, arquivos de conexão (por exemplo, arquivos JAR de Driver JDBC) ou arquivos executáveis (por exemplo, o **nmap.exe** é usado para descoberta sem credenciais).

Esses recursos devem ser reunidos no pacote na pasta External Resources. Qualquer recurso colocado nessa pasta é enviado automaticamente para qualquer Sonda que se conecta ao servidor HP Universal CMDB.

Além disso, quando a descoberta é iniciada, qualquer recurso de arquivo JAR é carregado no caminho de classe de Jython, tornando todas as classes contidas nele disponíveis para importação e utilização.

Execução do código

Depois que um trabalho é ativado, uma tarefa com todas as informações necessárias é baixada para a Sonda.

A Sonda começa a executar o código do DFM usando as informações especificadas na tarefa.

O fluxo de código Jython começa a ser executado de uma entrada principal no script, executa o código para descobrir ECs e fornece os resultados de um vetor de ECs descobertos.

Modificando scripts prontos

Ao fazer modificações em script prontos, faça somente alterações mínimas no script e coloque quaisquer métodos necessários em um script externo. Você pode controlar as alterações com mais eficiência e, ao migrar para uma versão mais recente do HP Universal CMDB, seu código não será substituído.

Por exemplo, a seguinte linha única de código em um script pronto chama um método que calcula um nome de servidor Web de uma maneira específica de aplicativo:

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

A lógica mais complexa que decide como calcular esse nome está contida em um script externo:

```
# 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
```

Salve o script externo na pasta External Resources. Para ver detalhes, consulte "Painel Recursos" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*. Se você adicionar esse script a um pacote, poderá usar esse script para outros trabalhos também. Para ver detalhes sobre como trabalhar com o Gerenciador de Pacotes, consulte "Gerenciador de Pacotes" no *Guia de Administração do HP Universal CMDB*.

Durante a atualização, a alteração feita na única linha de código é substituída pela nova versão do script pronto, por isso será necessário substituir a linha. Entretanto, o script externo não será substituído.

Estrutura do arquivo Jython

O arquivo Jython consiste em três partes em uma ordem específica:

1. Importações
2. Função principal - DiscoveryMain
3. Definições de funções (opcional)

Veja a seguir um exemplo de um script Jython:

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

# Function definition
def foo:
    # do something

#   Main Function
def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()

    ## Write implementation to return new result CIs here...

    return OSHVResult
```

Importações

As classes Jython são espalhadas pelos namespaces hierárquicos. Na versão 7.0 ou posterior, diferentemente das versões anteriores, não há importações implícitas e cada classe usada precisa ser importada explicitamente. (Essa alteração foi feita para melhorar o desempenho e permitir melhor entendimento do script Jython porque não oculta os detalhes necessários.)

➤ Para importar um script Jython:

```
importar agente de log
```

➤ Para importar uma classe Java:

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

Função principal – DiscoveryMain

Cada arquivo de script executável Jython contém uma função principal: DiscoveryMain.

A função DiscoveryMain é a principal entrada do script e a primeira função executada. A função principal pode chamar outras funções definidas nos scripts:

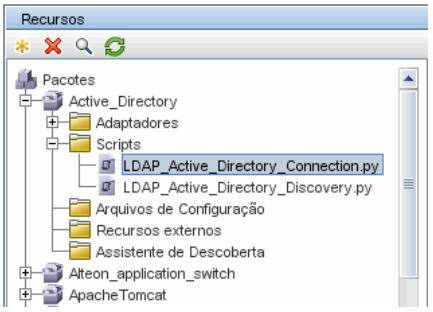
```
def DiscoveryMain(Framework):
```

O argumento Framework precisa ser especificado na definição da função principal. Esse argumento é usado pela função principal para recuperar informações necessárias à execução dos scripts (como informações sobre o EC Acionador e os parâmetros) e também serve para relatar erros que ocorrem durante a execução do script.

Você pode criar um script Jython sem nenhum método principal. Esses scripts são usados como scripts de biblioteca chamados a partir de outros scripts.

Definição de funções

Cada script pode conter funções adicionais chamadas a partir do código principal. Cada função dessas pode chamar outra, que existe no script atual ou em outro script (use a instrução import). Observe que, para usar outro script, você precisa adicioná-lo à seção Scripts do pacote:



Exemplo de uma função que chama outra função:

No exemplo a seguir, o código principal chama o método doQueryOSUsers(..) que chama um 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).createClie
nt()
    except:
        Framework.reportError('Connection failed')
    else:
        doQueryOSUsers(client, OSHVResult)
        client.close()
    return OSHVResult
```

Se esse script for uma biblioteca global relevante para muitos adaptadores, você poderá adicioná-lo à lista de scripts no arquivo de configuração jythonGlobalLibs.xml, em vez de adicioná-lo a cada adaptador (**Gerenciamento do Adaptador** > **Painel Recursos** > **AutoDiscoveryContent** > **Arquivos de Configuração**).

Geração de resultados pelo script Jython

Cada script Jython é executado em um EC Acionador específico e termina com os resultados retornados pelo valor de retorno da função DiscoveryMain.

O resultado do script na verdade é um grupo de ECs e links que serão inseridos ou atualizados no CMDB. O script retorna esse grupo de ECs e links no formato de ObjectStateHolderVector.

A classe ObjectStateHolder é uma maneira de representar um objeto ou link definido no CMDB. O objeto ObjectStateHolder contém o nome do TEC e uma lista de atributos e seus valores. O ObjectStateHolderVector é um vetor de instâncias ObjectStateHolder.

A sintaxe de ObjectStateHolder

Esta seção explica como criar os resultados do DFM em um modelo do UCMDB.

Exemplo de configuração de atributos nos ECs:

A classe ObjectStateHolder descreve o gráfico de resultados do DFM. Cada EC e link (relacionamento) é colocado em uma instância da classe ObjectStateHolder como no exemplo de código Jython a seguir:

```
# siebel application server
1 appServerOSH = ObjectStateHolder('siebelappserver' )
2 appServerOSH.setStringAttribute('data_name', sblsvrName)
3 appServerOSH.setStringAttribute ('application_ip', ip)
4 appServerOSH.setContainer(appServerHostOSH)
```

- A Linha 1 cria um EC de tipo **siebelappserver**.
- A Linha 2 cria um atributo chamado **data_name** com um valor de **sblsvrName** que é um conjunto de variáveis Jython com o valor descoberto para o nome do servidor.
- A Linha 3 define um atributo não-chave que é atualizado no CMDB.
- A Linha 4 é a criação de containment (o resultado é um gráfico). Ela especifica que esse servidor de aplicativo está condito em um host (outra classe ObjectStateHolder no escopo).

Observação: Cada EC sendo relatado pelo script Jython precisa incluir valores para todos os atributos-chave do Tipo de EC do EC.

Exemplo de relacionamentos (links):

O exemplo de link a seguir explica como o gráfico é representado:

```
1 linkOSH = ObjectStateHolder('route')
2 linkOSH.setAttribute('link_end1', gatewayOSH)
3 linkOSH.setAttribute('link_end2', appServerOSH)
```

- A Linha 1 cria o link (que também é a classe ObjectStateHolder. A única diferença é que route é um Tipo de EC de link).
- As Linhas 2 e 3 especificam os nós na extremidade de cada link. Para fazer isso, use os atributos **end1** e **end2** do link que precisa ser especificado (porque são os atributos-chave mínimos de cada link). Os valores de atributo são as instâncias de ObjectStateHolder. Para ver detalhes sobre End 1 e End 2, consulte "Vínculo" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Cuidado: Um link é direcional. Você deve verificar se os nós End 1 e End 2 correspondem a TECs válidos em cada extremidade. Se os nós não forem válidos, o objeto de resultado falhará na validação e não será relatado corretamente. Para ver detalhes, consulte "Relacionamentos de tipos de EC", no *Guia de Modelagem do HP Universal CMDB*.

Exemplo de vetor (reunindo ECs):

Depois de criar objetos com atributos, além de links com objetos em suas extremidades, você precisará agrupá-los. Para fazer isso, adicione-os a uma instância ObjectStateHolderVector da seguinte maneira:

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

Para ver detalhes sobre como relatar esse resultado composto para o Framework para que ele possa ser enviado ao servidor CMDB, consulte o método sendObjects.

Depois que o gráfico de resultados for montado em uma instância ObjectStateHolderVector, será necessário retorná-lo ao DFM Framework para que seja inserido ao CMDB. Para fazer isso, retorne a instância ObjectStateHolderVector como sendo o resultado da função DiscoveryMain().

Observação: para ver detalhes sobre como criar **OSH** para TECs comuns, consulte **modeling.py** em "Bibliotecas e utilitários Jython" na página 113.

Instância do Framework

A instância do Framework é o único argumento fornecido na função principal no script Jython. Trata-se de uma interface que pode ser usada para recuperar as informações necessárias para executar o script (por exemplo, informações sobre ECs Acionadores e parâmetros do adaptador) e que também serve para relatar erros que ocorrem durante a execução do script. Para ver detalhes, consulte "Referência da API do HP Gerenciamento de Fluxo de Dados" na página 64.

Esta seção descreve os principais usos do Framework:

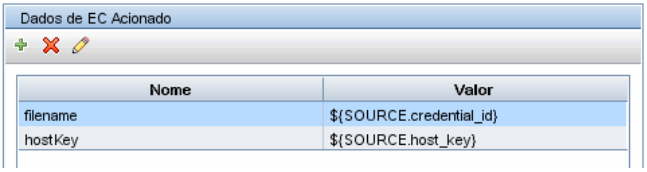
- "Framework.getTriggerCIData(String attributeName)" na página 74
- "Framework.createClient(credentialsId, props)" na página 75
- "Framework.getParameter (String parameterName)" na página 76
- "Framework.reportError(String message) e Framework.reportWarning(String message)" na página 77

Framework.getTriggerCIData(String attributeName)

Essa API proporciona a etapa intermediária entre os dados de EC Acionador definidos no adaptador e no script.

Exemplo de recuperação de informações de credenciais:

Solicite as seguintes informações de dados de EC Acionador:



Para recuperar as informações de credenciais da tarefa, use esta API:

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

Framework.createClient(credentialsId, props)

Faça uma conexão com uma máquina remota criando um objeto cliente e executando comandos nesse cliente. Para criar um cliente, recupere a classe ClientFactory. O método getClientFactory() recebe o tipo do protocolo de cliente solicitado. As constantes do protocolo são definidas na classe ClientsConsts. Para ver detalhes sobre credenciais e protocolos aceitos, consulte "Referências das credenciais de domínio" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Exemplo de criação de uma instância de cliente para a ID de Credenciais:

Para criar uma instância Client para a ID de credenciais:

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

Agora você pode usar a instância Client para se conectar à máquina ou aplicativo relevante.

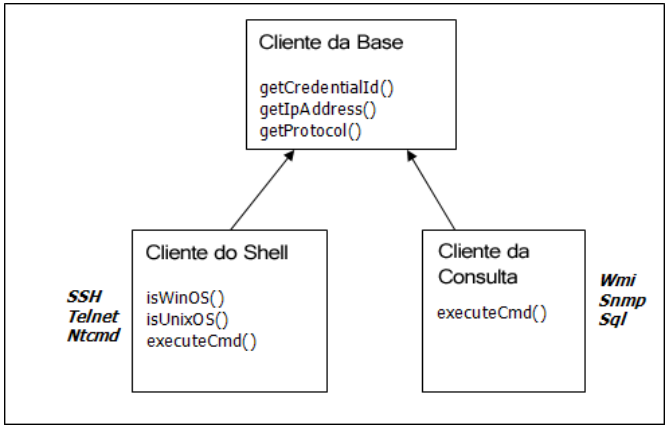
Exemplo de criação de um cliente WMI e execução de uma consulta WMI:

Para criar um cliente WMI e executar uma consulta WMI usando o cliente:

```
wmiClient = Framework.createClient(credential)
resultSet = wmiClient. executeQuery("SELECT TotalPhysicalMemory
                                    FROMWin32_LogicalMemoryConfiguration")
```

Observação: para fazer com que a API createClient() funcione, adicione o seguinte parâmetro aos parâmetros de dados de EC Acionador: **credentialsId = ${SOURCE.credentials_id}** in the Triggered CI Data pane. Ou adicione manualmente a ID de credenciais ao chamar a função: **wmiClient = clientFactory().createClient(credentials_id)**.

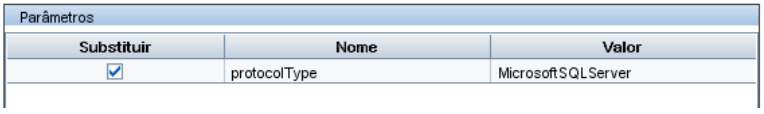
O diagrama a seguir ilustra a hierarquia dos clientes, com suas APIs comumente aceitas:



Para ver detalhes sobre os clientes e suas APIs aceitas, consulte BaseClient, ShellClient e QueryClient no *Referência da API de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Framework.getParameter (String parameterName)

Além de recuperar informações sobre o EC Acionador, geralmente você precisa recuperar um valor de parâmetro do adaptador. Por exemplo:



Exemplo de recuperação do valor do parâmetro protocolType:

Para recuperar o valor do parâmetro protocolType do script Jython, use a seguinte API:

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

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

Alguns erros (por exemplo, falha na conexão, problemas de hardware, tempos limites) podem ocorrer durante uma execução de script. Quando esse tipo de erro é detectado, o Framework pode relatar o problema. A mensagem relatada chega ao servidor e é exibida para o usuário.

Exemplo de um erro de relatório e mensagem:

O diagrama a seguir ilustra o uso da API reportError(<Mensagem_de_Erro>):

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

Você pode usar uma das APIs – Framework.reportError(String message), Framework.reportWarning(String message) – para relatar um problema. A diferença entre as duas APIs é que, ao relatar um erro, a Sonda salva um arquivo de log de comunicação com os parâmetros da sessão inteira no sistema de arquivos. Dessa maneira, é possível controlar a sessão e entender melhor o erro.

Para ver detalhes sobre mensagens de erro, consulte "Mensagens de erro" na página 117.

Localizando as credenciais corretas (para adaptadores de conexão)

Um adaptador que tenta se conectar a um sistema remoto precisa testar todas as credenciais possíveis. Um dos parâmetros necessários ao criar um cliente (através do ClientFactory) é a ID de credenciais. O script de conexão obtém acesso a possíveis conjuntos de credenciais e testa-os um a um com o método clientFactory.getAvailableProtocols(). Quando um conjunto de credenciais obtém êxito, o adaptador relata um objeto de conexão de EC no host desse EC acionador (com a ID de credenciais que corresponde ao IP) ao CMDB. Os adaptadores subsequentes podem usar esse EC de objeto de conexão diretamente para se conectar ao conjunto de credenciais (ou seja, os adaptadores não precisam testar todas as credenciais possíveis outra vez).

O exemplo a seguir mostra como obter todas as entradas do protocolo SNMP. Observe que aqui o IP é obtido dos dados de EC Acionador (# Get the Trigger CI data values).

O script de conexão solicita todas as credenciais de protocolo possíveis (# Go over all the protocol credentials) e testa-as em loop até que uma obtenha êxito (resultVector). Para ver detalhes, consulte a entrada **two-phase connect paradigm** em "Separando adaptadores" na página 36.

```
importar agente de log
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
```

Manipulando exceções de Java

Algumas classes Java lançam uma exceção após falha. É recomendável executar o comando catch na exceção e manipulá-la, caso contrário ela fará com que o adaptador seja encerrado inesperadamente.

Ao executar o comando catch em uma exceção conhecida, na maioria dos casos você deve imprimir o rastreamento de pilha no log e emitir uma mensagem adequada à UI, por exemplo:

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

Se a exceção não for fatal e o script puder prosseguir, você deverá omitir a chamada para o método reportError() e permitir que o script prossiga.

Oferecer suporte a localização em adaptadores Jython

O recurso de localidade multilíngue permite que o DFM funcione em diferentes idiomas de sistemas operacionais e possibilita personalizações apropriadas em tempo de execução.

Anteriormente ao Content Pack 3.00, o DFM usava codificação especificada estatisticamente para tratar a saída de todos os destinos da rede. Entretanto, essa abordagem não atende a uma rede de TI multilíngue: para descobrir hosts com diferentes idiomas de sistema operacional, os administradores da Sonda sempre precisavam repetir, manualmente e várias vezes, a execução dos trabalhos do DFM com diferentes parâmetros de trabalho. Esse procedimento gerava um grave impacto na carga da rede e, pior ainda, bloqueava vários recursos essenciais do DFM, como invocação imediata do trabalho em um EC acionador ou atualização automática de dados no UCMDB pelo Gerenciador de Programação.

Os seguintes idiomas de localidade são aceitos por padrão: japonês, russo e alemão. A localidade padrão é inglês.

Esta seção inclui:

Adicionar suporte para um novo idioma

Esta tarefa descreve como adicionar suporte a um novo idioma.

Esta tarefa inclui as seguintes etapas:

1 Adicionar um bundle de recursos (arquivos *.properties)

Adicione um bundle de recursos de acordo com o trabalho que será executado. A tabela a seguir lista os trabalhos do DFM e o bundle de recursos usado por cada trabalho:

Trabalho	Nome base do bundle de recursos
Monitor de Arquivos por Shell	langFileMonitoring
Recursos e Aplicativos de Host por Shell	langHost_Resources_By_TTY, langTCP
Hosts por Shell Usando NSLOOKUP no Servidor DNS	langNetwork
Conexão de Host por Shell	langNetwork
Coletar Dados de Rede por Shell ou SNMP	langTCP
Recursos e Aplicativos de Host por SNMP	langTCP
Conexão do Microsoft Exchange por NTCMD, Topologia do Microsoft Exchange por NTCMD	msExchange
MS Cluster por NTCMD	langMsCluster

Para obter detalhes sobre bundles, consulte "Trabalhar com bundles de recursos" na página 86.

2 Declarar e registrar o objeto de idioma

Para definir um novo idioma, adicione as duas linhas de código a seguir ao script **shellutils.py**, que no momento contém a lista de todos os idiomas aceitos. O script é incluído no pacote AutoDiscoveryContent. Para visualizar o script, acesse a janela Gerenciamento do Adaptador. Para ver detalhes, consulte "Janela Gerenciamento do Adaptador" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

a Declare o idioma da seguinte maneira:

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

Para ver detalhes sobre idioma de classe, consulte "Referência de API" na página 87. Para ver detalhes sobre o objeto Localidade de Classe, consulte http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html. Você pode usar uma localidade existente ou definir uma nova.

b Registre o idioma adicionando-o à seguinte coleção:

```
LANGUAGES = (LANG_ENGLISH, LANG_GERMAN, LANG_SPANISH,
LANG_RUSSIAN, LANG_JAPANESE)
```

Alterar o idioma padrão

Se não for possível determinar o idioma do sistema operacional, o padrão será usado. O idioma padrão é especificado no arquivo **shellutils.py**.

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

Para alterar o idioma padrão, inicialize a variável DEFAULT_LANGUAGE com um idioma diferente. Para ver detalhes, consulte "Adicionar suporte para um novo idioma" na página 80.

Determinar o conjunto de caracteres para codificação

O conjunto de caracteres adequado para decodificar a saída de comando é determinado em tempo de execução. A solução multilíngue se baseia nos seguintes fatos e pressuposições:

1. É possível determinar o idioma do sistema operacional de maneira independente da localidade, por exemplo, executando o comando **chcp** no Windows ou o comando **locale** no Linux.
2. O relacionamento entre idioma e codificação é bem conhecido e pode ser definido estaticamente. Por exemplo, o idioma russo tem duas das codificações mais conhecidas: Cp866 e Windows-1251.
3. Um conjunto de caracteres para cada idioma é preferencial, por exemplo, o conjunto de caracteres preferencial para o idioma russo é Cp866. Isso significa que a maioria dos comandos produz saída nessa codificação.

4 A codificação em que é fornecida a saída de comando seguinte é imprevisível, mas é uma das codificações possíveis em um determinado idioma. Por exemplo, ao trabalhar com uma máquina com Windows e localidade russo, o sistema fornece a saída de comando **ver** em Cp866, mas o comando **ipconfig** é fornecido em Windows-1251.

5 Um comando conhecido produz palavras-chave conhecidas em sua saída. Por exemplo, o comando **ipconfig** contém a forma traduzida da cadeia de caracteres **IP-Address**. Assim, a saída de comando **ipconfig** contém **IP-Address** para o sistema operacional em inglês, IP-Адрес para o sistema operacional em russo, **IP-Adresse** para o sistema operacional em alemão e assim por diante.

Depois de descoberto o idioma em que a saída de comando é produzida (# 1), os conjuntos de caracteres possíveis ficam limitados a um ou dois (# 2). Além disso, são conhecidas quais palavras-chave estão contidas nessa saída (# 5).

A solução, portanto, é decodificar a saída de comando com uma das codificações possíveis pesquisando uma palavra-chave no resultado. Se a palavra-chave for encontrada, o conjunto de caracteres atual será considerado o correto.

Definir um novo trabalho para operar com dados localizados

Esta tarefa descreve como escrever um novo trabalho que pode operar com dados localizados.

Os scripts Jython normalmente executam comandos e analisam sua saída. Para receber essa saída de comando com decodificação adequada, use a API para a classe**ShellUtils**. Para ver detalhes, consulte "HP Universal CMDB Visão geral da API WS do" na página 290.

Esse código normalmente assume a seguinte forma:

```
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 Crie um cliente:

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

2 Crie uma instância da classe **ShellUtils** e adicione o idioma do sistema operacional a ela. Se o idioma não for adicionado, o idioma padrão será usado (normalmente inglês):

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

Durante a inicialização do objeto, o DFM detecta automaticamente o idioma da máquina e define a codificação preferencial com base no objeto Language predefinido. A codificação preferencial é a primeira instância que aparece na lista de codificação.

3 Recupere o bundle de recursos apropriado do **shellclient** usando o método **getLanguageBundle**:

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

4 Recupere uma palavra-chave do bundle de recursos, adequado a um comando específico:

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

5 Invoque o método **executeCommandAndDecode** e repasse a palavra-chave a ele no objeto **ShellUtils**:

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

O objeto ShellUtils também é necessário para vincular um usuário à referência da API (em que esse método é descrito em detalhes).

6 Analise a saída normalmente.

Decodificar comandos sem uma palavra-chave

A abordagem atual para localização usa uma palavra-chave para decodificar toda a saída de comando. Para ver detalhes, consulte a etapa 4 na página 84 em "Definir um novo trabalho para operar com dados localizados" na página 83.

Entretanto, outra abordagem usa uma palavra-chave para decodificar somente a primeira saída de comando e decodifica outros comandos com o conjunto de caracteres usado para decodificar o primeiro comando. Para fazer isso, use os métodos **getCharsetName** e **useCharset** do objeto **ShellUtils**.

O caso de uso regular funciona da seguinte maneira:

1 Invoque o método **executeCommandAndDecode** uma vez.

2 Obtenha o nome do conjunto de caracteres de uso mais recente através do método **getCharsetName**.

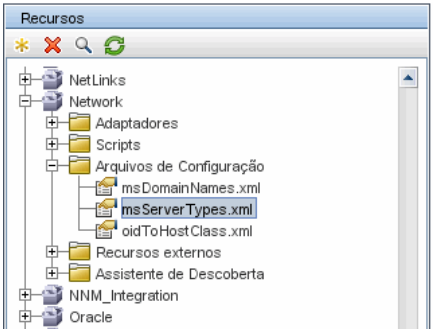
3 Faça com que **shellUtils** use esse conjunto de caracteres por padrão invocando o método **useCharset** no objeto **ShellUtils**.

4 Invoque o método **execCmd** de **ShellUtils** uma ou mais vezes. A saída é retornada com o conjunto de caracteres especificado na etapa 3. Não ocorre nenhuma operação de decodificação adicional.

Trabalhar com bundles de recursos

Um bundle de recursos é um arquivo que tem uma extensão properties (***.properties**). Um arquivo properties pode ser considerado um dicionário que armazena dados no formato de key = value. Cada linha em um arquivo properties contém uma associação key = value. A principal funcionalidade de um bundle de recursos é retornar um valor pela sua chave.

Os bundles de recursos estão localizados na máquina da Sonda: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles**. Eles são baixados do Servidor UCMDB como qualquer outro arquivo de configuração. Podem ser editados, adicionados ou removidos, na janela Recursos. Para ver detalhes, consulte "Painel Arquivo de Configuração" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.



Ao descobrir um destino, o DFM normalmente precisa analisar o texto da saída de comando ou do conteúdo de arquivo. Essa análise geralmente se baseia em uma expressão regular. Diferentes idiomas exigem diferentes expressões regulares para ser usadas para análise. Para que o código seja escrito somente uma vez para todos os idiomas, todos os dados específicos de idioma precisam ser extraídos para os bundles de recursos. Há um bundle de recursos para cada idioma. (Apesar de ser possível que um bundle de recursos contenha dados para diferentes idiomas, no DFM um bundle de recursos sempre contém dados para um idioma.)

O script Jython em si não inclui dados específicos de idioma embutidos no código-fonte (por exemplo, expressões regulares específicas de idioma). O script determina o idioma do sistema remoto, carrega o bundle de recursos adequado e obtém todos os dados específicos de idioma por uma chave específica.

No DFM, o bundles de recursos tem um formato de nome específico: <nome_base>_<identificador_de_idioma>.properties, por exemplo, langNetwork_spa.properties. (O bundle de recursos padrão tem o seguinte formato: <nome_base>.properties, por exemplo, langNetwork.properties.)

O formato nome_base reflete a finalidade desse bundle. Por exemplo, **langMsCluster** significa que o bundle de recursos contém recursos específicos de idioma usados pelos trabalhos do MS Cluster.

O formato identificador_de_idioma é um acrônimo de três letras usado para identificar o idioma. Por exemplo, rus indica o idioma russo e ger indica o idioma alemão. Esse identificador de idioma é incluído na declaração do objeto Language.

Referência de API

Esta seção inclui:

A classe do idioma

Essa classe encapsula informações sobre o idioma, como sufixo do bundle de recursos, codificação possível e assim por diante.

Campos

Nome	Descrição
locale	Objeto Java que representa uma localidade.
bundlePostfix	Sufixo do bundle de recursos. Este sufixo é usado nos nomes de arquivos de bundles de recursos para identificar o idioma. Por exemplo, o bundle **langNetwork_ger.properties** inclui um sufixo de bundle **ger**.
charsets	Os conjuntos de caracteres usados para codificar esse idioma. Cada idioma pode ter vários conjuntos de caracteres. Por exemplo, o idioma russo normalmente tem as codificações Cp866 e Windows-1251.
wmiCodes	A lista de códigos WMI usados pelo sistema operacional Microsoft Windows para identificar o idioma. Todos os códigos possíveis estão listados em http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (na seção OSLanguage). Um dos métodos para identificar o idioma do sistema operacional é consultar o sistema operacional de classe WMI para a propriedade OSLanguage.
codepage	Página de código usada com um idioma específico. Por exemplo, 866 é usado para máquinas em russo e 437 para máquinas em inglês. Um dos métodos para identificar o idioma do sistema operacional é recuperar a página de código padrão (por exemplo, pelo comando chcp).

O método executeCommandAndDecode

Esse método precisa ser usado pelos scripts Jython de lógica de negócios. Ele encapsula a operação de decodificação e retorna uma saída de comando decodificada.

Argumentos

Nome	Descrição
cmd	O comando real que será executado.
keyword	A palavra-chave que será usada para a operação de decodificação.
framework	O objeto Framework repassado para cada script Jython executável no DFM.
timeout	O tempo limite do comando.
waitForTimeout	Especifica se o cliente deve aguardar quando o tempo limite for excedido.
useSudo	Especifica se sudo deve ser usado (relevante somente para clientes de máquina com UNIX).
language	Permite especificar o idioma diretamente em vez de detectar automaticamente um idioma.

O método getCharsetName

Este método retorna o nome do conjunto de caracteres de uso mais recente.

O método useCharset

Este método define o conjunto de caracteres na instância ShellUtils, que usa esse conjunto de caracteres para decodificação de dados inicial.

Argumentos

Nome	Descrição
charsetName	O nome do conjunto de caracteres, por exemplo, tipo de EC, por exemplo, windows-1251 or UTF-8.

Consulte também "O método getCharsetName" na página 89.

O método getLanguageBundle

Este método deve ser usado para obter o bundle de recursos correto. Isso substitui a seguinte API:

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

Argumentos

Nome	Descrição
baseName	O nome do bundle sem o sufixo de idioma, por exemplo, langNetwork.
language	O objeto do idioma. ShellUtils.osLanguage deve ser repassado aqui.
framework	O Framework, objeto comum que é repassado para cada script Jython executável no DFM.

O campo osLanguage

Este campo contém um objeto que representa o idioma.

Trabalhar com o Analisador de Descoberta

A ferramenta Analisador de Descoberta serve para fins de depuração ao desenvolver pacotes, scripts ou qualquer outro tipo de conteúdo. A ferramenta executa um trabalho com base em um destino remoto e retorna logs que contêm informações, avisos e detalhes de erros, além de resultados de ECs descobertos.

Observe que os resultados nem sempre são relatados à UI. Isso ocorre porque os resultados são relatados de duas maneiras e só há suporte a uma delas. Além disso, não há suporte ao log de comunicação do Eclipse.

Ao executar a ferramenta do Eclipse, o arquivo **DiscoveryProbe.properties** (**C:\hp\UCMDB\DataFlowProbe\conf\DiscoveryProbe.properties**) precisa conter o seguinte parâmetro definido como **true**:

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

Para ver detalhes, consulte "Executar o Analisador de Descoberta do Eclipse" na página 100.

Em todos os outros casos (quando a ferramenta é executada do arquivo **cmd** ou enquanto a Sonda está em execução), esse sinalizador precisa ser definido como **false**:

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

Tarefas e registros

Um arquivo de tarefas contém dados relativos a uma tarefa que será executada. A tarefa consiste em informações como o nome do trabalho e os parâmetros obrigatórios que definem o EC acionador, por exemplo, o endereço de destino remoto.

Um arquivo de registro contém informações de tarefas, além dos resultados de uma execução específica, ou seja, a comunicação detalhada (incluindo uma resposta) entre a Sonda ou o Analisador de Descoberta (qualquer módulo que tenha executado a tarefa) e o destino remoto.

Uma tarefa que é definida por um arquivo de tarefa pode ser executada com base em um destino remoto, enquanto uma tarefa que é definida por um arquivo de registro (que contém dados adicionais sobre uma execução específica) pode ser executada e também reproduzida (ou seja, pode reproduzir a mesma execução documentada no arquivo de registro).

Logs

Os logs fornecem informações sobre a execução mais recente, da seguinte maneira:

- **Log geral**. Este log inclui todos os dados de informações, erros e avisos que ocorreram durante a execução.
- **Log de comunicação**. Este log contém a comunicação detalhada entre o Analisador de Descoberta e o destino remoto (incluindo sua resposta). Após a execução, o log pode ser salvo como um arquivo de registro.
- **Log de resultados**. Exibe uma lista de ECs descobertos. O tempo de aparição de cada EC depende do design dos adaptadores e scripts.

É possível salvar todos os logs juntos ou cada um separadamente. Quando você salva todos os logs, eles são salvos juntos sob um único nome.

Se você executar novamente um arquivo de registro, os mesmos dados serão exibidos no log de comunicação, sendo que a única diferença será o horário da execução.

Limitação: Os logs de Comunicação e Resultados não estão disponíveis ao executar o Analisador de Descoberta através do Eclipse.

Esta seção inclui as seguintes etapas:

➤ "Executar a tarefa" na página 97

➤ "Enviar um resultado de tarefa ao servidor" na página 98

➤ "Configurações de importação" na página 98

➤ "Pontos de interrupção" na página 99

1 Pré-requisitos

➤ A Sonda precisa estar instalada. (O Analisador de Descoberta é instalado como parte do processo de instalação da Sonda e compartilha recursos com ela.)

➤ A Sonda não precisa estar em execução enquanto você trabalha com o Analisador de Descoberta.

Entretanto, se a Sonda já tiver sido executada com base no Servidor a UCMDB , todos os recursos necessários já estarão baixados para o sistema de arquivos. Se a Sonda ainda não tiver sido executada, você poderá carregar os recursos necessários pelo Analisador de Descoberta através do menu Configurações. Para ver detalhes, consulte "Configurações de importação" na página 98.

➤ O Servidor CMDB não precisa estar instalado.

2 Acessar o Analisador de Descoberta

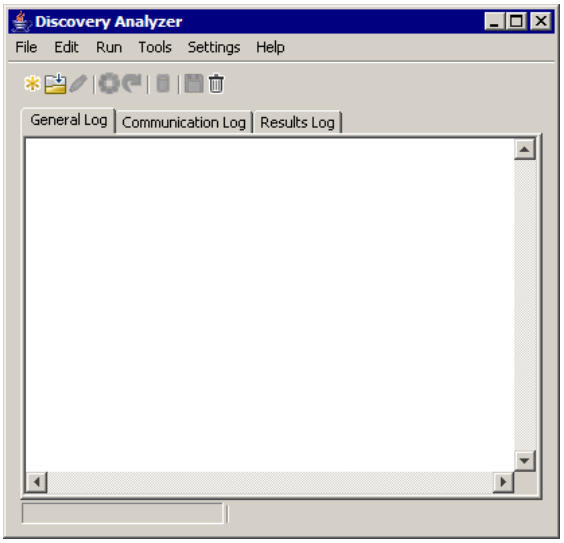
Acesse o Analisador de Descoberta desta maneira:

➤ Ao trabalhar com o Eclipse.

A instalação da Sonda vem com um espaço de trabalho do Eclipse padrão localizado em **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**. Esse espaço de trabalho inclui um script Jython para iniciar o Analisador de Descoberta (**startDiscoveryAnalyzerScript.py**) além de um link para todos os scripts do DFM. Se você iniciar a ferramenta dessa maneira, poderá localizar os pontos de interrupção nos scripts Jython para fins de depuração.

➤ Diretamente, clicando duas vezes no arquivo na seguinte pasta: **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzer.cmd**. Para ver detalhes, consulte a próxima seção.

A janela Analisador de Descoberta será aberta:



3 Definir uma tarefa

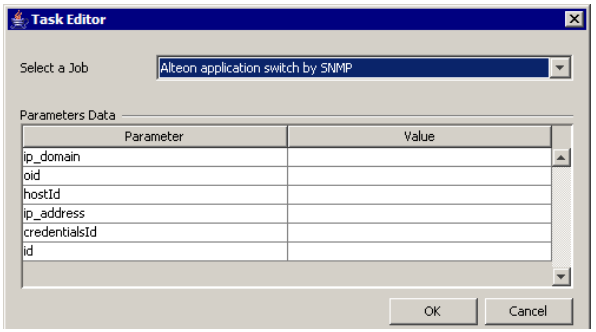
Defina uma tarefa usando um dos métodos a seguir:

- Definindo uma nova tarefa. Para ver detalhes, consulte "Definir uma nova tarefa" na página 95.
- Importando uma tarefa de um arquivo de registro. Para ver detalhes, consulte "Recuperar um registro" na página 96.
- Importando uma tarefa salva de um arquivo de tarefas. Para ver detalhes, consulte "Abrir um arquivo de tarefa" na página 96.
- Recuperando um trabalho do banco de dados interno da Sonda. Para ver detalhes, consulte "Importar uma tarefa do banco de dados" na página 96.

4 Definir uma nova tarefa

a Exiba o Editor de Tarefas: clique no botão **Nova Tarefa**.

O Editor de Tarefas exibe uma lista de trabalhos que existem no momento no sistema de arquivos. Essa lista é atualizada sempre que a Sonda recebe tarefas do servidor ou os pacotes são implantados manualmente do menu Configurações.



b Selecione um trabalho.

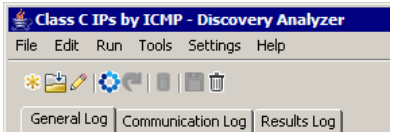
c Insira valores para todos os parâmetros.

Os parâmetros exibidos aqui são os parâmetros do adaptador do DFM. Eles podem ser visualizados no painel Parâmetros de Padrão de Descoberta na guia Assinatura de Padrão. Para ver detalhes, consulte "Painel Parâmetros do Adaptador" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Todos os campos são obrigatórios (a menos que o script de um trabalho exija que o campo seja vazio).

Para parâmetros que exigem uma ID ou valor de entrada de ID de credenciais, você pode usar IDs criados aleatoriamente: clique com o botão direito do mouse na caixa de valor e selecione **Gerar ID do CMDB aleatória** ou **Selecionador de Credencial**.

A tarefa agora está ativa e o nome da tarefa aberta é exibido na barra de título:



d Continue com o procedimento para definir uma tarefa. Para ver detalhes, consulte "Salvar a tarefa e os logs" na página 97.

5 Recuperar um registro

Você pode definir uma tarefa abrindo um arquivo de registro que contém dados relativos a uma execução específica. Se uma tarefa for definida dessa maneira, você poderá reproduzir a execução específica selecionando a opção de reprodução. (Se uma tarefa for executada novamente, as respostas serão recebidas dos dados armazenados no arquivo de registro e não do destino remoto.)

Selecione **Arquivo** > **Abrir Registro**. Navegue até a pasta onde você salvou o registro. O registro agora está ativo e o nome da tarefa é exibido na barra de título.

Para ver detalhes sobre como adquirir um arquivo de registro, consulte "Registrar código do DFM" na página 111.

6 Abrir um arquivo de tarefa

Você pode definir uma tarefa de um arquivo de registro: Selecione **Arquivo** > **Abrir Tarefa**.

7 Importar uma tarefa do banco de dados

Você poderá recuperar uma tarefa do banco de dados da Sonda desde que a Sonda já tenha sido executada e tenha tarefas ativas em seu banco de dados interno. Você pode usar os valores de parâmetro para definir a tarefa.

a Selecione **Arquivo** > **Importar Tarefa do Banco de Dados da Sonda**.

b Na caixa de diálogo que for aberta, selecione a tarefa que será executada e clique em **OK**.

c Continue com o procedimento para definir uma tarefa. Para ver detalhes, consulte "Salvar a tarefa e os logs" na página 97.

8 Editar uma tarefa

Depois que uma tarefa for definida, o nome da tarefa (ou do arquivo) será exibido na barra de título. Agora o arquivo poderá ser editado.

a Selecione **Editar** > **Editar Tarefa**.

b Faça quaisquer alterações na tarefa e clique em **OK**.

9 Salvar a tarefa e os logs

Você pode salvar os parâmetros da tarefa: Selecione **Arquivo** > **Salvar Tarefa**.

As seguintes opções estão disponíveis somente depois que uma tarefa é executada:

- Salve um registro da tarefa. Você pode salvar os parâmetros da tarefa e os resultados da execução da tarefa: Selecione **Arquivo** > **Salvar Registro**.
- Salve um log da tarefa: selecione **Arquivo** > **Salvar Log Geral**.
- Salve os resultados: selecione **Arquivo** > **Salvar Resultados**.

10 Executar a tarefa

A próxima etapa no procedimento é executar a tarefa criada.

a Importe o arquivo de configuração de credenciais/intervalos. Para ver detalhes, consulte "Configurações de importação" na página 98.

b Para executar a tarefa somente com base em um destino remoto, clique no botão **Executar Tarefa**.

O Analisador de Descoberta executa o trabalho e exibe informações nos três arquivos de log: **Geral**, **Comunicação** e **Resultados**.

c É possível salvar os arquivos de log, sejam juntos ou separadamente. selecione **Arquivo** > **Salvar Log Geral**, **Salvar Registro**, **Salvar Resultados** ou **Salvar Todos os Logs**. Para ver detalhes sobre os arquivos de log, consulte "Logs" na página 92.

d Se uma tarefa for recuperada de um arquivo de registro, a execução que está documentada nesse arquivo poderá ser reproduzida clicando no botão **Reproduzir**. O mesmo log de Comunicação será exibido, mas o tempo de execução será atualizado.

11 Enviar um resultado de tarefa ao servidor

Se a execução de uma tarefa for encerrada com os resultados (ou seja, a guia Log de Resultados exibir uma lista de ECs descobertos), você poderá enviar os resultados ao Servidor UCMDB. Isso é útil se, por exemplo, anteriormente você estava testando um script quando o servidor estava inoperante.

Observação: Só é possível enviar resultados para um Servidor UCMDB que receba tarefas da Sonda instalada na mesma máquina do Analisador de Descoberta.

---

12 Configurações de importação

Para executar tarefas no arquivo de registro de reprodução, é necessário importar o arquivo **domainScopeDocument.bin**. Durante a importação, digite uma senha.

a Inicie um navegador da Web e insira a seguinte URL: **http://localhost:8080/jmx-console**. Você pode precisar fazer logon com um nome do usuário e senha.

b Clique em **UCMDB:service=DiscoveryManager** para abrir a página Visualização do JMX MBEAN.

c Localize a operação **exportCredentialsAndRangesInformation**. Siga este procedimento:

- Insira a ID do cliente (o padrão é **1**).
- Insira um nome para o arquivo exportado.
- Insira a senha.
- Defina **isEncrypted** como **False**.

d Clique em **Invoke** para exportar o arquivo **domainScopeDocument.bin**.

Quando o processo de exportação for concluído com êxito, o arquivo será salvo no seguinte local:
C:\hp\UCMDB\UCMDBServer\conf\discovery\<customer_dir>.

e Copie o arquivo **domainScopeDocument.bin** para o sistema de arquivos da Sonda de Fluxo de Dados e importe-o selecionando: **Configurações** > **Importar domainScopeDocument**.

Observação: Durante a importação do arquivo **domainScopeDocument**, você será solicitado a fornecer uma senha. Essa solicitação também é exibida depois de cada reinício do Analisador de Descoberta e antes da execução da primeira tarefa ou registro.

---

13 Pontos de interrupção

Se você executar o Analisador de Descoberta do script Python, poderá adicionar pontos de interrupção ao script.

14 Configurar o Eclipse

Para ver detalhes sobre como executar scripts Jython em modo de depuração, consulte "Executar o Analisador de Descoberta do Eclipse" na página 100.

Executar o Analisador de Descoberta do Eclipse

Esta tarefa explica como configurar o Eclipse para permitir a execução de scripts Jython em modo de depuração, proporcionando assim maior visibilidade dos threads de trabalho, ECs acionadores e resultados.

Esta seção inclui as seguintes etapas:

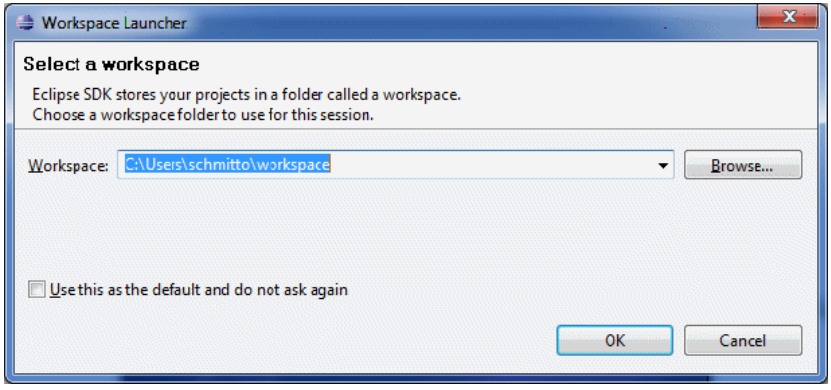
1 Pré-requisitos

- Instale a versão mais recente do Eclipse no computador. O aplicativo está disponível em www.eclipse.org.
- Verifique se a Sonda de Fluxo de Dados está instalada no mesmo computador.
- Verifique se o parâmetro **appilog.agent.local.discoveryAnalyzerFromEclipse** no arquivo **DiscoveryProbe.properties** está definido como **true**.

2 Descompactar o Eclipse e iniciá-lo

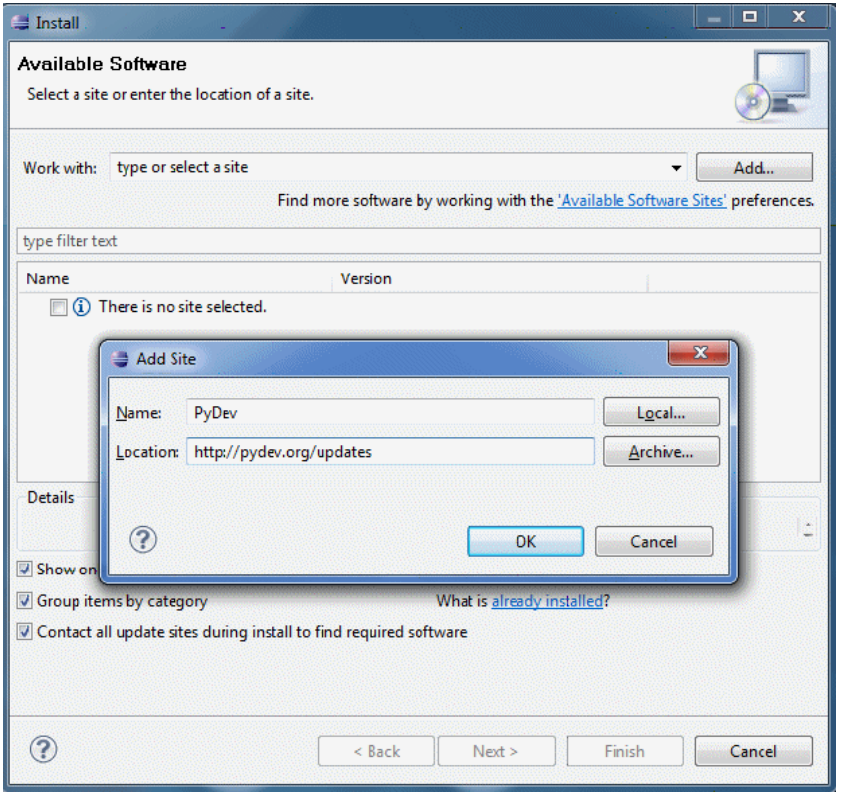
3 Configurar o espaço de trabalho padrão

Configure o espaço de trabalho padrão em que o Eclipse salva e armazena todos os projetos e dados relacionados.



4 Configurar as Extensões PyDev

a Acesse **Ajuda** > **Instalar Novo Software**, clique em **Adicionar**, digite um nome para o plugin PyDev e, no campo Local, adicione a URL do site em que pydev pode ser baixado: **http://pydev.org/updates**. Clique em **OK**.



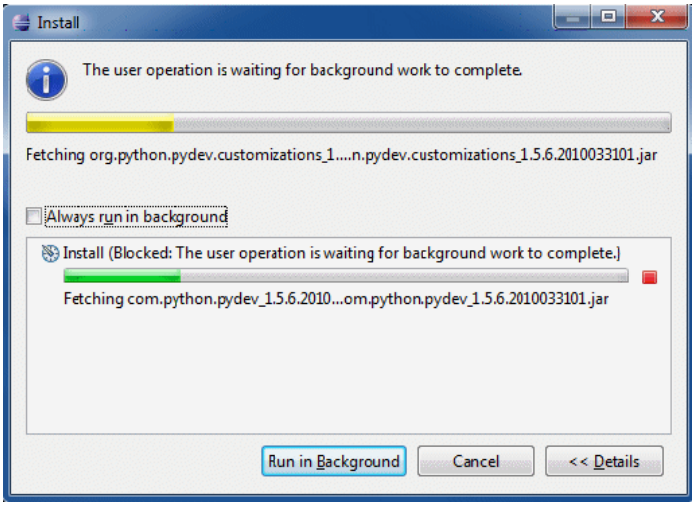
Observação: As Extensões PyDev e PyDev agora são mescladas em um único plugin porque as Extensões PyDev agora têm código-fonte aberto. Para ver informações adicionais, visite http://pydev.org.

---

b Na janela que for aberta, selecione **Pydev**. O segundo plugin serve para UI voltada a tarefas. Clique em **Avançar**, verifique os detalhes da instalação e clique em **Avançar** novamente.

c Aceite o contrato de licença e clique em **Avançar**.

d O Pydev está instalado. Se você for solicitado a instalar conteúdo não assinado, confirme clicando em **OK**.

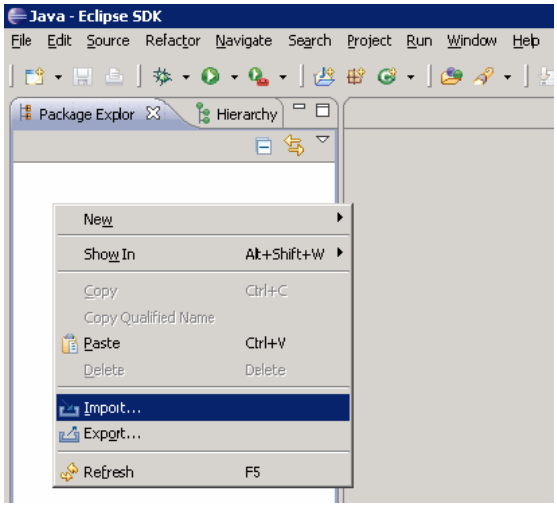


e Reinicie o Eclipse.

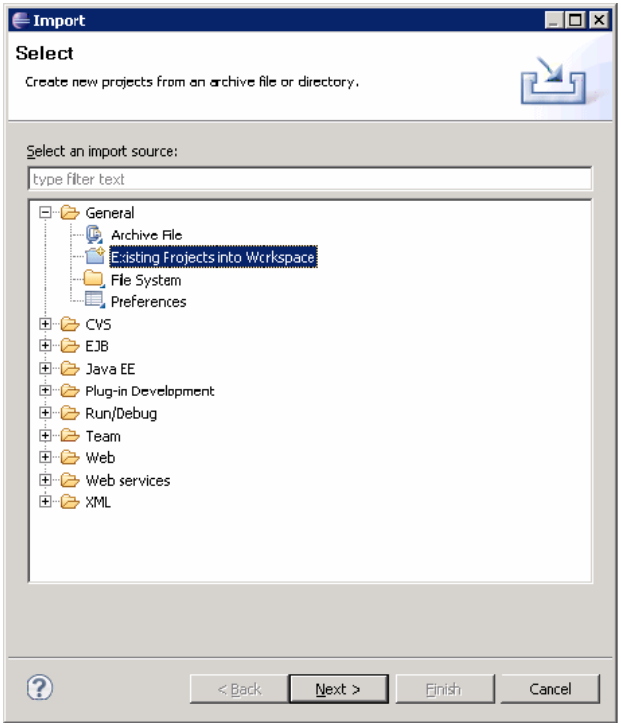
O PyDev agora está instalado na IDE Eclipse. Você tem novas perspectivas no Eclipse e a IDE pode interpretar scripts Python (realce de texto, opções de configuração adicionais e assim por diante).

5 Configurar o espaço de trabalho do Analisador de Descoberta

a Importe os arquivos necessários: Clique com o botão direito do mouse na área em branco no Explorador de Pacotes e clique em **Importar** para importar o **discoveryAnalyzerWorkspace** pré-configurado, incluído com a instalação da Sonda.



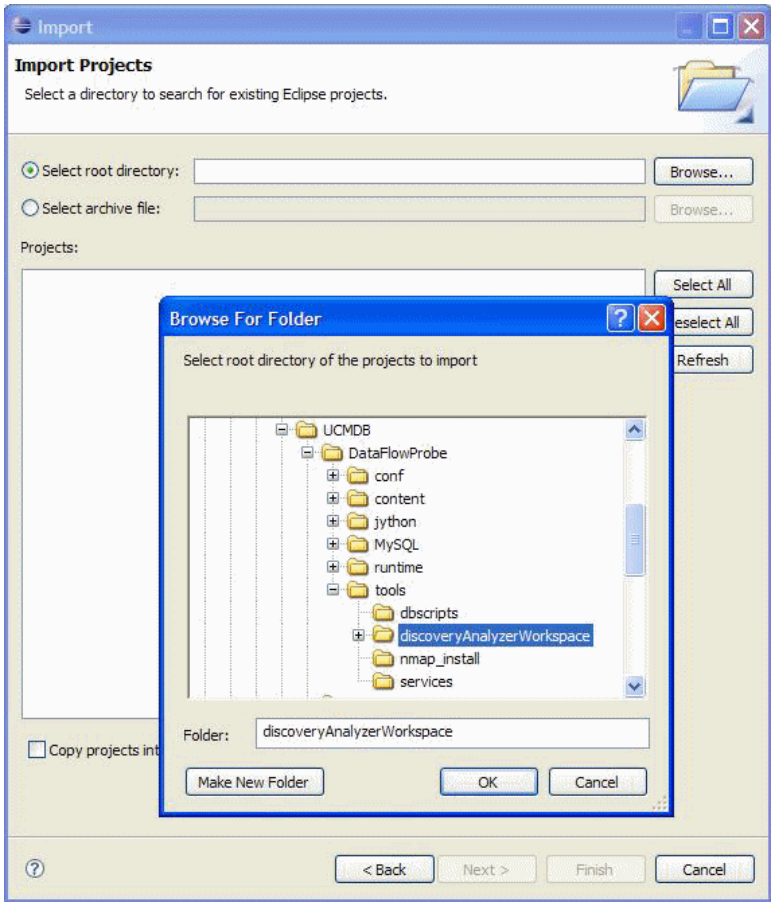
b Em **Geral**, selecione **Projetos existentes no espaço de trabalho** para importar o projeto no espaço de trabalho do Eclipse.

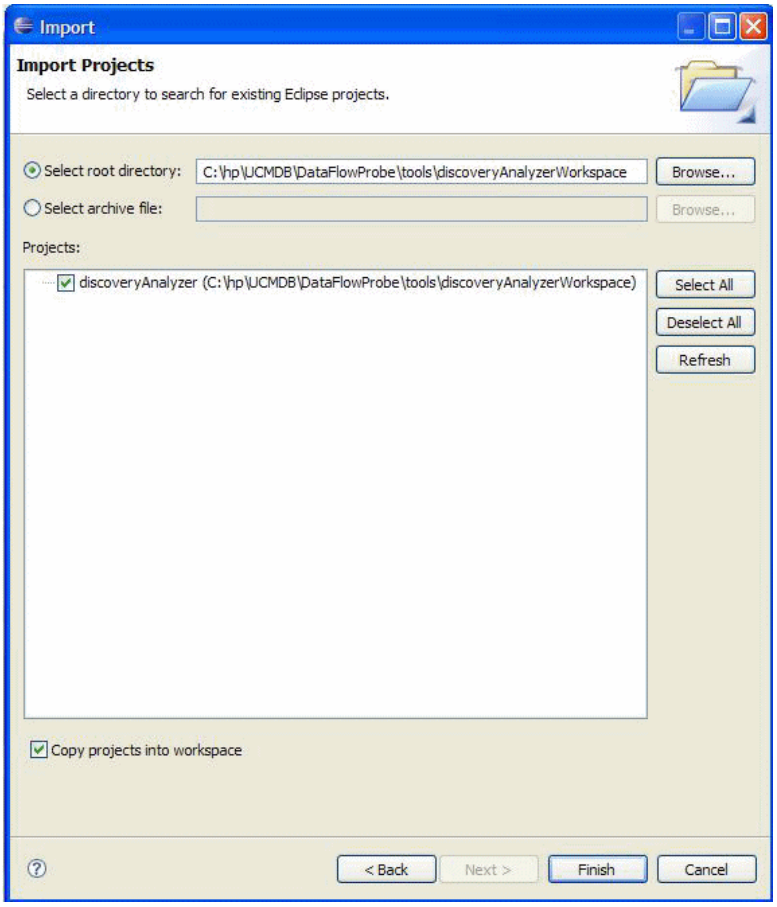


c Em **Selecionar diretório raiz**, selecione o espaço de trabalho do Analisador de Descoberta, normalmente localizado em **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**.

d Selecione **Copiar projetos no espaço de trabalho** para criar uma cópia real do espaço de trabalho existente. Esta é uma etapa importante: Em caso de falha, você pode importar novamente o **discoveryAnalyserWorkspace** original.

e Clique em **Concluir** para iniciar a Importação.





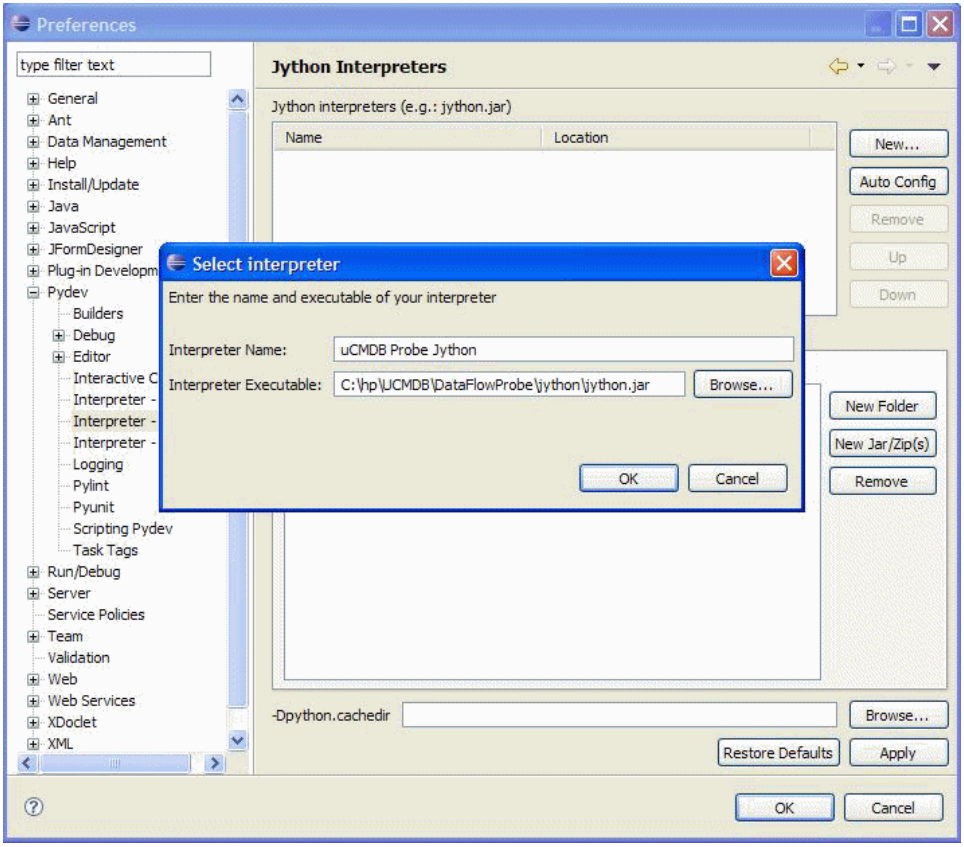
6 Configurar o caminho de classe e o interpretador

a Clique com o botão direito do mouse em **discoveryAnalyzerWorkspace** e selecione **Propriedades** para exibir as configurações específicas do Projeto.

b Vá para **Pydev > Interpreter/Grammar** e clique em **Configurar um interpretador nas preferências relacionadas antes de prosseguir**.

Essa etapa configura o mesmo interpretador Jython que a Sonda está usando para garantir que os scripts não sejam interpretados por uma versão Jython diferente.

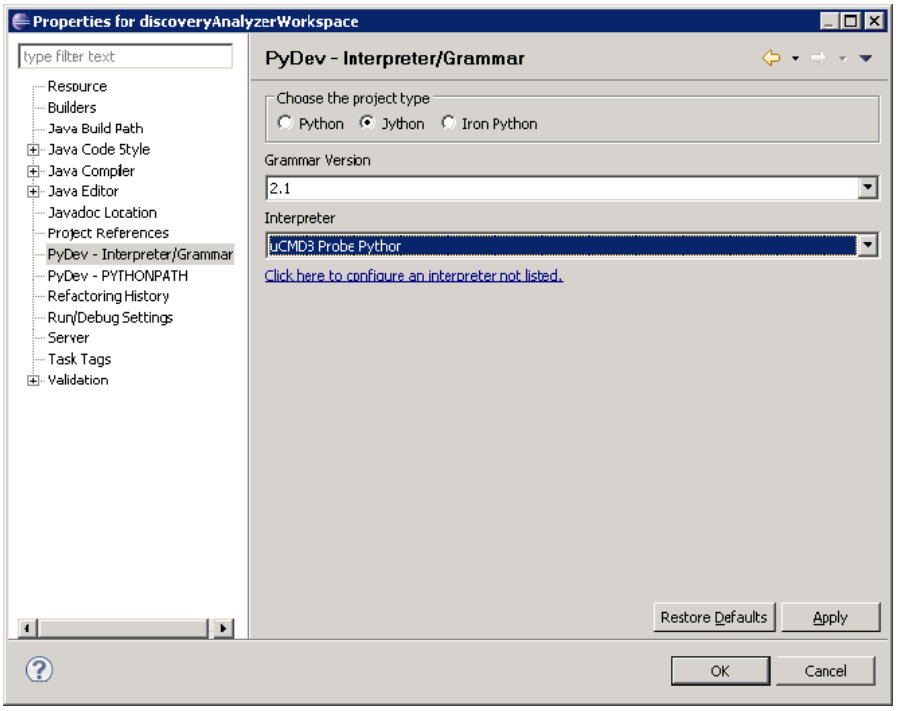
c Clique em **Novo**, digite um nome para o interpretador e selecione o arquivo na seguinte pasta: **C:\hp\UCMDB\DataFlowProbe\jython\jython.jar**.



d Clique em **OK**. Se uma janela for exibida solicitando para você selecionar as pastas que devem ser importadas para o caminho de sistema Python, não altere nada (deve ser **C:\hp\UCMDB\DataFlowProbe\jython** e **C:\hp\UCMDB\DataFlowProbe\jython\lib**) e clique em **OK**.

e Clique em **Aplicar** e em **OK**.

f Clique em **Interpretador** e selecione o interpretador recém-criado.



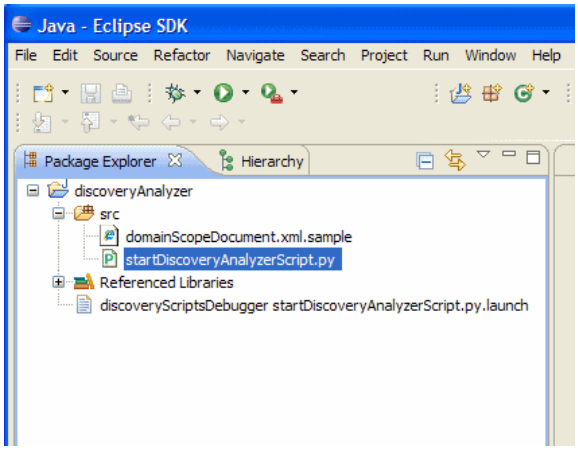
g Clique em **Aplicar** e em **OK**.

O interpretador Jython agora é o mesmo que a Sonda está usando.

7 Executar o Analisador de Descoberta

a Adicione um ponto de interrupção ao script Jython para ser depurado.

b Para iniciar o Analisador de Descoberta, selecione **startDiscoveryAnalyzerScript.py** no projeto **discoveryAnalyzerWorkspace\src** project. Clique com o botão direito do mouse no arquivo e escolha **Depurar como** > **Execução de Jython**.



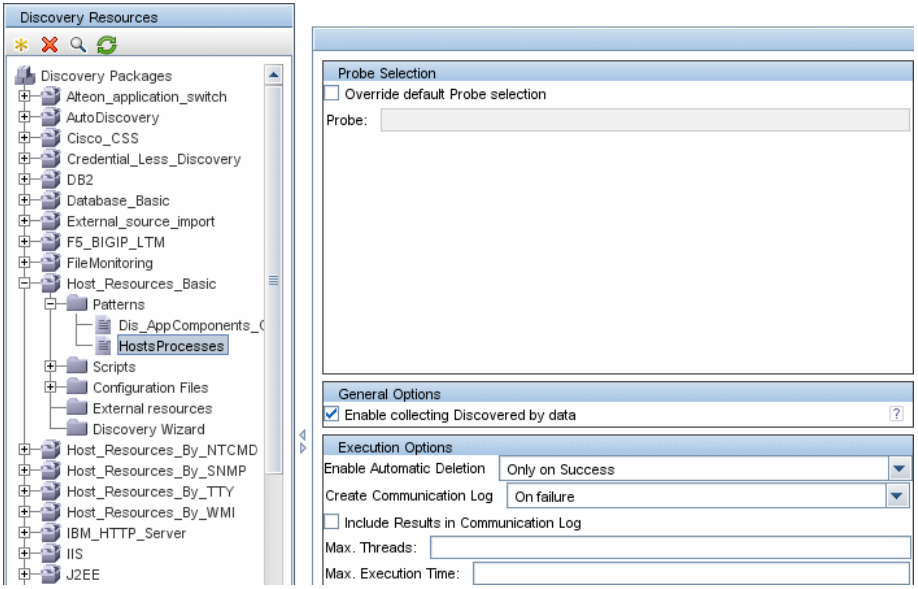
Registrar código do DFM

Pode ser muito útil registrar uma execução inteira, incluindo todos os parâmetros, por exemplo, ao depurar e testar código. Essa tarefa descreve como registrar uma execução inteira com todas as variáveis relevantes. Além disso, você pode visualizar informações de depuração adicionais que normalmente não são impressas nos logs de arquivo mesmo no nível de depuração.

Para registrar um código do DFM:

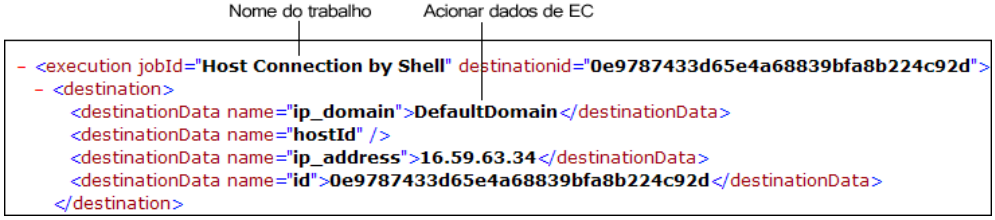
1 Acesse **Gerenciamento do Fluxo de Dados** > **Painel Controle de Descoberta**. Clique com o botão direito do mouse no trabalho cuja execução precisa ser registrada em log e selecione **Editar adaptador** para abrir o aplicativo Gerenciamento do Adaptador.

2 Localize o painel **Opções de Execução** na guia Gerenciamento de Padrões:.

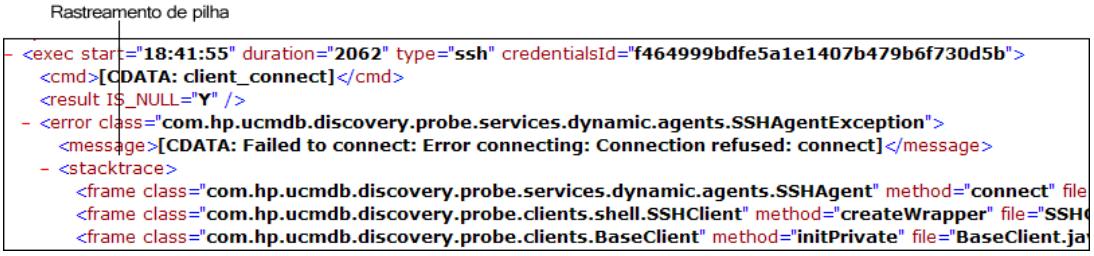


3 Altere a caixa **Criar logs de comunicação** para **Sempre**. Para ver detalhes sobre como configurar opções de registro em log, consulte "Painel Opções de Execução" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

O exemplo a seguir é o arquivo de log XML que é criado quando o trabalho Conexão de Host por Shell é executado e a caixa **Criar logs de comunicação** está definida como **Sempre** ou **Em Falhas**:



O exemplo a seguir mostra a mensagem e os parâmetros de rastreamento de pilha:



Referência

Bibliotecas e utilitários Jython

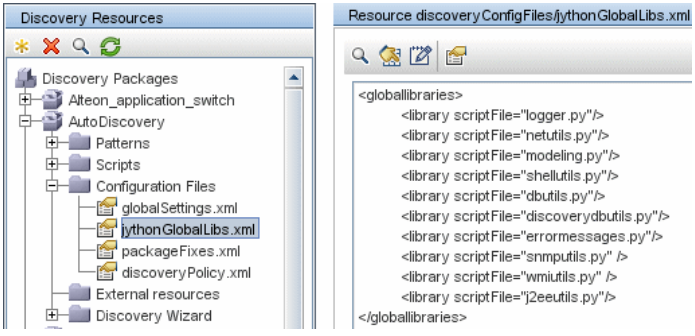
Vários scripts de utilitários são usados amplamente nos adaptadores. Esses scripts fazem parte do pacote AutoDiscovery e estão localizados em: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** com os outros scripts baixados para o Sonda.

Observação: A pasta discoveryScript é criada dinamicamente quando a Sonda inicia o trabalho.

Para usar um dos scripts de utilitário, adicione a seguinte linha de importação à seção de importação do script:

```
import <nome_do_script>
```

A biblioteca Python AutoDiscovery contém scripts de utilitário Jython. Esses scripts de biblioteca são considerados biblioteca externa do DFM. Eles são definidos no arquivo jythonGlobalLibs.xml (localizado na pasta **Arquivos de Configuração**).



Cada script que aparece no arquivo jythonGlobalLibs.xml é carregado por padrão na inicialização da Sonda, por isso não é necessário usá-los explicitamente na definição do adaptador.

Esta seção inclui os seguintes tópicos:

logger.py

O script **logger.py** contém utilitários de log e funções auxiliares para gerar relatórios de erros. Você pode chamar sua depuração, informações e APIs de erros para gravar nos arquivos de log. As mensagens de log são registradas em **C:\hp\UCMDB\DataFlowProbe\runtime\log**.

As mensagens são inseridas no arquivo de log de acordo com o nível de depuração definido para o anexador PATTERNS_DEBUG no arquivo **C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties**. Por padrão, o nível é DEBUG.) Para ver detalhes, consulte "Níveis de gravidade de erro" na página 123.

```
#########################################################################
################       PATTERNS_DEBUG log              ##################
#########################################################################
log4j.category.PATTERNS_DEBUG=DEBUG, PATTERNS_DEBUG
log4j.appender.PATTERNS_DEBUG=org.apache.log4j.RollingFileAppender
log4j.appender.PATTERNS_DEBUG.File=C:\hp\UCMDB\DataFlowProbe\runtime\log/pr
obeMgr-patternsDebug.log
log4j.appender.PATTERNS_DEBUG.Append=true
log4j.appender.PATTERNS_DEBUG.MaxFileSize=15MB
log4j.appender.PATTERNS_DEBUG.Threshold=DEBUG
log4j.appender.PATTERNS_DEBUG.MaxBackupIndex=10
log4j.appender.PATTERNS_DEBUG.layout=org.apache.log4j.PatternLayout
log4j.appender.PATTERNS_DEBUG.layout.ConversionPattern=<%d> [%-5p] [%t] -
%m%n
log4j.appender.PATTERNS_DEBUG.encoding=UTF-8
```

As mensagens de informações e de erros também aparecem no console Prompt de Comando.

Há dois conjuntos de APIs:

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

O primeiro conjunto emite a concatenação de todos os seus argumentos de cadeia de caracteres no nível de log apropriado e o segundo conjunto emite a concatenação, além de emitir o rastreamento de pilha da exceção de lançamento mais recente, para fornecer mais informações, por exemplo:

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

modeling.py

O script **modeling.py** contém APIs para criar hosts, IPs, ECs de processo e assim por diante. Essas APIs permitem a criação de objetos comuns e tornam o código mais legível. Por exemplo:

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

netutils.py

A biblioteca **netutils.py** é usada para recuperar informações de rede e TCP, por exemplo, recuperar nomes de sistemas operacionais, verificar se um endereço MAC é válido, verificar se um endereço IP é válido e assim por diante. Por exemplo:

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

shellutils.py

A biblioteca **shellutils.py** fornece uma API para executar comandos de shell e recuperar o status final de um comando executado, além de permitir a execução de vários comandos com base nesse status final. A biblioteca é inicializada com um Cliente Shell e usa o cliente para executar comandos e recuperar resultados. Por exemplo:

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

4

Mensagens de erro

Este capítulo inclui:

Conceitos

Referência

Conceitos

Visão geral de mensagens de erro

Durante a descoberta, muitos erros podem ser revelados, por exemplo, falhas de conexão, problemas de hardware, exceções, tempo limite esgotado e assim por diante. O DFM exibe esses erros no Painel de Controle de Descoberta, tanto em Modo Básico quanto em Modo Avançado, sempre que o fluxo de descoberta regular não tiver êxito. Você pode exibir os detalhes do EC Acionador que causou o problema para ver a mensagem de erro em si.

O DFM diferencia entre erros que podem ser ignorados (por exemplo, um host fora de alcance) e erros que precisam ser corrigidos (por exemplo, problemas de credenciais ou arquivos DLL ou de configuração ausentes). Além disso, o DFM relata erros uma vez, mesmo que o erro ocorra em execuções sucessivas, e relata um erro inclusive se ele ocorrer somente uma vez.

Ao criar um pacote, você pode adicionar ao pacote mensagens apropriadas como recursos. Durante a implementação do pacote, as mensagens também são implementadas no local correto. As mensagens devem obedecer a convenções, conforme descrito em "Convenções da criação de mensagens" na página 119.

O DFM dá suporte a mensagens de erro multilíngues. Você pode traduzir as mensagens que escreve de modo que sejam exibidas no idioma local.

Para ver detalhes sobre como procurar erros, consulte "Painel Status de Descoberta" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Para ver detalhes sobre os log de comunicação, consulte "Painel Opções de Execução" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Referência

Convenções da criação de mensagens

- Cada erro é identificado por um código de mensagem de erro e diversos argumentos (**int, String[]**). Um erro específico é definido pelo seu código de mensagem e seus vários argumentos. A matriz de parâmetros pode ser nula.
- Cada código de erro é associado a uma **mensagem curta**, que é uma cadeia fixa, e uma **mensagem detalhada**, que é uma cadeia de modelo com zero ou mais argumentos. Assume-se a correspondência entre o número de argumentos no modelo e o número real de parâmetros.

  Exemplo de código de mensagem de erro:

  10234 pode representar um erro com a mensagem curta:

  ```
  Erro de conexão
  ```

  e a mensagem detalhada:

  ```
  Impossível conectar via o protocolo {0} devido ao tempo limite de {1} ms
  ```

  em que

  {0} = primeiro argumento: nome do protocolo

  {1} = segundo argumento: duração do tempo limite em ms

Esta seção também inclui os seguintes tópicos:

Conteúdo de arquivo de propriedade

Um arquivo de propriedade deve conter duas chaves para cada código de mensagem de erro. Por exemplo, para o erro 45:

- **DDM_ERROR_MESSAGE_SHORT_45**. Descrição curta do erro.
- **DDM_ERROR_MESSAGE_LONG_45**. Descrição longa do erro (pode conter parâmetros, por exemplo: {**0**},{**1**}).

Arquivo de propriedade de mensagens de erro

Um arquivo de propriedade contém o mapeamento entre o código da mensagem de erro e duas mensagens (curta e detalhada).

Após implementar um arquivo de propriedade, seus dados são mesclados com os dados existentes, ou seja, novos códigos de mensagem de erro são adicionados e os códigos antigos são substituídos.

Arquivos de propriedade de infraestrutura são parte do pacote **AutoDiscoveryInfra**.

Convenções de nomenclatura de local

- Para o local padrão: **<nome do arquivo>.properties.errors**
- Para um local específico: **<nome do arquivo>_xx.properties.errors**

  em que **xx** é o local (por exemplo, **infraerr_fr.properties.errors** ou **infraerr_en_us.properties.errors**).

Códigos de mensagem de erro

Os códigos de erro a seguir estão incluídos por padrão no HP Universal CMDB. Você pode adicionar a essa lista suas próprias mensagens de erro.

Nome do Erro	Código do Erro	Descrição
Interno	100-199	Geralmente resolvidos de exceções resultantes da execução de scripts Jython
Conexão	200-299	Falha da conexão; agente ausente na máquina-alvo; destino fora de alcance e assim por diante
Relacionados a credenciais	300-399	Permissão negada; tentativa de conexão bloqueada devido à falta de credenciais
Tempo limite	400-499	Tempo limite alcançado durante conexão/comando
Comportamento inesperado ou inválido	500-599	Arquivos de configuração ausentes; interrupções inesperadas e assim por diante
Recuperação de informações	600-699	Informações ausentes nas máquinas-alvo; falha ao solicitar informações ao agente e assim por diante
Relacionados a recursos	700-799	Erros relacionados à falta de memória ou clientes que não foram liberados corretamente
Análise	800-899	Erro ao analisar texto
Codificação	900	Erro na entrada de dados; codificação sem suporte
Relacionados a SQL	901-903, 924	Erros recebidos durante operações SQL

Nome do Erro	Código do Erro	Descrição
Relacionados a HTTP	904-909	Erros gerados durante conexões HTTP; derivados de códigos de erro HTTP.
Específicos a aplicativos	910-923	Erro resultante de problemas específicos de aplicativos, por exemplo, versão LSOF incorreta, gerenciadores de fila ausentes e assim por diante

Erros de conteúdo não classificado

Para dar suporte a conteúdo antigo sem causar regressões, os métodos relevantes do SDK e do aplicativo lidam de modo diferente com os erros com código de mensagem 100 (erro de script não classificado).

Esses erros não são agrupados (ou seja, não são considerados erros do mesmo tipo) de acordo com seu código de mensagem, e sim são agrupados pelo conteúdo da mensagem. Isso significa que se um script retorna um erro via métodos antigos e obsoletos (com texto de mensagem mas sem um código de erro), essas mensagens recebem o mesmo código de erro, porém de acordo com os métodos relevantes do SDK ou do aplicativo, mensagens diferentes são exibidas como erros diferentes.

Mudanças na metodologia

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

Os métodos a seguir foram adicionados à interface:

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

Os métodos antigos a seguir ainda têm suporte com o propósito de compatibilidade com versões anteriores, porém são marcados como preteridos:

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

Níveis de gravidade de erro

Quando um adaptador termina de ser executado em um EC acionador, ele retorna um status. Se nenhum erro ou aviso for retornado, o status será **Sucesso**.

Os níveis de gravidade são listados abaixo, do escopo mais estreito ao mais amplo:

Erros Fatais

Esse nível inclui erros graves, como um problema com a infraestrutura, arquivos DLL ausentes ou exceções:

- Falha ao gerar a tarefa (a sonda não foi encontrada, variáveis não foram localizadas etc.)
- Não é possível executar o script
- O processamento dos resultados não ocorre no servidor e os dados não são gravados no CMDB

Erros

Esse nível identifica problemas que impedem que o DFM recupere dados. Analise esses erros, uma vez que exigem que medidas sejam tomadas (por exemplo, aumentar o tempo limite, mudar um intervalo, mudar um parâmetro, adicionar outra credencial de usuário etc.)

- Em casos em que a intervenção do usuário pode ajudar, um erro é retornado, o qual pode envolver um problema de credenciais ou rede que talvez exija investigação adicional. (Esses não são erros de descoberta, e sim de configuração.)
- Falha interna, geralmente causada por comportamento inesperado da máquina ou aplicativo descoberto, por exemplo, arquivos de configuração ausentes, etc.

Aviso

Quando uma execução foi bem-sucedida, mas produziu erros que não foram graves mas que devem ser relatados, o DFM define a gravidade como **Aviso**. Esses ECs devem ser verificados quanto a dados ausentes, antes do início de uma sessão de depuração mais detalhada. Um **Aviso** pode incluir mensagens sobre a falta de um agente instalado no host remoto, ou sobre dados inválidos que fazem com que um atributo não seja calculado corretamente.

- Agente de conexão ausente (SNMP, WMI)
- A descoberta foi bem-sucedida, porém nem todas as informações disponíveis foram descobertas

5

Desenvolvendo adaptadores de banco de dados genéricos

Este capítulo inclui:

Conceitos

Tarefas

Conceitos

Visão geral dos adaptadores de banco de dados genéricos

A finalidade da plataforma de adaptador de banco de dados genérico é criar adaptadores que podem se integrar com RDBMSs (sistemas de gerenciamento de bancos de dados relacionais) e executar consultas TQL e trabalhos de população com base no banco de dados. Os sistemas RDBMS aceitos pelo adaptador de banco de dados genérico são Oracle, Microsoft SQL Server e MySQL.

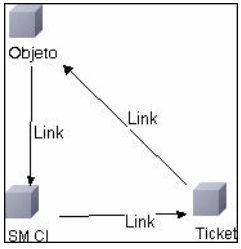
Esta versão da implementação do adaptador de banco de dados se baseia em um padrão JPA (Java Persistence API) tendo a biblioteca ORM do Hibernate como provedor de persistência.

Consultas TQL não aceitas

As seguintes limitações existem nas consultas TQL calculadas somente pelo Adaptador de Banco de Dados Genérico:

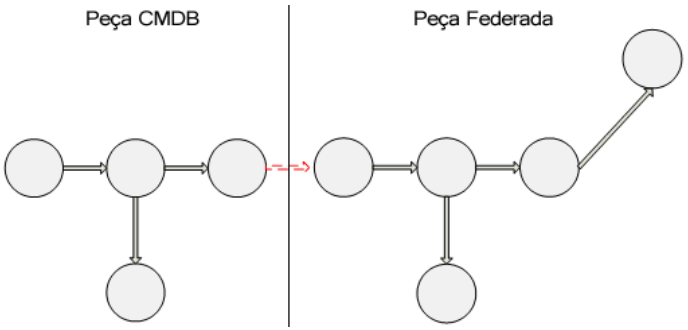
- Subgráfico não são aceitos
- Relacionamentos compostos não são aceitos
- Ciclos ou partes de ciclo não são aceitos

  A consulta TQL a seguir é um exemplo de ciclo:



- O layout de função não é aceito.
- Cardinalidade 0..0 não é aceita.

- O relacionamento Join não é aceito.
- Condições de qualificador não são aceitas.
- Para se conectar entre dois ECs, um relacionamento na forma de uma tabela ou chave estrangeira precisa existir na fonte de dados de dados externa.



Reconciliação

A reconciliação é executada como parte do cálculo TQL no lado do adaptador. Para que a reconciliação ocorra, o lado do CMDB é mapeado para uma entidade federada denominada TEC de reconciliação.

Mapeamento. Cada atributo no CMDB é mapeado para uma coluna na fonte de dados.

Apesar de o mapeamento ser executado diretamente, as funções de transformação nos dados de mapeamento também são aceitas. Você pode adicionar novas funções através de código Java (por exemplo, lowercase, uppercase). A finalidade dessas funções é permitir conversões de valor (valores armazenados no CMDB em um formato e no banco de dados federado em outro formato).

Observação:

- Para conectar o CMDB e a fonte de banco de dados externa, é necessário haver uma associação apropriada no banco de dados. Para ver detalhes, consulte "Pré-requisitos" na página 133.
- A reconciliação com o CMDB id também é aceita.

Hibernate como provedor de JPA

O Hibernate é uma ferramenta de mapeamento relacional a objeto, que permite mapear classes Java em tabelas em vários tipos de bancos de dados relacionais (por exemplo, Oracle e Microsoft SQL Server). Para ver detalhes, consulte "Limitações funcionais" na página 218.

Em um mapeamento elementar, cada classe Java é mapeada para uma única tabela. Um mapeamento mais avançado permite o mapeamento de herança (como pode ocorrer no banco de dados do CMDB).

Outros recursos aceitos incluem mapeamento de uma classe para várias tabelas, suporte a coleções e associações de tipos um-para-um, um-para-muitos e muitos-para-um. Para ver detalhes, consulte "Associações" na página 131.

Para as nossas finalidades, não é necessário criar classes Java. O mapeamento é definido dos TECs de modelo de classe do CMDB às tabelas de banco de dados.

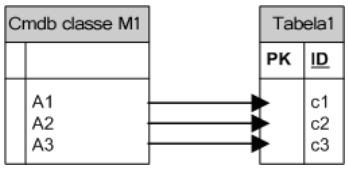
Esta seção também inclui os seguintes tópicos:

Exemplos de mapeamento relacional de objeto

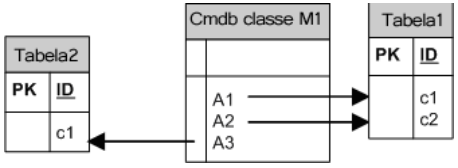
Os exemplos a seguir descrevem o mapeamento relacional de objeto:

Exemplo de 1 classe do CMDB mapeada para 1 tabela de banco de dados:

A classe M1, com atributos A1, A2 e A3, é mapeada para as colunas c1, c2 e c3 da tabela 1. Isso significa que qualquer instância de M1 tem uma linha de correspondência na tabela 1.

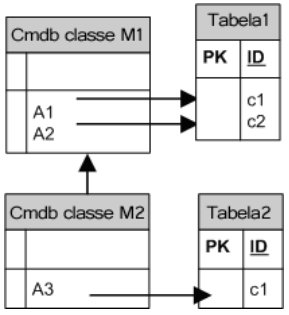


Exemplo de 1 classe do CMDB mapeada para 2 tabelas de banco de dados:



Exemplo de herança:

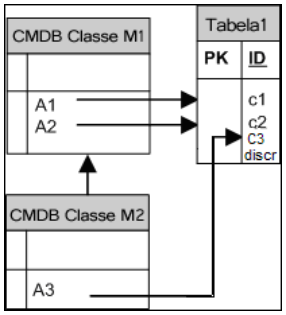
Este caso é usado no CMDB, em que cada classe tem a própria tabela de banco de dados.



Exemplo de herança de uma única tabela com discriminador:

Uma hierarquia inteira de classes é mapeada para uma única tabela de banco de dados, cujas colunas consistem em um superconjunto de todos os atributos das classes mapeadas. A tabela também contém uma coluna adicional (Discriminator), cujo valor indica qual classe específica deve ser mapeada para essa entrada.

Ao usar os recursos de discriminador, você não pode ignorar uma classe na hierarquia; ou seja, como C3 herda de C2, que por sua vez herda de C1, não é possível simplesmente definir C1 e C3, mas sim é necessário definir as três classes.



Associações

Existem três tipos de associações: um-para-muitos, muitos-para-um e muitos-para-muitos. Para se conectar entre os diferentes objetos de banco de dados, uma dessas associações precisa ser definida usando uma coluna de chave estrangeira (para o caso um-para-muitos) ou uma tabela de mapeamento (para o caso de muitos-para-muitos).

Facilidade de uso

Como o esquema JPA é muito amplo, um arquivo XML simplificado é fornecido para facilitar as definições.

O caso de uso deste arquivo XML é assim: os dados federados são modelados em uma classe federada. Essa classe tem bastantes relacionamentos muitos-para-um com uma classe do CMDB não-federada. Além disso, só há um tipo de relacionamento possível entre a classe federada e a classe não-federada.

Tarefas

Preparar criação do adaptador

Esta tarefa descreve os preparativos necessários para criar um adaptador.

Observação: Você pode visualizar exemplos do adaptador de banco de dados genérico na API do UCMDB. Especificamente, o exemplo de Adaptador DDMi contém um arquivo **orm.xml** complicado, além das implementações para algumas interfaces de plugin.

Esta tarefa inclui as seguintes etapas:

- "Pré-requisitos" na página 133
- "Criar um tipo de EC" na página 135
- "Criar um relacionamento" na página 136

1 Pré-requisitos

Para validar se é possível usar o adaptador de banco de dados com o seu banco de dados, verifique estes itens:

➤ Se as classes de reconciliação e seus atributos (também conhecidos como multinodes) existem no banco de dados. Por exemplo, se a reconciliação for executada por nome de nó, verifique se há uma tabela que contém uma coluna com nomes de nó. Se a reconciliação estiver sendo executada de acordo com o nó cmdb_id, verifique se há uma coluna com IDs do CMDB que corresponda aos IDs do CMDB dos nós no CMDB. Para ver detalhes sobre reconciliação, consulte "Reconciliação" na página 128.

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 dois TECs com um relacionamento, deve haver dados de correlação entre as tabelas de TEC. A correlação pode ser mediante uma coluna de chave estrangeira ou mediante uma tabela de mapeamento. Por exemplo, para correlacionar entre nó e ticket, deve haver uma coluna na tabela de tickets que contém a ID de nó, uma coluna na tabela de nó com a ID de ticket que está conectada a ela ou uma tabela de mapeamento cujo end1 é a ID de nó e end2 é a ID de ticket. Para ver detalhes sobre dados de correlação, consulte "Hibernate como provedor de JPA" na página 129.

A tabela a seguir mostra a coluna NODE_ID da chave estrangeira:

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	Controlador de barramento serial	Intel ® 82801EB USB Universal Host Controller
3581	2	Sistema	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3581	3	Vídeo	ATI ES1000
3581	4	Periférico de sistema base	HP ProLiant iLO 2 Legacy Support Function

➤ Cada TEC pode ser mapeado para uma ou mais tabelas. Para mapear um TEC para mais de uma tabela, verifique se há uma tabela primária cuja chave primária exista nas outras tabelas e seja uma coluna de valor exclusivo.

Por exemplo, um ticket é mapeado para duas tabelas: ticket1 e ticket2. A primeira tabela tem as colunas c1 e c2 e a segunda tabela tem as colunas c3 e c4. Para permitir que elas sejam consideradas como uma só tabela, ambas precisam ter a mesma chave primária. Como alternativa, a primeira chave primária da tabela pode ser uma coluna na segunda tabela.

No exemplo a seguir, as tabelas compartilham a mesma chave primária denominada CARD_ID:

CARD_ID	CARD_TYPE	CARD_NAME
1	Controlador de barramento serial	Intel ® 82801EB USB Universal Host Controller
2	Sistema	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3	Vídeo	ATI ES1000
4	Periférico de sistema base	HP ProLiant iLO 2 Legacy Support Function

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(Controlador de host USB padrão)
3	Hewlett-Packard Company
4	(Dispositivos de sistema padrão)
5	Hewlett-Packard Company

2 Criar um tipo de EC

Nesta etapa, você cria um TEC federado que deve ser mapeado para os dados no RDBMS (a fonte de dados externa).

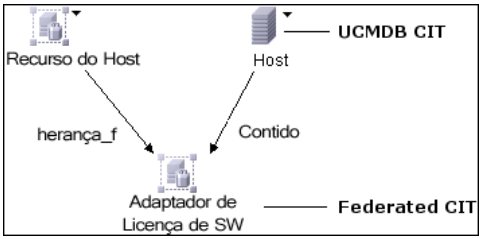
a No UCMDB, acesse o Gerenciador de Tipo de EC e crie um novo Tipo de EC. Para ver detalhes, consulte "Criar um tipo de EC" no *Guia de Modelagem do HP Universal CMDB*.

b Adicione os atributos necessários ao TEC, como o último horário de acesso, o fornecedor e assim por diante. Esses são os atributos que o adaptador recuperará da fonte de dados externa e importará para as visualizações do CMDB.

3 Criar um relacionamento

Nesta etapa, você adiciona um relacionamento entre o TEC do UCMDB e o novo TEC que representa os dados que serão federados a partir da fonte de dados externa.

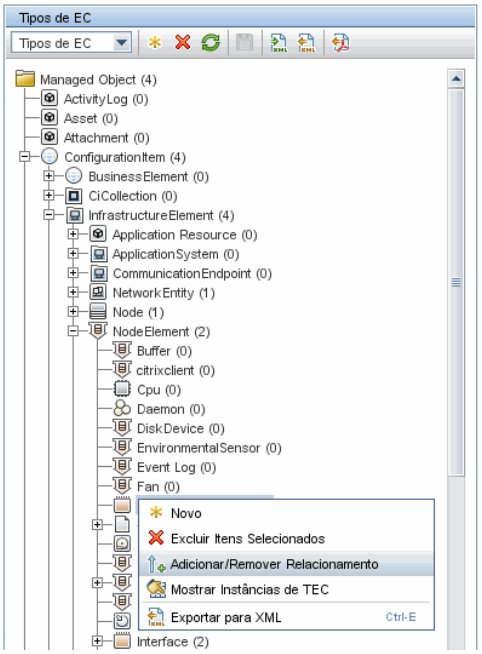
Adicione relacionamentos válidos apropriados ao novo TEC. Para ver detalhes, consulte "Caixa de diálogo Adicionar/Remover Relacionamento" no *Guia de Modelagem do HP Universal CMDB*.



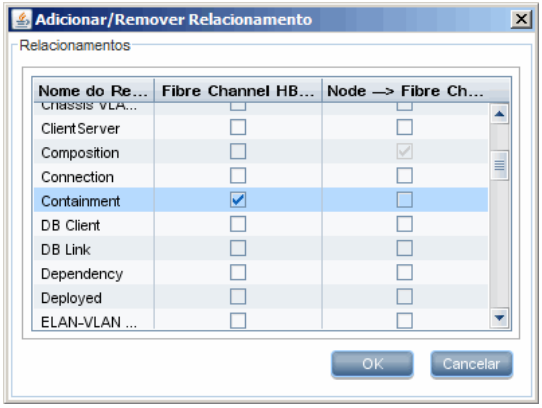
Observação: Nesta etapa, você ainda não pode visualizar os dados federados, já que ainda não definiu o método para importar os dados.

Exemplo de criação de um relacionamento Containment:

1 No Gerenciador de TEC, selecione os dois TECs:



2 Crie um relacionamento **Containment** entre dois TECs:



Preparar o pacote de adaptadores

Nesta etapa, você localiza e configura o pacote de adaptadores de banco de dados genérico.

1 Localize o pacote **db-adapter.zip** na pasta **C:\hp\UCMDB\UCMDBServer\content\adapters**.

2 Extraia o pacote para um diretório temporário local.

3 Edite o arquivo XML do adaptador:

- Abra o arquivo **discoveryPatterns\db_adapter.xml** em um editor de texto.
- Localize o atributo **adapter id** e substitua o nome:

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

Se o adaptador oferecer suporte a dados de replicação, o recurso a seguir deverá ser adicionado ao elemento <**adapter-capabilities**>:

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

O ID ou rótulo de exibição aparece na lista de adaptadores no painel Pontos de Integração do HP Universal CMDB.

Para ver detalhes sobre como popular os dados do CMDB, consulte "Página Integration Studio" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

➤ Se o adaptador está usando o mecanismo de mapeamento da versão 8.x (significando que ele não está usando o novo mecanismo de mapeamento de reconciliação), substitua o elemento a seguir:

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

por

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

Para reverter para o novo mecanismo de mapeamento, retorne o elemento para o seguinte valor:

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

➤ Localize a definição **category**:

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

Altere o nome da categoria **Generic** para a categoria de sua preferência.

Observação: Os adaptadores cujas categorias são especificadas como **Generic** não estão listados no Integration Studio ao criar um novo ponto de integração.

---

4 No diretório temporário, abra a pasta **adapterCode** e renomeie **GenericDBAdapter** para o valor de **adapter id** que foi usado na etapa 3.

Essa pasta contém os arquivos jar que executam a lógica de federação, por exemplo, o nome do adaptador, as consultas e as classes no CMDB, além dos campos no RDBMS que o adaptador aceita.

5 Configure o adaptador conforme necessário. Para ver detalhes, consulte "Configurar o adaptador" na página 141.

6 Crie um arquivo *.zip com o mesmo nome dado ao atributo **adapter id**, conforme descrito na etapa 3 na página 138.

Observação: O arquivo **descriptor.xml** é um arquivo padrão que existe em cada pacote.

---

7 Salve o novo pacote que você criou na etapa anterior. O diretório padrão dos adaptadores é: **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Atualizar o adaptador de banco de dados genérico da versão 9.00 ou 9.01 para 9.02 e posterior

1 Copie o pacote de adaptadores para um diretório temporário local.

2 Extraia os arquivos.

3 Remova os seguintes arquivos da pasta **adapterCode\<Nome_do_seu_adaptador>**:

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

4 Recrie o pacote de adaptadores.

Observação: Para quaisquer adaptadores de banco de dados genérico implantados que você possa ter, o instalador do UCMDB removerá os arquivos necessários do UCMDB e do sistema de arquivos da Sonda. Entretanto, você ainda precisará corrigir o pacote por conta própria, para reimplantá-lo quando for necessário.

Configurar o adaptador

Você pode usar um dos métodos a seguir para configurar o adaptador:

➤ "Configuração do Adaptador – método mínimo" na página 141

➤ "Configuração do Adaptador – método avançado" na página 144

Estes arquivos de configuração estão localizados no pacote **db-adapter.zip** da pasta **C:\hp\UCMDB\UCMDBServer\content\adapters** que você extraiu na etapa 2 de "Preparar o pacote de adaptadores" na página 138.

Configuração do Adaptador – método mínimo

Observação: O arquivo **orm.xml** que é gerado automaticamente como resultado da execução desse método é um ótimo exemplo que pode ser usado ao trabalhar com o método avançado.

O procedimento a seguir descreve um método de mapear o modelo de classe no CMDB para um RDBMS. Você usará este método mínimo quando precisar:

➤ Federar um único nó, como um atributo de nó.

➤ Demonstrar os recursos do adaptador de banco de dados genérico

Este método:

➤ aceita somente federação de um único nó

➤ aceita somente relacionamentos virtuais de muitos-para-um

Esta tarefa inclui as seguintes etapas:

➤ "Configurar o arquivo adapter.conf" na página 142

➤ "Configurar o arquivo simplifiedConfiguration.xml" na página 142

Configurar o arquivo adapter.conf

Nesta etapa, você altera as configurações no arquivo adapter.conf para que os dados sejam federados automaticamente.

1 Abra o arquivo **adapter.conf** em um editor de texto.

2 Localize a seguinte linha: **use.simplified.xml.config=<true/false>**.

3 Altere-a para **use.simplified.xml.config=true**.

Configurar o arquivo simplifiedConfiguration.xml

Nesta etapa, você configura o arquivo **simplifiedConfiguration.xml** mapeando o TEC no CMDB para os campos na tabela RDBMS.

1 Abra o arquivo **simplifiedConfiguration.xml** em um editor de texto.

Esse arquivo inclui um modelo usado para que cada entidade seja mapeada.

Observação: Não edite o arquivo **simplifiedConfiguration.xml** em nenhuma versão do Bloco de Notas da Microsoft Corporation. Use o Notepad++, o UltraEdit ou algum outro editor de texto de terceiros.

---

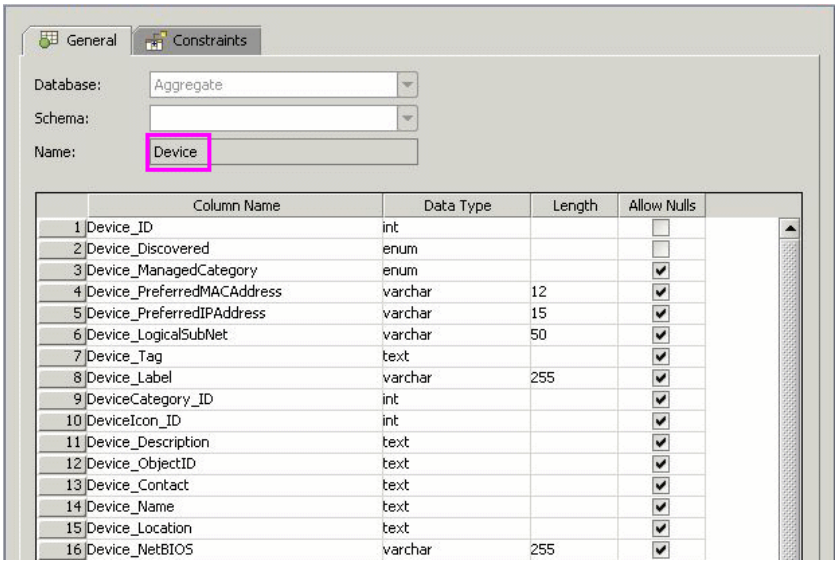
2 Faça alterações nos seguintes atributos:

➤ O nome do TEC no UCMDB (cmdb-class-name) e o nome da tabela correspondente no RDBMS (default-table-name):

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

O atributo cmdb-class-name é obtido no TEC de nó:

O atributo default-table-name é obtido na tabela Device:



	Column Name	Data Type	Length	Allow Nulls
1	Device_ID	int		☐
2	Device_Discovered	enum		☐
3	Device_ManagedCategory	enum		☑
4	Device_PreferredMACAddress	varchar	12	☑
5	Device_PreferredIPAddress	varchar	15	☑
6	Device_LogicalSubNet	varchar	50	☑
7	Device_Tag	text		☑
8	Device_Label	varchar	255	☑
9	DeviceCategory_ID	int		☑
10	DeviceIcon_ID	int		☑
11	Device_Description	text		☑
12	Device_ObjectID	text		☑
13	Device_Contact	text		☑
14	Device_Name	text		☑
15	Device_Location	text		☑
16	Device_NetBIOS	varchar	255	☑

- O identificador exclusivo no RDBMS:

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

- A regra de reconciliação (reconciliation-by-two-nodes):

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

- O atributo de reconciliação no UCMDB (cmdb-attribute-name) e no RDBMS (column-name):

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

- O nome do TEC (cmdb-class-name) e o nome da tabela correspondente no RDBMS (default-table-name). Além disso, o relacionamento do UCMDB (connected-cmdb-class-name) e o relacionamento do TEC (link-class-name):

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

- A chave primária e a chave estrangeira:

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

- O identificador exclusivo no RDBMS:

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

- O mapeamento entre o atributo CMDB (cmdb-attribute-name) e o nome da coluna no RDBMS (column-name):

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

3 Salve o arquivo.

Configuração do Adaptador – método avançado

Esta tarefa inclui as seguintes etapas:

- "Configurar o arquivo orm.xml" na página 145
- "Configurar o arquivo reconciliation_types.txt" na página 149
- "Configurar o arquivo reconciliation_rules.txt" na página 149

Configurar o arquivo orm.xml

Nesta etapa, você mapeia os TECs e relacionamentos no CMDB para as tabelas no RDBMS.

1 Abra o arquivo **orm.xml** em um editor de texto.

Esse arquivo, por padrão, contém um modelo que é usado para mapear quantos TECs e relacionamentos forem necessários para a federação.

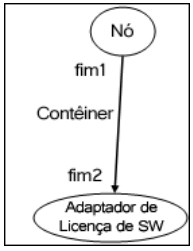
Observação: Não edite o arquivo **orm.xml** em nenhuma versão do Bloco de Notas da Microsoft Corporation. Use o Notepad++, o UltraEdit ou algum outro editor de texto de terceiros.

---

2 Faça alterações no arquivo de acordo com as entidades de dados que serão mapeadas. Para ver detalhes, consulte os exemplos a seguir.

Os seguintes tipos de relacionamentos podem ser mapeados no arquivo **orm.xml**:

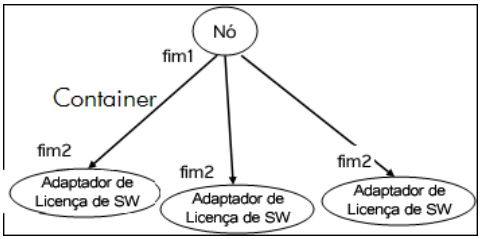
➤ Um-para-um:



O código desse tipo de relacionamento é:

```
<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>
```

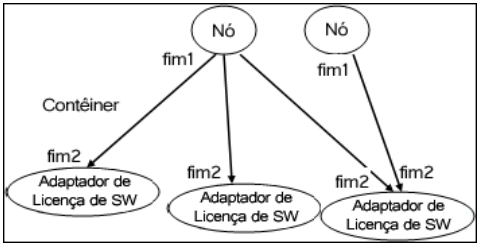
➤ Muitos-para-um:



O código desse tipo de relacionamento é:

```
<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>
```

➤ Muitos-para-muitos:



O código desse tipo de relacionamento é:

```
<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 ver detalhes sobre convenções de nomenclatura, consulte "Convenções de nomenclatura" na página 184.

Exemplo de mapeamento de entidades entre o modelo de dados e o RDBMS:

Observação: Atributos que não precisam ser configurados são omitidos dos exemplos a seguir.

➤ A classe do TEC do CMDB:

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

➤ O nome da tabela no RDBMS:

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

➤ O nome da coluna do identificador exclusivo na tabela RDBMS:

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

➤ O nome do atributo no TEC do CMDB:

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

➤ O nome do campo de tabela na fonte de dados externa:

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

➤ O nome do novo TEC criado no "Criar um tipo de EC" na página 135:

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

➤ O nome da tabela correspondente no RDBMS:

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

➤ A identidade exclusiva no 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>
```

➤ O nome do atributo no TEC do CMDB e o nome do atributo correspondente no RDBMS:

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

Exemplo de mapeamento de relacionamentos entre o modelo de dados e o RDBMS:

➤ A classe do relacionamento do CMDB:

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

➤ O nome da tabela RDBMS em que o relacionamento é executado:

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

➤ O ID exclusivo no 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>
```

➤ O tipo de relacionamento e o TEC do CMDB:

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

➤ Os campos de chave primária e chave estrangeira no RDBMS:

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

Configurar o arquivo reconciliation_types.txt

Abra o arquivo **reconciliation_types.txt** em um editor de texto.

Para ver detalhes, consulte "O arquivo reconciliation_types.txt" na página 192.

Configurar o arquivo reconciliation_rules.txt

Nesta etapa, você define as regras pelas quais o adaptador reconcilia o CMDB e o RDBMS (somente se o Mecanismo de Mapeamento for usado, para permitir compatibilidade com a versão 8.x):

1. Abra o arquivo **META-INF\reconciliation_rules.txt** em um editor de texto.
2. Faça alterações no arquivo de acordo com o TEC que você está mapeando. Por exemplo, para mapear um TEC de nó, use a seguinte expressão:

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

Observação:

- Se os dados no banco de dados diferenciarem maiúsculas de minúsculas, não exclua o caractere de controle (^).
- Verifique se cada colchete de abertura tem um colchete de fechamento correspondente.

Para ver detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 192.

Implementar um plugin

Esta tarefa descreve como implementar e implantar um adaptador de banco de dados genérico com plugins.

Observação: Antes de escrever um plugin para um adaptador, verifique se concluiu todas as etapas necessárias em "Preparar o pacote de adaptadores" na página 138.

---

1 Copie os seguintes arquivos jar do diretório de instalação do servidor UCMDB para o caminho de classe de desenvolvimento:

- Copie o arquivo **db-interfaces.jar** e o arquivo **db-interfaces-javadoc.jar** da pasta **tools\adapter-dev-kit**.
- Copie o arquivo **federation-api.jar** e o arquivo **federation-api-javadoc.jar** da pasta **\tools\adapter-dev-kit\SampleAdapters\production-lib**.

Observação: Para ver mais informações sobre como desenvolver um plugin, consulte os arquivos **db-interfaces-javadoc.jar** e **federation-api-javadoc.jar** e na documentação online em:

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

---

2 Escreva uma classe Java implementando a interface Java do plugin. As interfaces são definidas no arquivo **db-interfaces.jar**. A tabela a seguir especifica a interface que precisa ser implementada para cada plugin:

Tipo de plugin	Nome da interface	Método
Sincronizar topologia completa	FcmdbPluginForSyncGetFull Topology	getFullTopology
Sincronizar alterações	FcmdbPluginForSyncGet ChangesTopology	getChangesTopology
Sincronizar layout	FcmdbPluginForSyncGet Layout	getLayout
Recuperar consultas aceitas	FcmdbPluginForSyncGet SupportedQueries	getSupportedQueries
Alterar definições e resultados de consulta TQL	FcmdbPluginGetTopology CmdbFormat	getTopologyCmdbFormat
Alterar solicitação de layout para ECs	FcmdbPluginGetCIsLayout	getCisLayout
Alterar solicitação de layout para links	FcmdbPluginGetRelations Layout	getRelationsLayout

A classe do plugin precisa ter um construtor padrão público. Além disso, todas as interfaces expõem um método denominado initPlugin. Esse método tem a garantia de ser chamado antes de qualquer outro método e é usado para inicializar o adaptador com o objeto de ambiente do adaptador recipiente.

3 Verifique se tem o JAR do Federation SDK e os JARs do adaptador de banco de dados genérico em seu caminho de classe antes de compilar seu código Java. O Federation SDK é o arquivo **federation_api.jar**, que pode ser encontrado no diretório **C:\hp\UCMDB\UCMDBServer\lib**.

4 Compacte sua classe em um arquivo jar e na pasta adapterCode\<Seu_Nome_de_Adaptador> do pacote de adaptadores, antes de implantá-lo.

Os plugins são configurados usando o arquivo **plugins.txt**, localizado na pasta **\META-INF** do adaptador.

Veja a seguir um exemplo do arquivo do 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]
```

Legenda:

- Uma linha de comentário.

[<Tipo de Adaptador>] – Inicie a seção de definição de um tipo de adaptador específico.

Poderá haver uma linha vazia sob cada [<Tipo de Adaptador>], significando que não há uma classe de plugin associada ou poderá estar listado o nome totalmente qualificado de sua classe de plugin.

5. Compacte o adaptador com o novo arquivo jar e o arquivo **plugins.xml** atualizado. O restante dos arquivos no pacote deverá ser o mesmo de qualquer adaptador baseado no adaptador de banco de dados genérico.

Implantar o adaptador

1 No UCMDB, acesse o Gerenciador de Pacotes. Para ver detalhes, consulte "Página Gerenciador de Pacotes" no *Guia de Administração do HP Universal CMDB*.

2 Clique no ícone **Implantar pacotes no servidor (a partir do disco local)** e navegue até o pacote de adaptadores. Selecione o pacote e clique em **Abrir**, clique em **Implantar** para exibir o pacote no Gerenciador de Pacotes.

3 Selecione o pacote na lista e clique no ícone **Exibir recursos do pacote** para verificar se o conteúdo do pacote é reconhecido pelo Gerenciador de Pacotes.

Editar o adaptador

Depois de criar e implantar o adaptador, você poderá editá-lo no UCMDB. Para ver detalhes, consulte "Gerenciamento do Adaptador" na página 117.

Criar um ponto de integração

Nesta etapa, você verifica se a federação está funcionando, ou seja, se a conexão e o arquivo XML são válidos. Entretanto, essa verificação não confere se o XML está mapeando para os campos corretos no RDBMS.

1 No UCMDB, acesse o Integration Studio (**Gerenciamento de Fluxo de Dados** > **Integration Studio**).

2 Crie um ponto de integração. Para ver detalhes, consulte "Caixa de diálogo Criar Novo Ponto de Integração/Editar Ponto de Integração" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

A guia Federação exibe todos os TECs que podem ser federados usando esse ponto de integração. Para ver detalhes, consulte "Guia Federação" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

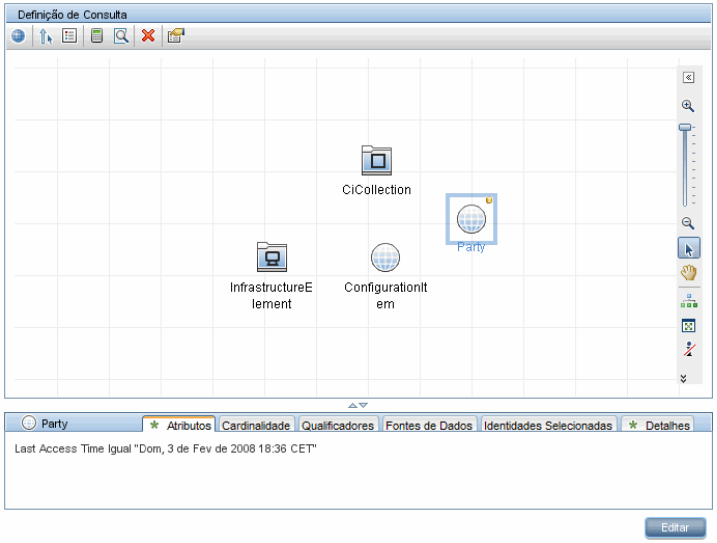
Criar uma visualização

Nesta etapa, você cria uma visualização que permite ver instâncias do TEC.

1 No UCMDB, acesse o Modeling Studio (**Modelagem** > **Modeling Studio**).

2 Crie uma visualização. Para ver detalhes, consulte "Criar uma visualização baseada em gabarito" no *Guia de Modelagem do HP Universal CMDB*.

3 Você pode adicionar condições ao TQL, por exemplo, o último horário de acesso é superior a seis meses:



Calcular os resultados

Nesta etapa, você verifica os resultados.

1 No UCMDB, acesse o Modeling Studio (**Modelagem** > **Modeling Studio**).

2 Abra uma visualização.

3 Calcule os resultados clicando no botão **Calcular Contagem de Resultados de Consulta**.

4 Clique no botão **Visualização** para exibir os ECs na visualização.

Visualizar os resultados

Nesta etapa, você exibe os resultados e depura os problemas no procedimento. Por exemplo, se nada for mostrado na visualização, verifique as definições no arquivo **orm.xml**; remova os atributos de relacionamento e recarregue o adaptador.

1. No UCMDB, acesse o Gerenciador de Universo de IT (**Modelagem** > **Gerenciador de Universo de TI**).
2. Selecione um EC.

   A guia Propriedades exibe os resultados da federação.

Visualizar relatórios

Nesta etapa, você exibe os relatórios de Topologia. Para ver detalhes, consulte "Visão geral dos Relatórios de Topologia" no *Guia de Modelagem do HP Universal CMDB*.

Habilitar arquivos de log

Para entender os fluxos de cálculo e o ciclo de vida do adaptador, bem como visualizar informações de depuração, você pode consultar os arquivos de log. Para ver detalhes, consulte "Arquivos de log do adaptador" na página 215.

Usar o Eclipse para mapear entre atributos de TEC e tabelas de banco de dados

---

Cuidado: Este procedimento é voltado aos usuários com conhecimento avançado em desenvolvimento de conteúdo. Em caso de dúvidas, entre em contato com Suporte da HP Software.

Esta tarefa descreve como instalar e usar o plugin JPA, fornecido com a edição J2EE do Eclipse, para:

- Habilitar o mapeamento gráfico entre os atributos de classe do CMDB e as colunas de tabela de banco de dados.
- Habilitar a edição manual do arquivo de mapeamento (orm.xml), ao mesmo tempo garantindo exatidão. A verificação de exatidão inclui uma verificação de sintaxe, além da verificação de se os atributos de classe e as colunas de tabela de banco de dados mapeadas estão declaradas corretamente.
- Habilitar a implantação do arquivo de mapeamento no servidor CMDB e para visualizar os erros, como uma verificação de exatidão adicional.
- Definir uma consulta de exemplo no servidor CMDB e executá-la diretamente do Eclipse, para testar o arquivo de mapeamento.

Esta tarefa inclui as seguintes etapas:

1 Pré-requisitos

Instalar o **Java Runtime Environment (JRE) 6 Update 7** na máquina em que você executará o Eclipse do seguinte site: **http://java.sun.com/javase/downloads/index.jsp**.

O procedimento funciona com o ambiente de tempo de execução Java 5 (ou posterior).

2 Instalação

a Baixe e extraia **Eclipse IDE for Java EE Developers** em **<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/ganymede/SR1/eclipse-jee-ganymede-SR1-win32.zip>** para uma pasta local, por exemplo, **C:\Arquivos de Programas\eclipse**.

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

c Inicie **C:\Arquivos de Programas\Eclipse\eclipse.exe** (exige pelo menos um ambiente de tempo de execução Java 5). Se uma mensagem for exibida informando que a máquina virtual Java não foi encontrada, inicie o **eclipse.exe** com a seguinte linha de comando:

```
"C:\Arquivos de Programas\eclipse\eclipse.exe" -vm "<pasta de instalação JRE>\bin"
```

3 Preparar o ambiente de trabalho

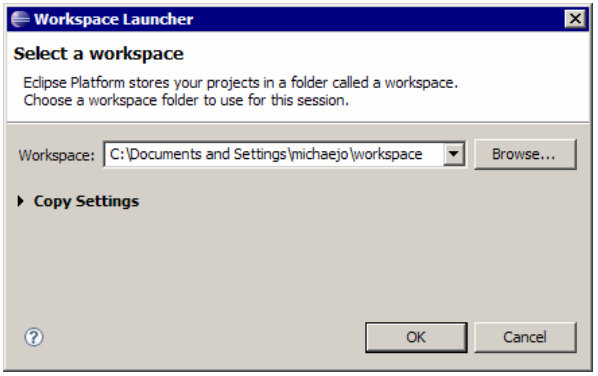
Nesta etapa, você vai configurar o espaço de trabalho, o banco de dados, as conexões e as propriedades de driver.

a Extraia o arquivo **workspaces_gdb.rar** de **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** para **C:\Documents and Settings\All Users\workspaces**.

Observação: Você precisa usar o caminho de pastas exato. Se descompactar o arquivo para o caminho errado ou deixar o arquivo descompactado, o procedimento não funcionará.

---

b No Eclipse, escolha **File (Arquivo)** > **Switch Workspace (Trocar espaço de trabalho)** > **Other (Outro)**:

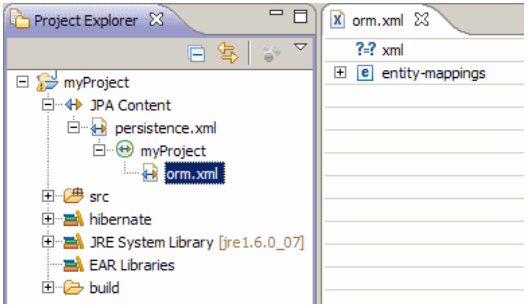


Se você estiver trabalhando com:

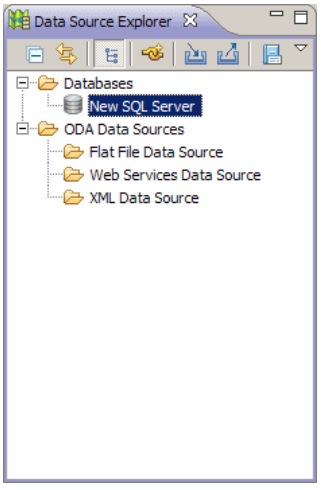
- SQL Server, selecione a seguinte pasta: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.
- MySQL, selecione a seguinte pasta: **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
- Oracle, selecione a seguinte pasta: **C:\Documents and Settings\All Users\workspace_gdb_oracle**.

c Clique em **OK**.

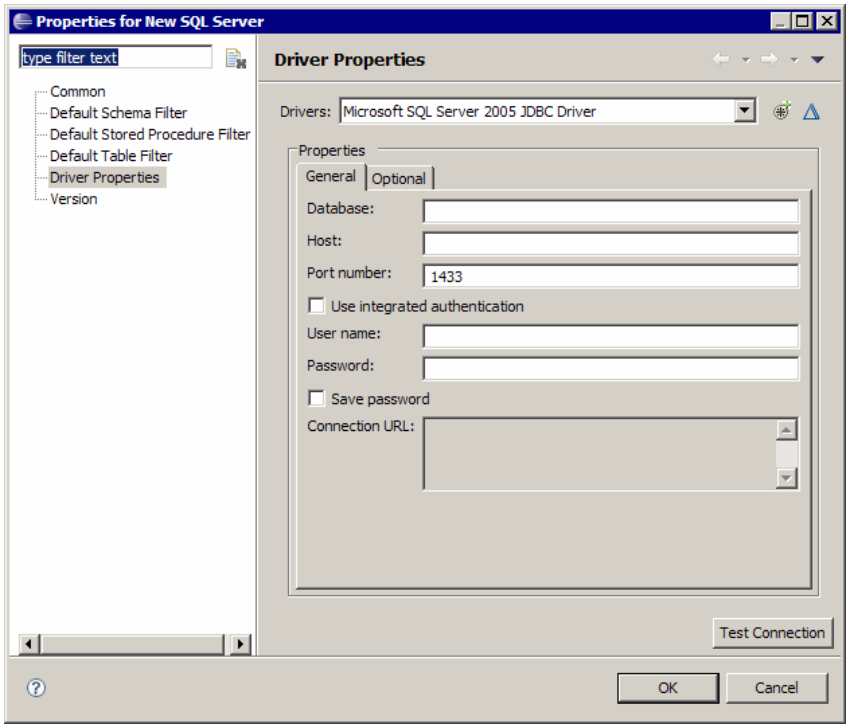
d No Eclipse, exiba a visualização Project Explorer e selecione **<Projeto ativo> > JPA Content (Conteúdo JPA) > persistence.xml > <nome do projeto ativo> > orm.xml**.



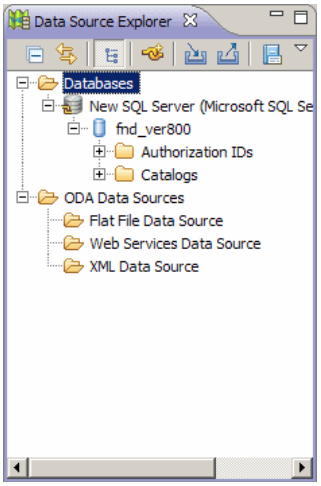
e Na visualização Data Source Explorer (painel inferior esquerdo), clique com o botão direito do mouse na conexão de banco de dados e selecione o menu **Properties (Propriedades)**.



f Na caixa de diálogo **Properties for (Propriedades para) <Nome da conexão>**, selecione **Common (Comum)** e marque a caixa de seleção **Connect every time the workbench is started (Conectar sempre que o workbench for iniciado)**. Selecione **Driver Properties (Propriedades do driver)** e preencha as propriedades de conexão. Clique em **Test Connection (Testar conexão)** e verifique se a conexão está funcionando. Clique em **OK**.



g Na visualização Data Source Explorer, clique com o botão direito do mouse na conexão de banco de dados e clique em **Connect (Conectar)**. Um árvore que contém os esquemas e tabelas de banco de dados é exibido no ícone de conexão de banco de dados.

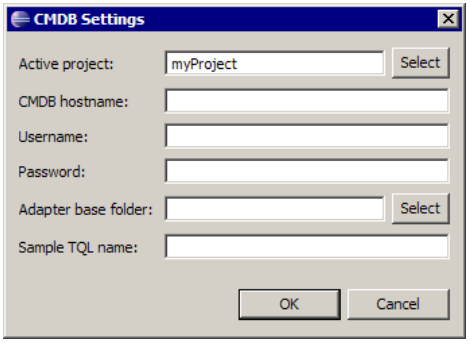


4 Criar um adaptador

Crie um adaptador usando as diretrizes em "Etapa 1: Criar um adaptador" na página 41.

5 Configurar o plugin do CMDB

a No Eclipse, clique em **UCMDB** > **Settings (Configurações)** para abrir a caixa de diálogo **CMDB Settings (Configurações do CMDB)**:



b Se ainda não estiver selecionado, selecione o projeto JPA recém-criado como projeto ativo.

c Insira o nome de host do CMDB, por exemplo, **localhost** ou **labm1.itdep1**. Não é necessário incluir o número da porta ou o prefixo http:// no endereço.

d Preencha o nome de usuário e senha para acessar a API do CMDB, normalmente **admin**/**admin**.

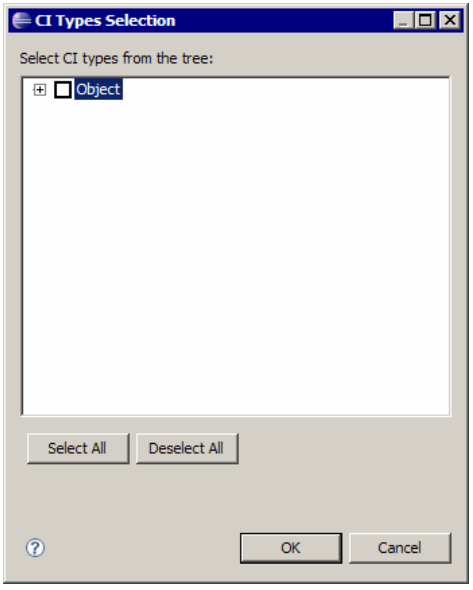
e Verifique se a pasta **C:\hp** no servidor CMDB está mapeada como uma unidade de rede.

f Selecione a pasta base do adaptador relevante em **C:\hp**. A pasta base é aquela que contém o arquivo **dbAdapter.jar** e a subpasta **META-INF**. Seu caminho deve ser **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<nome_do_adaptador>**. Verifique se não há barra invertida (\) no final.

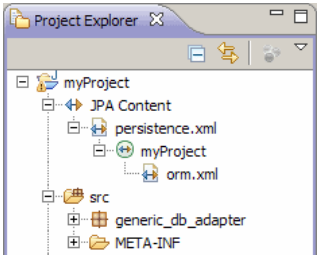
6 Importar o modelo de classe UCMDB

Nesta etapa, você seleciona os TECs que serão mapeados como entradas JPA.

a Clique em **UCMDB > Import CMDB Class Model (Importar Modelo de Classe do CMDB)** para abrir a caixa de diálogo **CI Type Selection (Seleção de Tipo de EC)**:



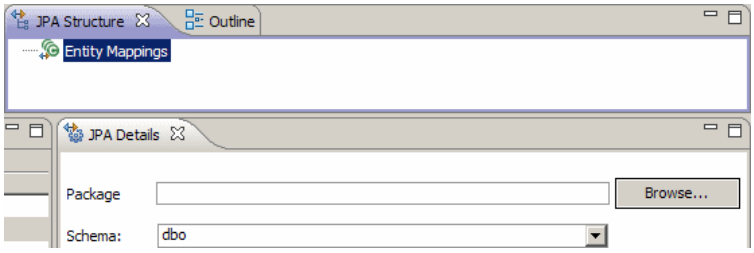
b Selecione os tipos de EC que você pretende mapear como entidades JPA. Clique em **OK**. Os tipos de EC são importados como classes Java. Verifique se eles aparecem sob a pasta **src** do projeto ativo:



7 Criar o arquivo ORM – mapear classes do UCMDB para tabelas de banco de dados

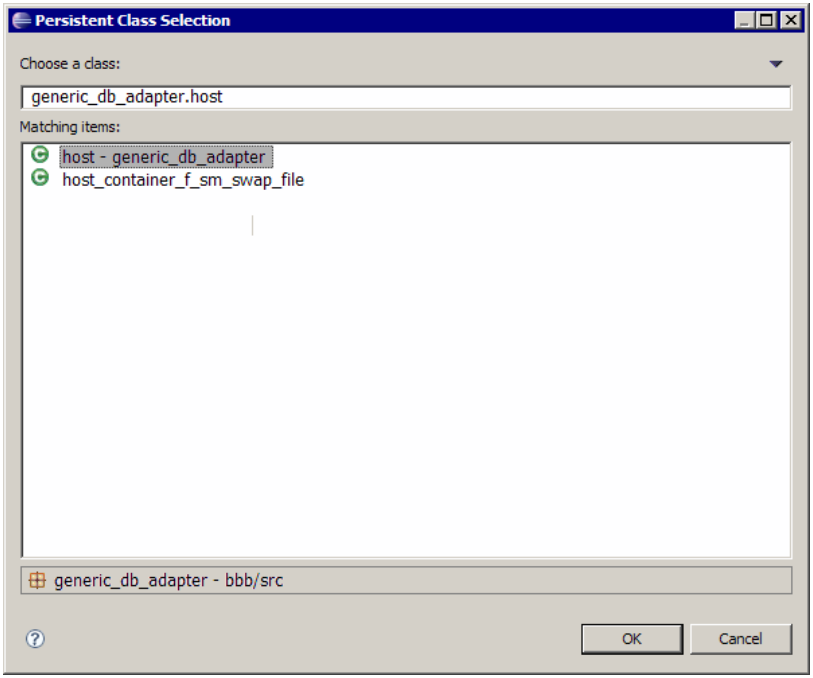
Nesta etapa, você mapeia as classes Java (importadas na etapa anterior) para as tabelas de banco de dados.

a Verifica se há conexão do banco de dados. Clique com o botão direito do mouse no projeto ativo (denominado myProject por padrão) no Project Explorer. Selecione a visualização JPA, marque a caixa de seleção **Override default schema from connection (Substituir esquema padrão da conexão)** e selecionar o esquema de banco de dados relevante. Clique em **OK**.



b Mapeie um TEC: Na visualização Estrutura de JPA, clique com o botão direito do mouse na ramificação **Entity Mappings (Mapeamentos de Entidade)** e selecione **Add Class (Adicionar Classe)**. A caixa de diálogo **Add Persistent Class (Adicionar Classe Persistente)** será aberta. Não altere o campo **Map as (Mapear como)** (**Entity**).

c Clique em **Procurar** e selecione a classe do UCMDB que será mapeada (todas as classes do UCMDB pertencem ao pacote **generic_db_adapter**).

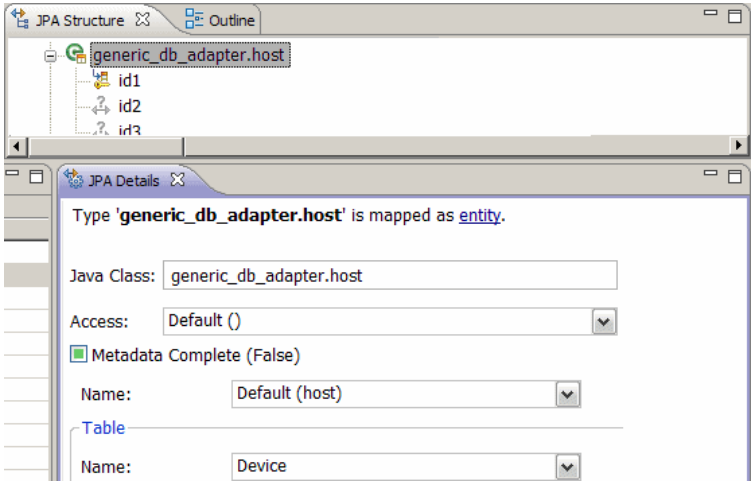


d Clique em **OK** em ambas as caixas de diálogo. A classe selecionada é exibida na ramificação **Entity Mappings (Mapeamentos de Entidade)** na visualização JPA Structure (Estrutura de JPA).

Observação: Se a entidade aparecer sem uma árvore de atributo, clique com o botão direito do mouse no projeto ativo na visualização Project Explorer. Escolha **Fechar** e **Abrir**.

---

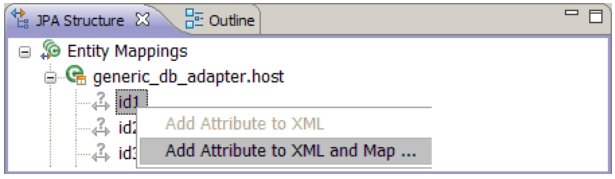
e Na visualização Detalhes de JPA, selecione a tabela de banco de dados primária para a qual a classe do UCMDB deve ser mapeada. Deixe todos os outros campos inalterados.



8 IDs de Mapa

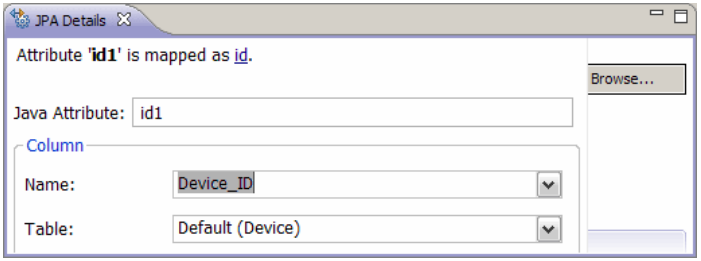
De acordo com os padrões JPA, cada classe persistente precisa ter pelo menos um atributo de ID. Para as classes do UCMDB, você pode mapear até três atributos como IDs. Os possíveis atributos de ID são denominados **id1**, **id2** e **id3**. Para mapear um atributo de ID:

a Expanda a classe correspondente na ramificação **Mapeamentos de Entidade** na visualização Estrutura de JPA, clique com o botão direito do mouse no atributo relevante (por exemplo, **id1**) e selecione **Adicionar atributo como XML e mapear...**:



b A caixa de diálogo **Add Persistent Attribute (Adicionar Atributo Persistente)** será aberta. Selecione **Id** no campo **Map as (Mapear como)** e clique em **OK**.

c Na visualização Detalhes de JPA, selecione a coluna de tabela de banco de dados para a qual o campo de ID deve ser mapeado.



9 Atributos de mapa

Nesta etapa, você mapeia os atributos para as colunas de banco de dados.

a Expanda a classe correspondente na ramificação **Mapeamentos de Entidade** na visualização Estrutura de JPA, clique com o botão direito do mouse no atributo relevante (por exemplo, **host_nomedohost**) e selecione **Adicionar atributo como XML e mapear...**.

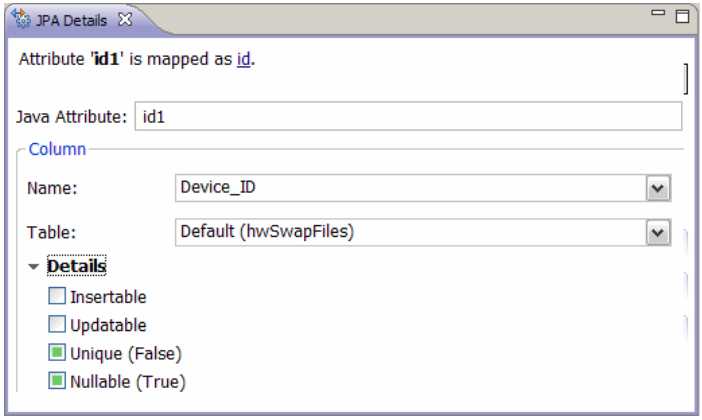
b A caixa de diálogo **Add Persistent Attribute (Adicionar Atributo Persistente)** será aberta. Selecione **Basic (Básico)** no campo **Map as (Mapear como)** e clique em **OK**.

c Na visualização JPA Details (Detalhes de JPA), selecione a coluna de banco de dados para a qual o campo de atributo deve ser mapeado.

10 Mapear um link válido

Execute estas fases descritas na etapa b na página 164 para mapear uma classe do UCMDB que indica um link válido. O nome de cada uma dessas classes tem a seguinte estrutura: **<end1 entity name>_<link name>_<end 2 entity name>**. Por exemplo, um link **Contains** entre um host e um local é indicado por uma classe Java cujo nome é **generic_db_adapter.host_contains_location**. Para ver detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 192.

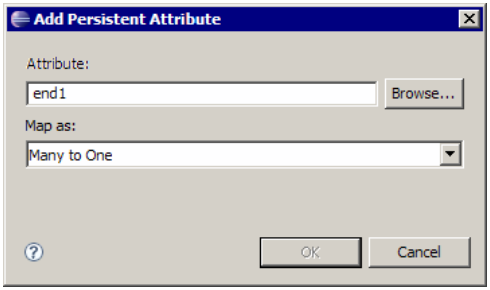
a Mapeie os atributos de ID da classe de link conforme descrito em "IDs de Mapa" na página 166. Para cada atributo de ID, expanda o grupo de caixas de seleção **Details (Detalhes)** na visualização Detalhes de JPA e desmarque as caixas de seleção **Insertable (Inserível)** e **Updateabl (Atualizável)**.



b Mapeie os atributos **end1** and **end2** da classe de link da seguinte maneira: Para cada um dos atributos **end1** and **end2** da classe de link:

➤ Expanda a classe correspondente na ramificação **Entity Mappings (Mapeamentos de Entidade)** na visualização JPA Structure, clique com o botão direito do mouse no atributo relevante (por exemplo, **end1**) e selecione **Add Attribute to XML and Map… (Adicionar atributo como XML e mapear…)**.

➤ Na caixa de diálogo **Add Persistent Attribute (Adicionar Atributo Persistente)**, selecione **Many to One (Muitos-para-Um)** ou **One to One (Um-para-Um)** no campo **Map as (Mapear como)**.

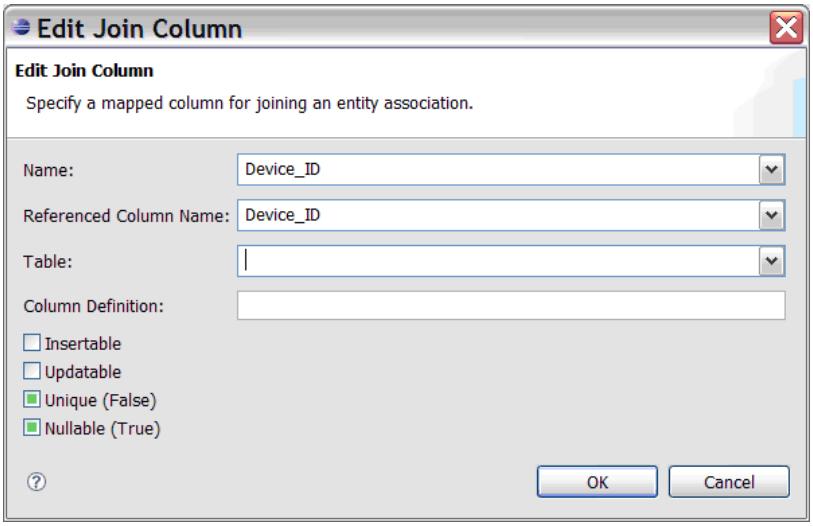


➤ Selecione **Many to One** se o EC **end1** ou **end2** especificado puder ter vários links desse tipo. Caso contrário, selecione **One to One**. Por exemplo, para um link **host_contains_ip** a extremidade de **host** deve ser mapeada como **Many to One (Muitos-para-Um)**, porque um host pode ter vários IPs, e a extremidade **ip** deveria ser mapeada como **One to One (Um-para-Um)**, porque um IP pode ter somente um host.

➤ Na visualização JPA Details (Detalhes de JPA), selecione **Target entity (Entidade de destino)**, por exemplo, **generic_db_adapter.host**.

➤ Na seção **Join Columns (Colunas de Junção)** da visualização Detalhes de JPA, marque **Override Default (Substituir Padrão)**. Clique em **Edit (Editar)**. Na caixa de diálogo **Edit Join Column (Editar Coluna de Junção)**, selecione a coluna de chave estrangeira da tabela de banco de dados do link que aponta para uma entrada na tabela da entidade de destino **end1**/**end2**. Se o nome da coluna referenciado na tabela

da entidade de destino **end1/end2** for mapeado para seu atributo de ID, deixe o **Referenced Column Name (Nome de Coluna Referenciado)** inalterado. Caso contrário, selecione o nome da coluna para a qual a coluna de chave estrangeira aponta. Desmarque as caixas de seleção **Insertable (Inserível)** e **Updatable (Atualizável)** e clique em **OK**.



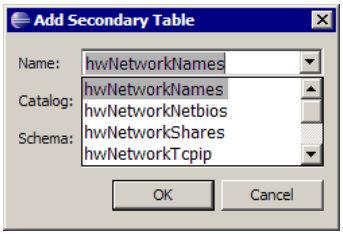
➤ Se a entidade de destino **end1/end2** tiver mais de uma ID, clique no botão **Add (Adicionar)** para adicionar colunas de junção e mapeá-las da mesma maneira descrita na etapa anterior.

11 Criar o arquivo ORM – usar tabelas secundárias

O JPA permite que uma classe Java seja mapeada para mais de uma tabela de banco de dados. Por exemplo, **Host** pode ser mapeado para a tabela **Device** para habilitar a persistência da maioria de seus atributos e para a tabela **NetworkNames** para habilitar a persistência de **host_hostName**. Nesse caso, **Device** é a tabela primária e **NetworkNames** é a tabela secundária. Qualquer número de tabelas secundárias pode ser definido. A única condição é que haja um relacionamento de um-para-um entre as entradas das tabelas primárias e secundárias.

12 Definir uma tabela secundária

Selecione a classe apropriada na visualização Estrutura de JPA. Na visualização **JPA Details (Detalhes de JPA)**, acesse a seção **Secondary Tables (Tabelas Secundárias)** e clique em **Add (Adicionar)**. Na caixa de diálogo **Add Secondary Table (Adicionar Tabela Secundária)**, selecione a tabela secundária apropriada. Deixe os outros campos inalterados.



Se a tabela primária e a secundária não tiverem as mesmas chaves primárias, configure as colunas de junção na seção **Primary Key Join Columns (Colunas de Junção de Chave Primária)** da visualização **JPA Details (Detalhes de JPA)**.

13 Mapear um atributo para uma tabela secundária

Você mapeia um atributo de classe para um campo de uma tabela secundária da seguinte maneira:

a Mapeie o atributo conforme descrito em "Atributos de mapa" na página 167.

b Na seção **Column** da visualização JPA Details, selecione o nome da tabela secundária no campo **Table** para substituir o valor padrão.

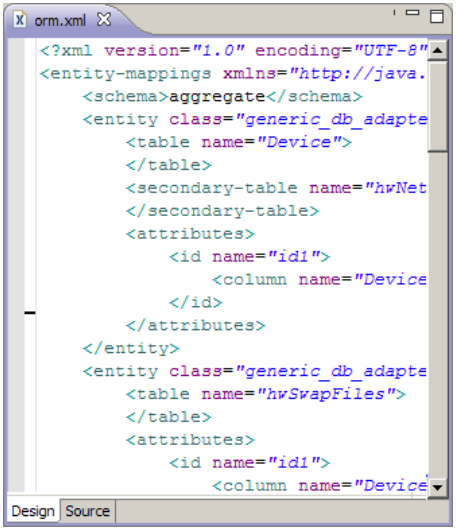
14 Usar um arquivo ORM existente como base

Para usar um arquivo orm.xml existente como base para aquele que você está desenvolvendo, execute as seguintes etapas:

a Verifique se todos os TECs mapeados no arquivo **orm.xml** existente são importados para o projeto do Eclipse ativo.

b Selecione e copie total ou parcialmente os mapeamentos de entidade do arquivo existente.

c Selecione a guia **Source (Fonte)** do arquivo **orm.xml** na perspectiva JPA do Eclipse.



d Cole todos os mapeamentos de entidades copiados abaixo da marca **<entity-mappings>** do arquivo **orm.xml** editado, abaixo da marca **<schema>**. Verifique se a marca do esquema está configurada conforme descrito na etapa b na página 164. Todas as entidades coladas agora aparecem na visualização Estrutura de JPA. De agora em diante, os mapeamentos poderão ser editados tanto gráfica quanto manualmente através do código XML do arquivo **orm.xml**.

e Clique em **Salvar**.

15 Importando um arquivo ORM existente de um adaptador

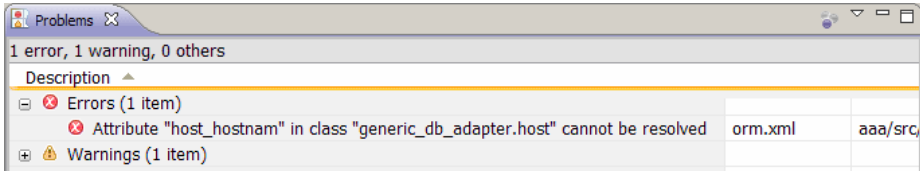
Se já existir um adaptador, o Plugin do Eclipse poderá ser usado para editar seu arquivo ORM graficamente. Importe o arquivo ORM para o Eclipse, edite-o usando o plugin e implante-o de volta na máquina do UCMDB. Para importar o arquivo ORM, pressione o botão na barra de ferramentas do Eclipse. Uma caixa de diálogo de confirmação será exibida. Clique em **OK**. O arquivo ORM é copiado da máquina do UCMDB para o projeto do Eclipse ativo e todas as classes relevantes são importadas do modelo de classe do UCMDB.

Se as classes relevantes não aparecerem na visualização Estrutura de JPA, clique com o botão direito do mouse no projeto ativo na visualização Project Explorer, escolha **Close (Fechar)** e **Open (Abrir)**.

De agora em diante, o arquivo ORM pode ser editado graficamente usando o Eclipse e implantado de volta na máquina do UCMDB conforme descrito em "Implantar o arquivo ORM no CMDB" na página 174.

16 Verificar a exatidão do arquivo ORM – verificação de exatidão interna

O plugin de JPA do Eclipse verifica se há erros e marca-os no arquivo **orm.xml**. Tanto erros de sintaxe (por exemplo, nome de marca errado, marca não-fechada, ID ausente) quanto de mapeamento (por exemplo, nome de atributo ou nome de campo de tabela de banco de dados errado) são verificados. Se houver erros, sua descrição aparecerá na visualização **Problemas**.



17 Criar um novo ponto de integração

Se não houver nenhum ponto de integração no CMDB para esse adaptador, você poderá criá-lo no Integration Studio. Para ver detalhes, consulte "Integration Studio" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Preenche o nome do ponto de integração na caixa de diálogo que for aberta. O arquivo **orm.xml** será copiado para a pasta do adaptador. Um ponto de integração será criado com todos os tipos de EC importados como sendo suas classes aceitas, com exceção de TECs multinode, se eles estiverem configurados no arquivo reconciliation_rules.txt. Para ver detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 192.

18 Implantar o arquivo ORM no CMDB

Salve o arquivo **orm.xml** e implante-o no servidor UCMDB. clicando em **UCMDB** > **Deploy ORM (Implantar ORM)**. O arquivo **orm.xml** será copiado para a pasta do adaptador, e o adaptador será recarregado. O resultado da operação é mostrado em uma caixa de diálogo **Operation Result (Resultado da Operação)**. Se algum erro ocorrer durante o processo de recarregamento, o rastreamento de pilha da exceção Java será exibido na caixa de diálogo. Se nenhum ponto de integração ainda não tiver sido definido usando o adaptador, nenhum erro de mapeamento será detectado após a implantação.

19 Executar uma consulta TQL de exemplo

a Defina uma consulta usando o Gerenciador de Consultas e não o Gerenciador de Visualizações.

b Crie um ponto de integração usando o adaptador **GenericDBAdapter**. Para ver detalhes, consulte "Caixa de diálogo Criar Novo Ponto de Integração/Editar Ponto de Integração" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

c Durante a criação do adaptador, verifique se os tipos de EC que devem participar da consulta são aceitos por esse ponto de integração.

d Ao configurar o plugin do CMDB, use este nome de consulta de exemplo na caixa de diálogo Settings (Configurações). Para ver detalhes, consulte "Configurar o plugin do CMDB" na página 161.

e Clique no botão **Run TWL (Executar TWL)** para executar um TQL de exemplo e verificar se ele retorna os resultados necessários usando o arquivo **orm.xml** recém-criado.

Referência

Arquivos de configuração do adaptador

Os arquivos tratados nesta seção estão localizados no pacote **db-adapter.zip** da pasta **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Esta seção inclui os seguintes tópicos:

Configuração geral

- **adapter.conf.** O arquivo de configuração do adaptador. Para ver detalhes, consulte "O arquivo adapter.conf" na página 176.

Configuração avançada

- **orm.xml**. O arquivo de mapeamento relacional a objeto em que você mapeia entre os TECs e as tabelas de banco de dados do CMDB. Para ver detalhes, consulte "O arquivo orm.xml" na página 180.
- **reconciliation_types.txt**. Contém as regras usadas para configurar os tipos de reconciliação. Para ver detalhes, consulte "O arquivo reconciliation_types.txt" na página 192.
- **reconciliation_rules.txt**. Contém as regras de reconciliação. Para ver detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 192.
- **transformations.txt.** Arquivo de transformações em que você especifica os conversores que serão aplicados para converter do valor do CMDB para o valor do banco de dados e vice-versa. Para ver detalhes, consulte "O arquivo transformations.txt" na página 195.

- **Discriminator.properties.** Este arquivo mapeia cada tipo de EC aceito para uma lista separada por vírgulas de possíveis valores correspondentes. Para ver detalhes, consulte "O arquivo discriminator.properties" na página 197.
- **Replication_config.txt.** Este arquivo contém uma lista separada por vírgulas de tipos de EC e de relacionamento cujas condições de propriedade são aceitas pelo plugin de replicação. Para ver detalhes, consulte "O arquivo replication_config.txt" na página 199.
- **Fixed_values.txt.** Este arquivo permite configurar os valores fixos para atributos específicos de determinados TECs. Para ver detalhes, consulte "O arquivo fixed_values.txt" na página 199.

Configuração do Hibernate

- **persistence.xml.** Usado para substituir configurações prontas do Hibernate. Para ver detalhes, consulte "O arquivo persistence.xml" na página 196.

Configuração simples

- **simplifiedConfiguration.xml**. Arquivo de configuração que substitui os arquivos **orm.xml**, **transformations.txt** e **reconciliation_rules.txt** com menos recursos. Para ver detalhes, consulte "O arquivo simplifiedConfiguration.xml" na página 177.

 O arquivo adapter.conf

Este arquivo contém as seguintes configurações:

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

Observação: O uso deste arquivo significa que os arquivos orm.xml, transformations.txt e reconciliation_rules.txt são substituídos por menos recursos.

- **dal.ids.chunk.size=300.** Não altere esse valor.

➤ **dal.use.persistence.xml=false**. **true**: o adaptador lê a configuração do Hibernate de persistence.xml.

Observação: Não é recomendável substituir a configuração do Hibernate.

O arquivo simplifiedConfiguration.xml

Este arquivo é usado para mapeamento simples de classes do UCMDB para tabelas de banco de dados. Para acessar o modelo para editar o arquivo, navegue para o **Gerenciamento do Adaptador** > **db-adapter** > **Arquivos de configuração**.

Esta seção inclui os seguintes tópicos:

➤ "O modelo de arquivo simplifiedConfiguration.xml" na página 177

➤ "Limitações" na página 179

O modelo de arquivo simplifiedConfiguration.xml

A propriedade **CMDB-class-name** é o tipo multinode (o nó ao qual os TECs federados se conectam no 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. A reconciliação pode ser executada usando um ou dois nós. Neste exemplo de caso, a reconciliação usa dois nós.

connected-node-CMDB-class-name. O segundo tipo de classe necessário no TQL de reconciliação.

CMDB-link-type. O tipo de relacionamento necessário no TQL de reconciliação.

link-direction. A direção do relacionamento no TQL de reconciliação (de node para ip_address ou de ip_address para node):

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

A expressão de reconciliação está na forma de ORs e cada OR inclui ANDs.

is-ordered. Determina se a reconciliação é executada na forma de ordem ou por uma comparação OR regular.

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

Se a propriedade de reconciliação for recuperada da classe principal (o multinó), use a marca **attribute**, caso contrário, use a marca **connected-node-attribute**.

ignore-case. true: quando os dados no modelo de classe UCMDB são comparados com os dados no RDBMS, o caso não importa:

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

O nome da coluna é o nome da coluna de chave estrangeira (a coluna com valores que apontam para a coluna de chave primária de multinó).

Se a coluna de chave primária de multinó for composta de várias colunas, será necessário haver diversas colunas de chave estrangeira, sendo uma para cada coluna de chave primária.

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

Se houver poucas colunas de chave primária, duplique essa coluna.

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

As propriedades **from-CMDB-converter** e **to-CMDB-converter** são classes Java que implementam as seguintes interfaces:

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

Use esses conversores se o valor no CMDB e no banco de dados não for o mesmo. Por exemplo, o nome do nó no CMDB tem o sufixo mer.com.

Neste exemplo, GenericEnumTransformer é usado para converter o enumerador de acordo com o arquivo XML escrito entre parênteses (**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.i
mpl.GenericEnumTransformer(generic-enum-transformer-example.xml)"
to-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)"/>
        <attribute CMDB-attribute-name="[CMDB_attribute_name]"
column-name="[column_name]"/>
        <attribute CMDB-attribute-name="[CMDB_attribute_name]"
column-name="[column_name]"/>
    </class>
</generic-DB-adapter-config>
```

Limitações

- Pode ser usado para mapear somente uma consulta TQL de nó (na fonte do banco de dados). Por exemplo, você pode executar um node > ticket e uma consulta TQL ticket. Para importar a hierarquia de nós do banco de dados, você precisa usar o arquivo **orm.xml** avançado.
- Somente relacionamentos um-para-muitos são aceitos. Por exemplo, você pode importar um ou mais tickets para cada nó. Você não pode importar tickets que pertencem a mais de um nó.
- Não pode conectar a mesma classe a diferentes tipos de TECs do CMDB. Por exemplo, se você definir que ticket está conectado a node, ele não poderá ser conectado a application também.

O arquivo orm.xml

Este arquivo é usado para mapeamento dos TECs do CMDB para tabelas de banco de dados.

Um modelo para ser usado na criação de um novo arquivo está localizado no diretório **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF META-INF**.

Para editar o arquivo XML para um adaptador implantado, navegue para **Gerenciamento do Adaptador** > **db-adapter** > **Arquivos de configuração**.

Esta seção inclui os seguintes tópicos:

O modelo de arquivo 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>
```

Não altere o nome do pacote.

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

entity. O nome do TEC do CMDB. Esta é a entidade multinode.

Verifique se **class** inclui um prefixo **generic_db_adapter.**.

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

Use uma tabela secundária se a entidade for mapeada para mais de uma tabela.

```
    <secondary-table name=""/>
    <attributes>
```

Para uma herança de tabela única com discriminador, use o código a seguir:

```
    <inheritance strategy="SINGLE_TABLE"/>
    <discriminator-value>node</discriminator-value>
    <discriminator-column name="[column_name]"/>
```

Os atributos com marca **id** são as colunas de chave primária. Verifique se a convenção de nomenclatura para essas colunas de chave primária são **idX** (id1, id2 e assim por diante) em que **X** é o índice de coluna na chave primária.

```
        <id name="id1">
```

Altere somente o nome da coluna da chave primária.

```
            <column updatable="false" insertable="false" name="[column_name]"/>
            <generated-value strategy="TABLE"/>
        </id>
```

basic. Usado para declarar os atributos do CMDB. Verifique se editou somente as propriedades **name** e **column_name**.

```
        <basic name="name">
            <column updatable="false" insertable="false" name="[column_name]"/>
        </basic>
```

Para uma herança de tabela única com discriminador, mapeie as classes estendidas da seguinte maneira:

```
    <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>
```

O exemplo a seguir mostra um nome de atributo do CMDB sem prefixo:

```
            <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 é uma entidade de relacionamento. A convenção de nomenclatura é **end1Type_linkType_end2Type**. Neste exemplo, **end1Type** é **node** e **linkType** é **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>
```

A entidade de destino é aquela para a qual esta propriedade está apontando. Neste exemplo, **end1** é mapeado para a entidade **node**.

many-to-one. Muitos relacionamentos podem ser conectados a um nó.

join-column. A coluna que contém IDs **end1** (os IDs de entidade de destino).

referenced-column-name. O nome de coluna na entidade de destino (**node**) que contém os IDs que são usados na coluna de junção.

```
            <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. Um relacionamento pode ser conectado a um **[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>
```

Vários arquivos ORM

Vários arquivos de mapeamento são aceitos. Cada nome de arquivo de mapeamento deve terminar com **orm.xml**. Todos os arquivos de mapeamento devem ser colocados na pasta META-INF do adaptador.

Convenções de nomenclatura

- Em cada entidade, a propriedade classe precisa corresponder à propriedade de nome com o prefixo de generic_db_adapter.
- As colunas de chave primária precisam ter nomes com a forma **idX** em que **X** = **1**, **2**, ..., de acordo com o número de chaves primárias na tabela.
- Os nomes de atributo precisam corresponder aos nomes de atributo de classe inclusive em relação a maiúsculas e minúsculas.
- O nome do relacionamento tem a forma end1Type_linkType_end2Type.
- Os TECs do CMDB, que também são nomes reservados em Java, devem ter como prefixo **gdba_**. Por exemplo, para o TEC do CMDB **goto**, a entidade ORM deve ter o nome **gdba_goto**.

Usando instruções SQL em linha em vez de nomes de tabela

Você pode mapear entidades para cláusulas select em linha em vez de mapear para tabelas de banco de dados. Isso equivale a definir uma visualização no banco de dados e mapear uma entidade para essa visualização. Por exemplo:

```
    <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)"/>
```

Nesse exemplo, os atributos de nó devem ser mapeados para as colunas id1, name e host_os, em vez de id, name e os.

As seguintes limitações se aplicam:

- A instrução SQL em linha está disponível somente ao usar o Hibernate como provedor de JPA.
- Parênteses ao redor da cláusula de seleção SQL em linha são obrigatórias.
- O elemento <**schema**> não deve estar presente no arquivo **orm.xml**. No caso do Microsoft SQL Server 2005, isso significa que todos os nomes de tabela devem ter como prefixo dbo., em vez de defini-los globalmente por <schema>dbo</schema>.

O esquema orm.xml

A tabela a seguir explica os elementos comuns do arquivo **orm.xml**. O esquema completo pode ser encontrado em http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. A lista não está completa e explica principalmente o comportamento específico da JPA (API de Persistência Java) padrão para o adaptador de banco de dados genérico.

Elemento		Atributos
Nome e caminho	**Descrição**	
entity-mappings	O elemento raiz do documento de mapeamento de entidade. Esse elemento deve ser exatamente o mesmo daquele fornecido nos arquivos de exemplo de adaptador de banco de dados genérico.	
description (entity-mappings)	Uma descrição de texto livre do documento de mapeamento de entidade. Opcional.	

Elemento		Atributos
Nome e caminho	**Descrição**	
package (entity-mappings)	O nome do pacote Java que conterá as classes de mapeamento. Deve sempre conter o texto generic_db_adapter.	**Nome**. name **Descrição.** O nome do tipo de EC do UCMDB ao qual essa entidade está mapeada. Se essa entidade estiver mapeada para um link no CMDB, o nome da entidade deverá estar no formato <end_1>_<link_name>_<end_2>. Por exemplo, node_composition_cpu define uma entidade que será mapeada para o link de composição entre um nó e uma CPU. Se o nome do tipo de EC for o mesmo do nome da classe Java sem o prefixo do pacote, esse campo poderá ser omitido. **É obrigatório.** Opcional **Tipo.** Cadeia
		Nome. class **Descrição.** O nome totalmente qualificado da classe Java que será criado para essa entidade de banco de dados. O nome do pacote da classe Java deve ser o mesmo do nome fornecido no elemento package. Você não pode usar palavras reservadas Java, como uma interface ou switch, como o nome da classe. Em vez disso, adicione o prefixo gdba_ ao nome (para que a interface seja generic_db_adapter.gdba_interface. **É obrigatório.** Obrigatório **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
table (entity-mappings > entity)	Esse elemento define a tabela primária da entidade de banco de dados. Pode aparecer somente uma vez. Obrigatório.	**Nome**. name **Descrição.** O nome da tabela primária. Se o nome da tabela não contiver o esquema ao qual pertence, a tabela será pesquisada somente no esquema do usuário que foi utilizado para criar o ponto de integração. Isso também pode ser qualquer instrução SELECT válida. Se essa for uma instrução SELECT, será necessário encapsulá-la com parênteses. **É obrigatório.** Obrigatório **Tipo.** Cadeia
secondary-table (entity-mappings > entity)	Esse elemento pode ser usado para definir uma tabela secundária da entidade de banco de dados. Essa tabela precisa ser conectada à tabela primária com um relacionamento 1-para-1. Você pode definir mais de uma tabela secundária. Opcional.	**Nome**. name **Descrição.** O nome da tabela secundária. Se o nome da tabela não contiver o esquema ao qual pertence, a tabela será pesquisada somente no esquema do usuário que foi utilizado para criar o ponto de integração. Isso também pém pode ser qualquer instrução SELECT válida. Se essa for uma instrução SELECT, será necessário encapsulá-la com parênteses. **É obrigatório.** Obrigatório **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
primary-key-join-column (entity-mappings > entity > secondary-table)	Se as tabelas secundária e primária não estiverem conectadas usando campos com o mesmo nome, esse elemento definirá o nome do campo de chave primária na tabela secundária que precisa estar conectada ao campo de chave primária da tabela primária.	**Nome**. name **Descrição.** O nome do campo de chave primária na tabela secundária. Se esse elemento não existe, pressupõe-se que o campo de chave primária tenha o mesmo nome do campo de chave primária da tabela primária. **É obrigatório.** Opcional **Tipo.** Cadeia
herança (entity-mappings > entity)	Se a entidade atual for a entidade pai de uma família de entidades de banco de dados, use esse elemento para marcá-la como tal. Opcional.	**Nome**. strategy **Descrição.** Define a maneira como a herança é implementada no banco de dados. **É obrigatório.** Obrigatório **Tipo.** Um dos seguintes valores: ➤ SINGLE_TABLE – Este entidade e todas as entidades filhos existem na mesma tabela. ➤ JOINED – As entidades filho estão em tabelas juntadas. ➤ TABLE_PER_CLASS – Cada entidade é definida completamente por uma tabela separada.
discriminator-column (entity-mappings > entity)	Se a herança for do tipo SINGLE_TABLE, este elemento será usado para definir o nome do campo usado para determinar o tipo de entidade para cada linha.	**Nome**. name **Descrição.** O nome da coluna do discriminador. **É obrigatório.** Obrigatório **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
discriminator-value (entity-mappings > entity)	Este elemento define o tipo da entidade específica na árvore de herança. Este nome precisa ser o mesmo do nome definido no arquivo **discriminator.properties** para o grupo de valores deste tipo de entidade específico.	
attributes (entity-mappings > entity)	O elemento raiz de todos os mapeamentos de atributo de uma entidade.	
id (entity-mappings > entity attributes)	Este elemento define o campo de chave da entidade. Deve haver pelo menos um campo id definido. Se existir mais de um elemento id, seus campos criarão uma chave composta para a entidade. Você deve testar e evitar chaves compostas para entidades de EC (não para links).	**Nome**. name **Descrição.** Uma cadeia de caracteres do tipo idX, em que X é um número entre 1 e 9. O primeiro id deve ser marcado como id1, o segundo como id2 e assim por diante. Este NÃO é o nome do atributo-chave do UCMDB. **É obrigatório.** Obrigatório **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
basic (entity-mappings > entity attributes)	Este elemento define um mapeamento entre um campo na tabela, que não faz parte da chave primária da tabela, e um atributo do UCMDB.	**Nome**. name **Descrição.** O nome do atributo do UCMDB para o qual esse campo está mapeado. Este atributo precisa existir no tipo de EC do UCMDB para o qual essa entidade atual está mapeada. **É obrigatório.** Obrigatório **Tipo.** Cadeia
column (entity-mappings > entity > attributes > id -OU- (entity-mappings > entity > attributes > basic)	Define o nome da coluna na tabela para mapeamento básico ou um campo id.	**Nome**. name **Descrição.** O nome do campo. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. table **Descrição.** O nome da tabela à qual o campo pertence. Esse precisa ser a tabela primária ou uma das tabelas secundárias definidas para a entidade. Se esse atributo for omitido, será pressuposto que o campo pertence à tabela principal. **É obrigatório.** Opcional **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
one-to-one (entity-mappings > entity > attributes)	Define uma coluna cujo valor está em outra tabela, e as duas tabelas estão conectadas usando um relacionamento um-para-um. Este elemento só é aceito para mapeamentos de entidade de link e não para outros tipos de EC. Essa é a única maneira de definir um mapeamento entre uma tabela e um link UCMDB.	**Nome**. name **Descrição.** Qual das duas extremidades este campo representa. **É obrigatório.** Obrigatório **Tipo.** end1 ou end2
		Nome. target-entity **Descrição.** O nome da entidade à qual a extremidade se refere. **É obrigatório.** Obrigatório **Tipo.** Um dos nomes de entidade definidos no documento de mapeamento de entidade
join-column (entity-mappings > entity attributes > one-to-one)	Define a maneira de juntar o target-entity definido no elemento um-para-um pai e a entidade atual.	**Nome**. name **Descrição.** O nome do campo na tabela atual que será usado para executar a junção um-para-um. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. name **Descrição.** O nome de um campo na entidade de junção pela qual será executada a junção. Se esse atributo for omitido, pressupõe-se que a tabela de junção tenha uma coluna com o mesmo nome do campo definido no atributo de nome. **É obrigatório.** Opcional **Tipo.** Cadeia

O arquivo reconciliation_types.txt

Este arquivo é usado para configurar os tipos de reconciliação.

Cada linha no arquivo representa um TEC do CMDB que está conectado a um TEC de banco de dados federado na consulta TQL.

O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)

Este arquivo é usado para configurar as regras de reconciliação se você quiser executar a reconciliação quando o DBMappingEngine estiver configurado no adaptador. Se você não usar o DBMappingEngine, o mecanismo de reconciliação genérico do UCMDB será usado e não haverá necessidade de configurar esse arquivo.

Cada linha no arquivo representa uma regra. Por exemplo:

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

O multinode é preenchido com o nome de multinó (o TEC do CMDB que está conectado a um TEC de banco de dados federado no TQL).

Esta expressão inclui a lógica que decide se dois multinós são iguais (um multinó no CMDB e o outro na fonte do banco de dados).

A expressão consiste em ORs ou ANDs.

A convenção em relação a nomes de atributos na parte da expressão é [className].[attributeName]. Por exemplo, attributeName na classe ip_address é escrito ip_address.name.

Para ver uma correspondência ordenada (se a primeira subexpressão OR retornar uma resposta de que os multinós não são iguais, a segunda subexpressão OR não será comparada), use ordered expression em vez de expression.

Para ignorar a diferenciação de maiúsculas e minúsculas durante uma comparação, use o sinal de controle (^).

Os parâmetros end1_type, end2_type e link_type são usados somente se o TQL de reconciliação contém dois nós e não só um multinó. Nesse caso, o TQL de reconciliação é end1_type > (link_type) > end2_type.

Não é necessário adicionar o layout relevante porque ele é obtido da expressão.

Tipos de regras de reconciliação

As regras de reconciliação têm a forma de condições OR e AND. Você pode definir essas regras em vários nós diferentes (por exemplo, o nó é identificado por name from node AND/OR name from ip_address).

As seguintes opções localizam uma correspondência:

- **Correspondência ordenada.** A expressão de reconciliação é lida da esquerda para a direita. Duas subexpressões OR são consideradas iguais se têm valores e estes são iguais. Duas subexpressões OR não são consideradas iguais se ambas têm valores e estes não são iguais. Para qualquer outro caso, não há decisão, e a subexpressão OR seguinte é testada para fins de igualdade.

  name from node OR from ip_address. Se tanto o CMDB quanto a fonte de dados incluir name e eles forem iguais, os nós serão considerados iguais. Se ambos tiverem name mas não forem iguais, os nós não serão considerados iguais sem testar o name do ip_address. Se tanto o CMDB quanto a fonte de dados não tiver o name of node, o name of ip_address será verificado.

- **Correspondência regular.** Se houver igualdade em uma das duas subexpressões OR, o CMDB e a fonte de dados serão considerados iguais.

  name from node OR from ip_address. Se não houver correspondência no name of node, o name of ip_address será verificado para ver se há igualdade.

Para reconciliações complexas, em que a entidade de reconciliação é modelada no modelo de classe como vários TECs e relacionamentos (como node), o mapeamento de um nó de incluirá todos os atributos relevantes de todos os TECs modelados.

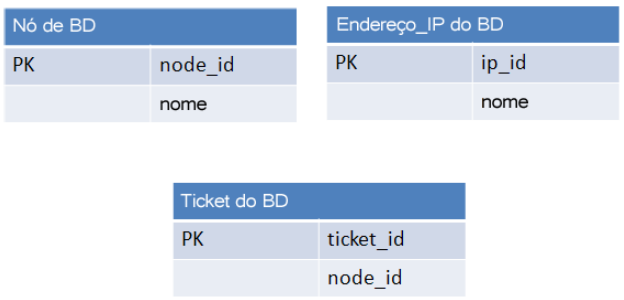
Observação: Consequentemente, haverá uma limitação de que todos os atributos de reconciliação na fonte de dados deverá residir em tabelas que compartilham a mesma chave primária.

Outra limitação informa que o TQL de reconciliação deve ter no máximo dois nós. Por exemplo, o TQL node > ticket tem um nó no CMDB e um a ticket na fonte de dados.

Para reconciliar os resultados, o name precisa ser recuperado do nó e/ou ip_address.

Se o name no CMDB estiver no formato de *.m.com, um conversor poderá ser usado do CMDB para o banco de dados federado, e vice-versa, para converter esses valores.

A coluna node_id na tabela de ticket de banco de dados será usada para conectar entre as entidades (a associação definida também pode ser feita em uma tabela de nó):



Observação: As três tabelas precisam fazer parte da fonte do RDBMS federada e não do banco de dados do CMDB.

O arquivo transformations.txt

Este arquivo contém todas as definições do conversor.

O formato é que cada linha contém uma nova definição.

O modelo de arquivo transformations.txt

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]]
to_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

entity. O nome da entidade conforme aparece no arquivo orm.xml.

attribute. O nome do atributo conforme aparece no arquivo orm.xml.

to_DB_class. O nome completo qualificado de uma classe que implementa a interface **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB**. Os elementos entre parênteses são fornecidos para esse construtor de classe. Use este conversor para transformar valores do CMDB em valores de banco de dados, por exemplo, para acrescentar o sufixo de **.com** a cada nome de nó.

from_DB_class. O nome completo qualificado de uma classe que implementa a interface **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB** Os elementos entre parênteses são fornecidos para esse construtor de classe. Use este conversor para transformar valores de banco de dados em valores do CMDB, por exemplo, para acrescentar o sufixo de **.com** a cada nome de nó.

Para ver detalhes, consulte "Conversores prontos" na página 199.

O arquivo persistence.xml

Este arquivo é usado para substituir as configurações do Hibernate padrão e adicionar suporte a tipos de banco de dados que não são prontos (os tipos de banco de dados OOB são Oracle Server, Microsoft MSSQL Server e MySQL).

Se você precisar oferecer suporte a um novo tipo de banco de dados, verifique se forneceu um provedor de pool de conexão (o padrão é c3p0) e um driver JDBC para o banco de dados (coloque os arquivos *.jar na pasta do adaptador).

Para ver todos os valores do Hibernate disponíveis que podem ser alterados, verifique a classe **org.hibernate.cfg.Environment**.

Exemplo do arquivo 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">
    <!-- Não mude esse valor -->
    <persistence-unit name="GenericDBAdapter">
        <properties>
            <!-- Não mude esse valor -->
            <property name="hibernate.archive.autodetection" value="class, hbm"/>
            <!--Nome da classe do driver"/-->
            <property name="hibernate.connection.driver_class"
value="com.mercury.jdbc.MercOracleDriver"/>
            <!--A url de conexão"/-->
            <property name="hibernate.connection.url" value="jdbc:mercury:oracle://
artist:1521;sid=cmdb2"/>
            <!-- Credenciais de logon no BD"/-->
            <property name="hibernate.connection.username" value="CMDB"/>
            <property name="hibernate.connection.password" value="CMDB"/>
            <!--propriedades de pool de conexão"/-->
            <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"/>
            <!--O dialeto a usar-->
            <property name="hibernate.dialect"
value="org.hibernate.dialect.OracleDialect"/>
        </properties>
    </persistence-unit>
</persistence>
```

O arquivo discriminator.properties

Este arquivo mapeia cada tipo de EC aceito (que também é usado como valor de discriminador em orm.xml) para uma lista separada por vírgulas de possíveis valores correspondentes da coluna do discriminador.

Se o adaptador que você está criando usar recursos de discriminador, será necessário definir todos os valores de discriminador no arquivo **discriminator.properties**.

Exemplo de mapeamento de discriminador:

O arquivo **discriminator.properties** inclui o seguinte código:

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

O arquivo orm.xml inclui o seguinte código:

```
<entity class="generic_db_adapter.node" >
    <table name="[table_name]"/>
    …
    <inheritance strategy="SINGLE_TABLE"/>
    <discriminator-value>node</discriminator-value>
    <discriminator-column name="[discriminator_column]"/>
    …
</entity>
<entity class="generic_db_adapter.nt" name="nt">
    <discriminator-value>nt</discriminator-value>
    <attributes/>
</entity>
<entity class="generic_db_adapter.unix" name="unix">
    <discriminator-value>unix</discriminator-value>
    <attributes/>
</entity>
```

O atributo [discriminator_column] é calculado da seguinte maneira:

- A coluna do discriminador da tabela correspondente contém 10002 para uma determinada entrada. A entrada é mapeada para o TEC **nt** .
- A coluna do discriminador da tabela correspondente contém 10006 para uma determinada entrada. A entrada é mapeada para o TEC **unix** .
- A coluna do discriminador da tabela correspondente contém 10010 para uma determinada entrada. A entrada é mapeada para o TEC **node** .

Observe que o TEC **node** também é o pai de **nt** e **unix**.

O arquivo replication_config.txt

Este arquivo contém uma lista separada por vírgulas de tipos de EC e de relacionamento cujas condições de propriedade são aceitas pelo plugin de replicação. Para ver detalhes, consulte "Plugins" na página 203.

O arquivo fixed_values.txt

Este arquivo permite configurar os valores fixos para atributos específicos de determinados TECs. Dessa maneira, cada um desses atributos pode receber um valor fixo que não está armazenado no banco de dados.

O arquivo deve conter zero ou mais entradas do seguinte formato:

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

Por exemplo:

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

Conversores prontos

Você pode usar os seguintes conversores (transformadores) para converter consultas federadas e trabalhos de replicação em dados de banco de dados e vice-versa.

Esta seção inclui os seguintes tópicos:

O conversor enum-transformer

Este conversor usa um arquivo XML fornecido como um parâmetro de entrada.

O arquivo XML mapeia entre valores do CMDB embutidos no código-fonte e valores de banco de dados (enums). Se um dos valores não existir, você poderá optar por retornar o mesmo valor, retornar nulo ou lançar uma exceção.

Use um arquivo de mapeamento XML para cada atributo de entidade.

Observação: Este conversor pode ser usado para os campos to_DB_class e from_DB_class no arquivo **transformations.txt**.

Exemplo do XSD do arquivo 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:complexType>
    </xs:element>
    <xs:element name="value">
        <xs:complexType>
            <xs:attribute name="CMDB-value" type="xs:string" use="required"/>
            <xs:attribute name="external-DB-value" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Exemplo de conversão do valor 'sys' no valor 'System':

Neste exemplo, o valor sys no CMDB é transformado no valor System no banco de dados federado, enquanto o valor System no banco de dados federado é transformado no valor sys no CMDB.

Se o valor não existir no arquivo XML (por exemplo, a cadeia de caracteres demo), o conversor retornará o mesmo valor de entrada recebido.

```
<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>
```

O conversor SuffixTransformer

Este conversor é usado para adicionar ou remover sufixos do valor do CMDB ou do valor da fonte de banco de dados federado.

Há duas implementações:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddSuffixTransformer**. Adiciona o sufixo (fornecido como entrada) ao converter do valor de banco de dados federado para o valor CMDB e remove o sufixo ao converter do valor CMDB para o valor de banco de dados federado.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemoveSuffixTransformer**. Remove o sufixo (fornecido como entrada) ao converter do valor de banco de dados federado para o valor CMDB e adiciona o sufixo ao converter do valor CMDB para o valor de banco de dados federado.

O conversor PrefixTransformer

Este conversor é usado para adicionar ou remover um prefixo do valor do CMDB ou do valor da fonte de banco de dados federado.

Há duas implementações:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbAddPrefixTransformer**. Adiciona o prefixo (fornecido como entrada) ao converter do valor de banco de dados federado para o valor CMDB e remove o prefixo ao converter do valor CMDB para o valor de banco de dados federado.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdbRemovePrefixTransformer**. Remove o prefixo (fornecido como entrada) ao converter do valor de banco de dados federado para o valor CMDB e adiciona o prefixo ao converter do valor CMDB para o valor de banco de dados federado.

O conversor BytesToStringTransformer

Este conversor é usado para converter matrizes de bytes do CMDB em sua representação de cadeia de caracteres da fonte de banco de dados federado.

O conversor é:
com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytesToStringTransformer.

Plugins

O adaptador de banco de dados genérico oferece suporte aos seguintes plugins:

- Um plugin opcional para sincronização de topologia completa.
- Um plugin opcional para sincronização de alterações na topologia. Se não estiver implementado nenhum plugin para sincronizar alterações, será possível executar uma sincronização diferencial, mas ela na verdade será total.
- Um plugin opcional para layout de sincronização.
- Um plugin opcional para recuperar consultas aceitas para sincronização. Se esse plugin não estiver definido, todos os nomes TQL serão retornados.
- Um plugin opcional interno para alterar a definição e o resultado de TQL.
- Um plugin opcional interno para alterar uma solicitação de layout e o resultado de ECs.
- Um plugin opcional interno para alterar uma solicitação de layout e o resultado de relacionamentos.

Para ver detalhes sobre como implementar e implantar plugins, consulte "Implementar um plugin" na página 150.

Exemplos de configuração

Esta seção fornece exemplos de configurações.

Esta seção inclui os seguintes tópicos:

Caso de uso

Caso de uso. Um TQL é:

node > **(composition)** > **card**

onde:

node é a entidade do CMDB

card é a entidade da fonte de banco de dados federado

composition é o relacionamento entre eles

O exemplo é executado com base no banco de dados ED. ED nodes são armazenados na tabela Device e card é armazenado na tabela hwCards. Nos exemplos a seguir, card sempre é mapeado da mesma maneira.

Reconciliação de nó único

Neste exemplo, a reconciliação é executada com base na propriedade name.

Definição simplificada

A reconciliação é executada por node e enfatizada pela marca 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>
```

Definição avançada

O arquivo orm.xml

Preste atenção à inclusão do mapeamento de relacionamentos. Para ver detalhes, consulte a seção de definições em "O arquivo orm.xml" na página 180.

Exemplo do arquivo 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>
```

O arquivo reconciliation_types.txt

Para ver detalhes, consulte "O arquivo reconciliation_types.txt" na página 192.

```
nó
```

O arquivo reconciliation_rules.txt

Para ver detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 192.

```
multinode[node] expression[node.name]
```

O arquivo transformations.txt

Este arquivo continua vazio porque nenhum valor precisa ser convertido neste exemplo.

Reconciliação de nó duplo

Neste exemplo, a reconciliação é calculada de acordo com a propriedade name de node e de ip_address com diferentes variações.

O TQL de reconciliação é **node** > **(containment)** > **ip_address**.

Definição simplificada

A reconciliação é 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>
```

A reconciliação é 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">
            <e>
                <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>
```

A reconciliação é 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>
```

Definição avançada

O arquivo orm.xml

Como a expressão de reconciliação não é definida neste arquivo, a mesma versão deve ser usada para qualquer expressão de reconciliação.

O arquivo reconciliation_types.txt

Para ver detalhes, consulte "O arquivo reconciliation_types.txt" na página 192.

```
nó
```

O arquivo reconciliation_rules.txt

Para ver detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 192.

```
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]
```

O arquivo transformations.txt

Este arquivo continua vazio porque nenhum valor precisa ser convertido neste exemplo.

Usando uma chave primária que contém mais de uma coluna

Se a chave primária for composta por mais de uma coluna, o seguinte código será adicionado às definições de XML:

Definição simplificada

Há mais de uma marca de chave primária e, para cada coluna, há uma marca.

```
    <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>
```

Definição avançada

O arquivo orm.xml

É adicionada uma nova entidade id que mapeia para as colunas de chave primária. As entidades que usam essa entidade id precisam adicionar uma marca especial.

Se você usar uma chave estrangeira (marca join-column) para essa chave primária, deverá mapear entre cada coluna na chave estrangeira para uma coluna na chave primária.

Para ver detalhes, consulte "O arquivo orm.xml" na página 180.

Exemplo do arquivo 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>
```

Usando transformações

No exemplo a seguir, o transformador **enum** genérico é convertido dos valores 1, 2, 3 para os valores a, b, c respectivamente na coluna name.

O arquivo de mapeamento é 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>
```

Definição 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.i
mpl.GenericEnumTransformer(generic-enum-transformer-example.xml)"
to-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)"/>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress"/>
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
.
.
.
```

Definição avançada

Há uma alteração somente no arquivo **transformation.txt**.

O arquivo transformations.txt

Verifique se os nomes de atributo e de entidade são os mesmos no arquivo orm.xml.

```
entity[node] attribute[name]
to_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Generic
EnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.Gene
ricEnumTransformer(generic-enum-transformer-example.xml)]
```

Arquivos de log do adaptador

Para entender os fluxos de cálculo e o ciclo de vida do adaptador, bem como visualizar informações de depuração, você pode consultar os seguintes arquivos de log.

Esta seção inclui os seguintes tópicos:

➤ "Níveis de log" na página 215

➤ "Locais dos arquivos" na página 216

Níveis de log

É possível configurar o nível de log de cada um dos logs.

Em um editor de texto, abra o arquivo **C:\hp\UCMDB\UCMDBServer\conf\log\fcmdb.gdba.properties**.

O nível de log padrão é **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL
loglevel=ERROR
```

➤ Para aumentar o nível de log para todos os arquivos de log, altere **loglevel=ERROR** para **loglevel=DEBUG** ou **loglevel=INFO**.

➤ Para alterar o nível de log para um arquivo específico, altere a linha de categoria **log4j** específica adequadamente. Por exemplo, para alterar o nível de log de fcmdb.gdba.dal.sql.log para **INFO**, altere

```
log4j.category.fcmdb.gdba.dal.SQL=${loglevel},fcmdb.gdba.dal.SQL.appender
```

para:

```
log4j.category.fcmdb.gdba.dal.SQL=INFO,fcmdb.gdba.dal.SQL.appender
```

Locais dos arquivos

Os arquivos de log ficam localizados no diretório **C:\hp\UCMDB\UCMDBServer\runtime\log** .

➤ **Fcmdb.gdba.log**

O log do ciclo de vida do adaptador. Fornece detalhes sobre quando o adaptador iniciou ou parou, além de quais TECs são aceitos por esse adaptador.

Consulte sobre erros de iniciação (carregamento/descarregamento do adaptador).

➤ **fcmdb.log**

Consulte sobre exceções.

➤ **cmdb.log**

Consulte sobre exceções.

➤ **Fcmdb.gdba.mapping.engine.log**

O log do mecanismo de mapeamento. Fornece detalhes sobre o TQL de reconciliação que o mecanismo de mapeamento usa, além das topologias de reconciliação que são comparadas com a fase de conexão.

Consulte este log quando uma consulta TQL não fornecer resultados, mesmo que você saiba que há ECs relevantes no banco de dados ou que os resultados sejam inesperados (verifique a reconciliação).

➤ **Fcmdb.gdba.TQL.log**

O log TQL. Fornece detalhes sobre as consultas TQL e seus resultados.

Consulte este log quando uma consulta TQL não retornar resultados e o log do mecanismo de mapeamento mostrar que não há resultados na fonte de dados federados.

➤ **Fcmdb.gdba.dal.log**

O log do ciclo de vida de DAL. Fornece detalhes sobre conexão de banco de dados e geração de TECs.

Consulte este log quando não puder se conectar ao banco de dados ou quando houver TECs ou atributos não aceitos pela consulta.

- **Fcmdb.gdba.dal.command.log**

  O log de operações de DAL. Fornece detalhes sobre operações de DAL internas que são chamadas. (Este log é semelhante ao cmdb.dal.command.log).

- **Fcmdb.gdba.dal.SQL.log**

  O log de consultas SQL de DAL. Fornece detalhes sobre JPAQLs chamadas (consultas SQL orientadas a objeto) e seus resultados.

  Consulte este log quando não puder se conectar ao banco de dados ou quando houver TECs ou atributos não aceitos pela consulta.

- **Fcmdb.gdba.hibrnate.log**

  O log do Hibernate. Fornece detalhes sobre as consultas SQL que são executadas, a análise de cada JPAQL em relação a SQL, os resultados das consultas, os dados referentes ao cache do Hibernate e assim por diante. Para ver detalhes sobre o Hibernate, consulte "Hibernate como provedor de JPA" na página 129.

Referências externas

Para ver detalhes sobre a especificação JavaBeans 3.0, consulte http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

Solução de problemas e limitações

Esta seção descreve a solução de problemas e as limitações do adaptador de banco de dados genérico.

Limitações gerais

- Não há suporte à autenticação NTLM do SQL Server.
- Ao atualizar um pacote de adaptadores, use o Notepad++, o UltraEdit ou algum outro editor de texto de terceiros em vez do Bloco de Notas (qualquer versão) da Microsoft Corporation para editar os arquivos de modelo. Isso impede o uso de símbolos especiais, que causam falha na implantação do pacote preparado.

Limitações de JPA

- Todas as tabelas precisam ter uma coluna de chave primária.
- Os nomes de atributo de classe do CMDB precisam seguir a convenção de nomenclatura JavaBeans (por exemplo, os nomes precisam iniciar com letras minúsculas).
- Dois ECs conectados a um relacionamento no modelo de classe precisam ter associação direta no banco de dados (por exemplo, se node estiver conectado a ticket, deverá haver uma chave estrangeira ou tabela vinculável que se conecte a ele).
- Várias tabelas mapeadas para o mesmo TEC precisam compartilhar a mesma tabela de chave primária.

Limitações funcionais

- Você não pode criar um relacionamento manual entre o CMDB e os TECs federados. Para poder definir relacionamentos virtuais, uma lógica de relacionamento especial precisa estar definida (ela pode se basear nas propriedades da classe federada).
- Os TECs federados não podem ser TECs acionadores em uma regra de impacto, mas podem ser incluídos em uma consulta TQL de análise.
- Um TEC federado pode fazer parte de um TQL de melhoria, mas não pode ser usado como o nó em que a melhoria é executada (você não pode adicionar, atualizar ou excluir o TEC federado).
- Não há suporte ao uso de um qualificador de classe em uma condição.
- Não há suporte a subgráficos.
- Não há suporte a relacionamentos compostos.
- O id do CMDB de EC externo é composto por sua chave primária, e não seus atributos-chave.
- Uma coluna de tipo bytes não pode ser usada como coluna de chave primária no Microsoft SQL Server.
- O cálculo de consulta TQL falhará se as condições de atributo definidas em um nó federado não tiverem tido seus nomes mapeados no arquivo **orm.xml**.
- O adaptador de banco de dados genérico não aceita a Autenticação do Windows para o SQL Server.

6

Desenvolvendo adaptadores Java

Este capítulo inclui:

Conceitos

Tarefas

Referência

Conceitos

Visão geral do Federation Framework

Observação:

- O termo **relationship** é equivalente ao termo **link**.
- O termo **CI** é equivalente ao termo **object**.
- Um gráfico é uma coleção de nós e links.
- Para ver um glossário de definições e termos, consulte "Glossário" no *Guia de Administração do HP Universal CMDB*.

A funcionalidade do Federation Framework usa uma API para recuperar informações de fontes federadas. O Federation Framework fornece três recursos principais:

- **Federação** em tempo real. Todas as consultas são executadas sobre repositórios de dados originais e os resultados são criados em tempo real no CMDB.
- **População.** Popula os dados (dados de topologia e propriedades de EC) de uma fonte de dados externa para o CMDB.
- **Push de dados.** Faz push dos dados (dados de topologia e propriedades de EC) de uma fonte de dados remota para o CMDB local.

Todos os tipos de ação exigem um adaptador para cada repositório de dados, que pode fornecer os recursos específicos do repositório de dados e recuperar e/ou atualizar os dados necessários. Cada solicitação do repositório de dados é feita por meio de seu adaptador.

Esta seção também inclui os seguintes tópicos:

- "Federação em tempo real" na página 221
- "Push de Dados" na página 222
- "População" na página 223

Federação em tempo real

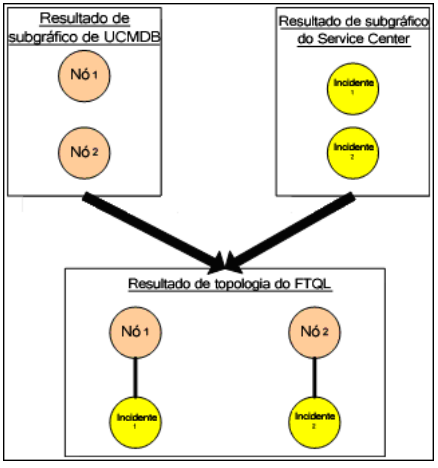
As consultas TQL federadas permitem a recuperação de dados de qualquer repositório de dados externo sem replicar seus dados.

Uma consulta TQL federada usa adaptadores que representam repositórios de dados externos, para criar relacionamentos externos apropriados entre ECs de diferentes repositórios de dados externos e os ECs do UCMDB.

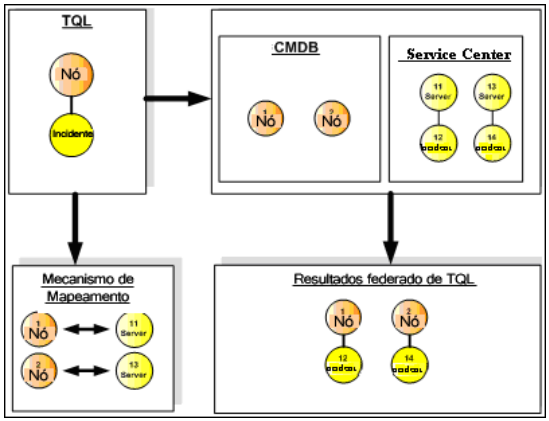
Exemplo de fluxo de federação em tempo real:

1 O Federation Framework divide uma consulta TQL federada em vários subgráficos, em que todos os nós de um subgráfico se referem ao mesmo repositório de dados. Cada subgráfico é conectado a outros subgráficos por um virtual relationship (mas ele próprio não contém relacionamentos virtuais).

2 Depois que a consulta TQL federada é dividida em subgráficos, o Federation Framework calcula a topologia de cada subgráfico e conecta dois subgráficos apropriados criando relacionamentos virtuais entre os nós apropriados.



3 Depois que a topologia TQL federada é calculada, o Federation Framework recupera um layout para o resultado da topologia.



Push de Dados

Use o fluxo de push de dados para sincronizar dados do CMDB local para um serviço remoto ou repositório de dados de destino.

No push de dados, os repositórios de dados são divididos em duas categorias: origem (CMDB local) e destino. Os dados são recuperados do repositório de dados de origem e atualizados no repositório de dados de destino. O processo de push de dados se baseia em nomes de consulta, significando que os dados são sincronizados entre os repositórios de dados de origem (CMDB local) e de destino e são recuperados por um nome de consulta TQL do CMDB local.

O fluxo do processo de push de dados inclui as seguintes etapas:

1 Recuperar o resultado de topologia com assinaturas do repositório de dados de origem.

2 Comparar os novos resultados com os anteriores.

3 Recuperar um layout completo (ou seja, todas as propriedades de EC) de ECs e relacionamentos – somente para os resultados alterados.

4 Atualizar o repositório de dados de destino com o layout completo recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

O CMDB tem 2 fontes de dados ocultas (**hiddenRMIDataSource** e **hiddenChangesDataSource**), que sempre são a fonte de dados de 'origem' nos fluxos de push de dados. Para implementar um novo adaptador para fluxos de push de dados, basta apenas implementar o adaptador de 'destino'.

População

Use o fluxo de população para popular o CMDB com dados de fontes externas.

O fluxo sempre usa um repositório de dados de 'origem' para recuperar os dados, e faz push dos dados recuperados para a Sonda em um processo semelhante ao fluxo de um trabalho de descoberta.

Para implementar um novo adaptador para fluxos de população, basta apenas implementar o adaptador de origem, já que a Sonda de Fluxo de Dados atua como destino.

O adaptador no fluxo de população é executado na Sonda. A depuração e o registro em log devem ser executados na Sonda e não no CMDB.

O fluxo de população se baseia em nomes de consulta, ou seja, os dados são sincronizados entre o repositório de dados de origem e a Sonda de Fluxo de Dados e são recuperados por um nome de consulta no repositório de dados de origem. Por exemplo, no UCMDB, o nome de consulta é o nome da consulta TQL. Entretanto, em outro repositório de dados, o nome da consulta pode ser um nome de código que retorna dados. O adaptador foi projetado para manipular corretamente o nome da consulta.

Cada trabalho pode ser definido como um trabalho exclusivo. Isso significa que os ECs e relacionamentos nos resultados do trabalho são exclusivos no CMDB local, e nenhuma consulta pode transmiti-los ao destino. O adaptador do repositório de dados de origem oferece suporte a consultas específicas e pode recuperar os dados desse repositório de dados. O adaptador do repositório de dados de destino permite a atualização dos dados recuperados nesse repositório de dados.

Fluxo SourceDataAdapter

- Recupera o resultado de topologia com assinaturas do repositório de dados de origem.
- Compara os novos resultados com os anteriores.
- Recupera um layout completo (ou seja, todas as propriedades de EC) de ECs e relacionamentos – somente para os resultados alterados.
- Atualiza o repositório de dados de destino com o layout completo recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

Fluxo SourceChangesDataAdapter

- Recupera o resultado de topologia que ocorreu desde a última data determinada.
- Recupera um layout completo (ou seja, todas as propriedades de EC) de ECs e relacionamentos – somente para os resultados alterados.
- Atualiza o repositório de dados de destino com o layout completo recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

Fluxo PopulateDataAdapter

- Recupera a topologia completa com o resultado de layout solicitado.
- Usa o mecanismo de partes de topologia para recuperar os dados em partes.

- A sonda filtra e desconsidera quaisquer dados que já foram transmitidos em execuções anteriores.
- Atualiza o repositório de dados de destino com o layout recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

Fluxo PopulateChangesDataAdapter

- Recupera a topologia com o resultado de layout solicitado que tem alterações desde a última execução.
- Usa o mecanismo de partes de topologia para recuperar os dados em partes.
- A sonda filtra e desconsidera quaisquer dados que já foram transmitidos em execuções anteriores (incluindo este fluxo).
- Atualiza o repositório de dados de destino com o layout recebido de ECs e relacionamentos. Se algum EC ou relacionamento for excluído no repositório de dados de origem e a consulta for exclusiva, o processo de replicação também removerá os ECs e relacionamentos no repositório de dados de destino.

 Interação de adaptador e mapeamento com o Federation Framework

Uma adaptador é uma entidade no UCMDB que representa os dados externos (dados que não são salvos no UCMDB). Em fluxos federados, todas as interações com fontes de dados externos são executadas através de adaptadores. O fluxo de interação e as interfaces de adaptador do Federation Framework são diferentes para replicação e consultas TQL federadas.

Esta seção também inclui os seguintes tópicos:

- "Ciclo de vida do adaptador" na página 226
- "Métodos assist do adaptador" na página 226

Ciclo de vida do adaptador

Uma instância de adaptador é criada para cada repositório de dados externo. O adaptador começa seu ciclo de vida com a primeira ação aplicada a ele (por exemplo, calculate TQL ou retrieve/update data). Quando o método **start** é chamado, o adaptador recebe informações do ambiente, como a configuração do repositório de dados, o logger e assim por diante. O ciclo de vida do adaptador termina quando o repositório de dados é removido da configuração, e o método **shutdown** é chamado. Isso significa que o adaptador tem informações de estado e pode conter a conexão ao repositório de dados externo se for necessário.

Métodos assist do adaptador

O adaptador tem vários métodos assist que podem adicionar configurações de repositório de dados externo. Esses métodos não fazem parte do ciclo de vida do adaptador e criam um novo adaptador sempre que são chamados.

- O primeiro método testa a conexão ao repositório de dados externo para uma determinada configuração. testConnection pode ser executado no servidor UCMDB ou na Sonda de Fluxo de Dados, dependendo do tipo de adaptador.
- O segundo método é relevante apenas para o adaptador de origem e retorna as consultas válidas para replicação. (Esse método é executado somente na Sonda.)
- O terceiro método é relevante apenas para fluxos de federação e de população e retorna external classs aceitas pelo repositório de dados externo. (Esse método é executado somente no servidor UCMDB.)

Todos esses métodos são usados ao criar ou visualizar configurações de integração.

Fluxo do Federation Framework para consultas TQL federadas

Esta seção inclui os seguintes tópicos:

Definições e termos

Dados de reconciliação. A regra para corresponder ECs do tipo especificado que são recebidos do CMDB e do repositório de dados externos. A regra de reconciliação pode ser de três tipos:

- **Reconciliação de ID.** Pode ser usada somente se o repositório de dados externo contém a ID do CMDB de objetos de reconciliação.
- **Reconciliação de propriedade.** É usada quando a correspondência pode ser feita por propriedades somente do tipo de EC de reconciliação.
- **Reconciliação de topologia.** É usada quando você precisa das propriedades de TECs adicionais (não só do TEC de reconciliação) para executar uma correspondência em ECs de reconciliação. Por exemplo, é possível executar reconciliação do tipo de nó pela propriedade name que pertence ao TEC ip_address .

Objeto de reconciliação. O objeto é criado pelo adaptador de acordo com os dados de reconciliação recebidos. Esse objeto deve se referir a um EC externo e é usado pelo Mecanismo de Mapeamento para se conectar entre os ECs externos e os ECs do CMDB.

Tipo de EC de reconciliação. O tipo de ECs que representam objetos de reconciliação. Esses ECs precisam ser armazenados tanto no CMDB quanto em repositórios de dados externos.

Mecanismo de mapeamento. Um componente que identifica os relacionamentos entre os ECs de diferentes repositórios de dados que têm um relacionamento virtual entre eles. A identificação é executada através dos objetos de reconciliação do CMDB e dos objetos de reconciliação de ECs externos.

Mecanismo de mapeamento

O Federation Framework usa o Mecanismo de Mapeamento para calcular a consulta TQL federada. O Mecanismo de Mapeamento se conecta entre os ECs que são recebidos de diferentes repositórios de dados e são conectados por relacionamentos virtuais. O Mecanismo de Mapeamento também fornece dados de reconciliação para o relacionamento virtual. Uma extremidade do relacionamento virtual precisa se referir ao CMDB. Essa extremidade é um tipo reconciliation. Para o cálculo dos dois subgráficos, um relacionamento virtual pode começar de qualquer nó final.

Adaptador federado

O adaptador Federado oferece dois tipos de dados dos repositórios de dados externos: dados de EC externos e objetos de reconciliação que pertencem a ECs externos.

- **Dados de EC externos.** Os dados de EC externos que não existem no CMDB. São os dados de destino do repositório de dados externo.
- **Dados de objeto de reconciliação.** Os dados auxiliares que são usados pelo Federation Framework para se conectar entre os ECs do CMDB e os dados externos. Cada objeto de reconciliação deve se referir a um EC externo. O tipo de objeto de reconciliação é o tipo (ou subtipo) de uma das extremidades de relacionamento virtual das quais os dados são recuperados. Os objetos de reconciliação devem ajustar o adaptador recebido aos dados de reconciliação. O objeto de reconciliação pode ser de um destes três tipos: IdReconciliationObject, PropertyReconciliationObject ou TopologyReconciliationObject.

Nas interfaces baseadas em DataAdapter (DataAdapter, PopulateDataAdapter e PopulateChangesDataAdapter), a reconciliação é solicitada como parte da definição da consulta.

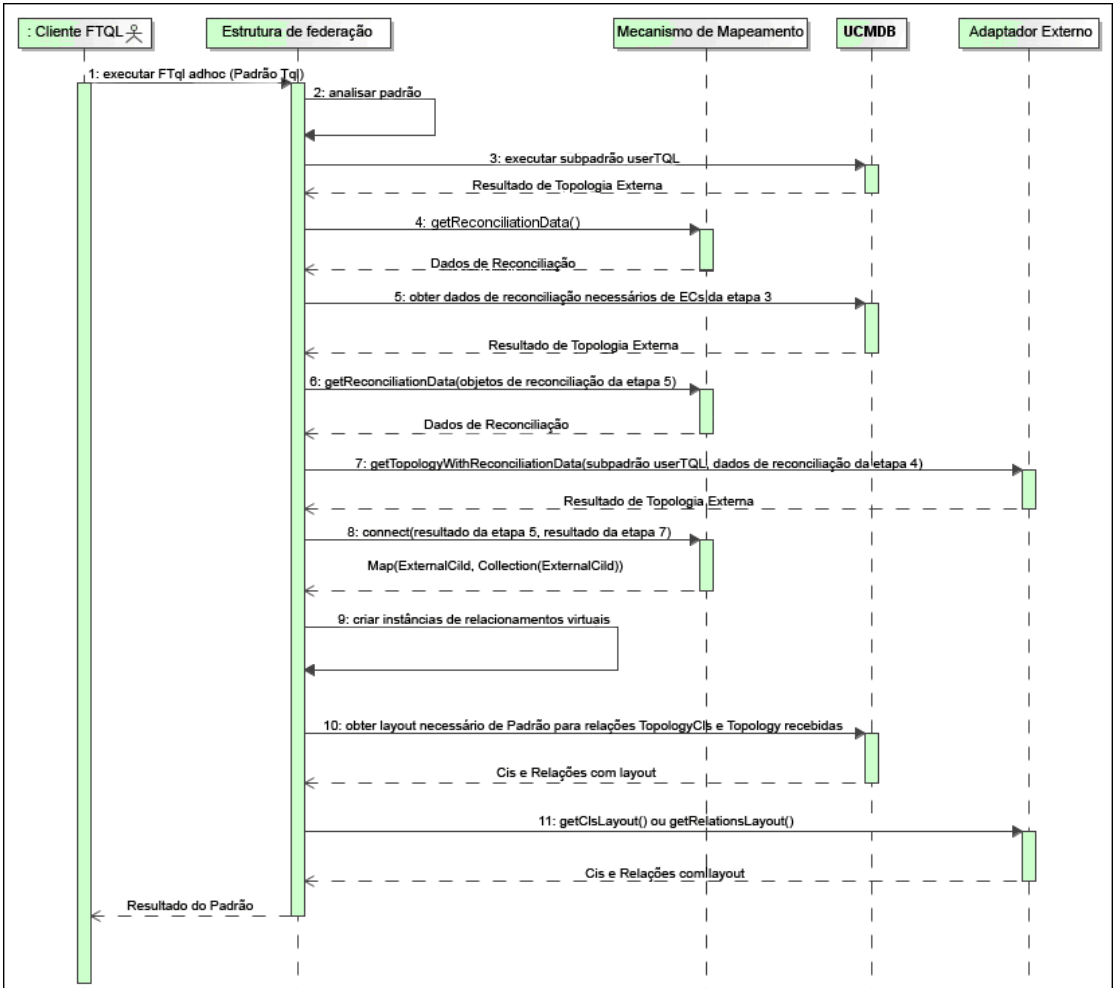
Diagramas de fluxo

Os links diagramas ilustram as interações entre o Federation Framework, o UCMDB, o adaptador e o Mecanismo de Mapeamento. A consulta TQL federada nos diagramas de exemplo tem somente um relacionamento virtual, por isso somente o UCMDB e um repositório de dados externo estão envolvidos na consulta TQL federada.

No primeiro diagrama, o cálculo começa no UCMDB e no segundo diagrama no adaptador externo. Cada etapa no diagrama inclui referências à chamada de método apropriada do adaptador ou da interface do mecanismo de mapeamento.

O cálculo começa na extremidade do HP Universal CMDB

O diagrama de sequência a seguir ilustra a interação entre o Federation Framework, o UCMDB, o adaptador e o Mecanismo de Mapeamento. A consulta TQL federada no diagrama de exemplo tem somente um relacionamento virtual, por isso somente o UCMDB e um repositório de dados externo estão envolvidos na consulta TQL federada.

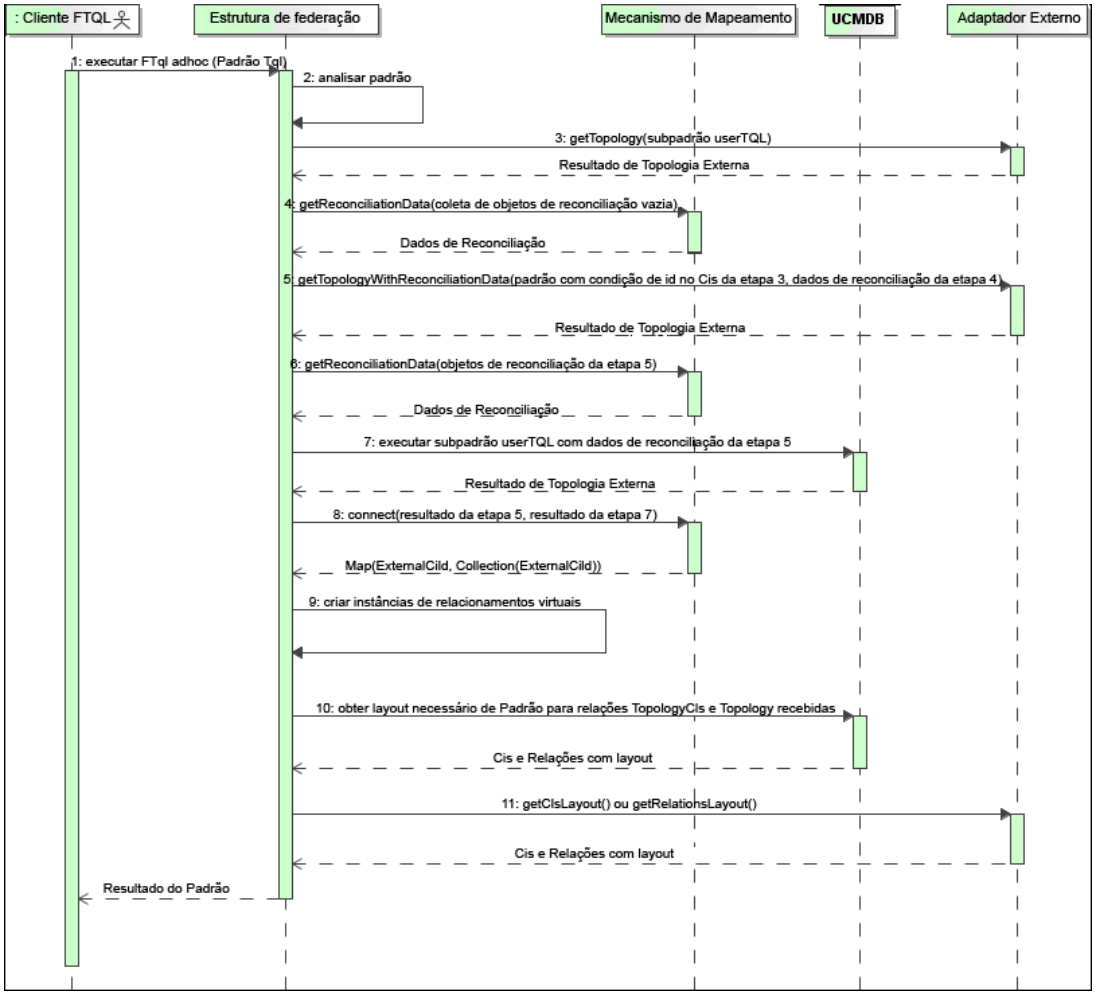


Os números nesta imagem são explicados a seguir:

Número	Explicação
1	O Federation Framework recebe uma chamada para um cálculo de TQL federado.
2	O Federation Framework analisa o adaptador, localiza o relacionamento virtual e divide o TQL original em dois subadaptadores – um para o UCMDB e outro para o repositório de dados externo.
3	O Federation Framework solicita a topologia do sub-TQL do UCMDB.
4	Depois de receber os resultados de topologia, o Federation Framework chama o Mecanismo de Mapeamento apropriado para o relacionamento virtual atual e solicita os dados de reconciliação. O parâmetro reconciliationObject está vazio nessa etapa, ou seja, nenhuma condição é adicionada aos dados de reconciliação nessa chamada. Os dados de reconciliação retornados definem quais dados são necessários para corresponder os ECs de reconciliação no UCMDB ao repositório de dados externo. Os dados de reconciliação podem ser de um destes tipos: ➤ **IdReconciliationData.** Os ECs são reconciliados de acordo com seu ID. ➤ **PropertyReconciliationData.** Os ECs são reconciliados de acordo com as propriedades de um dos ECs. ➤ **TopologyReconciliationData.** Os ECs são reconciliados de acordo com a topologia (por exemplo, para reconciliar ECs de nó, o endereço IP de **IP** é necessário também).
5	O Federation Framework solicita dados de reconciliação para os ECs das extremidades do relacionamento virtual que foram recebidas na etapa 3 do UCMDB.
6	O Federation Framework chama o Mecanismo de Mapeamento para recuperar os dados de reconciliação. Nesse estado (em contraste com a etapa 3), o Mecanismo de Mapeamento recebe os objetos de reconciliação da etapa 5 como parâmetros. O Mecanismo de Mapeamento traduz o objeto de reconciliação recebido para a condição nos dados de reconciliação.

Número	Explicação
7	O Federation Framework solicita a topologia do sub-TQL do repositório de dados externo. O adaptador externo recebe os dados de reconciliação da etapa 6 como um parâmetro.
8	O Federation Framework chama o Mecanismo de Mapeamento para se conectar entre os resultados recebidos. O parâmetro firstResult é o resultado de topologia externa recebido do UCMDB na etapa 5 e o parâmetro secondResult é o resultado de topologia externa recebido do Adaptador Externo na etapa 7. O Mecanismo de Mapeamento retorna um mapa em que a ID de EC Externo do primeiro repositório de dados externo (UCMDB, neste caso) é mapeado para os IDs de EC Externo do segundo repositório de dados (externo).
9	Para cada mapeamento, o Mecanismo de Mapeamento cria um relacionamento virtual.
10	Depois do cálculo dos resultados da consulta TQL federada (somente na etapa de topologia), o Federation Framework recupera o layout TQL original para os ECs e relacionamentos resultantes dos repositórios de dados apropriados.

O cálculo começa na extremidade do adaptador externo



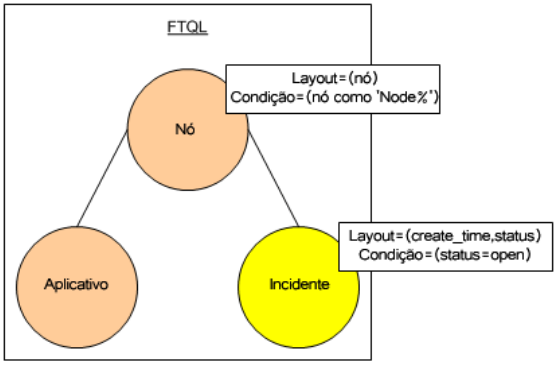
Os números nesta imagem são explicados a seguir:

Número	Explicação
1	O Federation Framework recebe uma chamada para um cálculo de TQL federado.
2	O Federation Framework analisa o adaptador, localiza o relacionamento virtual e divide o TQL original em dois subadaptadores – um para o UCMDB e outro para o repositório de dados externo.
3	O Federation Framework solicita a topologia do sub-TQL do Adaptador Externo. O ExternalTopologyResult retornado não deve conter nenhum objeto de reconciliação, porque os dados de reconciliação não fazem parte da solicitação.
4	Depois de receber os resultados de topologia, o Federation Framework chama o Mecanismo de Mapeamento apropriado com o relacionamento virtual atual e solicita os dados de reconciliação. O parâmetro reconciliationObjects está vazio nessa etapa, ou seja, nenhuma condição é adicionada aos dados de reconciliação nessa chamada. Os dados de reconciliação retornados definem quais dados são necessários para corresponder os ECs de reconciliação no UCMDB ao repositório de dados externo. Os dados de reconciliação podem ser de um destes três tipos: ➤ **IdReconciliationData.** Os ECs são reconciliados de acordo com seu ID. ➤ **PropertyReconciliationData.** Os ECs são reconciliados de acordo com as propriedades de um dos ECs. ➤ **TopologyReconciliationData.** Os ECs são reconciliados de acordo com a topologia (por exemplo, para reconciliar ECs de nó, o endereço IP de **IP** é necessário também).
5	O Federation Framework solicita objetos de reconciliação para os ECs que foram recebidos na etapa 3 do repositório de dados externo. O Federation Framework chama o método **getTopologyWithReconciliationData()** no Adaptador Externo, em que a topologia solicitada é uma topologia de um nó com ECs recebidos na etapa 3 como sendo os dados de reconciliação e condição de ID da etapa 4.

Número	Explicação
6	O Federation Framework chama o Mecanismo de Mapeamento para recuperar os dados de reconciliação. Nesse estado (em contraste com a etapa 3), o Mecanismo de Mapeamento recebe os objetos de reconciliação da etapa 5 como parâmetros. O Mecanismo de Mapeamento traduz o objeto de reconciliação recebido para a condição nos dados de reconciliação.
7	O Federation Framework solicita a topologia do sub-TQL com dados de reconciliação da 6 do UCMDB.
8	O Federation Framework chama o Mecanismo de Mapeamento para se conectar entre os resultados recebidos. O parâmetro firstResult é o resultado de topologia externa recebido do Adaptador Externo na etapa 5 e o parâmetro secondResult é o resultado de topologia externa do UCMDB na etapa 7. O Mecanismo de Mapeamento retorna um mapa em que a ID de EC Externo do primeiro repositório de dados (o repositório de dados externo, nesse caso) é mapeado para os IDs de EC Externo do segundo repositório de dados (UCMDB).
9	Para cada mapeamento, o Mecanismo de Mapeamento cria um relacionamento virtual.
10	Depois do cálculo dos resultados da consulta TQL federada (somente na etapa de topologia), o Federation Framework recupera o layout TQL original para os ECs e relacionamentos resultantes dos repositórios de dados apropriados.

Exemplo do fluxo do Federation Framework para consultas TQL federadas

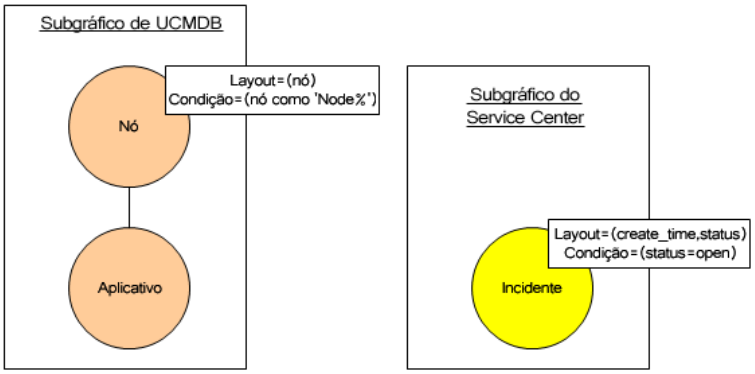
Este exemplo explica como visualizar todos os incidentes abertos em nós específicos. O repositório de dados do ServiceCenter é o repositório de dados externo. As instâncias de nó são armazenadas no UCMDB, enquanto as instâncias de incidente são armazenadas no ServiceCenter. Pressupõe-se que, para conectar as instâncias de incidente ao nó apropriado, as propriedades node e ip_address do host e do IP são necessárias. Essas são propriedades de reconciliação que identificam os nós do ServiceCenter no UCMDB.



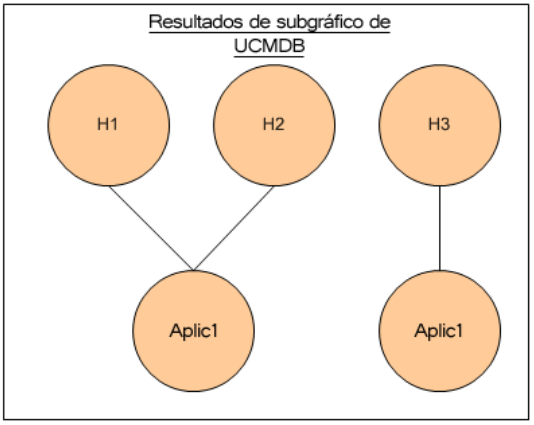
Observação: Para federação de atributos, o método **getTopology** do adaptador é chamado. Os dados de reconciliação são adaptados no TQL do usuário (nesse caso, o elemento de EC).

---

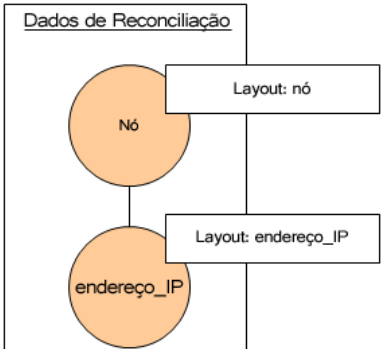
1 Depois de analisar o adaptador, o Federation Framework reconhece o relacionamento virtual entre Node e Incident e divide a consulta TQL federada em dois subgráficos.



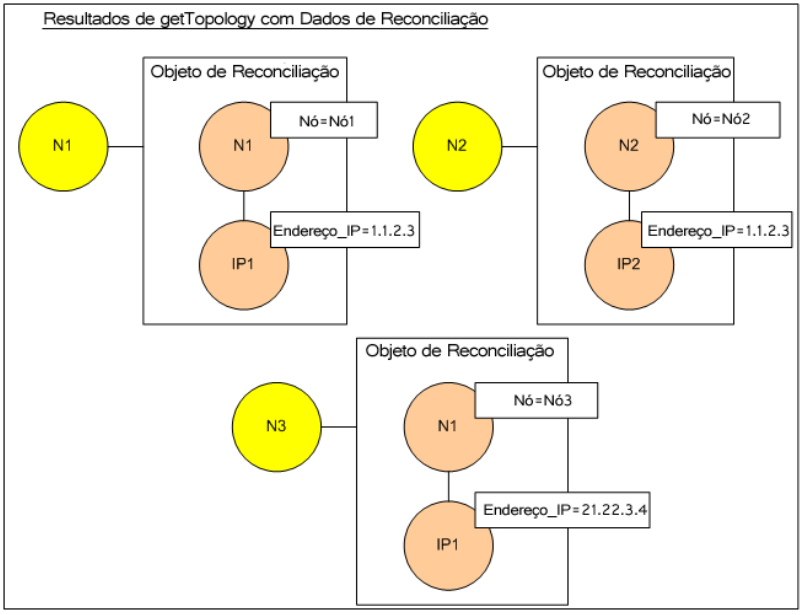
2 O Federation Framework executa o subgráfico do UCMDB para solicitar a topologia e recebe os seguintes resultados:



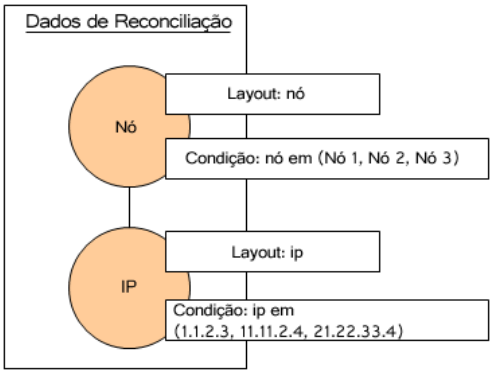
3 O Federation Framework solicita, no Mecanismo de Mapeamento, os dados de reconciliação para o primeiro repositório de dados (UCMDB) que contém as informações para se conectar entre os dados recebidos dos dois repositórios de dados. Os dados de reconciliação nesse caso são:



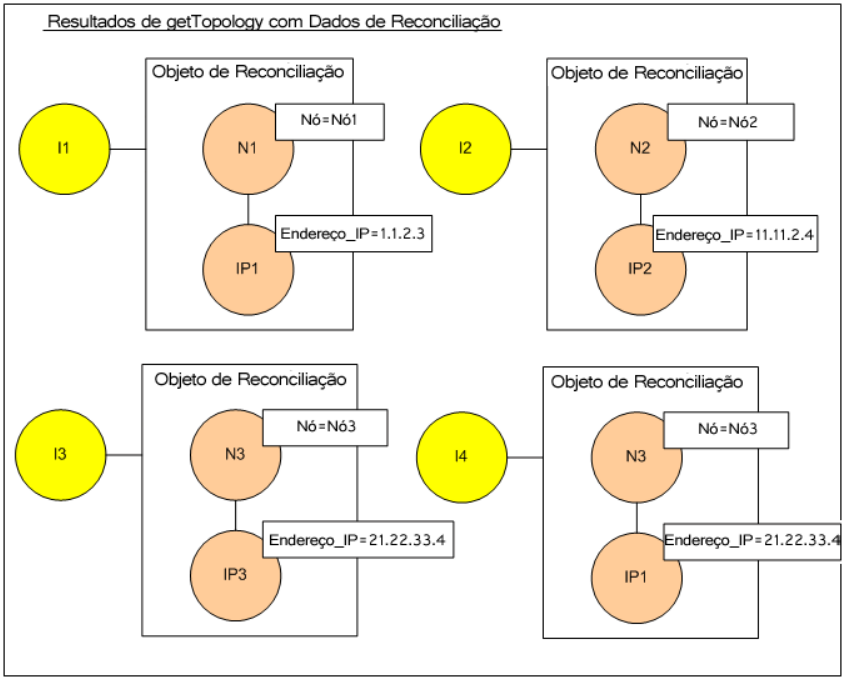
4 O Federation Framework cria uma consulta de topologia de um nó com as condições de Nó e ID nela do resultado anterior (node em H1, H2, H3) e executa essa consulta com os dados de reconciliação necessários no UCMDB. O resultado inclui ECs de Nó que são relevantes para a condição de ID e o objeto de reconciliação apropriado para cada EC:



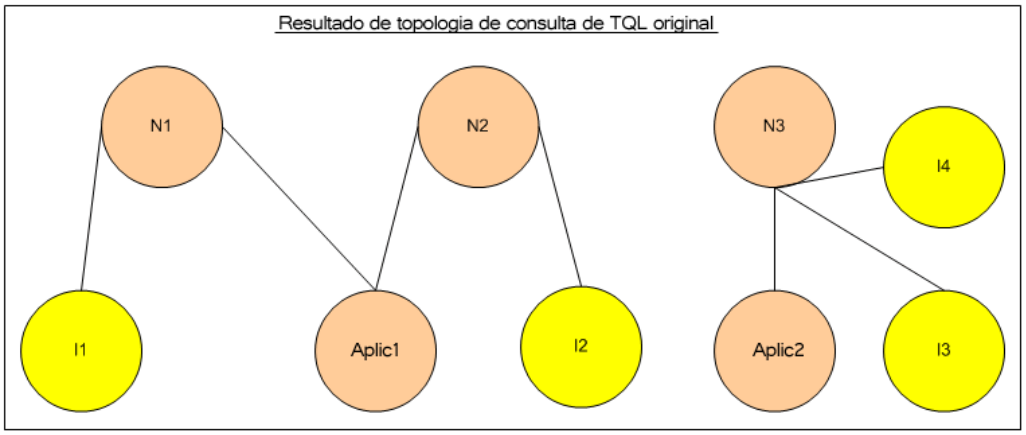
5 Os dados de reconciliação para o ServiceCenter devem conter uma condição para node e ip que é derivada dos objetos de reconciliação recebidos do UCMDB:



6 O Federation Framework executa o subgráfico do ServiceCenter com os dados de reconciliação para solicitar a topologia e os objetos de reconciliação apropriados e recebe os seguintes resultados:



7 O resultado após a conexão no Mecanismo de Mapeamento e a criação de relacionamentos virtuais é:



8 O Federation Framework solicita o layout de TQL original para instâncias recebidas do UCMDB e ServiceCenter.

Fluxo do Federation Framework para população

Esta seção inclui os seguintes tópicos:

➤ "Definições e termos" na página 240

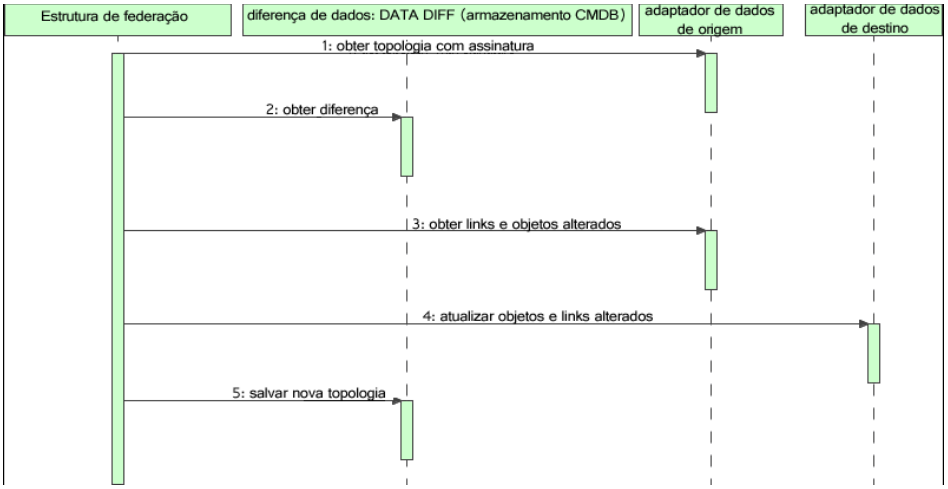
➤ "Diagrama de fluxo" na página 241

Definições e termos

Assinatura. Indica o estado das propriedades no EC. Se alterações forem feitas nos valores de propriedade em um EC, também será necessário alterar a assinatura de EC. A assinatura de EC ajuda a detectar se um EC foi alterado sem recuperar e comparar todas as propriedades de EC. Tanto o EC quanto a assinatura de EC são fornecidos pelo adaptador apropriado. O adaptador é responsável por alterar a assinatura de EC quando as propriedades de EC são alteradas.

Diagrama de fluxo

O diagrama de sequência a seguir ilustra a interação entre o Federation Framework e os adaptadores de origem e de destino em um fluxo de população:



1 O Federation Framework recebe a topologia do resultado da consulta proveniente do adaptador de origem. O adaptador reconhece a consulta por seu nome e a executa no repositório de dados externo. O resultado de topologia contém a ID e a assinatura para cada EC e relacionamento no resultado. A ID é a ID lógica que define o EC como exclusivo no repositório de dados externo. A assinatura deverá ser modificada se o EC ou relacionamento for modificado.

2 O Federation Framework usa assinaturas para comparar os resultados de consulta de topologia recém-recebidos com os resultados salvos, e para determinar quais ECs foram alterados.

3 Depois que o Federation Framework localiza os ECs e relacionamentos que foram alterados, ele chamará o adaptador de origem com os IDs dos ECs e relacionamentos alterados como um parâmetro para recuperar o layout completo.

4 O Federation Framework envia a atualização ao adaptador de destino. O adaptador de destino atualiza a origem de dados externos com os dados recebidos.

5 Após a atualização, o Federation Framework salva o último resultado da consulta.

Interfaces de adaptador

Esta seção inclui os seguintes tópicos:

Definições e termos

O relacionamento externo. O relacionamento entre dois tipos de EC externos aceitos pelo mesmo adaptador.

Interfaces de adaptador para consultas TQL federadas

Use a interface de adaptador apropriada para cada adaptador, da seguinte maneira.

Uma **one Node topology interface** é usada quando o adaptador não oferece suporte a nenhum relacionamento externos, ou seja, o adaptador nunca está para receber uma solicitação com mais de um EC externo. Todas as OneNode são criadas para simplificar fluxo de trabalho; para esses casos em que é necessário usar uma consulta mais extensiva, use a interface DataAdapter.

Rejeitado a partir do UCMDB 9.00: Interface de topologia de padrões

Uma **interface DataAdapter** é usada para definir adaptadores que oferecem suporte a consultas federadas complexas. A solicitação de reconciliação nesses adaptadores faz parte do único parâmetro QueryDefinition. Esses adaptadores também podem ser usados para População.

Interfaces OneNode

As interfaces a seguir têm diferentes tipos de dados de reconciliação:

- **OneNodeTopologyIdReconciliationDataAdapter.** Use se o adaptador oferecer suporte a um **TQL de nó único** e a reconciliação entre repositórios de dados for calculada pelo ID.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Use se o adaptador oferecer suporte a um **TQL de nó único** e a reconciliação entre repositórios de dados for executada pelas propriedades de um EC.
- **OneNodeTopologyDataAdapter.** Use se o adaptador oferecer suporte a um **TQL de nó único** e a reconciliação entre repositórios de dados for executada pela topologia.

Interfaces de adaptador de dados

- **DataAdapter.** Use este adaptador para oferecer suporte a consultas TQL federadas complexas. Permite a maior diversidade.
- **PopulateDataAdapter.** Use este adaptador para oferecer suporte a consultas TQL federadas complexas e fluxos de população. Em um fluxo de população, este adaptador recupera todo o conjunto de dados, além de deixar no filtro de sonda a diferença desde a última execução do trabalho.
- **PopulateChangesDataAdapter.** Use este adaptador para oferecer suporte a consultas TQL federadas complexas e fluxos de população. Em um fluxo de população, este adaptador oferece suporte à recuperação apenas das alterações que ocorreram desde a última execução do trabalho.

Interfaces de topologia de padrões (rejeitadas a partir do UCMDB 9.00)

As interfaces a seguir têm diferentes tipos de dados de reconciliação:

- **PatternTopologyIdReconciliationDataAdapter.** Use se o adaptador oferecer suporte a um **TQL complexo** e a reconciliação entre repositórios de dados for executada pelo ID.
- **PatternTopologyPropertyReconciliationDataAdapter.** Use se o adaptador oferecer suporte a um **TQL complexo** e a reconciliação entre repositórios de dados for executada por propriedades de nó único.
- **PatternTopologyDataAdapter.** Use se o adaptador oferecer suporte a um **TQL complexo** e a reconciliação entre repositórios de dados for executada pela topologia.

Interfaces adicionais

- **SortResultDataAdapter.** Use se for possível classificar os ECs resultantes no repositório de dados externo.
- **FunctionalLayoutDataAdapter.** Use se for possível calcular o layout funcional no repositório de dados externo.

Interfaces de adaptador para sincronização

- **SourceDataAdapter.** Use para adaptadores de origem nos fluxos de população.
- **TargetDataAdapter.** Use para adaptadores de destino nos fluxos de push de dados.

Tarefas

Adicionar um adaptador para uma nova fonte de dados externos

Esta tarefa explica como definir um adaptador para oferecer suporta a uma nova fonte de dados externos.

Esta tarefa inclui as seguintes etapas:

- "Pré-requisitos" na página 245
- "Definir relacionamentos válidos para relacionamentos virtuais" na página 246
- "Definir uma configurações de adaptador" na página 247
- "Definir classes aceitas" na página 250
- "Implementar o adaptador" na página 251
- "Definir as regras de reconciliação ou implementar o mecanismo de mapeamento" na página 251
- "Adicionar JARs necessários para implementação no caminho de classe" na página 251
- "Implantar o adaptador" na página 252
- "Atualizar o adaptador" na página 253

1 Pré-requisitos

Classes de adaptador aceitas pelo modelo para ECs e relacionamentos no Modelo de Dados do UCMDB: como desenvolvedor de adaptadores, você deve:

- ter conhecimento da hierarquia dos tipos de EC do UCMDB para entender como os TECs externos estão relacionados aos TECs do UCMDB
- modelar os TECs externos no modelo de classe do UCMDB

- adicionar as definições para novos tipos de EC e seus relacionamentos
- definir relacionamentos válidos no modelo de classe do UCMDB para os relacionamentos válidos entre as classes internas do adaptador. (Os TECs podem ser colocados em qualquer nível da árvore do modelo de classe do UCMDB.)

A modelagem deve ser a mesma independentemente do tipo de federação (em tempo real ou replicação). Para ver detalhes sobre como adicionar novas definições de TEC ao modelo de classe do UCMDB, consulte "Trabalhando com o Seletor de EC" no *Guia de Modelagem do HP Universal CMDB*.

Para que o adaptador ofereça suporte a atributos federados em TECs, adicione esse TEC às classes aceitas com atributos aceitos e à regra de reconciliação para esse TEC.

2 Definir relacionamentos válidos para relacionamentos virtuais

---

Observação: Esta seção é relevante apenas para federação.

Para recuperar TECs federados que estão conectados aos TECs do CMDB locais, é necessário haver uma definição de link válida entre os dois TECs no CMDB.

a Crie um arquivo XML de links válido que contenha esses links (se ainda não existirem).

b Adicione o arquivo XML de links ao pacote de adaptadores na pasta **\validlinks**. Para ver detalhes, consulte "Gerenciador de Pacotes" no *Guia de Administração do HP Universal CMDB*.

Exemplo de uma definição de relacionamento válido:

No exemplo a seguir, o relacionamento de tipo containment entre as instâncias de tipo node com as instâncias de tipo myclass1 é uma definição de relacionamento válido.

```
<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 Definir uma configurações de adaptador

a Navegue até **Gerenciamento do Adaptador** .

b Clique no botão **Criar novo recurso**.

c Na caixa de diálogo Novo adaptador, selecione **Integração** e **Adaptador Java**.

d Clique com o botão direito do mouse no adaptador que você criou e selecione **Editar Origem do Adaptador** no menu de atalho.

e Edite as seguintes marcas 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.BaseDestinationData">
            <!-- verificação -->
            <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>
                <!--<classes com suporte/> <!—consulte a seção sobre classes com suporte-->
            <topology>
                <pattern-topology /> <!—ou <one-node-topology> -->
            </topology>
            </support-federated-query>
            <!--<support-replicatioin-data>
            <source>
                <changes-source/>
            </source>
    <target/>
        </adapter-capabilities>
        <default-mapping-engine />
            <queries />
    <removedAttributes />
        <full-population-days-interval>-1</full-population-days-interval>
    </adapterInfo>
    <inputClass>destination_config</inputClass>
    <protocols />
```

```
<parameters>
    <!--O atributo de descrição pode ser escrito em texto simples ou em HTML.-->
    <!--O atributo de host é tratado como um caso especial pelo UCMDB-->
    <!--e selecionará automaticamente o nome da sonda (se possível)-->
    <!--de acordo com o valor do atributo.-->
    <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 ver detalhes sobre as marcas XML, consulte "Marcas e propriedades de configuração XML" na página 257.

4 Definir classes aceitas

Defina as classes aceitas do código do adaptador implementando o método *getSupportedClasses()* ou usando o arquivo XML de padrão.

```
<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	O nome do tipo de EC
is-derived	Especifica se essa definição inclui todos os filhos herdeiros
is-reconciliation-supported	Especifica se essa classe é usada para reconciliação
is-id-reconciliation-supported	Especifica se essa classe é usada para id-reconciliation
federation-not-supported	Especifica se esse TEC não deve ser permitido para federação (bloqueando determinados TECs, por exemplo, um TEC definido exclusivamente para federação)
<supported-conditions>	Especifica as condições aceitas em cada atributo

5 Implementar o adaptador

Selecione a classe de implementação de adaptadores correta de acordo com seus recursos definidos. A classe de implementação de adaptadores implementa as interfaces apropriadas de acordo com os recursos definidos.

6 Definir as regras de reconciliação ou implementar o mecanismo de mapeamento

Se o adaptador oferecer suporte a consultas TQL federadas, você terá três opções para definir seu Mecanismo de Mapeamento:

- Use o mecanismo de mapeamento padrão do CMDB 9.0x, que usa as regras de reconciliação interna para mapeamento do CMDB. Para usá-lo, deixe a marca XML **<default-mapping-engine/>** vazia.

  Para ver detalhes, consulte "O arquivo reconciliation_types.txt" na página 192.

- Use o mecanismo de mapeamento do CMDB 8.0x. Para fazer isso, use a seguinte Marca XML:
  <default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>

  Para ver detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 192.

- Escreva seu próprio mecanismo de mapeamento implementando a interface do mecanismo de mapeamento e colocando o JAR com o restante do código do adaptador. Para fazer isso, use a seguinte marca XML:
  <default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>

7 Adicionar JARs necessários para implementação no caminho de classe

Para implementar suas classes, adicione o arquivo **federation_api.jar** ao caminho de classe do editor de código.

8 Implantar o adaptador

a Implante o pacote de adaptadores. Para ver detalhes gerais sobre como implantar um pacote, consulte "Gerenciador de Pacotes" no *Guia de Administração do HP Universal CMDB*.

O pacote deve conter as seguintes entidades:

- Nova definição de TEC (opcional):

  Usada somente se o adaptador oferecer suporte a novos tipos de EC que ainda não existem no UCMDB.

  As novas definições de TEC estão localizadas na pasta class do pacote.

- Nova definição de tipo de dados (opcional):

  Usada somente se os novos TECs exigirem novos tipos de dados.

  As novas definições de tipo de dados estão localizadas na pasta typedef do pacote.

- Nova definição de relacionamentos válidos (opcional):

  Usada somente se o adaptador oferecer suporte a consultas TQL federadas.

  As novas definições de relacionamentos válidos estão localizadas na pasta validlinks do pacote.

- O arquivo XML de configuração de padrões deve estar localizado na pasta discoveryPatterns do pacote.

- **Descriptor.** Determina as definições do pacote.

- Coloque suas classes compiladas (normalmente um arquivo jar) no pacote na pasta **adapterCode\<id_do_adaptador>**.

Observação: O nome da pasta id_do_adaptador tem o mesmo valor da configuração do adaptador.

- Se você criar seu próprio arquivo de configuração, deverá colocá-lo no pacote na pasta **adapterCode\<id_do_adaptador>**.

9 Atualizar o adaptador

Alterações em qualquer um dos arquivos não-binários do adaptador podem ser feitas no módulo Gerenciamento do Adaptador. Fazer alterações em arquivos de configuração no módulo Gerenciamento do Adaptador faz com que o adaptador seja recarregado com as novas configurações.

As atualizações também podem ser feitas editando os arquivos no pacote (tanto arquivos binários quanto não-binários) e reimplantando o pacote usando o Gerenciador de Pacotes. Para ver detalhes, consulte "Implantar um pacote" no *Guia de Administração do HP Universal CMDB*.

Implementar o mecanismo de mapeamento

A configuração do mecanismo de mapeamento depende de qual mecanismo de mapeamento você está usando.

Esta tarefa inclui as seguintes etapas:

- "Configure o arquivo reconciliation_types.txt (para o mecanismo de mapeamento padrão do UCMDB 9.0x)" na página 253
- "Configure o arquivo reconciliation_rules.txt (para o mecanismo de mapeamento do UCMDB 8.0x)" na página 254

1 Configure o arquivo reconciliation_types.txt (para o mecanismo de mapeamento padrão do UCMDB 9.0x)

O arquivo é usado para definir quais tipos de EC são usados para reconciliação no adaptador.

Escreva cada tipo de EC usado para reconciliação em uma única linha, da seguinte maneira:

```
nó
business_application
```

Coloque o arquivo no pacote de adaptadores na pasta **adapterCode\<ID_do_adaptador>\META-INF**.

2 Configure o arquivo reconciliation_rules.txt (para o mecanismo de mapeamento do UCMDB 8.0x)

Este arquivo é usado para configurar as regras de reconciliação. Cada linha no arquivo representa uma regra. Por exemplo:

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

O parâmetro **reconcilition_type** é preenchido com o tipo de EC no qual a reconciliação é executada (o nome de classeUCMDB que está conectado à classe federada no TQL).

O parâmetro **expression** é a lógica que decide se dois objetos de reconciliação são iguais (um objeto de reconciliação do lado do UCMDB e o outro do lado do adaptador Federado).

A expressão consiste em ORs e ANDs.

A convenção em relação a nomes de atributos na parte da expressão é **[nome_de_classe].[nome_de_atributo]**. Por exemplo, o atributo **ip_address** na classe **ip** é escrito **ip.ip_address**.

Você pode definir correspondências ordenadas. A correspondência ordenada verifica a primeira subexpressão OR. Se dois objetos de reconciliação tiverem o valor nos atributos da subexpressão e for retornado falso (os objetos de reconciliação não são iguais), isso significará que a segunda subexpressão OR não é comparada.

Para ver uma correspondência ordenada, use **ordered expression** em vez de **expression**.

O sinal se circunflexo (^) é usado para não diferenciar maiúsculas de minúsculas durante as comparações.

Os outros parâmetros (**end1_type**, **end2_type** e **link_type**) são usados somente se os dados de reconciliação contêm dois nós e não só o nó do tipo de reconciliação (os dados de reconciliação de topologia). Nesse caso, os dados de reconciliação são **end1_type -(link_type)> end2_type**.

Não é necessário adicionar o layout relevante porque ele é recuperado da expressão.

Para executar a reconciliação por ID do UCMDB, use **cmdb_id** como nome do atributo na expressão.

Coloque o arquivo no pacote de adaptadores na pasta **adapterCode**\<**ID_do_adaptador**>**META-INF**\.

Exemplos:

- Você pode adicionar uma regra de reconciliação somente para um TEC de nó. Isso ocorre porque somente TECs de nó têm relacionamentos válidos com TECs externos. Por exemplo, um EC de nó no CMDB tem correspondência com um EC de nó no ServiceCenter através do atributo node.name ou do atributo ip_address.name.
- A regra de reconciliação nesse caso é uma regra de topologia e a expressão é ordenada. A regra executa as seguintes verificações nos ECs em comparação:
  - Se o atributo node.name for igual, a regra fará correspondência dos nós.
  - Se o atributo node.name não for igual, a regra não fará correspondência dos nós.
  - Se o atributo node.name for nulo em um dos ECs comparados, a regra verificará o atributo ip_address.name. Se o atributo ip_address.name for igual, a regra fará correspondência dos nós.

Criar um adaptador de amostra

Este exemplo ilustra como criar um adaptador de exemplo.

Esta tarefa inclui as seguintes etapas:

- "Selecionar lógica do adaptador" na página 256
- "Carregar o projeto" na página 256

1 Selecionar lógica do adaptador

Ao implementar um adaptador, você precisa escolher como manipular a lógica de condição na implementação (condições de propriedade, condições de ID, condições de reconciliação e condições de link).

a Recupere todos os dados na memória do adaptador e deixe-o selecionar ou filtrar as Instâncias de EC necessárias.

b Converta todas as condições na linguagem de fonte de dados e deixe-a filtrar e selecionar os dados. Por exemplo:

- ➤ Converta a condição em uma consulta SQL.
- ➤ Converta a condição em um objeto de filtro de API Java.

c O meio termo é filtrar alguns dos dados no serviço remoto e fazer com que o adaptador selecione e filtre o restante.

No exemplo MyAdapter, a lógica na etapa a é usada.

2 Carregar o projeto

Copie os arquivos da pasta **C:\hp\UCMDB\UCMDBServer\tools\adapter-dev-kit\SampleAdapters** e siga as instruções nos arquivos leiame.

Observação: Se você usar um adaptador com conjuntos de dados extensos, talvez precisar usa armazenamento em cache e indexação para melhorar o desempenho para Federação.

A documentação de javadocs online está disponível em:

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

Referência

Marcas e propriedades de configuração XML

id="newAdapterIdName"		Define o nome real do adaptador. É usado para logs e pesquisas de pasta.
displayName="New Adapter Display Name"		Define o nome de exibição do adaptador, conforme aparece na UI.
<className>…</className>		Define a interface do adaptador que implementa a classe Java.
<category >My Category</ category>		Define a categoria do adaptador.
<parameters>		Define as propriedades para configuração disponíveis na UI ao configurar um novo ponto de integração.
	name	O nome da propriedade (usado principalmente pelo código).
	description	A dica de exibição da propriedade.
	type	Cadeia de caracteres ou inteiro (use valores válidos com cadeia de caracteres para booliano).
	display-name	O nome da propriedade na UI.
	mandatory	Especifica se essa propriedade de configuração é obrigatória para o usuário.
	order-index	A ordem de colocação da propriedade (small = up).
	valid-values	Uma lista de possíveis valores válidos separados por caracteres ‘;’ (por exemplo, valid-values="Oracle;SQLServer;MySQL” ou valid-values=”True;False”).
<adapterInfo>		Contém a definição das configurações estáticas e recursos do adaptador.
	<support-federated-query>	Define esse adaptador como capaz de federação.
	<one-node-topology>	A capacidade de consultas federadas com um nó de consulta federado.

	<pattern-topology>	A capacidade de federar consultas complexas.
	<support-replicatioin-data>	Define o recurso para executar fluxos de push de dados e de população.
	<source>	Esse adaptador também pode ser usado para fluxos de população.
	<changes-source/>	Esse adaptador pode ser usado para fluxos de alterações de população.
	<target>	Esse adaptador pode ser usado para fluxos de push de dados.
	<default-mapping-engine>	Permite a definição de um mecanismo de mapeamento para o adaptador (por padrão, o adaptador usa o mecanismo de mapeamento padrão). Para qualquer outro mecanismo de mapeamento, insira o nome de classe implementador do mecanismo de mapeamento (para o mecanismo de mapeamento do UCMDB 8.0x, use: com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine)
	<removedAttributes>	Obriga a remoção de atributos específicos do resultado.
	<full-population-days-interval>	Especifica quando executar um trabalho de população completo em vez de um trabalho diferencial (a cada 'x' dias). Usa o mecanismo de envelhecimento juntamente com o fluxo de alterações.

7

Desenvolvendo adaptadores push

Este capítulo inclui:

Conceitos

Tarefas

Referência

Conceitos

Visão geral do desenvolvimento de adaptadores push

O Adaptador Push Genérico fornece uma plataforma que permite desenvolver rapidamente integrações que enviam por push dados do UCMDB 9.0x para repositórios de dados externos (bancos de dados e aplicativos de terceiros). O desenvolvimento de uma integração personalizada baseada no Adaptador Push Genérico exige:

- um arquivo XML para o mapeamento entre os tipos de vínculo de EC do UCMDB e itens de dados externos.
- um script Jython para enviar por push itens de dados ao repositório de dados externo.

Sincronização diferencial

DiscoveryMain no script Jython no qual o adaptador push foi baseado retornar uma instância de **OSHVResult** vazia, o adaptador não dá suporte à sincronização diferencial. Isso significa que mesmo que um trabalho de sincronização diferencial seja executado, na realidade uma sincronização completa será realizada. Portanto, nenhum dado poderá ser atualizado ou removido do sistema remoto, uma vez que todos os dados serão adicionados ao CMDB durante cada sincronização.

Para que o adaptador push permita a sincronização diferencial, a função **DiscoveryMain** deve retornar um objeto que implemente a interface **DataPushResults**, que contém os mapeamentos entre as IDs que o script Jython recebe do arquivo XML e as IDs que o script Jython cria na máquina remota. Essas últimas são IDs do tipo Externalid.

Tarefas

Preparar os arquivos de mapeamento

Há dois modos diferentes de preparar arquivos de mapeamento:

- Você pode preparar um arquivo de mapeamento único e global.

  Todos os mapeamentos são colocados em um único arquivo nomeado **mappings.xml**.

- Você pode preparar um arquivo separado para cada consulta push.

  Esses arquivos de mapeamento são nomeados com o formato **<nome da consulta>.xml**.

Para ver detalhes, consulte "Esquema do arquivo de mapeamento" na página 269.

Esta tarefa inclui as seguintes etapas:

- "Criar arquivo de mapeamento" na página 262
- "Mapear ECs" na página 262
- "Mapear vínculos" na página 263

1 Criar arquivo de mapeamento

A estrutura do arquivo de mapeamento é a seguinte:

```
<?xml version="1.0" encoding="UTF-8"?>
<integration>
    <info>
        <source name="UCMDB"  versions="9.x" vendor="HP" />
        <!-- exemplo: -->
        <target name="Oracle" versions="11g" vendor="Oracle" />
    </info>
    <targetcis>
        <!--- Mapeamentos de EC --->
    </targetcis>
    <targetrelations>
        <!--- Mapeamentos de vínculo --->
    </ targetrelations>
</integration>
```

2 Mapear ECs

Cada TEC do CMDB é mapeado de acordo com o exemplo a seguir:

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

Observação: Os possíveis valores de mode dependem da implementação do script.

---

3 Mapear vínculos

Cada vínculo válido é mapeado de acordo com o exemplo a seguir:

```
<link source_link_type="dependency" target_link_type="dependency"
mode="update_else_insert" source_ci_type_end1="webservice"
source_ci_type_end2="sap_gateway">
    <target_ci_type_end1 name="webservice" />
    <target_ci_type_end2 name="sap_gateway" />
</link>
```

Escrever scripts Jython

O script de mapeamento é um script Jython comum, e deve seguir as regras aplicáveis a scripts Jython. Para ver detalhes, consulte "Desenvolvendo adaptadores Jython" na página 63.

O script deve conter a função **DiscoveryMain**, que pode retornar uma instância de **OSHVResult** vazia ou uma de **DataPushResults** ao ser bem-sucedido.

Para relatar falhas, o script deve retornar uma exceção, por exemplo:

```
raise Exception('Falha ao inserir no UCMDB remoto usando TopologyUpdateService.
Consulte o log do UCMDB remoto')
```

Na função **`DiscoveryMain`**, os itens de dados aos quais enviar por push ou que devem ser excluídos do aplicativo externo podem ser obtidos da seguinte forma:

```
# obter objetos de resultado de adicionar/atualizar/excluir (em formato XML) da
metodologia (Framework)
addResult = Framework.getTriggerCIData('addResult')
updateResult = Framework.getTriggerCIData('updateResult')
deleteResult = Framework.getTriggerCIData('deleteResult')
```

O objeto-cliente do aplicativo externo pode ser obtido do seguinte modo:

```
oracleClient = Framework.createClient()
```

Esse objeto-cliente usa automaticamente a ID, nome de host e número da porta de credenciais passadas pelo adaptador pela metodologia.

Se você precisar usar os parâmetros de conexão que definiu para o adaptador (para obter detalhes, consulte etapa 2 em"Criar um pacote de adaptador" na página 268), use o seguinte código:

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

Por exemplo:

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

Esta seção inclui também:

- "Trabalhando com resultados do mapeamento" na página 264
- "Lidando com a conexão de teste no script" na página 265

Trabalhando com resultados do mapeamento

O Adaptador Push Genérico cria cadeias XML que descrevem os dados a serem adicionados, atualizados ou excluídos do sistema de destino. O script Jython precisa analisar esse XML para, em seguida, realizar a adição, atualização ou exclusão no destino.

No XML da operação de adição que o script Jython recebe, o atributo mamId dos objetos e vínculos é sempre o identificador do UCMDB do objeto ou vínculo original antes que seu tipo, atributo ou outra informação tenha sido alterada no esquema do sistema remoto.

No XML das operações de atualização e remoção, o atributo mamId de cada objeto ou vínculo contém a representação da cadeia do mesmo ExternalId que foi retornado pelo script Jython na sincronização anterior.

Exemplo do resultado do XML

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

Lidando com a conexão de teste no script

Um script Jython pode ser chamado para testar a conexão a um aplicativo externo. Nesse caso, o atributo testConnection do destino retornará true. Esse atributo pode ser obtido da metodologia do seguinte modo:

```
testConnection = Framework.getTriggerCIData('testConnection')
```

Quando executado no modo de conexão de teste, um script deve resultar numa exceção quando a conexão ao aplicativo externo não pode ser estabelecida. Do contrário, se a conexão for bem-sucedida, a função **DiscoveryMain** deverá retornar um **OSHVResult** vazio.

Suporte à sincronização diferencial

Importante: Se você estiver implementando a sincronização diferencial em um adaptador existente criado na versão 9.00 ou 9.01, deverá usar o arquivo push-adapter.zip da versão 9.02 ou posterior para recriar seu pacote de adaptador. Para ver detalhes, consulte "Criar um pacote de adaptador" na página 268.

Essa tarefa permite que o adaptador push realize a sincronização diferencial. Para ver detalhes, consulte "Sincronização diferencial" na página 260.

O script Jython retorna o objeto **DataPushResults** que contém dois mapas Java - um para mapeamentos de ID de objeto (chaves e valores são objetos do tipo ExternalCiId) e um para IDs de vínculos (chaves e valores são objetos do tipo ExternalRelationId).

➤ Adicione as seguintes declarações **from** ao seu script Jython:

```
from com.hp.ucmdb.federationspi.data.query.types import ExternalIdFactory
from com.hp.ucmdb.adapters.push import DataPushResults
from com.hp.ucmdb.adapters.push import DataPushResultsFactory
from com.mercury.topaz.cmdb.server.fcmdb.spi.data.query.types import
ExternalIdUtil
```

➤ Use a classe de fábrica **DataPushResultsFactory** para obter o objeto **DataPushResults** da função **DiscoveryMain**.

```
# Criar o objeto UpdateResult
updateResult = DataPushResultsFactory.createDataPushResults(objectMappings,
linkMappings);
```

➤ Use os comandos a seguir para criar mapas Java para o objeto **DataPushResults**:

```
# Preparar os mapas para armazenar mapeamento se IDs
objectMappings = HashMap()
linkMappings = HashMap()
```

- Use a classe **ExternalIdFactory** para criar as seguintes IDs ExternalId:
  - ExternalId para objetos ou vínculos originários de um CMDB (por exemplo, todos os ECs em uma operação de adição são do CMDB):

```
externaCIIId = ExternalIdFactory.createExternalCmdbCiId(ciType, ciIDAsString)
externalRelationId = ExternalIdFactory.createExternalCmdbRelationId(linkType,
end1ExternalCIId, end2ExternalCIId, linkIDAsString)
```

  - ExternalId para objetos ou vínculos que não são originários de um CMDB (por exemplo, todas as operações de atualização e remoção contêm tais objetos):

```
myIDField = TypesFactory.createProperty("systemID", "1")
myExternalId = ExternalIdFactory.createExternalCiId(type, myIDField)
```

Observação: Se o script Jython atualizou informações existentes e a ID do objeto (ou vínculo) for alterada, você deverá retornar um mapeamento entre a ID externa anterior e a nova.

- Use os métodos **restoreCmdbCiIDString** ou **restoreCmdbRelationIDString** da classe **ExternalIdFactory** para recuperar a cadeia de ID do UCMDB de uma ID externa de um objeto ou vínculo originário do UCMDB.
- Use os métodos **restoreExternalCiId** e **restoreExternalRelationId** da classe **ExternalIdUtil** para restaurar o objeto **ExternalId** do valor do atributo mamId do XML das operações de atualização ou remoção.

Observação: Objetos ExternalId são, de fato, uma matriz de propriedades. Isso significa que você pode usar um objeto ExternalId para armazenar quaisquer informações de que possa precisar e que identifiquem os dados no sistema remoto.

Criar um pacote de adaptador

1. Extraia o conteúdo do arquivo **C:\hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip** em uma pasta temporária.
2. Edite o arquivo **discoveryPatterns\push_adapter.xml**.
   a. Modifique a marca <pattern> usando uma nova id e rótulo de exibição. Substitua:

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

   por

   ```
   <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. Atualize a lista de parâmetros para que reflita os atributos de conexão necessários. Não remova o atributo **probeName**.
3. Renomeie a pasta **adapterCode\PushAdapter** com a ID de adaptador usada na etapa 2 (por exemplo, **adapterCode\MyPushAdapter**).
4. Substitua **discoveryScripts\pushScript.py** pelo script que escreveu (para obter detalhes, consulte "Escrever scripts Jython" na página 263). Se você renomear o script, a propriedade jythonScript.name em **adapterCode\<adapter ID>\push.properties** deverá ser atualizada de acordo.
5. Substitua o arquivo **adapterCode\<adapter ID>\mappings\mappings.xml** pelos arquivos de mapeamento que preparou (para obter detalhes, consulte "Preparar os arquivos de mapeamento" na página 261.

   Se quiser usar um arquivo de mapeamento para cada método TQL,atribua o nome do TQL correspondente a cada arquivo XML, seguido por **.xml**. Nesse caso, o arquivo **mappings.xml** será usado como padrão, caso nenhum arquivo de mapeamento específico seja encontrado para o nome do TQL atual. O nome do arquivo de mapeamento padrão pode ser modificado alterando-se a propriedade mappingFile.default em **adapterCode\<adapter ID>\push.properties**.

Referência

Esquema do arquivo de mapeamento

Elemento		Atributos
Nome e caminho	**Descrição**	
integration	Define o conteúdo de mapeamento do arquivo. Deve ser o bloco mais periférico no arquivo, exceto pela linha de início e quaisquer comentários.	
info (integration)	Define informações sobre os repositórios de dados sendo integrados	
source (integration > info)	Define informações sobre o repositório de dados de origem	**Nome**. type **Descrição.** Nome do repositório de dados de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. versions **Descrição.** Versão(ões) do repositório de dados de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. vendor **Descrição.** Fornecedor do repositório de dados de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
target (integration > info)	Define informações sobre o repositório de dados de destino	**Nome.** type **Descrição.** Nome do repositório de dados de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. versions **Descrição.** Versão(ões) do repositório de dados de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. vendor **Descrição.** Fornecedor do repositório de dados de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia
targetcis (integration)	Elemento de contêiner de todos os mapeamentos de TEC.	

Elemento		Atributos
Nome e caminho	**Descrição**	
source_ci_type (integration > targetcis)	Define um TEC de origem	**Nome.** name **Descrição.** Nome do TEC de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. mode **Descrição.** Tipo da atualização necessária ao tipo de EC atual. **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: ➤ **insert** – Use-a apenas se o EC ainda não existir. ➤ **update** – Use-a apenas se o EC já existir. ➤ **update_else_insert** – Se o EC existir, será atualizado; caso contrário, um novo EC será criado. ➤ **ignore** – Não faz nada com esse tipo de EC.

Elemento		Atributos
Nome e caminho	**Descrição**	
target_ci_type (integration > targetcis > source_ci_type)	Define um TEC de destino	**Nome.** name **Descrição.** Nome do tipo de EC de destino. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. schema **Descrição.** Nome do esquema que será usado para armazenar esse tipo de EC no destino. **É obrigatório.** Não é obrigatório **Tipo.** Cadeia
		Nome. namespace **Descrição.** Indica o namespace desse tipo de EC no destino **É obrigatório.** Não é obrigatório **Tipo.** Cadeia
targetprimarykey (integration > targetcis > source_ci_type -OU- integration > targetrelations > link)	Identifica atributos de chave primária do TEC	
pkey (integration > targetcis > source_ci_type > targetprimarykey -OU- integration > targetrelations > link > targetprimarykey)	Identifica um atributo de chave primária Obrigatório somente se o modo for **update** ou **insert_else_update**	

Elemento		Atributos
Nome e caminho	**Descrição**	
target_attribute (integration > targetcis > source_ci_type -OU- integration > targetrelations > link)	Define o atributo do TEC de destino	**Nome.** name **Descrição.** Nome do atributo do TEC de destino. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. datatype **Descrição.** Tipo de dado do atributo do TEC de destino. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. length **Descrição.** Para tipos de dados cadeia/ caractere, esse é o tamanho inteiro do atributo de destino. **É obrigatório.** Não é obrigatório **Tipo.** Inteiro
		Nome. option **Descrição.** A função de conversão a ser aplicada ao valor. **É obrigatório.** Falso **Tipo.** Uma das seguintes cadeias: ➤ **uppercase** – Converte para maiúsculas ➤ **lowercase** – Converte para minúsculas ➤ Se esse atributo estiver vazio, nenhuma função de conversão será aplicada.

Elemento		Atributos
Nome e caminho	**Descrição**	
map (integration > targetcis > source_ci_type > target_attribute -OU- integration > targetrelations > link > target_attribute)	Especifica como obter o valor do atributo de TEC de origem	**Nome.** type **Descrição.** O tipo de mapeamento entre os valores de origem e destino. **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: ➤ **direct** – Especifica um mapeamento de 1 para 1 entre o valor do atributo de origem e o valor do atributo de destino ➤ **compoundstring –** Subelementos são unidos em uma única cadeia e o valor do atributo de destino é definido ➤ **childattr** – Subelementos são um ou mais atributos de um TEC filho. Um TEC filho é aquele com um relacionamento do tipo **container_f** ou **contained** ➤ **constant** – Cadeia estática
		Nome. value **Descrição.** Cadeia constante para tipo=**constant** **É obrigatório.** Necessário apenas quando tipo=**constant** **Tipo.** Cadeia
		Nome. attr **Descrição.** Nome de atributo de origem para tipo=**direct** **É obrigatório.** Necessário apenas quando tipo=**direct** **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
aggregation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetrelations > link > target_attribute > map Válido apenas quando o tipo do mapa é **childattr**)	Especifica como os valores de atributo de EC filho do EC de origem são combinados em um único valor para o mapeamento ao atributo de EC de destino. Opcional.	**Nome.** type **Descrição.** O tipo da função de agregação **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: ➤ **csv** – Agrega todos os valores em uma lista separada por vírgula (numérico ou cadeia/caractere) ➤ **count** – Retorna o total numérico de todos os valores incluídos. ➤ **sum** – Retorna o total numérico de todos os valores incluídos. ➤ **average** – Retorna a média numérica de todos os valores incluídos. ➤ **min** – Retorna o menor valor numérico/ caractere dentre os valores incluídos. ➤ **max** – Retorna o maior valor numérico/ caractere dentre os valores incluídos.

Elemento		Atributos
Nome e caminho	**Descrição**	
validation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetrelations > link > target_attribute > map Válido apenas quando o tipo do mapa é **childatt**)	Permite a filtragem para exclusão de ECs filhos do EC de origem com base em valores de atributo. Usado com o subelemento aggregation para alcançar a granularidade que demonstre exatamente que atributos de filhos serão mapeados ao valor de atributo do TEC de destino. Opcional.	**Nome.** minlength **Descrição.** Exclui cadeias mais curtas do que o valor determinado. **Obrigatório.** Não é obrigatório **Tipo.** Inteiro
		Nome. maxlength **Descrição.** Exclui cadeias mais longas do que o valor determinado. **Obrigatório.** Não é obrigatório **Tipo.** Inteiro
		Nome. minvalue **Descrição.** Exclui números menores do que o valor determinado. **Obrigatório.** Não é obrigatório **Tipo.** Numérico
		Nome. maxvalue **Descrição.** Exclui números maiores do que o valor determinado. **Obrigatório.** Não é obrigatório **Tipo.** Numérico
targetrelations (integration)	Elemento de contêiner de todos os mapeamentos de relacionamento. Opcional.	

Elemento		Atributos
Nome e caminho	**Descrição**	
link (integration > targetrelations)	Faz o mapeamento entre um relacionamento de origem e um relacionamento de destino. Obrigatório apenas se **targetrelation** estiver presente.	**Nome.** source_link_type **Descrição.** Nome do relacionamento de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. target_link_type **Descrição.** Nome do relacionamento de destino. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. nameSpace **Descrição.** O namespace do vínculo que será criado com o destino. **É obrigatório.** Não é obrigatório **Tipo.** Cadeia
		Nome. mode **Descrição.** Tipo da atualização necessária para o vínculo atual. **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: ➤ **insert** – Use-a apenas se o EC ainda não existir. ➤ **update** – Use-a apenas se o EC já existir. ➤ **update_else_insert** – Se o EC existir, será atualizado; caso contrário, um novo EC será criado. ➤ **ignore** – Não faz nada com esse tipo de EC.

Elemento		Atributos
Nome e caminho	**Descrição**	
link (continuação)		**Nome.** source_ci_type_end1 **Descrição.** Tipo de EC End1 do relacionamento de origem **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. source_ci_type_end2 **Descrição.** Tipo de EC End2 do relacionamento de origem **É obrigatório.** Obrigatório **Tipo.** Cadeia
target_ci_type_end1 (integration > targetrelations > link)	Tipo de EC End1 do relacionamento de destino	**Nome.** name **Descrição.** Nome do tipo de EC End1 do relacionamento de destino. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. superclass **Descrição.** Nome da superclasse do tipo de EC End1. **É obrigatório.** Não é obrigatório **Tipo.** Cadeia
target_ci_type_end2 (integration > targetrelations > link)	Tipo de EC End2 do relacionamento de destino	**Nome.** name **Descrição.** Nome do tipo de EC End2 do relacionamento de destino. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. superclass **Descrição.** Nome da superclasse do tipo de EC End2. **É obrigatório.** Não é obrigatório **Tipo.** Cadeia

Esquema de resultados de mapeamento

Elemento		Atributos
Nome e caminho	**Descrição**	
root	Raiz do documento de resultado	
data (root)	Raiz do próprio dado	
objects (root > data)	O elemento raiz dos objetos a atualizar	
Object (root > data > objects)	Descreve a operação de atualização de um único objeto e todos os seus atributos.	**Nome.** name **Descrição.** O nome do tipo de EC **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. mode **Descrição.** O tipo da atualização necessária para o tipo de EC atual. **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: ➤ **insert** – Use-a apenas se o EC ainda não existir. ➤ **update** – Use-a apenas se o EC já existir. ➤ **update_else_insert** – Se o EC existir, será atualizado; caso contrário, um novo EC será criado. ➤ **ignore** – Não faz nada com esse tipo de EC.

Elemento		Atributos
Nome e caminho	**Descrição**	
Object (continuação)		**Nome.** operation **Descrição.** A operação a ser realizada com esse EC. **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: ➤ **add** – O EC deve ser adicionado ➤ **update** – O EC deve ser atualizado ➤ **delete** – O EC deve ser excluído Se nenhum valor estiver definido, o valor padrão de **add** será usado.
		Nome. mamId **Descrição.** O ID do objeto no CMDB de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
field (root > data > objects > Object -OU- root > data > links > link)	Descreve o valor de um único campo de um objeto. O texto do campo é o novo valor no campo, e se o campo contiver um vínculo, o valor será a ID de uma das extremidades. Cada ID de extremidade aparece como um objeto (sob <objects>).	**Nome.** name **Descrição.** O nome do campo. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. key **Descrição.** Especifica se esse campo é uma chave do objeto. **É obrigatório.** Obrigatório **Tipo.** Booliano
		Nome. datatype **Descrição.** O tipo do campo. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. length **Descrição.** Para tipos de dados cadeia/ caractere, esse é o tamanho inteiro do atributo de destino. **É obrigatório.** Não é obrigatório **Tipo.** Inteiro

Elemento		Atributos
Nome e caminho	**Descrição**	
links (root > data)	O elemento raiz dos vínculos a atualizar	**Nome.** targetRelationshipClass **Descrição.** O nome do relacionamento (vínculo) no sistema de destino. **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. targetParent **Descrição.** O tipo da primeira extremidade do vínculo (pai). **É obrigatório.** Obrigatório **Tipo.** Cadeia
		Nome. targetChild **Descrição.** O tipo da segunda extremidade do vínculo (filho). **É obrigatório.** Obrigatório **Tipo.** Cadeia

Elemento		Atributos
Nome e caminho	**Descrição**	
links (continuação)		**Nome.** mode **Descrição.** Tipo da atualização necessária ao tipo de EC atual. **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: ➤ **insert** – Use-a apenas se o EC ainda não existir. ➤ **update** – Use-a apenas se o EC já existir. ➤ **update_else_insert** – Se o EC existir, será atualizado; caso contrário, um novo EC será criado. ➤ **ignore** – Não faz nada com esse tipo de EC.
		Nome. operation **Descrição.** A operação a ser realizada com esse EC. **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: ➤ **add** – O EC deve ser adicionado ➤ **update** – O EC deve ser atualizado ➤ **delete** – O EC deve ser excluído Se nenhum valor estiver definido, o valor padrão de **add** será usado.
		Nome. mamId **Descrição.** O ID do objeto no CMDB de origem. **É obrigatório.** Obrigatório **Tipo.** Cadeia

Parte II

Usando APIs

8

Introdução a APIs

Este capítulo inclui:

Conceitos

Conceitos

Visão geral das APIs

As seguintes APIs estão incluídas com o HP Universal CMDB:

- **API de Serviço Web do UCMDB.** Permite a criação de definições de itens de configuração e seus relacionamentos topológicos com o UCMDB (Universal Configuration Management Database), além da pesquisa de informações usando consultas TQL e ad hoc. Para ver detalhes, consulte "API WS do HP Universal CMDB" na página 289.
- **API Java do UCMDB.** Explica como ferramentas de terceiros ou personalizadas podem usar a API Java para extrair dados e cálculos e gravar dados no UCMDB (Universal Configuration Management Database). Para ver detalhes, consulte "HP Universal CMDB API" na página 375.

9

API WS do HP Universal CMDB

Este capítulo inclui:

Conceitos

Tarefas

Referência

Conceitos

Convenções

Este capítulo usa as seguintes convenções:

- **UCMDB** refere-se ao banco de dados de Gerenciamento de Configuração Universal em si. **HP Universal CMDB** refere-se ao aplicativo.
- Os elementos e argumentos de método do UCMDB são escritos levando em conta maiúsculas e minúsculas do mesmo modo como são especificados no esquema. Um elemento ou argumento de um método não tem inicial maiúscula. Por exemplo, um relation é um elemento de tipo Relation repassado para um método.

HP Universal CMDB Visão geral da API WS do

Use este capítulo junto com a documentação de esquema do UCMDB, disponível na Biblioteca de Documentação online.

A API WS do HP Universal CMDB é usada para integrar aplicativos ao HP Universal CMDB (UCMDB). A API fornece métodos para:

- adicionar, remover e atualizar EC e relações no CMDB
- recuperar informações sobre o modelo de classe
- recuperar análises de impacto
- recuperar informações sobre elementos de configuração e relacionamentos
- gerenciar credenciais: exibir, adicionar, atualizar e remover
- gerenciar trabalhos: exibir status, ativar e desativar
- gerenciar intervalos de Sonda: exibir, adicionar e atualizar
- gerenciar acionadores: adicionar ou remover um EC acionador e adicionar, remover ou desabilitar um TQL acionador

➤ exibir dados gerais sobre domínios e Sondas

Métodos para recuperar informações sobre elementos de configuração e relacionamentos geralmente usam o TQL (Topology Query Language). Para ver detalhes, consulte "Topology Query Language" no *Guia de Modelagem do HP Universal CMDB*.

Usuários da API WS do HP Universal CMDB devem ter familiaridade com:

➤ A especificação SOAP

➤ Uma linguagem de programação orientada a objeto como C++, C# ou Java

➤ HP Universal CMDB

➤ Gerenciamento de Fluxo de Dados

Esta seção inclui os seguintes tópicos:

➤ "Usos da API" na página 291

➤ "Permissões" na página 292

Usos da API

A API é usada para cumprir inúmeros requisitos de negócios. Por exemplo:

➤ Um sistema de terceiros pode consultar o modelo de classe para ver informações sobre os ECs (elementos de configuração) disponíveis.

➤ Uma ferramenta de gerenciamento de ativos de terceiros pode atualizar o CMDB com informações disponíveis somente a essa ferramenta, unificando assim seus dados com os dados coletados pelos aplicativos HP.

➤ Inúmeros sistemas de terceiros podem popular o CMDB para criar um CMDB central que consiga controlar as alterações e executar análise de impacto.

➤ Um sistema de terceiros pode criar entidades e relacionamentos de acordo com sua lógica de negócios e gravar os dados no CMDB para aproveitar os recursos de consulta do CMDB.

➤ Outros sistemas, como o sistema CCM (Release Control, podem usar os métodos de Análise de Impacto para análise de alterações.

Permissões

O administrador fornece credenciais de logon para conexão com o Serviço Web. As credenciais necessárias dependem de você estar ou não usando o HP Universal CMDB como um aplicativo autônomo ou do próprio Gerenciamento de Serviços de Negócios:

- **HP Universal CMDB autônomo.** Faça logon usando as credenciais de um usuário do UCMDB que recebeu permissões nos recursos de descoberta e integração.

  Para ver detalhes, consulte "Página Gerenciador de Segurança" no *Guia de Administração do HP Universal CMDB*.

- **HP Universal CMDB incorporado em Gerenciamento de Serviços de Negócios.** Faça logon usando as credenciais de um usuário do Gerenciamento de Serviços de Negócios. O usuário precisa receber as permissões relevantes no recurso do HP Universal CMDB no Gerenciamento de Serviços de Negócios.

Quando as permissões são atribuídas através do HP Universal CMDB, os níveis de permissão são Exibir, Atualizar e Executar. Quando as permissões são atribuídas usando o Gerenciamento de Serviços de Negócios, os níveis são Exibir e Atualizar, em que Atualizar também inclui Execução. Para exibir as permissões necessárias para cada operação, consulte a documentação de solicitação de cada operação, consulte *Gerenciamento de Fluxo de Dados Referência de Esquema*.

Referência da API WS do HP Universal CMDB

Para ver a documentação completa sobre as estruturas de solicitação e resposta, consulte a Referência de API do Serviço Web do HP UCMDB. Esses arquivos estão localizados na seguinte pasta:

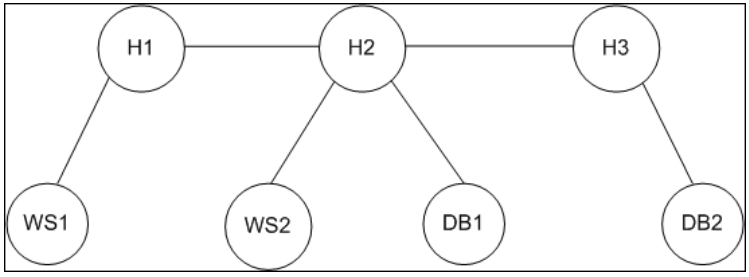
C:\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\doc_lib\DevRef_guide\CMDB_Schema\webframe.html

Retornando elementos de mapa de topologia não ambíguos

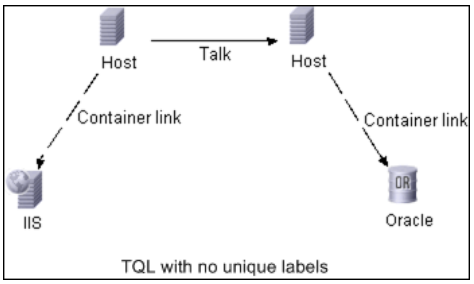
Os métodos de consulta que retornam os dados nos elementos topology ou topologyMap pesquisam no sistema se há uma correspondência de uma consulta TQL. Os diagramas a seguir ilustram como as estruturas topology e topologyMap resultantes são influenciadas pelo uso de rótulos exclusivos na consulta.

Os rótulos são nomes especificados pelo usuário na consulta para relacionamentos e elementos de configuração em configurações específicas. Os rótulos especificados na consulta são usados como os rótulos de nó no mapa retornado. Se nenhum rótulo for especificado, o CI ou Relation Nome do tipo será usado como rótulo no mapa resultante. O exemplo a seguir ilustra a especificação dos rótulos IISHost eDBHost no lugar dos rótulos Host padrão, e os rótulos ContainerIIS e ContainsDB no lugar do rótulo Container Link padrão.

O exemplo a seguir representa um pequeno modelo do universo de TI. Há três hosts: H1, H2 e H3, que hospedam servidores Web (WS) e gerenciadores de banco de dados (DB). WS1 reside em H1. DB1 e WS2 residem em H2. DB2 reside em H3.



Esta consulta é definida usando os rótulos padrão:



O resultado da execução desta consulta TQL no universo de TI pode ser um elemento Topology ou TopologyMap.

Resposta de Topology

```
ECs: H1, H2, H3, WS1, WS2, DB1, DB2
Relacionamentos: H1-WS1, H1-H2, H2-H3, WS2-H2, DB1-H2, DB2-H3
```

Resposta de TopologyMap

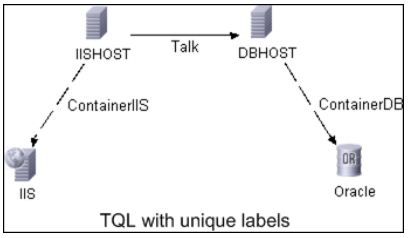
```
CINode:
    label: Host
    CIs: H1, H2
CINode:
    label: Host
    CIs: H2, H3
CINode:
    label: DB
    CIs: DB1, DB2
CINode:
    label: Webserver
CIs: IIS
relationNode:
    label: talk
    relations: H1-H2, H2-H3

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

Na resposta de TopologyMap anterior, os dois primeiros CINodes contêm rótulos Host idênticos, correspondendo aos dois ECs Host na consulta. Ambos os CINodes contêm o host H2, sem nenhuma indicação de por que H2 é duplicado.

Os dois últimos relationNodes contêm rótulos Contained idênticos, correspondendo aos dois relacionamentos Container link na consulta.

As duplicações ocorrem porque nenhum rótulo exclusivo é especificado na consulta, resultando no uso de rótulos padrão (os nomes de tipo Host e Container) no mapa. Para extrair um mapa mais utilizável, defina as consultas com rótulos exclusivos para cada configuração que terá correspondência, conforme mostrado na consulta a seguir:



O resultado de topology é idêntico àquele do TQL sem rótulos exclusivos. O resultado de topologyMap, entretanto, é diferente: Cada rótulo agora é exclusivo.

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

...

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

Neste mapa, está claro por que H2 é retornado duas vezes. Os rótulos exclusivos indicam que ele é retornado uma vez como host de servidor Web e uma vez como host de banco de dados.

Dica: Sempre que possível no CMDB, aplique rótulos exclusivos definidos pelo usuário a configurações específicas.

Tarefas

Chamar o serviço Web

Você pode usar técnicas de programação SOAP padrão no Serviço Web do HP Universal CMDB para habilitar a chamada de métodos no servidor. Se não for possível analisar a instrução ou se houver um problema ao invocar o método, os métodos de API lançarão uma exceção SoapFault. Quando uma exceção SoapFault é lançada, UCMDB popula um ou mais campos de mensagem de erro, código de erro e mensagem de exceção. Se não houver nenhum erro, os resultados da invocação serão retornados.

Os programadores SOAP podem acessar o WSDL em:

http://<server>[:port]/axis2/services/UcmdbService?wsdl

A especificação de porta só é necessária para instalações não padrão. Consulte o administrador do sistema sobre o número de porta correto.

A URL para chamar o serviço é:

http://<server>[:port]/axis2/services/UcmdbService

Para ver exemplos de como se conectar ao CMDB, consulte "Casos de uso" na página 333.

Consultar o CMDB

O CMDB é consultado usando as APIs descritas em"Métodos de consulta do UCMDB" na página 308.

As consultas e os elementos CMDB retornados sempre contêm IDs UMDB reais.

Para ver exemplos de uso dos métodos de consulta, consulte "Exemplo de consulta" na página 339.

Esta seção inclui os seguintes tópicos:

Cálculo de resposta JIT (Just In Time)

Para todos os métodos de consulta, o servidor UMDB calcula os valores solicitados pelo método de consulta quando a solicitação é recebida e retorna os resultados com base nos dados mais recentes. O resultado sempre é calculado no momento em que a solicitação é recebida, mesmo se a consulta TQL estiver ativa e houver um resultado previamente calculado. Portanto, os resultados da execução de uma consulta retornada ao aplicativo cliente podem ser diferentes dos resultados da mesma consulta exibida na interface do usuário.

Dica: Se seu aplicativo usar os resultados de uma determinada consulta mais de uma vez e não houver expectativa de que os dados sejam alterados significativamente entre os usos dos dados dos resultados, você poderá melhorar o desempenho fazendo com que o aplicativo cliente armazene os dados, em vez de executar a consulta várias vezes.

Processando respostas extensas

A resposta a uma consulta sempre incluirá as estruturas dos dados solicitados pelo método de consulta, mesmo se nenhum dado real estiver sendo transmitido. Para muitos métodos em que os dados consistem em uma coleção ou mapa, a resposta também inclui a estrutura ChunkInfo, constituída por chunksKey e numberOfChunks. O campo numberOfChunks indica o número de partes que contêm dados que precisam ser recuperados.

O tamanho máximo de transmissão dos dados é definido pelo administrador do sistema. Se os dados retornados da consulta forem maiores do que o tamanho máximo, as estruturas de dados na primeira resposta não conterão nenhuma informação significativa, e o valor do campo numberOfChunks será 2 ou maior. Se os dados não forem maiores do que o tamanho máximo, o campo numberOfChunks será 0 (zero), e os dados serão transmitidos na primeira resposta. Portanto, ao processar uma resposta, verifique primeiro o valor numberOfChunks. Se ele for maior do que 1, descarte os dados na transmissão e solicite as partes dos dados. Caso contrário, use os dados na resposta.

Para ver informações sobre como lidar com dados em partes, consulte "pullTopologyMapChunks" na página 321 e "releaseChunks" na página 323.

Especificando propriedades para retornar

ECs e relacionamentos geralmente têm muitas propriedades. Alguns métodos que retornam coleções ou gráficos desses elementos aceitam parâmetros de entrada que especificam quais valores de propriedade retornar com cada elemento que corresponde à consulta. O CMDB não retorna propriedades vazias. Portanto, a resposta de uma consulta pode ter menos propriedades do que foram solicitadas na consulta.

Esta seção descreve os tipos de conjuntos usados para especificar as propriedades a retornar.

As propriedades podem ser referenciadas de duas maneiras:

- Pelos seus nomes
- Pelos nomes das regras de propriedades predefinidas. As regras de propriedades predefinidas são usadas pelo CMDB para criar uma lista de nomes de propriedade reais.

Quando um aplicativo referencia propriedades por nome, ele repassa um elemento PropertiesList.

Dica: Sempre que possível, use PropertiesList para especificar os nomes das propriedades em que você tem interesse, em vez de um conjunto baseado em regras. O uso de regras de propriedades predefinidas quase sempre resulta no retorno de mais propriedades do que são necessárias e acarreta em prejuízo no desempenho.

Há dois tipos de propriedades predefinidas: propriedades de qualificador e propriedades simples.

- **Propriedades de qualificador.** Use quando for necessário que o aplicativo cliente repasse um elemento QualifierProperties (uma lista de qualificadores que podem ser aplicados às propriedades). O CMDB converte a lista de qualificadores repassada pelo aplicativo cliente à lista das propriedades à qual pelo menos um dos qualificadores se aplica. Os valores dessas propriedades são retornados com os elementos CI ou Relation.
- **Propriedades simples.** Para usar propriedades simples baseadas em regras, o aplicativo cliente repassa um elemento SimplePredefinedProperty ou SimpleTypedPredefinedProperty. Esses elementos contêm o nome da regra pela qual o CMDB gera a lista de propriedades a retornar. As regras que podem ser especificadas em um elemento SimplePredefinedProperty ou SimpleTypedPredefinedProperty são CONCRETE, DERIVED e NAMING.

Propriedades concretas

As propriedades concretas são o conjunto de propriedades definidas para o TEC especificado. As propriedades adicionadas pelas classes derivadas não são retornadas para instâncias dessas classes derivadas.

Uma coleção de instâncias retornadas por um método pode consistir em instâncias de um TEC especificado na invocação de método e nas instâncias de TECs que herdam desse TEC. Os TECs derivados herdam as propriedades do TEC especificado. Além disso, os TECs derivados estendem o TEC pai adicionando propriedades.

Exemplo de propriedades concretas:

O TEC T1 tem propriedades P1 e P2. O TEC T11 herda de T1 e estende T1 com propriedades P21 e P22.

A coleção de ECs de tipo T1 inclui as instâncias de T1 e T11. As propriedades concretas de todas as instâncias nessa coleção são P1 e P2.

Propriedades derivadas

As propriedades derivadas são o conjunto de propriedades definidas para o TEC especificado e, para cada TEC derivado, as propriedades adicionados pelo TEC derivado.

Exemplo de propriedades derivadas:

Continuando o exemplo das propriedades concretas, as propriedades derivadas das instâncias de T1 são P1 e P2. As propriedades derivadas das instâncias de T11 são P1, P2, P21 e P22.

Propriedades de nomenclatura

As propriedades de nomenclatura são display_label e data_name.

Outros elementos de especificação de propriedades

➤ **PredefinedProperties**

PredefinedProperties pode conter um elemento QualifierProperties e um elemento SimplePredefinedProperty para cada uma das outras regras possíveis. Um conjunto de PredefinedProperties não necessariamente contém todos os tipos de listas.

➤ **PredefinedTypedProperties**

PredefinedTypedProperties é usado para aplicar um conjunto diferente de propriedades a cada TEC. PredefinedTypedProperties pode conter um elemento QualifierProperties e um elemento SimpleTypedPredefinedPropertyroperty para cada uma das outras regras aplicáveis. Como PredefinedTypedProperties é aplicado a cada TEC individualmente, as propriedades derivadas não são relevantes. Um conjunto de PredefinedProperties não necessariamente contém todos os tipos aplicáveis de listas.

➤ **CustomProperties**

CustomProperties pode conter qualquer combinação do PropertiesList básico e das listas de propriedades baseadas em regras. O filtro de propriedades é a união de todas as propriedades retornadas por todas as listas.

➤ **CustomTypedProperties**

CustomTypedProperties pode conter qualquer combinação do PropertiesList básico e das listas de propriedades baseadas em regras aplicáveis. O filtro de propriedades é a união de todas as propriedades retornadas por todas as listas.

➤ **TypedProperties**

TypedProperties é usado para repassar um conjunto diferente de propriedades para cada TEC. TypedProperties é uma coleção de pares compostos de nomes de tipo e conjuntos de propriedades de todos os tipos. Cada conjunto de propriedades é aplicado somente ao tipo correspondente.

Atualizar o UCMDB

Você atualiza o CMDB com as APIs de atualização. Para ver detalhes dos métodos de API, consulte "Métodos de atualização do UCMDB" na página 323.

Para ver exemplos do uso dos métodos de atualização, consulte "Exemplo de atualização" na página 355.

Esta tarefa inclui as seguintes etapas:

➤ "Parâmetros de atualização do UCMDB" na página 303

➤ "Uso dos tipos de ID com métodos de atualização" na página 303

➤ "Métodos de atualização do UCMDB" na página 323

Parâmetros de atualização do UCMDB

Este tópico descreve os parâmetros usados somente pelos métodos de atualização do serviço. Para ver detalhes, consulte a documentação do esquema.

CIsAndRelationsUpdates

O tipo CIsAndRelationsUpdates consiste em CIsForUpdate, relationsForUpdate, referencedRelations e referencedCIs. Uma instância CIsAndRelationsUpdates não necessariamente inclui os três elementos.

CIsForUpdate é uma coleção de ECs. relationsForUpdate é uma coleção Relations. Os elementos CI e relation nos conjuntos têm um elemento props. Ao criar um EC ou relacionamento, as propriedades que têm o atributo required ou o atributo key na definição do Tipo de EC precisarão ser populadas com os valores. Os elementos nessas coleções são atualizados ou criados pelo método.

referencedCIs e referencedRelations são coleções de ECs que já estão definidas no CMDB. Os elementos na coleção são identificados com uma ID temporária em conjunto com todas as propriedades de chave. Esses elementos são usados para resolver as identidades dos ECs e relacionamentos para fins de atualização. Eles nunca são criados ou atualizados pelo método.

Cada um dos elementos CI e relation nessas coleções tem uma coleção de propriedades. Novos elementos são criados com os valores de propriedade nessas coleções.

Uso dos tipos de ID com métodos de atualização

A seguir estão descritos TECs de ID, além de ECs e relacionamentos. Quando a ID não é uma ID do CMDB real, os atributos-chave e de tipo são necessários.

Excluindo ou atualizando elementos de configuração

Um ID temporário ou vazio pode ser usado pelo cliente ao chamar um método para excluir ou atualizar um elemento. Nesse caso, o tipo de EC e os atributos-chave que identificam o EC precisam ser definidos.

Excluindo ou atualizando relacionamentos

Ao excluir ou atualizar relacionamentos, a ID de relacionamento pode ser vazia, temporária ou real.

Se a ID de um EC for temporária, o EC precisará ser repassado na coleção referencedCIs e seus atributos-chave precisarão ser especificados. Para ver detalhes, consulte referencedCIs no "CIsAndRelationsUpdates" na página 303.

Inserindo novos elementos de configuração no CMDB

É possível usar uma ID vazia ou uma ID temporária para inserir um novo EC. Entretanto, se a ID for vazia, o servidor não poderá retornar a ID do CMDB real na estrutura createIDsMap porque não haverá clientID. Para ver detalhes, consulte "addCIsAndRelations" na página 324 e "Métodos de consulta do UCMDB" na página 308.

Inserindo novos relacionamentos no CMDB

O ID de relacionamento pode ser temporário ou vazio. Entretanto, se o relacionamento for novo, mas os elementos de configuração em qualquer das extremidades do relacionamento já estiverem definidos no CMDB, esses ECs que já existem precisarão ser identificados por uma ID do CMDB real ou especificados em uma coleção referencedCIs.

Consultar o modelo de classe do UCMDB

Os métodos de modelo de classe retornam informações sobre TECs e relacionamentos. O modelo de classe é configurado usando o Gerenciador de Tipo de EC. Para ver detalhes, consulte "Gerenciador de Tipo de EC" no *Guia de Modelagem do HP Universal CMDB*.

Para ver exemplos do uso dos métodos de modelo de classe, consulte "Exemplo de modelo de classe" na página 359.

Esta seção fornece informações sobre os seguintes métodos que retornam informações sobre TECs e relacionamentos:

- "getClassAncestors" na página 305
- "getAllClassesHierarchy" na página 305
- "getCmdbClassDefinition" na página 306

getClassAncestors

O método getClassAncestors recupera o caminho entre o TEC específico e sua raiz, além da própria raiz.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
className	O nome do tipo. Para ver detalhes, consulte "Nome do tipo" na página 371.

Saída

Parâmetro	Comentário
classHierarchy	Uma coleção de pares de nomes de classe e nome de classe pai.
comments	Somente para uso interno.

getAllClassesHierarchy

O método getAllClassesHierarchy recupera toda a árvore de modelo de classe.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.

Saída

Parâmetro	Comentário
classesHierarchy	Uma coleção de pares de nome de classe e nome de classe pai.
comments	Somente para uso interno.

getCmdbClassDefinition

O método getCmdbClassDefinition recupera todas as informações sobre a classe especificada.

Se você usar getCmdbClassDefinition para recuperar os atributos-chave, também precisará consultar as classes pai até a classe base. getCmdbClassDefinition identifica como atributos-chave somente os atributos com a ID_ATTRIBUTE definida na definição de classe especificada por className. Os atributos-chave herdados não são reconhecidos como atributos-chave da classe especificada. Portanto, a lista completa dos atributos-chave da classe especificada é a união de todas as chaves da classe e de todos os seus pais, até a raiz.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
className	O nome do tipo. Para ver detalhes, consulte "Nome do tipo" na página 371.

Saída

Parâmetro	Comentário
cmdbClass	A definição de classe, consistindo em name, classType, displayLabel, description, parentName, qualificadores e atributos.
comments	Somente para uso interno.

Consulta para análise de impacto

O Identifier nos métodos de análise de impacto aponta para os dados de resposta do serviço. Ele é exclusivo da resposta atual e é descartado do cache de memória do servidor após 10 minutos de não utilização.

Para ver exemplos do uso dos métodos de análise de impacto, consulte "Exemplo de análise de impacto" na página 361.

Referência

Métodos de consulta do UCMDB

Esta seção fornece informações sobre os seguintes métodos:

executeTopologyQueryByName

O método executeTopologyQueryByName recupera o mapa de topologia que corresponde à consulta especificada.

Dica: O mapa contém mais informações e é mais fácil de entender se o rótulo para cada CINode e cada relationNode no TQL é exclusivo. Para ver detalhes, consulte "Retornando elementos de mapa de topologia não ambíguos" na página 293.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
queryName	O nome do TQL no CMDB com que o mapa será recuperado.
queryTypedProperties	Uma coleção de conjuntos de propriedades que serão recuperadas em elementos de um Tipo de Elemento de Configuração específico.

Saída

Parâmetro	Comentário
topologyMap	Para ver detalhes, consulte "TopologyMap" na página 374.

executeTopologyQueryByNameWithParameters

O método executeTopologyQueryByNameWithParameters recupera um elemento topologyMap que corresponde à consulta parametrizada especificada.

Os valores dos parâmetros da consulta são repassados para o argumento parameterizedNodes. A TQL especificada precisa ter rótulos exclusivos definidos para cada CINode e cada relationNode ou então haverá falha na invocação do método.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
queryName	O nome do TQL parametrizada no CMDB para o qual o mapa será obtido.

Parâmetro	Comentário
parameterizedNodes	As condições que cada nó precisa atender para ser incluído nos resultados da consulta.
queryTypedProperties	Uma coleção de conjuntos de propriedades que serão recuperadas em elementos de um Tipo de Elemento de Configuração específico.

Saída

Parâmetro	Comentário
topologyMap	Para ver detalhes, consulte "TopologyMap" na página 374.
chunkInfo	Para ver detalhes, consulte: "ChunkInfo" na página 374, "Processando respostas extensas" na página 299.

executeTopologyQueryWithParameters

O método executeTopologyQueryWithParameters recupera um elemento topologyMap que corresponde à consulta parametrizada.

A consulta é repassada no argumento queryXML. Os valores dos parâmetros da consulta são repassados para o argumento parameterizedNodes. A TQL precisa ter rótulos exclusivos definidos para cada CINode e cada relationNode.

O método executeTopologyQueryWithParameters é usado para repassar consultas ad-hoc, em vez de acessar uma consulta definida no CMDB. Você pode usar esse método quando não tem acesso à interface do usuário do UCMDB para definir uma consulta ou quando não deseja salvá-la no banco de dados.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
queryXML	Uma cadeia de caracteres XML que representa um TQL sem marcas de recursos.
parameterizedNodes	As condições que cada nó precisa atender para ser incluído nos resultados da consulta.

Saída

Parâmetro	Comentário
topologyMap	Para ver detalhes, consulte "TopologyMap" na página 374.
chunkInfo	Para ver detalhes, consulte "ChunkInfo" na página 374 e "Processando respostas extensas" na página 299.

 getChangedCIs

O método getChangedCIs retorna os dados de alteração para todos os ECs relacionados aos ECs especificados.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
ids	A lista dos IDs dos ECs raiz cujos ECs relacionados são verificados para ver se há alterações. Somente IDs do CMDB reais são válidos nessa coleção.

Parâmetro	Comentário
fromDate	O início do período em que os ECs são verificados para ver se houve alterações.
toDate	O término do período em que os ECs são verificados para ver se houve alterações.

Saída

Parâmetro	Comentário
changeDataInfo	Zero ou mais coleções de elementos ChangedDataInfo.

getCINeighbours

O método getCINeighbours retorna os vizinhos imediatos do EC especificado.

Por exemplo, se a consulta for nos vizinhos do EC A e EC A contiver EC B que usa EC C, EC B será retornado, mas EC C não. Isso significa que somente os vizinhos do tipo especificado serão retornados.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
ID	O ID do EC com o qual os vizinhos serão recuperados. Ele precisa ser uma ID do CMDB real.
neighbourType	O nome do TEC dos vizinhos que serão recuperados. Os vizinhos do tipo especificado e dos tipos derivados desse tipo são retornados. Para ver detalhes, consulte "Nome do tipo" na página 371.

Parâmetro	Comentário
CIProperties	Os dados que serão retornados em cada elemento de configuração, chamados Layout de Consulta na interface do usuário. Para ver detalhes, consulte "TypedProperties" na página 302.
relationProperties	Os dados que serão retornados em cada relacionamento (chamados Layout de Consulta na interface do usuário). Para ver detalhes, consulte "TypedProperties" na página 302

Saída

Parâmetro	Comentário
topology	Para ver detalhes, consulte "Topology" na página 373.
comments	Somente para uso interno.

 getCIsByID

O método getCIsByID recupera os elementos de configuração por seus IDs do CMDB .

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
CIsTypedProperties	Uma coleção de propriedades de tipos. Para ver detalhes, consulte "Outros elementos de especificação de propriedades" na página 301.
IDs	Somente IDs do CMDB reais são válidos nessa coleção.

Saída

Parâmetro	Comentário
CIs	Coleção de elementos EC.
chunkInfo	Para ver detalhes, consulte: "ChunkInfo" na página 374, "Processando respostas extensas" na página 299.

 getCIsByType

O método getCIsByType retorna a coleção dos elementos de configuração do tipo especificado e de todos os tipos que herdam do tipo especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
type	O nome de classe. Para ver detalhes, consulte "Nome do tipo" na página 371.
properties	Os dados que serão retornados em cada elemento de configuração. Para ver detalhes, consulte "CustomProperties" na página 302.

Saída

Parâmetro	Comentário
CIs	Coleção de elementos EC.
chunkInfo	Para ver detalhes, consulte: "ChunkInfo" na página 374, "Processando respostas extensas" na página 299.

getFilteredCIsByType

O método getFilteredCIsByType recupera os ECs do tipo especificado que atende às condições usadas pelo método. Uma condição consiste em:

- um campo de nome que contém o nome de uma propriedade
- um campo de operador que contém um operador de comparação
- um campo de valor opcional que contém um valor ou uma lista de valores

Juntos, eles formam uma expressão booliana:

```
<elemento>.property.value [operator] <condição>.value
```

Por exemplo, se o nome da condição for root_actualdeletionperiod, o valor da condição for 40 e o operador for Equal, a instrução booliana será:

```
<item>.root_actualdeletionperiod.value = = 40
```

A consulta retornará todos os elementos cujo root_actualdeletionperiod seja 40, pressupondo que não há outras condições.

Se o argumento conditionsLogicalOperator for AND, a consulta retornará os elementos que atendem a todas as condições na coleção conditions. Se o argumento conditionsLogicalOperator for OR, a consulta retornará os elementos que atendem a pelo menos uma das condições na coleção conditions.

A tabela a seguir lista os operadores de comparação:

Operador	Tipo de condição/comentários
ChangedDuring	Data Esta é uma verificação de intervalo. O valor da condição é especificado em horas. Se o valor da propriedade de data estiver no intervalo do tempo em que o método é invocado mais ou menos o valor da condição, a condição será verdadeira. Por exemplo, se o valor da condição for 24, a condição será verdadeira se o valor da propriedade de data estiver entre ontem nesta hora e amanhã nesta hora. **Observação:** o nome ChangedDuring é mantido para preservar a compatibilidade com versões anteriores. Nas versões anteriores, o operador era usado somente com as propriedades de criação e modificação de tempo.
Equal	Cadeia de caracteres e numérica
EqualIgnoreCase	Cadeia
Greater	Numérico
GreaterEqual	Numérico
In	Cadeia de caracteres, numérico e lista O valor da condição é uma lista. A condição será verdadeira se o valor da propriedade for um dos valores na lista.
InList	Lista O valor da condição e o valor da propriedade são listas. A condição será verdadeira se todos os valores na lista da condição também aparecerem na lista de propriedades do elemento. Pode haver mais valores de propriedades do que os especificados na condição sem afetar a validade da condição.

Operador	Tipo de condição/comentários
IsNull	Cadeia de caracteres, numérico e lista A propriedade do item não tem valor. Quando o operador IsNull é usado, o valor da condição é ignorado e, em alguns casos, pode ser nulo.
Less	Numérico
LessEqual	Numérico
Like	Cadeia O valor da condição é uma subcadeia do valor da propriedade. O valor da condição precisa estar entre parênteses com sinais de porcentagem (%). Por exemplo, %Bi% corresponde Bismark com Bay of Biscay, mas não com biscuit.
LikeIgnoreCase	Cadeia Use o operador LikeIgnoreCase como usa o operador Like. A correspondência, entretanto, não diferencia maiúsculas de minúsculas. Portanto, %Bi% corresponde a biscuit.
NotEqual	Cadeia de caracteres e numérica
UnchangedDuring	Data Esta é uma verificação de intervalo. O valor da condição é especificado em horas. Se o valor da propriedade de data estiver no intervalo do tempo em que o método é invocado mais ou menos o valor da condição, a condição será falsa. Se estiver fora desse intervalo, a condição será verdadeira. Por exemplo, se o valor da condição for 24, a condição será verdadeira se o valor da propriedade de data for anterior a ontem nesta hora ou posterior a amanhã nesta hora. **Observação:** o nome UnchangedDuring é mantido para preservar a compatibilidade com versões anteriores. Nas versões anteriores, o operador era usado somente com as propriedades de criação e modificação de tempo.

Exemplo de configuração de uma condição:

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.Equal);
```

Exemplo de consulta de propriedades herdadas:

O EC de destino é sample, que tem dois atributos, name e size. sampleII estende o EC com dois atributos, level e grade. Este exemplo configura uma consulta das propriedades de sampleII que foram herdadas de sample especificando-as por nome.

```
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

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
type	O nome de classe. Para ver detalhes, consulte "Nome do tipo" na página 371. O tipo pode ser qualquer um dos tipos definidos usando o Gerenciador de Tipo de EC. Para ver detalhes, consulte "Gerenciador de Tipo de EC", no *Guia de Modelagem do HP Universal CMDB*.

Parâmetro	Comentário
properties	Os dados que serão retornados em cada EC (chamados Layout de Consulta na interface do usuário). Para ver detalhes, consulte "CustomProperties" na página 302.
conditions	Uma coleção de pares de valor de nome e dos operadores que relacionam um ao outro. Por exemplo, host_hostname like QA.
conditionsLogicalOperator	➤ **AND.** Todas as condições precisam ser atendidas. ➤ **OR**. Pelo menos uma das condições precisa ser atendida.

Saída

Parâmetro	Comentário
CIs	Coleção de elementos EC.
chunkInfo	Para ver detalhes, consulte "ChunkInfo" na página 374 e "Processando respostas extensas" na página 299.

getQueryNameOfView

O método getQueryNameOfView recupera o nome do TQL que serve de base para a visualização especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
viewName	O nome de uma visualização, ou seja, um subconjunto do modelo de classe no CMDB.

Saída

Parâmetro	Comentário
queryName	O nome do TQL no CMDB que serve de base para a visualização.

 getTopologyQueryExistingResultByName

O método getTopologyQueryExistingResultByName recupera o resultado mais recente da execução do TQL especificado. A chamada não executa o TQL. Se não houver nenhum resultado de uma execução anterior, nada será retornado.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
queryName	O nome de um TQL.
queryTypedProperties	Uma coleção de conjuntos de propriedades que serão recuperadas para elementos de um Tipo de Elemento de Configuração específico.

Saída

Parâmetro	Comentário
queryName	O nome do TQL no CMDB que serve de base para a visualização.

getTopologyQueryResultCountByName

O método getTopologyQueryResultCountByName recupera o número de instâncias de cada nó que corresponde à consulta especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
queryName	O nome de um TQL.
countInvisible	Se for verdadeiro, a saída incluirá ECs definidos como invisíveis na consulta.

Saída

Parâmetro	Comentário
queryName	O nome do TQL no CMDB que serve de base para a visualização.

pullTopologyMapChunks

O método pullTopologyMapChunks recupera uma das partes que contêm a resposta a um método.

Cada parte contém um elemento topologyMap que faz parte da resposta. A primeira parte é numerada 1, para que o contador de loop de recuperação faça iteração de 1 para <*objeto de resposta*>.getChunkInfo().getNumberOfChunks().

Para ver detalhes, consulte "ChunkInfo" na página 374 e "Consultar o CMDB" na página 298.

O aplicativo cliente precisa poder manipular os mapas parciais. Veja o seguinte exemplo para manipular uma coleção de ECs e o exemplo para mesclar partes para um mapa em "Exemplo de consulta" na página 339.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
ChunkRequest	O número da parte a ser recuperada e o ChunkInfo que é retornado pelo método de consulta.

Saída

Parâmetro	Comentário
topologyMap	Para ver detalhes, consulte "TopologyMap" na página 374.
comments	Somente para uso interno.

Exemplo para manipular partes:

```
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++) {
                    // seu código para processar os ECs
               }
          }
}
```

releaseChunks

O método releaseChunks libera a memória das partes que contêm os dados da consulta.

Dica: O servidor descarta os dados após dez minutos. Chamar esse método para descartar os dados logo após a leitura destes conserva os recursos do servidor.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
chunksKey	O identificador dos dados no servidor que foi dividido em partes. A chave é um elemento do ChunkInfo.

Métodos de atualização do UCMDB

Esta seção fornece informações sobre os seguintes métodos:

- "addCIsAndRelations" na página 324
- "addCustomer" na página 325
- "deleteCIsAndRelations" na página 325
- "removeCustomer" na página 326
- "updateCIsAndRelations" na página 326

addCIsAndRelations

O método addCIsAndRelations adiciona ou atualiza ECs e relacionamentos.

Se os ECs ou relacionamentos não existirem no CMDB, eles serão adicionados e suas propriedades serão definidas de acordo com o conteúdo do argumento CIsAndRelationsUpdates.

Se os ECs ou relacionamentos não existirem no CMDB, eles serão atualizados com os novos dados, se updateExisting for **true**.

Se updateExisting for **false**, CIsAndRelationsUpdates não poderá referenciar os elementos de configuração ou relacionamentos existentes. Qualquer tentativa de referenciar os elementos existentes quando updateExisting for falso resultará em uma exceção.

Se updateExisting for**true**, a operação de adição ou atualização será executada sem validar os ECs, independentemente do valor de ignoreValidation.

Se updateExisiting for **false** e ignoreValidation for **true**, a operação de adição será executada sem validar os ECs.

Se updateExisiting for **false** e ignoreValidation também for **false**, os ECs serão validados antes da operação de adição.

Os relacionamentos nunca são validados.

CreatedIDsMap é um mapa ou dicionário do tipo ClientIDToCmdbID que conecta os IDs temporários do cliente aos IDs do CMDB reais correspondentes.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
updateExisting	Defina como *true* para atualizar os elementos que já existem no CMDB. Defina como *false* para lançar uma exceção se algum item já existir.

Parâmetro	Comentário
CIsAndRelationsUpdates	Os elementos que serão atualizados ou criados. Para ver detalhes, consulte "CIsAndRelationsUpdates" na página 303.
ignoreValidation	Se for verdadeiro, nenhuma verificação será executada antes de atualizar o CMDB.

Saída

Parâmetro	Comentário
CreatedIDsMap	O mapa dos IDs do cliente para os IDs do CMDB. Para ver detalhes, consulte "addCIsAndRelations" na página 324.
comments	Somente para uso interno.

addCustomer

O método addCustomer adiciona um cliente.

Entrada

Parâmetro	Comentário
CustomerID	O ID numérico do cliente.

deleteCIsAndRelations

O método deleteCIsAndRelations remove os elementos de configuração e relacionamentos especificados do CMDB.

Quando um EC é excluído e o EC é uma extremidade de um ou mais elementos Relation, esses elementos Relation também são excluídos.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
CIsAndRelationsUpdates	Os itens que serão excluídos. Para ver detalhes, consulte "CIsAndRelationsUpdates" na página 303.

removeCustomer

O método removeCustomer exclui um registro de cliente.

Entrada

Parâmetro	Comentário
CustomerID	O ID numérico do cliente.

updateCIsAndRelations

O método updateCIsAndRelations atualiza os ECs e relacionamentos especificados.

A atualização usa os valores de propriedade do argumento CIsAndRelationsUpdates. Se algum dos ECs ou relacionamentos não existir no CMDB, uma exceção será lançada.

CreatedIDsMap é um mapa ou dicionário do tipo ClientIDToCmdbID que conecta os IDs temporários do cliente aos IDs do CMDB reais correspondentes.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.

Parâmetro	Comentário
CIsAndRelationsUpdates	Os itens que serão atualizados. Para ver detalhes, consulte "CIsAndRelationsUpdates" na página 303.
ignoreValidation	Se for verdadeiro, nenhuma verificação será executada antes de atualizar o CMDB.

Saída

Parâmetro	Comentário
CreatedIDsMap	O mapa dos IDs do cliente para os IDs do CMDB. Para ver detalhes, consulte "addCIsAndRelations" na página 324.

Métodos de análise de impacto do UCMDB

Esta seção fornece informações sobre os seguintes métodos:

- "calculateImpact" na página 327
- "getImpactPath" na página 328
- "getImpactRulesByNamePrefix" na página 329

calculateImpact

O método calculateImpact calcula quais ECs são afetados por um determinado EC de acordo com as regras definidas no CMDB.

Isso mostra o efeito do acionamento de um evento da regra. A saída identifier de calculateImpact é usada como entrada para getImpactPath.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
impactCategory	O tipo de evento que acionaria a regra sendo simulada.
IDs	Uma coleção de elementos de EC.
impactRulesNames	Uma coleção de elementos ImpactRuleName.
severity	A gravidade do evento acionado.

Saída

Parâmetro	Comentário
impactTopology	Para ver detalhes, consulte "Topology" na página 373.
identifier	A chave da resposta do servidor.

 getImpactPath

O método getImpactPath recupera o gráfico de topologia do caminho entre o EC afetado e o EC que o afeta.

A saída identifier de calculateImpact é usada como o argumento de entrada identifier de getImpactPath.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
identifier	A chave da resposta do servidor que foi retornada por calculateImpact.
relation	UmaRelacionamento que se baseia em um dos ShallowRelations retornados por calculateImpact no elemento impactTopology.

Saída

Parâmetro	Comentário
impactPathTopology	Uma coleção de ECs e uma coleção ImpactRelations.
comments	Somente para uso interno.

Um elemento ImpactRelations consiste em ID, type, end1ID, end2ID, rule e action.

 getImpactRulesByNamePrefix

O método getImpactRulesByNamePrefix recupera as regras usando um filtro de prefixo.

Esse método se aplica a regras de impacto que são nomeadas com um prefixo que indica o contexto ao qual se aplicam, por exemplo, SAP_myrule, ORA_myrule e assim por diante. Esse método filtra todos os nomes de regras de impacto para aqueles que começam com o prefixo especificado pelo argumento ruleNamePrefixFilter.

Entrada

Parâmetro	Comentário
cmdbContext	Para ver detalhes, consulte "CmdbContext" na página 369.
ruleNamePrefixFilter	Uma cadeia de caracteres que contém as primeiras letras dos nomes de regras com os quais haverá correspondência.

Saída

Parâmetro	Comentário
impactRules	impactRules consiste em zero ou mais impactRule. Um impactRule, que especifica o efeito de uma alteração, consiste em ruleName, description, queryName e isActive.

Gerenciamento de Fluxo de Dados

Esta seção contém uma lista das operações de serviço Web e um breve resumo de seu uso. Para ver uma documentação completa da solicitação e resposta de cada operação, consulte *Gerenciamento de Fluxo de Dados Referência de Esquema*.

Esta seção inclui os seguintes tópicos:

Gerenciando métodos de trabalho do DFM

- **activateJob**

  Ativa o trabalho especificado.

- **deactivateJob**

  Desativa o trabalho especificado.

- **dispatchAdHocJob**

  Distribui um trabalho na Sonda ad-hoc. O trabalho precisa estar ativo e conter o EC acionador especificado.

- **getDiscoveryJobsNames**

  Retorna a lista de nomes de trabalho.

- **isJobActive**

  Verifica se o trabalho está ativo.

Gerenciando métodos de acionador

- **addTriggerCI**

  Adiciona um novo EC acionador ao trabalho especificado.

➤ **addTriggerTQL**

Adiciona um novo TQL acionador ao trabalho especificado.

➤ **disableTriggerTQL**

Impede o TQL de acionar o trabalho, mas não o remove permanentemente da lista de consultas que acionam o trabalho.

➤ **removeTriggerCI**

Remove o EC especificado da lista de ECs que acionam o trabalho.

➤ **removeTriggerTQL**

Remove o TQL especificado da lista de consultas que acionam o trabalho.

➤ **setTriggerTQLProbesLimit**

Restringe as Sondas em que o TQL está ativo no trabalho à lista especificada.

Métodos de dados de sonda e domínio

➤ **getDomainType**

Retorna o tipo de domínio.

➤ **getDomainsNames**

Retorna os nomes dos domínios atuais.

➤ **getProbeIPs**

Retorna os endereços IP da Sonda especificada.

➤ **getProbesNames**

Retorna os nomes das Sondas no domínio especificado.

➤ **getProbeScope**

Retorna a definição de escopo da Sonda especificada.

➤ **isProbeConnected**

Verifica se a Sonda especificada está conectada.

➤ **updateProbeScope**

Define o escopo da Sonda especificada, substituindo o escopo existente.

Métodos de dados de credenciais

➤ **addCredentialsEntry**

Adiciona uma entrada de credenciais ao protocolo especificado para o domínio especificado.

➤ **getCredentialsEntriesIDs**

Retorna os IDs das credenciais definidas para o protocolo especificado.

➤ **getCredentialsEntry**

Retorna as credenciais definidas para o protocolo especificado. Os atributos criptografados são retornados vazios.

➤ **removeCredentialsEntry**

Retorna as credenciais especificadas do protocolo.

➤ **updateCredentialsEntry**

Define novos valores para propriedades da entrada de credenciais especificada.

Métodos de atualização de dados

➤ **rediscoverCIs**

Localiza os acionadores que descobriram os objetos de EC especificados e executa novamente esses acionadores. (Observe que o comando de nova execução tem maior prioridade do que outros elementos agendados.)

Os **rediscoverCIs** são executados de modo assíncrono. Chame **checkDiscoveryProgress** para determinar quando a redescoberta estará concluída.

➤ **checkDiscoveryProgress**

Retorna o andamento da chamada **rediscoverCIs** mais recente nos IDs especificados. A resposta será um valor de 0 a 1. Quando a resposta for 1, a chamada **rediscoverCIs** terá sido concluída.

➤ **rediscoverViewCIs**

Localiza os acionadores que criaram os dados para popular a visualização especificada e executa novamente esses acionadores. (Observe que o comando de nova execução tem maior prioridade do que outros elementos agendados.)

Os **rediscoverViewCIs** são executados de modo assíncrono. Chame **checkViewDiscoveryProgress** para determinar quando a redescoberta estará concluída.

➤ **checkViewDiscoveryProgress**

Retorna o andamento da chamada **rediscoverViewCIs** mais recente na visualização especificada. A resposta é um valor entre 0 e 1. Quando a resposta for 1, a chamada **rediscoverCIs** terá sido concluída.

Casos de uso

Os seguintes casos de uso pressupõem dois sistemas:

➤ Servidor HP Universal CMDB

➤ Um sistema de terceiros que contém um repositório dos elementos de configuração

Esta seção inclui os seguintes tópicos:

Populando o CMDB

Casos de uso:

➤ Um gerenciamento de ativos de terceiros atualiza o CMDB com informações disponíveis somente no gerenciamento de ativos.

➤ Inúmeros sistemas de terceiros populam o CMDB para criar um CMDB central que consiga controlar as alterações e executar análise de impacto.

➤ Um sistema de terceiros cria Elementos de Configuração e Relacionamentos de acordo com uma lógica de negócios de terceiros para aproveitar os recursos de consulta do CMDB.

Consultando o CMDB

Casos de uso:

➤ Um sistema de terceiros obtém os Elementos de Configuração e Relacionamentos que representam o sistema SAP obtendo os resultados do TQL SAP.

➤ Um sistema de terceiros obtém a lista de servidores Oracle que foram adicionados ou alterados nas últimas cinco horas.

➤ Um sistema de terceiros obtém a lista de servidores cujo nome de host contém a subcadeia *lab*.

➤ Um sistema de terceiros localiza os elementos relacionados a um determinado EC obtendo seus vizinhos.

Consultando o modelo de classe

Casos de uso:

➤ Um sistema de terceiros permite que os usuários especifiquem o conjunto de dados que serão recuperados do CMDB. Uma interface do usuário pode ser criada com base no modelo de classe para mostrar aos usuários as propriedades possíveis e lhes solicitar os dados necessários. O usuário poderá escolher as informações que serão recuperadas.

➤ Um sistema de terceiros explora o modelo de classe quando o usuário não pode acessar a interface do usuário do UCMDB.

Analisando o impacto das alterações

Caso de uso:

Um sistema de terceiros gera saída de uma lista dos serviços comerciais que poderiam sofrer impacto de uma alteração em um host especificado.

Exemplos

Esta seção inclui os seguintes tópicos:

Exemplo de classe base

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
        } catch (MalformedURLException e) {

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

Exemplo 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 relations2;
    }
```

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


}
```

Exemplo de atualização

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

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

import java.rmi.RemoteException;

public class UpdateDemo extends Demo{
```

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

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

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

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

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

    }
```

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

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

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

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

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

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

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

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

    }
```

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

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

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

    }


}
```

Exemplo de modelo de classe

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

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

import java.rmi.RemoteException;

public class ClassmodelDemo extends Demo{
```

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

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

    }
```

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

    }
```

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

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

    }


}
```

Exemplo de análise 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
        }
}

}
```

Exemplo de adição de credenciais

```
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 main(String[] args) throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(new
GetDomainsNamesRequest(cmdbContext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credentials");
            return;
        }
        String domainName = domains.getStrValue(0);

        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user", newCredsProperties);
        setStringProperties("ntadminprotocol_ntdomain", "test doamin",
newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(new
AddCredentialsEntryRequest(domainName,  "ntadminprotocol",  newCredsProperties,
cmdbContext));

        System.out.println("new credentials craeted for domain: " + domainName + " in
ntcmd protocol");
    }
```

```
    private static void setPasswordProperty(CIProperties newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }
```

```
    private static void setStringProperties(String propertyName, String value,
CIProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }
```

```
    private static void getProbesInfo(DiscoveryService discoveryService) throws
Exception {
        GetDomainsNamesResponse result =
discoveryService.getDomainsNames(new GetDomainsNamesRequest(cmdbContext
));

        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName = result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(new
GetProbesNamesRequest(domainName, cmdbContext));

            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames().sizeStrValueList(); i++) {
                String probeName = probesResult.getProbesNames().getStrValue(i);

                // Check if connected
                IsProbeConnectedResponce connectedRequest =
                    discoveryService.isProbeConnected(new
IsProbeConnectedRequest(domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected();

                // Do something …
                System.out.println("probe " + probeName + " isconnect=" +
isConnected);
            }
        }
    }
```

```
    private static DiscoveryService getDiscoveryService() throws Exception {
        DiscoveryService discoveryService = null;
        try {

            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub = new
DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth = new
HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);

serviceStub._getServiceClient().getOptions().setProperty(HTTPConstants.AUTHENTIC
ATE,auth);

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

        return discoveryService;

    }
} // End class
```

Parâmetros gerais do UCMDB

Esta seção descreve os parâmetros mais comuns dos métodos do serviço. Para ver detalhes, consulte a documentação do esquema.

Esta seção inclui os seguintes tópicos:

CmdbContext

Todas as invocações de serviço da API do serviço Web do UCMDB exigem um argumento CmdbContext. CmdbContext é uma cadeia de caracteres callerApplication que identifica o aplicativo que invoca o serviço. CmdbContext é usado para registro em log e solução de problemas.

ID

Cada CI e Relation tem um campo ID. Ele consiste em uma cadeia de caracteres de ID que diferencia maiúsculas de minúsculas e um sinalizador temp opcional, indicando se a ID é temporária.

Atributos-chave

Para identificar um CI ou Relation em alguns contextos, os atributos-chave podem ser usados em lugar de um CMDB ID. Os atributos-chave são aqueles com a ID_ATTRIBUTE definida na definição de classe.

Na interface do usuário, os atributos-chave têm um ícone de chave ao lado deles na lista de atributos do Tipo de Elemento de Configuração na interface do usuário. Para ver detalhes, consulte "Caixa de diálogo Adicionar/Editar Atributo" no *Guia de Modelagem do HP Universal CMDB*. Para ver informações sobre como identificar os atributos-chave do próprio aplicativo cliente da API, consulte "getCmdbClassDefinition" na página 306.

Tipos de ID

Um elemento ID pode conter uma ID real, uma ID temporária ou pode estar vazio.

Um ID real é uma cadeia de caracteres atribuída pelo CMDB que identifica uma entidade no banco de dados. Um ID temporário pode ser qualquer cadeia de caracteres exclusiva na solicitação atual. Um ID vazio significa que nenhum valor está atribuído.

Um ID temporário pode ser atribuído pelo cliente e geralmente representa a ID do EC conforme armazenada pelo cliente. Ele não necessariamente representa uma entidade já criada no CMDB. Quando uma ID temporária é repassada pelo cliente, se o CMDB pode identificar um elemento de configuração de dados existente que usa as propriedades de chave do EC, esse EC é usado conforme apropriado para o contexto como se estivesse identificado com uma ID real.

A ID real de um CI é calculada pelo CMDB com base em uma combinação do tipo e das propriedades de chave do EC. A ID real de uma Relation se baseia no tipo do relacionamento, nas IDs dos dois ECs que fazem parte do relacionamento e nas propriedades de chave do relacionamento. Portanto, os valores de atributo-chave precisam ser definidos durante a criação de um CI ou Relation. Se os valores de propriedade de chave não forem especificados ao criar um EC, haverá duas possibilidades:

- Se o TEC incluir um qualificador RANDOM_GENERATED_ID, o servidor gerará uma ID exclusiva.
- Se o CIT não tiver um qualificador RANDOM_GENERATED_ID, será lançada uma exceção.

Para ver detalhes, consulte "Gerenciador de Tipo de EC", no *Guia de Modelagem do HP Universal CMDB*.

CIProperties

Um elemento CIProperties consiste em coleções, sendo que cada uma contém uma sequência de elementos de valor de nome que especifica as propriedades do tipo indicado pelo nome da coleção. Nenhuma das coleções é necessária, por isso o elemento CIProperties pode conter qualquer combinação de coleções.

CIProperties são usados pelos elementos CI e Relation. Para ver detalhes, consulte "Elemento de configuração (EC)" na página 372 e "Relacionamento" na página 372.

As coleções de propriedades são:

- datesProps - coleção de elementos DateProp
- doubleProps - coleção de elementos DoubleProp
- floatProps - coleção de elementos FloatProp
- intListProps - coleção de elementos intListProp
- intProps - coleção de elementos IntProp
- strProps - coleção de elementos StrProp
- strListProps - coleção de elementos StrListProp
- longProps - coleção de elementos LongProp
- bytesProps - coleção de elementos BytesProp
- xmlProps - coleção de elementos XmlProp

Nome do tipo

O nome to tipo é o nome de classe de um tipo de elemento de configuração ou tipo de relacionamento. O nome do tipo é usado em código para se referir à classe. Ele não deve ser confundido com o nome de exibição, que é visto na interface do usuário em que a classe é mencionada, mas que não faz sentido algum em termos de código.

Elemento de configuração (EC)

Um elemento CI (EC) consiste em uma coleção de ID, type e props.

Ao usar o Métodos de atualização do UCMDB para atualizar um CI, o elemento ID pode conter uma ID do CMDB real ou uma ID temporária atribuída pelo cliente. Se uma ID temporária for usada, defina o sinalizador temp como verdadeiro. Ao excluir um item, o ID pode ficar vazio. O Métodos de consulta do UCMDB obtém os IDs reais como parâmetros de entrada e retorna IDs reais nos resultados de consulta.

O type pode ser qualquer nome de tipo definido no Gerenciador de Tipo de EC. Para ver detalhes, consulte "Gerenciador de Tipo de EC", no *Guia de Modelagem do HP Universal CMDB*.

O elemento props é uma coleção CIProperties. Para ver detalhes, consulte "CIProperties" na página 371.

Relacionamento

Um Relation (Relacionamento) é uma entidade que vincula dois elementos de configuração. Um elemento Relation consiste em uma coleção de ID, type, identificadores dos dois itens sendo vinculados (end1ID e end2ID) e props.

Ao usar o Métodos de atualização do UCMDB para atualizar um Relation, o valor da ID do Relation pode ser uma ID do CMDB real ou uma ID temporária. Ao excluir um elemento, o ID pode ficar vazio. O Métodos de consulta do UCMDB obtém IDs reais como parâmetros de entrada e retorna IDs reais nos resultados de consulta.

O tipo de relacionamento é o Type Name da classe do UCMDB da qual o relacionamento é instanciado. O tipo pode ser qualquer um dos tipos de relacionamento definidos no CMDB. Para ver mais informações sobre classes ou tipos, consulte "Consultar o modelo de classe do UCMDB" na página 304.

Para ver detalhes, consulte "Gerenciador de Tipo de EC", no *Guia de Modelagem do HP Universal CMDB*.

Os dois IDs de extremidade do relacionamento precisam ser IDs vazios porque eles são usados para criar a ID do relacionamento atual. Entretanto, ambos podem ter IDs temporários atribuídos a eles pelo cliente.

O elemento props é uma coleção CIProperties. Para ver detalhes, consulte "CIProperties" na página 371.

Parâmetros de saída do UCMDB

Esta seção descreve os parâmetros mais comuns dos métodos do serviço. Para ver detalhes, consulte a documentação do esquema.

Esta seção inclui os seguintes tópicos:

CIs

CIs é uma coleção de elementos de CI (EC).

ShallowRelation

ShallowRelation é uma entidade que vincula dois elementos de configuração, que consistem em ID, type e identificadores dos dois elementos sendo vinculados (end1ID e end2ID). O tipo de relacionamento é o Type Name da classe do CMDB da qual o relacionamento é instanciado. O tipo pode ser qualquer um dos tipos de relacionamento definidos no CMDB.

Topology

Topology é um gráfico de elementos CI e relacionamentos. Um Topology consiste em uma coleção de CIs e uma coleção de Relations que contêm um ou mais elementos Relation.

CINode

CINode consiste em uma coleção de CIs com um label. O label no CINode é o rótulo definido no nó do TQL usado na consulta.

RelationNode

RelationNode consiste em um conjunto de coleções de Relations com um label. O label no RelationNode é o rótulo definido no nó do TQL usado na consulta.

TopologyMap

TopologyMap é a saída de um cálculo de consulta que corresponde à consulta TQL. Os labels no TopologyMap são os rótulos de nó definidos no TQL usado na consulta.

Os dados de um TopologyMap são retornados na seguinte forma:

- CINodes. Este consiste em um ou mais CINode (consulte "CINode" na página 373).
- relationNodes. Este consiste em um ou mais RelationNode (consulte "RelationNode" na página 374).

Os labels nessas duas estruturas ordenam as listas de elementos de configuração e relacionamentos.

ChunkInfo

Quando uma consulta retorna uma grande quantidade de dados, o servidor armazena-os, divididos em segmentos chamados partes. As informações que o cliente usa para recuperar os dados divididos em partes estão localizadas na estrutura de ChunkInfo retornada pela consulta. ChunkInfo consiste em no numberOfChunks que precisa ser recuperado e no chunksKey. O chunksKey é um identificador exclusivo dos dados no servidor para essa invocação de consulta específica.

Para ver mais informações, consulte "Processando respostas extensas" na página 299.

10

HP Universal CMDB API

Este capítulo inclui:

Conceitos

Tarefas

Referência

Conceitos

Convenções

Este capítulo usa as seguintes convenções:

- **UCMDB** refere-se ao banco de dados de Gerenciamento de Configuração Universal em si. **HP Universal CMDB** refere-se ao aplicativo.
- Os elementos e argumentos de método do UCMDB são escritos levando em conta maiúsculas e minúsculas do mesmo modo como são especificados nas interfaces.

Usando a HP Universal CMDB API

Use este capítulo junto com a documentação da API Java, disponível na Biblioteca de Documentação online.

A API do HP Universal CMDB é usada para integrar aplicativos ao Universal CMDB (CMDB). A API fornece métodos para:

- adicionar, remover e atualizar ECs e relações no CMDB
- recuperar informações sobre o modelo de classe
- executar cenários hipotéticos
- recuperar informações sobre elementos de configuração e relacionamentos

Métodos para recuperar informações sobre elementos de configuração e relacionamentos geralmente usam o TQL (Topology Query Language). Para ver detalhes, consulte "Topology Query Language" no *Guia de Modelagem do HP Universal CMDB*.

Usuários da API do HP Universal CMDB devem ter familiaridade com:

- linguagem de programação Java
- HP Universal CMDB

Esta seção inclui os seguintes tópicos:

Usos da API

A API é usada para cumprir inúmeros requisitos de negócios. Por exemplo, um sistema de terceiros pode consultar o modelo de classe para ver informações sobre os ECs (elementos de configuração) disponíveis. Para ver mais cenários de usos, consulte "Casos de uso" na página 383.

Permissões

O administrador fornece credenciais de logon para conexão com a API. O cliente da API precisa de nome de usuário e senha de um usuário de integração definido no CMDB. Esses usuários não representam pessoas que usam o CMDB, e sim aplicativos que se conectam ao CMDB.

Para ver detalhes, consulte "Criar um usuário de integração" na página 380.

Estrutura geral de um aplicativo

Há apenas uma fábrica estática, a UcmdbServiceFactory. Essa fábrica é o ponto de entrada de um aplicativo. A UcmdbServiceFactory expõe métodos getServiceProvider. Esses métodos retornam uma instância da interface **UcmdbServiceProvider**.

O cliente cria outros objetos usando métodos de interface. Por exemplo, para criar uma nova definição de consulta, o cliente:

1 obtém o serviço de consulta do objeto de serviço CMDB principal

2 obtém um objeto de fábrica de consulta do objeto de serviço

3 obtém uma nova definição de consulta da fábrica

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

Os serviços disponíveis no **UcmdbService** são:

Métodos de serviço	Uso
getClassModelService	informação sobre tipos de ECs e relações
getDDMConfigurationService	Configurar o sistema de Gerenciamento de Dependência e Descoberta
getDDMManagementService	Analisar e exibir o andamento, resultados e erros do sistema de Gerenciamento de Dependência e Descoberta
getImpactAnalysisService	Executar o cenário de análise de impacto (também conhecido como **correlação**).

Métodos de serviço	Uso
getQueryManagementService	Gerenciar acesso a consultas - salvar, excluir, listar existente. Fornece também validação de consulta e a descoberta de dependências de consultas.
getResourceBundleManagement Service	Marcação de recurso (serviços de "agrupamento"). Permite a criação explícita de novas marcas e a remoção de marcas de todos os recursos marcados.
getSoftwareSignatureService	Definir itens de software a serem descobertos pelo sistema de Gerenciamento de Dependência e Descoberta
getTopologyQueryService	Obter informações sobre o universo de TI
getTopologyUpdateService	Alterar informações no universo de TI
getViewService	Exibir o serviço de execução (definição de execução, salvamento de execução) e serviço de gerenciamento (salvar, excluir, listar existente). Fornece também validação da exibição e a descoberta de dependências.
getViewArchiveService	Exibir resultados de estatísticas. Permite salvar o resultado da exibição atual e recuperar resultados salvos anteriormente.

O cliente comunica-se com o servidor usando HTTP.

Tarefas

Colocar o arquivo Jar da API no caminho de classe

O uso desse conjunto de API exige o arquivo **ucmdb-api.jar**. Você pode baixar o arquivo inserindo http://localhost:8080 em um navegador da Web e clicando no link de **download do cliente da API**.

Coloque o arquivo jar no caminho de classe antes de compilar ou executar seu aplicativo.

Criar um usuário de integração

Você pode criar um usuário dedicado para integração entre outros produtos e o UCMDB. Esse usuário permite que um produto que use o SDK cliente do UCMDB seja autenticado no SDK servidor e execute as APIs. Aplicativos escritos com esse conjunto de API devem fazer logon usando credenciais de usuário de integração.

Cuidado: Apenas um usuário de integração pode se conectar ao CMDB usando esse conjunto de API. Uma tentativa de conexão por outros tipos de usuários poderá causar erros, mesmo quando a verificação LDAP for usada.

Para criar um usuário de integração:

1. Inicie o navegador da Web e insira o endereço do servidor, da seguinte maneira:

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

   Você pode precisar fazer logon com um nome de usuário e senha (os valores padrão são sysadmin/sysadmin).

2. Em UCMDB, clique em **service=UCMDB Security Services** para abrir a página JMX MBEAN View.

3. Localize a operação **CreateIntegrationUser**. Esse método aceita os seguintes parâmetros:
   - **customerId**. ID do cliente.
   - **username**. O nome do usuário de integração.
   - **password**. A senha do usuário de integração.
   - **dataStoreOrigin**. O nome do produto que usará esse usuário de integração.

   As operações a seguir são úteis para o gerenciamento do usuário de integração:
   - **DeleteIntegrationUser**. Exclui o usuário de integração indicado.
   - **ExportIntegrationUser**. Exporta o usuário de integração para um arquivo XML no caminho indicado (no servidor).
   - **getIntegrationUser**. Exibe informações do usuário de integração.
   - **changeIntegrationUserPassword**. Muda a senha do usuário de integração.
   - **canUserAuthenticate**. **isIntegrationUser** is **true**: é possível autenticar o usuário de integração com as credenciais fornecidas?
4. Clique em **Invocar**.

   Clique em **Back to MBean View** para criar mais usuários ou feche o console JMX.
5. Faça logon no UCMDB como administrador.
6. Na guia **Administração** , execute o **Gerenciador de Pacotes**.
7. Clique no ícone **Novo**.
8. Digite um nome para o novo pacote e clique em **Avançar**.
9. Na guia Seleção de Recurso, sob **Administração**, clique em **Usuários de Integração**.
10. Selecione um ou mais usuários que tenha criado usando o console JMX.
11. Clique em **Avançar** e em **Concluir**. Seu novo pacote aparecerá na lista Nome do Pacote no Gerenciador de Pacotes.

12 Implemente o pacote aos usuários que executarão os aplicativos API.

Para ver detalhes, consulte "Implantar um pacote" no *Guia de Administração do HP Universal CMDB*.

Observação:

A senha do usuário de integração é por cliente. Para criar um usuário de integração mais forte para uso entre clientes, utilize um **systemUser** com o sinalizador **isSuperIntegrationUser** definido como **verdadeiro**. Use os métodos **systemUser** (**createSystemUser**, **removeSystemUser**, **showAllSystemUsers**, **changeSystemUserPassword**, **canSuperIntegrationUserAuthenticate** e assim por diante).

Há dois usuários de sistema predefinidos; é recomendável mudar suas senhas após a instalaçãousando o método **changeSystemUserPassword**.

- **sysadmin**/**sysadmin**
- **UISysadmin**/**UISysadmin** (Esse usuário é também o superusuário de integração **SuperIntegrationUser**).

Se mudar a senha UISysadmin usando **changeSystemUserPassword**, você deverá executar o seguinte método: no console JMX, localize o serviço **UCMDB-UI:name=UCMDB Integration**. Execute **setCMDBSuperIntegrationUser** com o nome de usuário e a nova senha do usuário de integração.

Referência

Referência da API do HP Universal CMDB

Para ver a documentação completa sobre as APIs disponíveis, consulte "Introdução a APIs" na página 287.

Casos de uso

Os seguintes casos de uso pressupõem dois sistemas:

- Servidor HP Universal CMDB
- Um sistema de terceiros que contém um repositório dos elementos de configuração

Esta seção inclui os seguintes tópicos:

Populando o CMDB

Casos de uso:

- Um gerenciamento de ativos de terceiros atualiza o CMDB com informações disponíveis somente no gerenciamento de ativos.
- Inúmeros sistemas de terceiros populam o CMDB para criar um CMDB central que consiga controlar as alterações e executar análise de impacto.
- Um sistema de terceiros cria Elementos de Configuração e Relacionamentos de acordo com uma lógica de negócios de terceiros para aproveitar os recursos de consulta do UCMDB.

Consultando o CMDB

Casos de uso:

- Um sistema de terceiros obtém os Elementos de Configuração e Relacionamentos que representam o sistema SAP obtendo os resultados do TQL SAP.
- Um sistema de terceiros obtém a lista de servidores Oracle que foram adicionados ou alterados nas últimas cinco horas.
- Um sistema de terceiros obtém a lista de servidores cujo nome de host contém a subcadeia lab.
- Um sistema de terceiros localiza os elementos relacionados a um determinado EC obtendo seus vizinhos.

Consultando o modelo de classe

Casos de uso:

- Um sistema de terceiros permite que os usuários especifiquem o conjunto de dados que serão recuperados do CMDB. Uma interface do usuário pode ser criada com base no modelo de classe para mostrar aos usuários as propriedades possíveis e lhes solicitar os dados necessários. O usuário poderá escolher as informações que serão recuperadas.
- Um sistema de terceiros explora o modelo de classe quando o usuário não pode acessar a interface do usuário do UCMDB.

Analisando o impacto das alterações

Caso de uso:

Um sistema de terceiros gera saída de uma lista dos serviços comerciais que poderiam sofrer impacto de uma alteração em um host especificado.

Exemplos

Esta seção inclui os seguintes tópicos:

Exemplo de ponto de entrada

```
final String HOST_NAME = "localhost";
final int PORT = 8080;
UcmdbServiceProvider provider =
    UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
final String USERNAME = "integration_user";
final String PASSWORD = "integration_password";
Credentials credentials =
    provider.createCredentials(USERNAME, PASSWORD),
ClientContext clientContext = provider.createClientContext("Example");
UcmdbService ucmdbService = provider.connect(credentials, clientContext);
```

Exemplos de consulta

Os exemplos a seguir demonstram como obter uma definição de classe única e uma lista de todas as definições de TEC e seus atributos.

Recuperando uma definição de classe

```
ClassModelService classModelService
    = ucmdbService.getClassModelService();
String typeName = "disk";
ClassDefinition def =
    classModelService.getClassDefinition(typeName);
System.out.println("Type " + typeName + " is derived from type "
    + def.getParentClassName());
System.out.println("Has " + def.getChildClasses().size() +
    " derived types");
System.out.println("Defined and inherited attributes:");
for (Attribute attr : def.getAllAttributes().values()) {
    System.out.println("Attribute " + attr.getName() +
    " of type " + attr.getType());
}
```

Recuperando uma lista de definições e atributos de TEC

Esse exemplo consulta os atributos de um TEC e imprime seus nomes e tipos.

```
ClassModelService classModelService =
     ucmdbService.getClassModelService();
for (ClassDefinition def : classModelService.getAllClasses()) {
    System.out.println("Type " + def.getName() +
        " (" + def.getDisplayName() + ") is derived from type "
        + def.getParentClassName());
    System.out.println
        ("Has " + def.getChildClasses().size() + " derived types");
    System.out.println
        ("Defined and inherited attributes:");
    for (Attribute attr : def.getAllAttributes().values()) {
        System.out.println
          ("Attribute " + attr.getName() +
          " of type " + attr.getType());
    }
}
```

Exemplo de consulta de topologia

```
TopologyQueryService queryService =
    ucmdbService.getTopologyQueryService();
TopologyQueryFactory queryFactory =
    queryService.getFactory();
QueryDefinition queryDefinition =
    queryFactory.createQueryDefinition
        ("Get hosts with more than one network interface");
String hostNodeName = "Host";
QueryNode hostNode =

queryDefinition.addNode(hostNodeName).ofType("host").queryProperty("display_label"
);
QueryNode ipNode =
    queryDefinition.addNode("IP").ofType("ip").queryProperty("ip_address");
hostNode.linkedTo(ipNode).withLinkOfType("contained").atLeast(2);
Topology topology = queryService.executeQuery(queryDefinition);
Collection<TopologyCI> hosts = topology.getCIsByName(hostNodeName);
for (TopologyCI host : hosts) {
    System.out.println("Host " + host.getPropertyValue("display_label"));
    for (TopologyRelation relation : host.getOutgoingRelations()) {
        System.out.println
        ("  has IP " + relation.getEnd2CI().getPropertyValue("ip_address"));
    }
}
```

Exemplo de atualização de topologia

```
TopologyUpdateService topologyUpdateService =
    ucmdbService.getTopologyUpdateService();
TopologyUpdateFactory topologyUpdateFactory =
    topologyUpdateService.getFactory();
TopologyModificationData topologyModificationData =
    topologyUpdateFactory.createTopologyModificationData();
CI host = topologyModificationData.addCI("host");
host.setPropertyValue("host_key", "test1");
CI ip = topologyModificationData.addCI("ip");
ip.setPropertyValue("ip_address", "127.0.0.10");
ip.setPropertyValue("ip_domain", "DefaultDomain");
topologyModificationData.addRelation("contained", host, ip);
topologyUpdateService.create
    (topologyModificationData, CreateMode.IGNORE_EXISTING);
```

Exemplo de análise de impacto

```
ImpactAnalysisService impactAnalysisService =
    ucmdbService.getImpactAnalysisService();
ImpactAnalysisFactory impactFactory =
    impactAnalysisService.getFactory();
ImpactAnalysisDefinition definition =
    impactFactory.createImpactAnalysisDefinition();
definition.addTriggerCI(disk).withSeverity
    (impactFactory.getSeverityByName("Warning(2)"));
definition.useAllRules();
ImpactAnalysisResult impactResult =
    impactAnalysisService.analyze(definition);
AffectedTopology affectedCIs =
    impactResult.getAffectedCIs();
for (AffectedCI affectedCI : affectedCIs.getAllCIs()) {
     System.out.println("Affected " +
     affectedCI.getType() + " " + affectedCI.getId() +
     " - severity " + affectedCI.getSeverity());
}
```

Índice

A

B

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

U

V