HP Universal CMDB

Versão do software: 10.10

Guia de Referência do Desenvolvedor

Data de lançamento do documento: Novembro de 2013

Data de lançamento do documento: Novembro de 2013

Avisos Legais

Garantia

As únicas garantias para produtos e serviços HP estão estipuladas nas declarações de garantia expressa que acompanham esses produtos e serviços. Nenhum conteúdo deste documento deve ser interpretado como parte de uma garantia adicional. A HP não se responsabiliza por erros técnicos ou editoriais ou por omissões presentes neste documento.

As informações contidas neste documento estão sujeitas a mudanças sem aviso prévio.

Legenda de Direitos Restritos

Software de computador confidencial. Uma licença válida da HP é necessária para posse, utilização ou cópia. Consistentes com o FAR 12.211 e 12.212, o Software de Computador Comercial, a Documentação de Software de Computador e os Dados Técnicos para Itens Comerciais estão licenciados junto ao Governo dos Estados Unidos sob a licença comercial padrão do fornecedor.

Aviso de Direitos Autorais

Avisos de Marcas Comerciais

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Atualizações da Documentação

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

- Número de versão do software, que indica a versão do software.
- Data de lançamento do documento, que é alterada a cada 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 se você está utilizando a edição mais recente, vá para: **http://h20230.www2.hp.com/selfsolve/manuals**

Esse site exige que você se registre para obter um HP Passport e para se conectar. Para se registrar e obter uma ID do HP Passport, vá para: **http://h20229.www2.hp.com/passport-registration.html**

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

Você também receberá edições novas ou atualizadas se assinar o serviço de suporte adequado ao produto. Entre em contato com seu representante de vendas HP para saber mais detalhes.

Suporte

Visite o site de Suporte Online da HP Software em: **http://www.hp.com/go/hpsoftwaresupport**

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

O suporte on-line da HP Software fornece recursos de auto-ajuda aos clientes. Ele oferece uma maneira rápida e eficiente de acessar ferramentas de suporte técnico interativas necessárias para gerenciar seus negócios. Como um estimado cliente de suporte, você pode aproveitar o site de suporte para:

- Pesquisar documentos com informações de interesse
- Enviar e rastrear os casos de suporte e solicitações de aperfeiçoamentos
- Fazer download dos patches de software
- Gerenciar contratos de suporte
- Procurar contatos de suporte HP
- Revisar informações sobre os serviços disponíveis
- Participar de discussões com outros clientes de software
- Pesquisar e registrar-se para treinamentos de software

A maior parte das áreas de suporte exige que você se registre como usuário de um HP Passport e, em seguida, se conecte. Muitas também requerem um contrato de suporte ativo. Para se cadastrar e obter uma ID do HP Passport, acesse:

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

Para mais informações sobre níveis de acesso, vá para:

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

HP Software Solutions Now acessa o site de portal HPSW Solution and Integration. Este site permite que você explore as páginas de HP Product Solutions, que inclui uma lista completa das integrações entre os produtos HP, bem como uma lista de processos ITIL. A URL para este site é **http://h20230.www2.hp.com/sc/solutions/index.jsp**

Informações sobre este versão em PDF da ajuda on-line

Este documento é uma versão em PDF da ajuda on-line. Este arquivo em PDF é fornecido para que você possa imprimir facilmente vários tópicos das informações da ajuda ou ler a ajuda on-line em formato PDF. Este conteúdo foi originalmente criado para ser visto como ajuda on-line em um navegador da Web, por esta razão alguns tópicos não podem ser formatados corretamente. Alguns tópicos interativos podem não estar presentes nesta versão PDF. Esses tópicos podem ser impressos com êxito a partir da ajuda on-line.

Conteúdo

Conteúdo 4
Criando Adaptadores de Descoberta e Integração 11
Capítulo 1: Criação e desenvolvimento de adaptadores 12
Visão geral da criação e desenvolvimento de adaptadores 12
Criação de conteúdo 12
O ciclo de desenvolvimento do adaptador 13
Gerenciamento de fluxo de dados e integração 16
Associando valor comercial ao desenvolvimento da descoberta 17
Pesquisando requisitos de integração 18
Desenvolvendo conteúdo de integração 21
Desenvolvendo conteúdo de descoberta 23
Adaptadores de descoberta e componentes relacionados 23
Separando adaptadores 24
Implementar um adaptador de descoberta 25
Etapa 1: Criar um adaptador 28
Etapa 2: Atribuir um trabalho ao adaptador 34
Etapa 3: Criar código Jython 36
Configurar execução de processo remoto 36
Capítulo 2: Desenvolvimento de adaptadores Jython 38
Referência da API de Gerenciamento de Fluxo de Dados da HP 38
Criar código Jython 38
Usar arquivos JAR Java externos em Jython 39
Execução do código 39
Modificando scripts prontos 40
Estrutura do arquivo Jython 40
Importações 41
Função principal – DiscoveryMain 41
Definição de funções 42
Geração de resultados pelo script Jython 43

A sintaxe de ObjectStateHolder 43
Enviando grandes quantidades de dados 45
Instância do Framework 46
Localizando as credenciais corretas (para adaptadores de conexão) 49
Manipulando exceções de Java 51
Solucionando problemas de migração do Jython versão 2.1 para 2.5.3 52
Oferecer suporte a localização em adaptadores Jython 54
Adicionar suporte para um novo idioma 54
Alterar o idioma padrão 55
Determinar o conjunto de caracteres para codificação 56
Definir um novo trabalho para operar com dados localizados 56
Decodificar comandos sem uma palavra-chave 57
Trabalhar com bundles de recursos 58
Referência de API 59
Registrar código do DFM 61
Bibliotecas e utilitários Jython 63
Capítulo 3: Mensagem de erro 66
Visão geral de mensagens de erro 66
Convenções da criação de mensagens 66
Níveis de gravidade de erro 69
Capítulo 4: Desenvolvimento de adaptadores de banco de dados genéricos 71
Visão geral dos adaptadores de banco de dados genéricos 72
Consultas TQL para o adaptador de banco de dados genérico 72
Reconciliação 73
Hibernar como provedor de JPA 74
Preparar criação do adaptador 76
Preparar o pacote de adaptadores 81
Configurar o adaptador - Método mínimo 84
Configurar o arquivo adapter.conf 85
Exemplo: Preenchendo um nó e endereço IP usando o método simplificado 85
Configurar o adaptador - Método avançado 89

Implementar um plugin ... 94
Implantar o adaptador ... 97
Editar o adaptador ... 97
Criar um ponto de integração ... 97
Criar uma visualização ... 98
Calcular os resultados ... 98
Visualizar os resultados ... 98
Visualizar relatórios ... 98
Habilitar arquivos de log ... 98
Usar o Eclipse para mapear entre atributos de TEC e tabelas de banco de dados ... 99
Arquivos de configuração do adaptador ... 107
O arquivo adapter.conf ... 109
O arquivo simplifiedConfiguration.xml ... 110
O arquivo orm.xml ... 112
O arquivo reconciliation_types.txt ... 126
O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores) ... 126
O arquivo transformations.txt ... 128
O arquivo discriminator.properties ... 129
O arquivo replication_config.txt ... 130
O arquivo fixed_values.txt ... 130
Arquivo Persistence.xml ... 131
Conectar a banco de dados usando autenticação do NT ... 132
Conversores prontos ... 132
Plug-ins ... 138
Exemplos de Configuração ... 138
Arquivos de log do adaptador ... 147
Referências externas ... 149
Solução de problemas e limitações ... 149
Capítulo 5: Desenvolvimento de adaptadores Java ... 151
Visão geral do Federation Framework ... 151
Interação de adaptador e mapeamento com o Federation Framework ... 156

Fluxo do Federation Framework para consultas TQL federadas 157

Interações entre o Federation Framework, servidor, adaptador e mecanismo de mapeamento 158

Fluxo do Federation Framework para população 168

Interfaces de adaptador 169

Depurar recursos do adaptador 171

Adicionar um adaptador para uma nova fonte de dados externos 171

Criar um adaptador de amostra 180

Marcas e propriedades de configuração XML 181

A Interface DataAdapterEnvironment 183

OutputStream openResourceForWriting(String resourceName) throws FileNotFoundException; 183

InputStream openResourceForReading(String resourceName) throws FileNotFoundException; 183

Properties openResourceAsProperties(String propertiesFile) throws IOException; 184

String openResourceAsString(String resourceName) throws IOException; 185

public void saveResourceFromString(String relativeFileName, String value) throws IOException; 185

boolean resourceExists(String resourceName); 186

boolean deleteResource(String resourceName); 186

Collection<String> listResourcesInPath(String path); 186

DataAdapterLogger getLogger(); 186

DestinationConfig getDestinationConfig(); 187

int getChunkSize(); 187

int getPushChunkSize(); 187

ClassModel getLocalClassModel(); 187

CustomerInformation getLocalCustomerInformation(); 187

Objeto getSettingValue(String name); 187

Map<String, Object> getAllSettings(); 188

boolean isMTEnabled(); 188

String getUcmdbServerHostName(); 188

Capítulo 6: Desenvolvimento de adaptadores push 189

Desenvolvimento e implementação de adaptadores push 189

Criar um pacote de adaptador ... 190
Solução de Problemas ... 193
Práticas recomendadas de TQL para adaptadores push ... 194
Criar Mapeamentos ... 194
Compilar um Arquivo de Mapeamento ... 194
Preparar os arquivos de mapeamento ... 195
Escrever scripts Jython ... 198
Suporte à sincronização diferencial ... 202
Consultas SQL do Adaptador Push XML Genérico ... 203
Adaptador Push de Serviço Web Genérico ... 204
Referência do arquivo de mapeamento ... 223
Esquema do arquivo de mapeamento ... 225
Esquema de resultados de mapeamento ... 238
Personalização ... 242
Capítulo 7: Desenvolvendo adaptadores push genéricos aprimorados ... 244
Visão geral ... 244
O arquivo de mapeamento ... 244
O viajante do Groovy ... 247
Escrever scripts Groovy ... 250
Implementar interface PushConnector ... 251
Criar um pacote de adaptador ... 252
Esquema do arquivo de mapeamento ... 253
Usando APIs ... 260
Capítulo 8: Introdução a APIs ... 261
Visão geral sobre APIs ... 261
Capítulo 9: API do HP Universal CMDB ... 262
Convenções ... 262
Usando a API do HP Universal CMDB ... 262
Estrutura geral de um aplicativo ... 263
Colocar o arquivo Jar da API no classpath ... 266
Criar um usuário de integração ... 266

Casos de uso da API do UCMDB ... 268
Exemplos ... 269
Capítulo 10: API de serviço Web do HP Universal CMDB ... 271
Convenções ... 271
Visão geral da API de serviço Web do HP Universal CMDB ... 272
Chamar o serviço Web ... 275
Consultar o CMDB ... 275
Atualizar o UCMDB ... 279
Consultar o modelo de classe do UCMDB ... 280
getClassAncestors ... 280
getAllClassesHierarchy ... 281
getCmdbClassDefinition ... 281
Consulta para análise de impacto ... 282
Parâmetros gerais do UCMDB ... 282
Parâmetros de saída do UCMDB ... 285
Métodos de consulta do UCMDB ... 286
executeTopologyQueryByNameWithParameters ... 286
executeTopologyQueryWithParameters ... 287
getChangedCIs ... 288
getCINeighbours ... 289
getCIsByID ... 289
getCIsByType ... 290
getFilteredCIsByType ... 290
getQueryNameOfView ... 294
getTopologyQueryExistingResultByName ... 295
getTopologyQueryResultCountByName ... 295
pullTopologyMapChunks ... 295
releaseChunks ... 297
Métodos de atualização do UCMDB ... 297
addCIsAndRelations ... 298
addCustomer ... 299

deleteCIsAndRelations ... 299
removeCustomer ... 299
updateCIsAndRelations ... 300
Métodos de análise de impacto do UCMDB ... 300
calculateImpact ... 300
getImpactPath ... 301
getImpactRulesByNamePrefix ... 302
API de serviço Web de estado real ... 302
Casos de uso da API de Serviço Web do UCMDB ... 304
Exemplos ... 305
Capítulo 11: API Java do Gerenciamento de Fluxo de Dados ... 306
Usando a API Java do Gerenciamento de Fluxo de Dados ... 306
Capítulo 12: API do Serviço Web do Gerenciamento de Fluxo de Dados ... 308
Visão Geral da API do Serviço Web do Gerenciamento de Fluxo de Dados ... 308
Convenções ... 309
Chamando o serviço Web ... 309
Métodos de Gerenciamento de Fluxo de Dados e Estruturas de Dados ... 309
Estruturas de dados ... 310
Gerenciando métodos de trabalho de descoberta ... 311
Gerenciando métodos de acionador ... 312
Métodos de dados de sonda e domínio ... 314
Métodos de dados de credenciais ... 316
Métodos de atualização de dados ... 318
Amostra de código ... 320
Exemplo de adição de credenciais ... 323
Agradecemos seu feedback! ... 327

Criando Adaptadores de Descoberta e Integração

Capítulo 1: Criação e desenvolvimento de adaptadores

Este capítulo inclui:

Visão geral da criação e desenvolvimento de adaptadores 12

Criação de conteúdo 12

Desenvolvendo conteúdo de integração 21

Desenvolvendo conteúdo de descoberta 23

Implementar um adaptador de descoberta 25

Etapa 1: Criar um adaptador 28

Etapa 2: Atribuir um trabalho ao adaptador 34

Etapa 3: Criar código Jython 36

Configurar execução de processo remoto 36

Visão geral da criação e desenvolvimento de adaptadores

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 prático do HP 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, informações 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.)

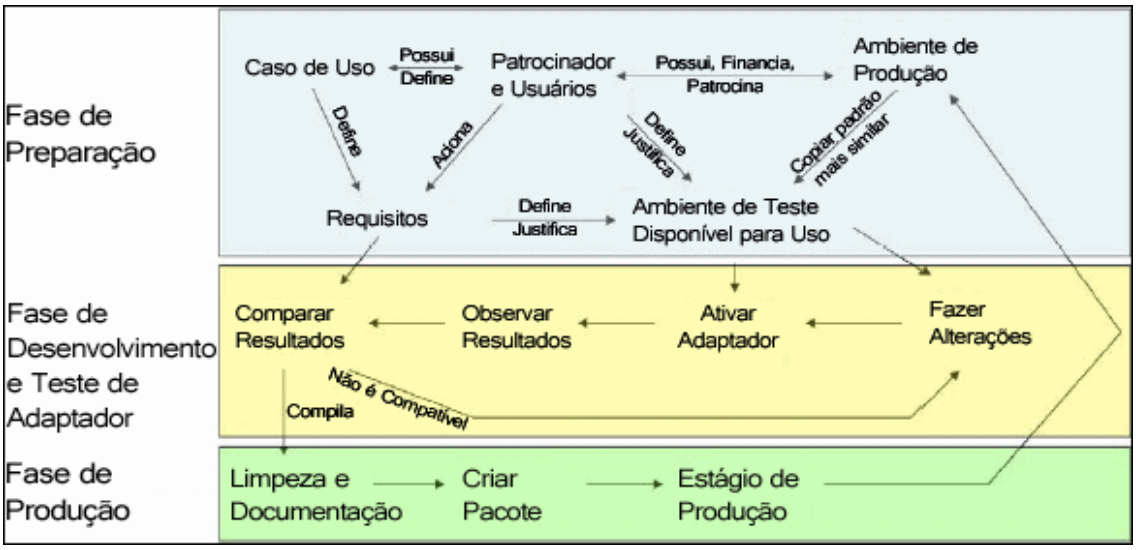
Criação de conteúdo

Esta seção inclui:

- "O ciclo de desenvolvimento do adaptador" abaixo
- "Gerenciamento de fluxo de dados e integração" na página 16
- "Associando valor comercial ao desenvolvimento da descoberta" na página 17
- "Pesquisando requisitos de integração" na página 18

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.



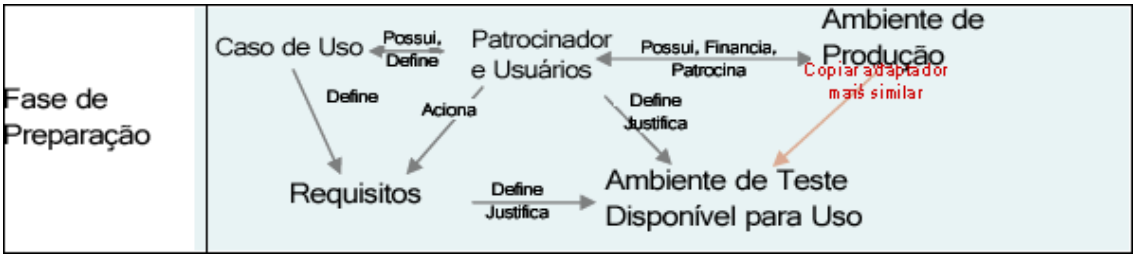
Cada fase do desenvolvimento do adaptador baseia-se na fase anterior.

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 "Package Manager" 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" na página seguinte
- "Desenvolvimento e teste do adaptador" na página seguinte
- "Empacotamento e produtização do adaptador " na página 15

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 obter detalhes, consulte "Painel Recursos" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

2. Pesquisar o método que o adaptador deve usar para coletar dados:

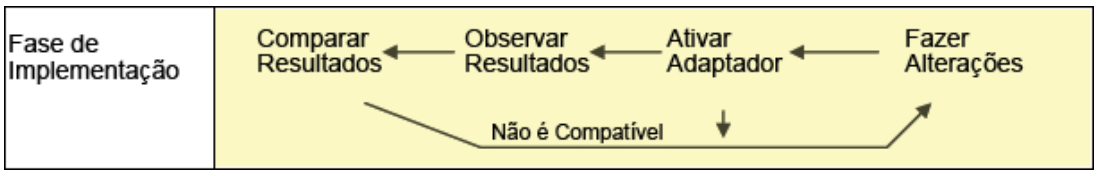
   - Use ferramentas/protocolos externos para obter os dados
   - Desenvolva como o adaptador deve criar ECs baseado nos dados
   - Você já sabe como deve ser um adaptador similar

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. Copiar o pacote inteiro.

5. Descompacte o conteúdo do pacote 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

A fase 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 obter detalhes, consulte "Package Manager" 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 obter detalhes, consulte "Package Manager" 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 comuns do CMDB 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

Tipo de integração	Exemplo de referência a reutilizar
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:

- 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 requisitos de 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 obter detalhes, consulte "Visão geral da criação e desenvolvimento de adaptadores" na página 12.

Esta seção inclui os seguintes tópicos:

- "Alterando um adaptador existente" abaixo
- "Escrevendo um novo adaptador" na página seguinte
- "Pesquisa de modelo" na página seguinte
- "Pesquisa da tecnologia" na página seguinte
- "Diretrizes para escolher modos de acesso aos dados" na página 20
- "Resumo" na página 20

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 com suporte atualmente do HP Universal CMDB em **Gerenciamento de Fluxo de Dados > Configuração do Data Flow Probe > painel Domínios e Sondas**. Para obter detalhes sobre cada protocolo, consulte a seção que descreve os protocolos com suporte em *HP UCMDB Discovery and Integrations Content Guide*.

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?

  Se você responder sim a essas perguntas, o **Preenchimento** é necessário.

- 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?

  Se você responder sim a essas perguntas, a **Federação** é necessária.

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

  Se você responder sim a essas perguntas, 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*.

Cinco opções diferentes estão disponíveis para criar adaptadores de integração:

1. Adaptador Jython:

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

   Para obter detalhes, consulte "Desenvolvimento de adaptadores Jython" na página 38.

2. 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 a uma única fonte de dados ou destino.

Para obter detalhes, consulte "Desenvolvimento de adaptadores Java" na página 151.

3. 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 mapeamento entre classes do UCMDB e colunas de banco de dados.
   - Apropriado para trabalhos que conectam a uma única fonte de dados.

   Para obter detalhes, consulte "Desenvolvimento de adaptadores de banco de dados genéricos" na página 71.

4. 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 a um único destino de dados.
   - Usado para o envio por push de dados.

   Para obter detalhes, consulte "Desenvolvimento de adaptadores push" na página 189.

5. Adaptador push genérico aprimorado:

   - Todos os recursos acuma do Adaptador Push Genérico
   - Um adaptador baseado em elemento raiz
   - Mapeia uma estrutura de dados do UCMDB para uma estrutura de dados de árvore de destino

Para obter detalhes, consulte "Desenvolvendo adaptadores push genéricos aprimorados" na página 244.

A tabela a seguir mostra as características de cada adaptador:

Fluxo/adaptador	Adaptador Jython	Adaptador Java	Adaptador de banco de dados genérico	Adaptador push genérico	Adaptador push genérico aprimorado
População	✓	✓	✓	×	×
Federação	×	✓	✓	×	×
Push de dados	×	✓	×	✓	✓

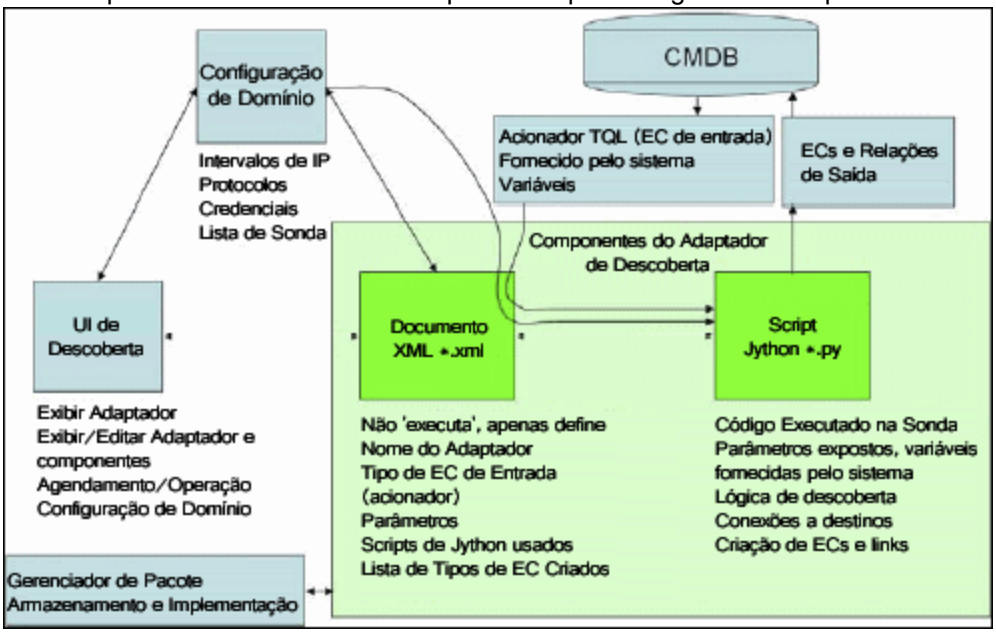
Desenvolvendo conteúdo de descoberta

Esta seção inclui:

- "Adaptadores de descoberta e componentes relacionados " abaixo
- "Separando adaptadores" na página seguinte

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 Universal Discovery, Gerenciamento do Adaptador e Configuração do Data Flow Probe 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

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.

Implementar 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. **Criar um adaptador.**

   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 obter detalhes, consulte "Etapa 1: Criar um adaptador" na página 28.

2. **Criar um assistente de Descoberta.**

   Você configura um trabalho com informações de programação e uma consulta acionadora. Para obter detalhes, consulte "Etapa 2: Atribuir um trabalho ao adaptador" na página 34.

3. **Editar código de descoberta.**

   Você pode editar o Jython ou o código Java contido nos arquivos do adaptador e que se refere à estrutura do DFM. Para obter detalhes, consulte "Etapa 3: Criar código Jython" na página 36.

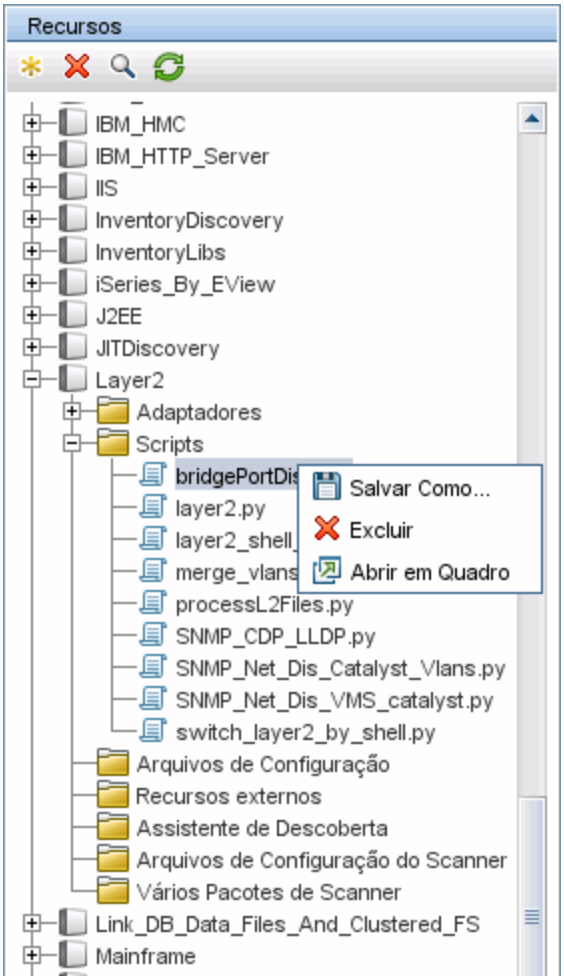
Para escrever novos adaptadores, você cria os componentes acima, e cada um deles é automaticamente associado ao componente da etapa precedente. Por exemplo, após você criar um trabalho e selecionar o adaptador relevante, o arquivo do adaptador será associado ao trabalho.

Código do adaptador

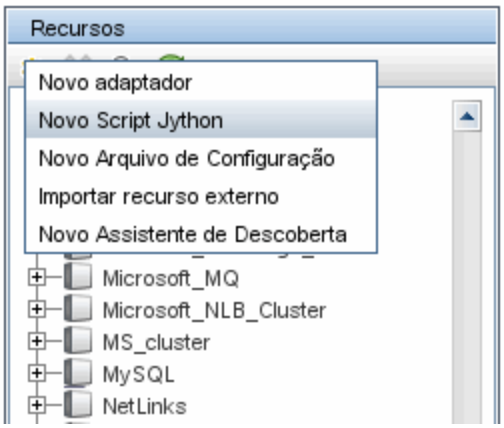
A implementação da conexão ao sistema remoto, a consulta aos seus dados e o seu mapeamento como dados do CMDB são executados pelo código Jython. Por exemplo, o código contem a lógica para a conexão ao banco de dados e a extração de seus dados. Nesse caso, o código espera receber uma URL do JDBC, nome 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.

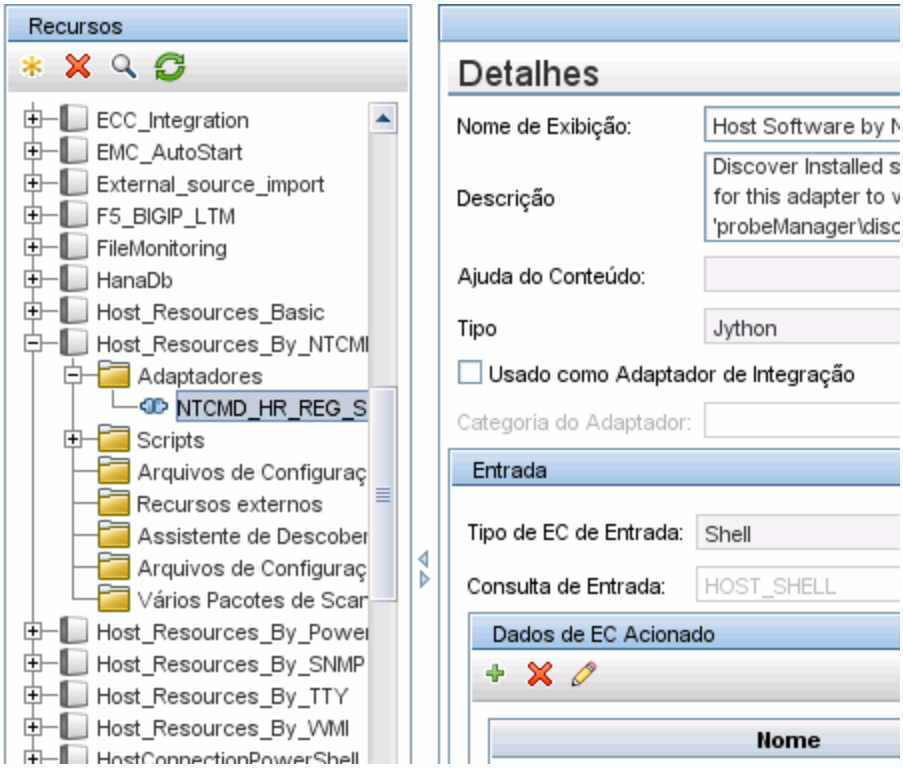
Um adaptador pode conter uma lista de scripts a serem usados ao executar uma descoberta. Ao criar um novo adaptador, você geralmente cria um novo script e o atribui ao adaptador. Um novo script inclui modelos básicos, mas você pode usar um dos outros scripts como modelos clicando com o botão direito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



Para ver detalhes sobre como escrever novos scripts Jython, consulte "Etapa 3: Criar código Jython" na página 36. 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 "Janela 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 Conceitos de Gerenciamento de Fluxo de Dados 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)" abaixo
- "Definir saída do adaptador" na página 30
- "Substituir parâmetros do adaptador" na página 32
- "Substituir seleção de sonda - Opcional" na página 33
- "Configurar um classpath para um processo remoto - Opcional" na página 34

1. ## 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 obter detalhes, consulte Como 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.

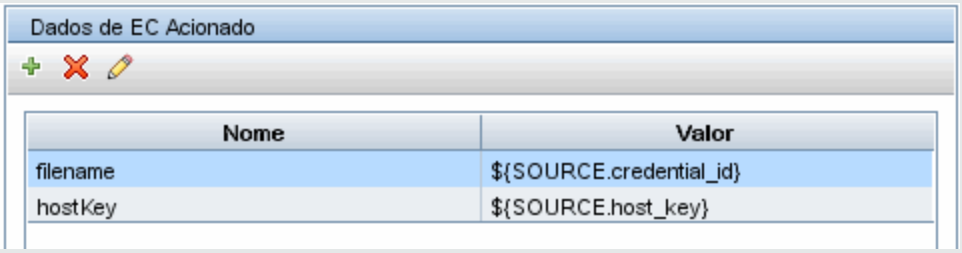
- Se o valor dos dados de destino é um lista, você pode definir o número de itens na lista a ser enviada à sonda. Para defini-lo, adicione dois-pontos (:) após o valor padrão seguido pelo número de itens. Se não houver nenhum valor padrão para os dados de destino, insira dois dois-pontos.

  Por exemplo, se os dados de destino a seguir estiverem inseridos: `name=portId, value= ${PHYSICALPORT.root_id:NA:1}` ou `name=portId, value= ${PHYSICALPORT.root_id::1}`, apenas a primeira porta da lista de portas será enviada à sonda.

Exemplo da substituição de variáveis por dados reais:

Nesse exemplo, as variáveis substituem os dados do EC **IpAddress** pelos valores verdadeiros das instâncias reais do EC **IpAddress** em seu sistema.

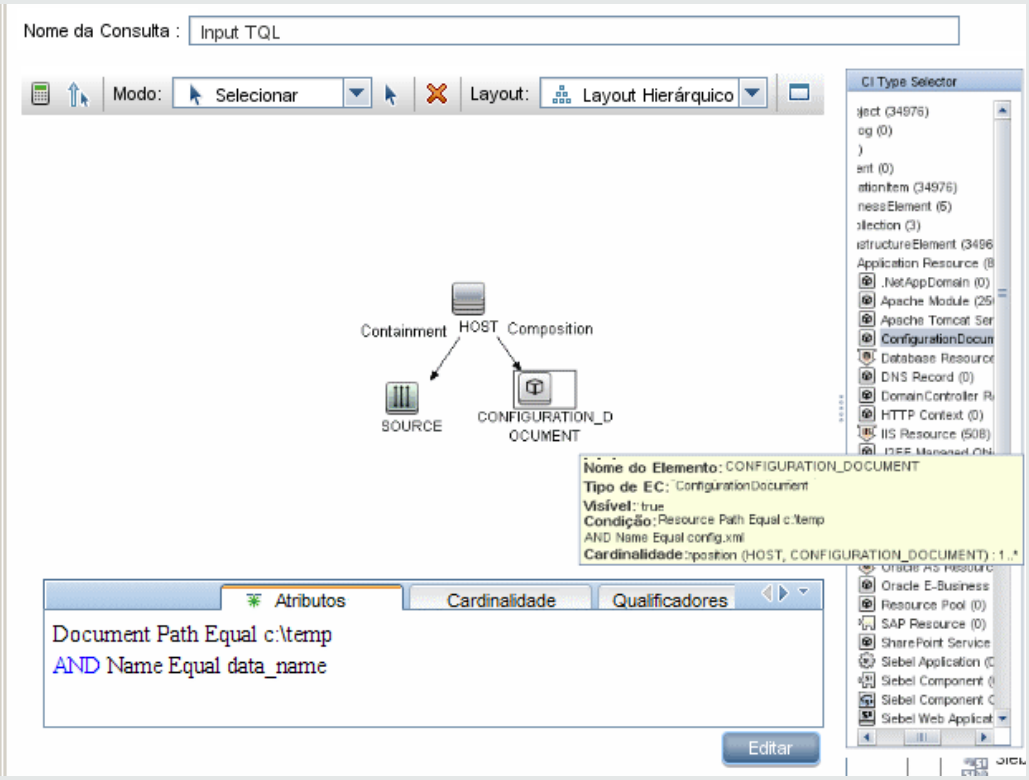
Os dados de EC acionado do EC **IpAddress** incluem uma variável `fileName`. Essa variável permite a substituição do nó **CONFIGURATION_DOCUMENT** na TQL de entrada pelos valores reais do arquivo de configuração situado em um host:



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 o DFM Framework para recuperar os valores reais das variáveis definidas:

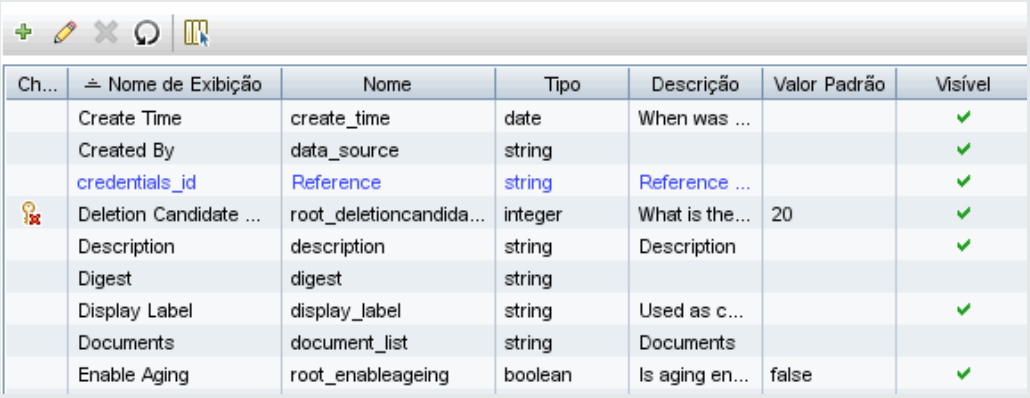
```
Framework.getTriggerCIData (“ip_address ")
```

As variáveis `fileName` e `path` usam os atributos `data_name` e `document_path` do nó **CONFIGURATION_DOCUMENT** (definido no Editor de Consulta de Entrada – consulte o exemplo anterior).



Clique na miniatura para exibir a imagem em tamanho integral.

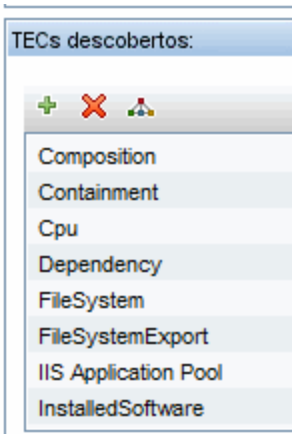
As variáveis `Protocol`, `credentialsId` e `ip_address` usam os atributos `root_class`, `credentials_id` e `application_ip`:



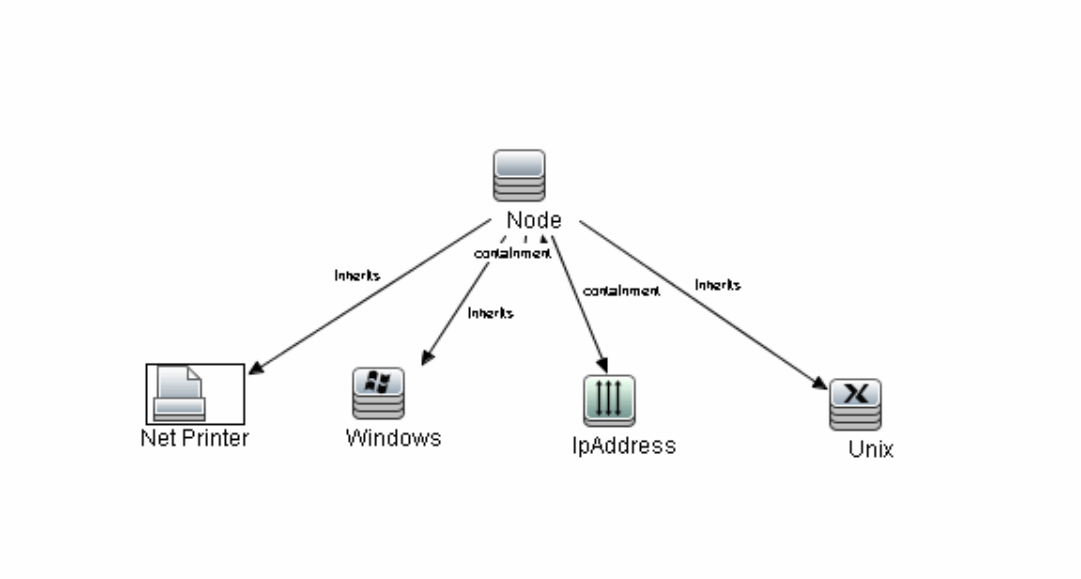
Ch...	Nome de Exibição	Nome	Tipo	Descrição	Valor Padrão	Visível
	Create Time	create_time	date	When was ...		✔
	Created By	data_source	string			✔
	credentials_id	Reference	string	Reference ...		✔
	Deletion Candidate ...	root_deletioncandida...	integer	What is the...	20	✔
	Description	description	string	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	✔

2. **Definir saída do adaptador**

A saída do adaptador é uma lista de ECs descobertos (**Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador > guia Definição do Adaptador > 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**):



Os ECs descobertos são retornados pelo código do DFM (isto é, o script Jython) no formato `ObjectStateHolderVector` do UCMDB. Para obter detalhes, consulte "Geração de resultados pelo script Jython" na página 43.

Exemplo da saída do adaptador:

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

a. Acesse **Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador**.

b. No painel Recursos, selecione **Rede > Adaptadores > NSLOOKUP_on_Probe**.

c. Na guia Definição do Adaptador, localize o painel TECs Descobertos.

d. Os TECs que devem fazer parte da saída do adaptador estarão listados. Adicione ou remova TECs da lista. Para obter detalhes, consulte "Guia Definição do Adaptador" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

3. ## 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`

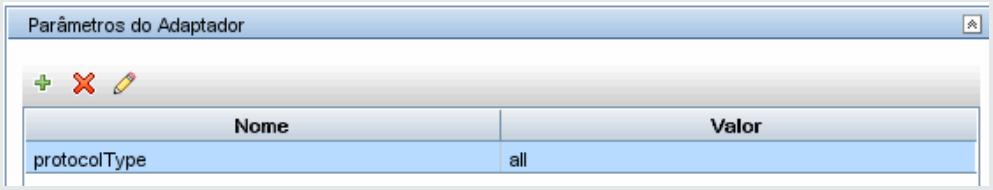
Exemplo da substituição de um parâmetro do adaptador:

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.

a. Acesse **Gerenciamento de Fluxo de Dados > Gerenciamento do Adaptador**.

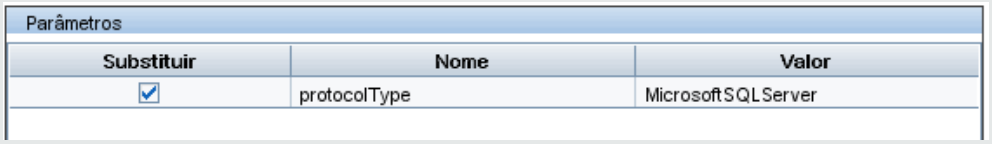
b. No painel Recursos, selecione **Database_Basic > Adaptadores > SQL_NET_Dis_Connection**.

c. Na guia Definição do Adaptador, localize o painel **Parâmetros do Adaptador**. O parâmetro `protocolType` tem o valor **todos**:



d. 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**.

e. 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 Guia Definição 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.

4. **Substituir seleção de sonda - Opcional**

No servidor do UCMDB, há um mecanismo de distribuição que leva os ECs acionadores recebidos pelo UCMDB e escolhe automaticamente qual sonda deve executar o trabalho para cada EC acionador de acordo com uma das opções a seguir:

- **Para o tipo de EC de endereço IP:** Adote a sonda definida para esse IP.
- **Para o tipo de EC de software em execução:** Use os atributos **application_ip** e **application_ip_domain** e escolha a sonda definida para o IP domínio relevante.
- **Para outros tipos de EC:** Adote o IP do nó de acordo com o nó relacionado do EC (se ele existir).

A seleção de sonda automática é feita de acordo com o nó relacionado do EC. Após obter o nó relacionado ao EC, o mecanismo de distribuição escolhe um dos IPs do nó e escolhe a sonda de acordo com as definições do escopo de rede da Sonda.

Nos casos a seguir, você precisa especificar a sonda manualmente e não usar o mecanismo de distribuição automático:

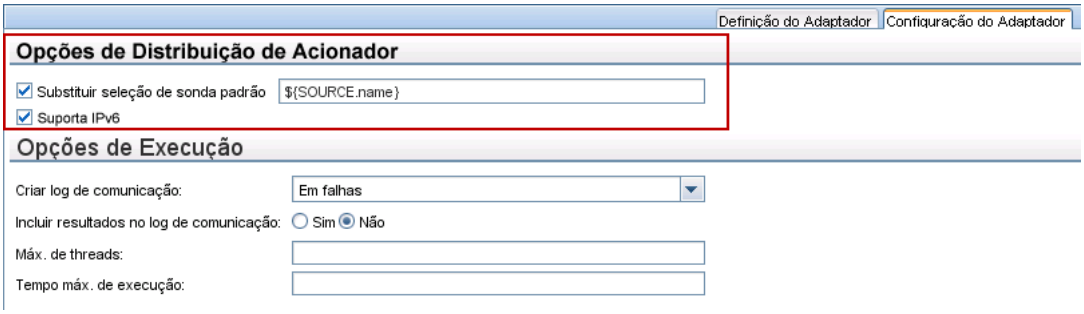
- Você já sabe qual sonda deve ser executada para o adaptador e não precisa do mecanismo de distribuição automático para selecionar a sonda (por exemplo, se o EC acionador for o gateway da sonda).
- A seleção de sonda automática pode falahr. Isso pode acontecer nas seguintes situações:
  - Um EC acionador não possui um nó relacionado (como o TEC de rede).
  - Um nó do EC acionador possui IPs múltiplos, cada um pertencente a uma sonda diferente.

Para especificar manualmente a sonda que deve ser usada com o adaptador:

- Selecione o adaptador e clique na guia **Gerenciamento do Adaptador**.
- Em **Opções de Distribuição de Acionador**, selecione **Substituir seleção de sonda padrão**.
- Na caixa, insira a Sonda em um dos seguintes formatos:

Nome da sonda	O nome da sonda
Endereço IP	O endereço IP da sonda — pode ser definido em formato IPv4 ou IPv6
IP,Domínio	Formato IPv4: **16.59.63.86,DefaultDomain** Formato IPv6: **2001:0:9d46:953c:34a9:1e6b:f2ff:fffe,CustomDomain**
Nome do domínio	O domínio a partir do qual a Sonda deve ser selecionada.

Por exemplo:



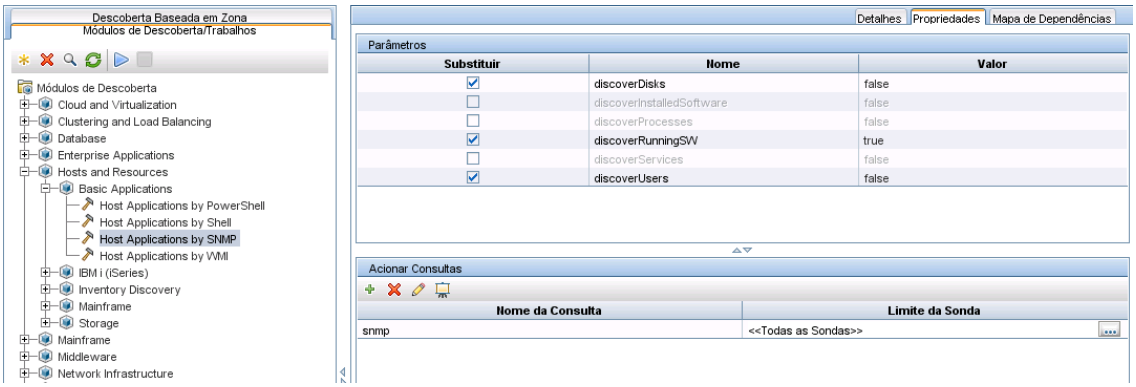
5. **Configurar um classpath para um processo remoto - Opcional**

   Para obter detalhes, consulte "Configurar execução de processo remoto" na página 36.

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, conforme a imagem abaixo.



Escolher um TQL acionador

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.

Definir informações de agendamento

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 obter detalhes, consulte Discovery Scheduler Dialog Box no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Substituir parâmetros do adaptador

Ao configurar um trabalho, você pode substituir os parâmetros do adaptador. Para obter detalhes, consulte "Substituir parâmetros do adaptador" na página 32.

Etapa 3: 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

Para obter detalhes, consulte "Criar código Jython" na página 38.

Configurar execução de processo remoto

Você pode executar a descoberta para um trabalho de descoberta em um processo separado do processo do Data Flow Probe.

Por exemplo, você pode executar o trabalho em um processo remoto separado se o trabalho usa bibliotecas **.jar** que são de uma versão diferente das bibliotecas da sonda ou incompatíveis com as bibliotecas da sonda.

Você também pode executar o trabalho em um processo remoto separado se o trabalho potencialmente consumir muita memória (apresentar muitos dados) e você desejar isolar a sonda da possíveis problemas **OutOMemory**.

Para configurar um trabalho para ser executado como um processo remoto, defina os seguintes parâmetros no arquivo de configuração do adaptador:

Parâmetro	Descrição
remoteJVMArgs	Parâmetros JVM para o processo de Java remoto.
runInSeparateProcess	Quando definido como **verdadeiro**, o trabalho de descoberta será executado em um processo separado.

Parâmetro	Descrição

remoteJVMClasspath	(Opcional) Permite a personalização do classpath do processo remoto, substituindo o classpath da sonda padrão. Isso é útil se houver uma incompatibilidade de versão entre os jars da sonda e os jars personalizados necessários para a descoberta definida pelo cliente.
	
	Se o parâmetro **remoteJVMClasspath** não estiver definido ou estiver vazio, o classpath da sonda padrão será usado.
	
	Se você desenvolver um novo trabalho de descoberta e você desejar garantir que a versão da biblioteca do jar da sonda não colida com as bibliotecas do jar do trabalho, você deve usar pelo menos o classpath mínimo necessário para executar a descoberta básica. O classpath mínimo é definido no arquivo **DataFlowProbe.properties** no parâmetro **basic_discovery_minimal_classpath**.
	
	Exemplos de personalização de **remoteJVMClasspath**:
	
	• Para pré-anexar ou anexar jars personalizados ao classpath da sonda padrão, personalize o parâmetro **remoteJVMClasspath** como a seguir:
	
	`custom1.jar;%classpath%;custom2.jar -`
	
	Nesse caso, **custom1.jar** é colocado antes do classpath da sonda padrão e **custom2.jar** é anexado ao classpath da sonda.
	
	• Para usar o classpath mínimo, personalize o parâmetro **remoteJVMClasspath** como a seguir:
	
	`custom1.jar;%minimal_classpath%;custom2.jar`

Capítulo 2: Desenvolvimento de adaptadores Jython

Este capítulo inclui:

Referência da API de Gerenciamento de Fluxo de Dados da HP 38

Criar código Jython 38

Oferecer suporte a localização em adaptadores Jython 54

Registrar código do DFM 61

Bibliotecas e utilitários Jython 63

Referência da API de Gerenciamento de Fluxo de Dados da HP

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

<diretório de instalação do UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_JavaDoc\index.html

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 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

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 Universal Discovery devem ser compatíveis com Jython versão 2.5.3.
- Para ver a documentação completa sobre as APIs disponíveis, consulte *Referência da API de Gerenciamento de Fluxo de Dados da HP*.

Esta seção inclui os seguintes tópicos:

- "Usar arquivos JAR Java externos em Jython" abaixo
- "Execução do código" abaixo
- "Modificando scripts prontos" na página seguinte
- "Estrutura do arquivo Jython" na página seguinte
- "Geração de resultados pelo script Jython" na página 43
- "Instância do Framework" na página 46
- "Localizando as credenciais corretas (para adaptadores de conexão)" na página 49
- "Manipulando exceções de Java" na página 51

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_uti
ls)
```

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

```
# implement customer specific processing for 'servername' attribute of httpplugi
n
#
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 f
ind. this join can't be done with IP address:port
                # because multiple apps on a web server share the same IP:port f
or multiple websphere applications
                logger.debug('httpcontext_webapplicationserver attribute has bee
n changed from [' + servername + '] to [' + transportHN[:5] + '] to facilitate w
ebsphere enrichment')
                servername = transportHN[:5]
        return servername
```

Salve o script externo na pasta External Resources. Para obter detalhes, consulte Resources Pane 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 "Package Manager" 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:

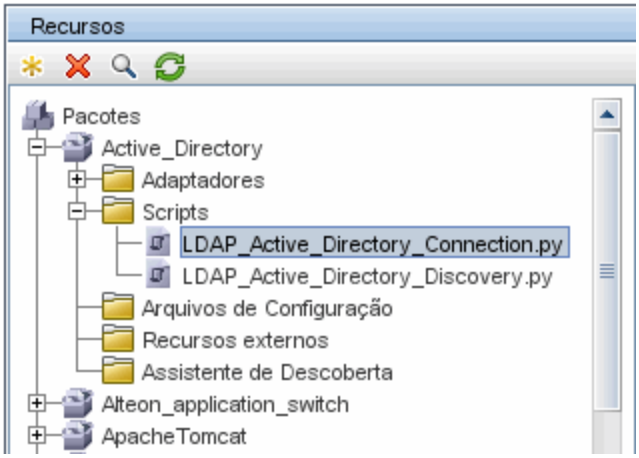
```
def DiscoveryMain(Framework):
```

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,str
ing'
    resultSet = client.executeQuery(data_name_mib)
    while resultSet.next():
        UserName = resultSet.getString(2)
        ########## send object ###############
        OSUserOSH = doOSUserOSH(UserName)
        OSUserOSH.setContainer(_hostObj)
        OSHVResult.add(OSUserOSH)
def DiscoveryMain(Framework):
    OSHVResult = ObjectStateHolderVector()
    try:
        client = Framework.createClient(Framework.getTriggerCIData
```

```
(BaseClient.CREDENTIALS_ID))
    except:
        Framework.reportError('Connection failed')
    else:
        doQueryOSUsers(client, OSHVResult)
        client.close()
    return OSHVResult
```

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 "Link" 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 obter detalhes, consulte CI Type Relationships 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" na página 65.

Enviando grandes quantidades de dados

Enviar grande quantidades de dados (normalmente mais de 20 KB) é difícil de processar no UCMDB. Dados desse tamanho devem ser divididos em blocos menores antes do envio ao UCMDB. Para que todos os blocos sejam inseridos corretamente no UCMDB, cada bloco precisa conter informações de identificação obrigatórias para os ECs do bloco. Esse é um cenário comum ao desenvolver integrações Jython. O método sendObjects é usado para enviar os resultados em blocos . Se o script Jython envia um grande número de resultados (o valor padrão será 20.000, mas esse valor pode ser configurado no "Arquivo DataFlowProbe.properties" usando a chave **appilog.agent.local.maxTaskResultSize**), ele deve agrupar os resultados em blocos de acordo com sua topologia. Esse agrupamento deve ser realizado levando-se em conta regras de identificação para que os resultados sejam inseridos corretamente no UCMDB. Se o script Jython não agrupar os resultados, a sonda tentará agrupá-los em partes. No entanto, isso pode levar a um desempenho insatisfatório para um grande conjunto de resultados.

Observação: O agrupamento em partes deve ser usado para adaptadores de integração Jython e não para trabalhos de descoberta regulares. Isso porque trabalhos de descoberta normalmente descobrem informações com relação a um acionamento específico e não enviam grandes quantias de informações. Com as integrações Jython, grandes quantias de dados são descobertas no único acionamento da integração.

Também é possível usar o agrupamento em partes para um pequeno número de resultados. Nesse caso, há um relacionamento entre ECs em diferentes partes e o desenvolvedor do script Jython tem duas opções:

- Enviar o EC inteiro e todas as suas informações de identificação novamente em todas as partes que contêm um vínculo para ele.

- Use a ID do UCMDB do EC. Para isso, o script Jython precisa esperar que cada parte seja processada no servidor UCMDB para obter as IDs do UCMDB. Para habilitar esse modo (chamado de envio de resultado assíncrono), adicione a tag SendJythonResultsSynchronously ao adaptador. Essa tag garante que, quando você terminar de enviar a parte, as IDs do UCMDB dos ECs da parte já tenham sido recebidas pela sonda. O desenvolvedor do adaptador pode usar as IDs do UCMDB para gerar a próxima parte. Para usar as IDs do UCMDB, use a API de estrutura getIdMapping.

Exemplo de uso de getIdMapping

Na segunda parte, você envia nós. No segundo bloco, você envia processos. O contêiner raiz do processo é um nó. Em vez de enviar o objectStateHolder inteiro do nó no atributo root_container do processo, você pode obter a ID do UCMDB do nó usando a API getIdMapping e usar somente a ID do nó no atributo root_container do atributo para tornar a parte menor.

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 obter detalhes, consulte "Referência da API de Gerenciamento de Fluxo de Dados da HP" na página 38.

O uso correto da instância do Framework é transmiti-la como argumento para cada método que a usa.

Exemplo:

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

Esta seção descreve os principais usos do Framework:

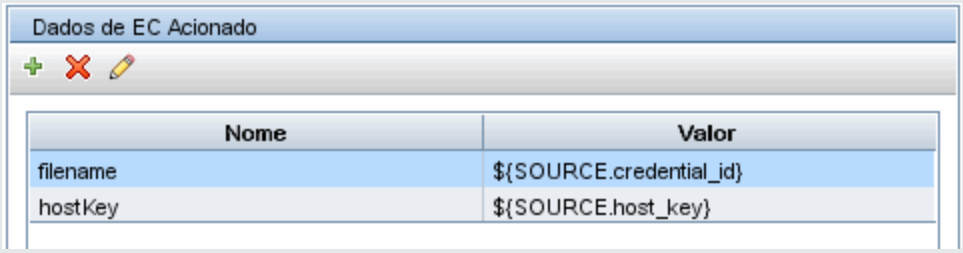
- "Framework.getTriggerCIData(String attributeName)" na página seguinte
- "Framework.createClient(credentialsId, props)" na página seguinte
- "Framework.getParameter (String parameterName)" na página 48
- "Framework.reportError(String message) e Framework.reportWarning(String message)" na página 49

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 *HP UCMDB Discovery and Integrations Content Guide*.

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:

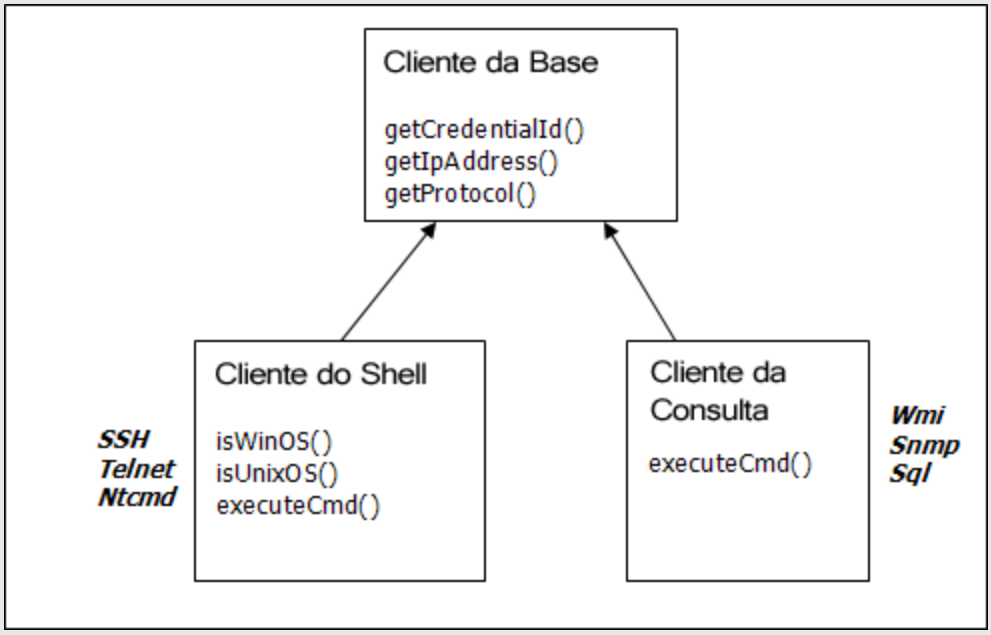
Para criar um cliente WMI e executar uma consulta WMI usando o cliente:

```
wmiClient = Framework.createClient(credential)
resultSet = wmiClient. executeQuery("SELECT TotalPhysicalMemory
                                                    FROM    Win32_Logical
MemoryConfiguration")
```

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 obter detalhes sobre os clientes e suas APIs aceitas, consulte BaseClient, ShellClient e QueryClient no DFM Framework. Esses arquivos estão localizados na seguinte pasta:

<Diretório Raiz do UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

Framework.getParameter (String parameterName)

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

Parâmetros

Substituir	Nome	Valor
☑	protocolType	MicrosoftSQLServer

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.createClient(Framework.getTriggerCIData(BaseClient.CR
EDENTIALS_ID))

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 "Mensagem de erro" na página 66.

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 `Framework.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).

Observação: O acesso a dados confidenciais (senhas, chaves privadas e assim por diante) é bloqueado para os seguintes tipos de protocolos:

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

A utilização desses tipos de protocolos deve ser concluída usando clientes dedicados.

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 24.

Exemplo

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

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

TRUE = 1
FALSE = 0

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

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

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

          connected = 0
          # Go over all the protocol credentials
          for protocol in protocols:
              client = None
              try:
                  try:
                      logger.debug('try to get snmp agent for: %s:%s' % (ip_
```

```
        address, ip_domain))
                    if (isClient == TRUE):
                        properties = Properties()
                        properties.setProperty(CollectorsConstants.DESTINA
        TION_DATA_IP_ADDRESS, ip_address)
                        properties.setProperty(CollectorsConstants.DESTINA
        TION_DATA_IP_DOMAIN, ip_domain)
                        client = Framework.createClient(protocol, properti
        es)
                    else:
                        properties = Properties()
                        properties.setProperty(CollectorsConstants.DESTINA
        TION_DATA_IP_ADDRESS, ip_address)
                        client = Framework.createClient(protocol, properti
        es)
                    logger.debug('Running test connection queries')
                    testConnection(client)
                    Framework.saveState(protocol)
                    logger.debug('got snmp agent for: %s:%s' % (ip_address,
        ip_domain))
                    isMultiOid = client.supportMultiOid()
                    logger.debug('snmp server isMultiOid state=%s' %isMult
        iOid)

                 client.close()
                 client = None

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

    return (_vector, error)
```

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.

Observação: É muito importante importar a classe de exceção de base Java conforme mostrado no seguinte exemplo devido à presença da classe de exceção de base em Python com o mesmo nome.

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

Solucionando problemas de migração do Jython versão 2.1 para 2.5.3

O Universal Discovery usa agora Jython versão 2.5.3. Todos os scripts prontos para o uso foram migrados adequadamente. Se você desenvolveu seu próprio script Jython antes desse upgrade para uso por Descoberta, você pode executá-lo nos seguintes problemas e terá que fazer as correções indicadas.

Observação: Você deve ser um desenvolvedor experiente em Jython para fazer essas alterações.

Formatação de Cadeia

- **Mensagem de erro:** `TypeError: argumento int necessário`
- **Causa possível:** Usar formatação de cadeia para inteiro decimal a partir de variável de cadeia contendo dados de inteiros.
- **Código Jython 2.1 problemático:**

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

- **Corrigir código Jython 2.5.3:**

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

  or

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

Verificando Tipo de Cadeia

O código abaixo pode não funcionar corretamente se a entrada contiver cadeias unicode:

- **Código Jython 2.1 problemático:** `isinstance(unicodeStringVariable,'')`
- **Corrigir código Jython 2.5.3:** `isinstance(unicodeStringVariable,basestring)`

  A comparação deve ser concluída com `basestring` para testar se um objeto é uma instância de `str` ou unicode.

Caractere não ASCII em arquivo

- **Mensagem de erro:**
  `SyntaxError: Caractere não ASCII em arquivo 'x', mas sem codificação declarada; consulte http://www.python.org/peps/pep-0263.html para obter mais detalhes`
- **Corrigir código Jython 2.5.3:** (add this to the first line in the file)

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

Importar subpacotes

- **Mensagem de erro:**
  `AttributeError: objeto 'module' não tem atributo 'sub_package_name'`
- **Causa possível:** Um subpacote é importado sem especificar explicitamente o nome do subpacote na instrução de importação.
- **Código Jython 2.1 problemático:**

  ```
  importar um
  print dir(a.b)
  ```

  O subpacote não foi importado explicitamente.
- **Corrigir código Jython 2.5.3:**

  ```
  import a.b
  ```

  or

  ```
  de a importar b
  ```

Alterações do Iterador

A partir do Jython 2.2, o método **__iter__** é usado para causar um loop sobre uma coleção no

escopo de um bloqueio **for-in**. O iterador deve implementar o método **avançar**, retornando um elemento apropriado ou lançar o erro **StopIteration** se ele tiver atingido o final da coleção. Se o método **__iter__** não for implementado, o método **getitem** será usado.

Levantando Exceções

- **Método Jython 2.1 para levantar exceções está obsoleto:**
  ```
  raise Exception, 'Failed getting contents of file'
  ```
- **Método Jython 2.5.3 recomendado para levantar exceções:**
  ```
  raise Exception('Failed getting contents of file')
  ```

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.

Esta seção inclui:

- "Adicionar suporte para um novo idioma" abaixo
- "Alterar o idioma padrão" na página seguinte
- "Determinar o conjunto de caracteres para codificação" na página 56
- "Definir um novo trabalho para operar com dados localizados" na página 56
- "Decodificar comandos sem uma palavra-chave" na página 57
- "Trabalhar com bundles de recursos" na página 58
- "Referência de API" na página 59

Adicionar suporte para um novo idioma

Esta tarefa descreve como adicionar suporte a um novo idioma.

Esta tarefa inclui as seguintes etapas:

- "Adicionar um bundle de recursos (arquivos *.properties)" abaixo
- "Declarar e registrar o objeto de idioma" na página seguinte

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 58.

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 obter detalhes, consulte Adapter Management Window 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'), (104
   9,), 866)
   ```

   Para ver detalhes sobre idioma de classe, consulte "Referência de API" na página 59. 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 obter detalhes, consulte "Adicionar suporte para um novo idioma" na página 54.

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 obter detalhes, consulte " Visão geral da API de serviço Web do HP Universal CMDB" na página 272.

Esse código normalmente assume a seguinte forma:

```
client = Framework.createClient(protocol, properties)
shellUtils = shellutils.ShellUtils(client)
languageBundle = shellutils.getLanguageBundle ('langNetwork', shellUtils.osLangu
age, Framework)
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_addres
s')
ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig /all', strWindowsI
PAddress)
#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.osL
   anguage, Framework)
   ```

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

   ```
   strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_addr
   ess')
   ```

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

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

   O ShellUtils object 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 obter detalhes, consulte a etapa sobre como recuperar uma palavra-chave do bundle de recursos no "Definir um novo trabalho para operar com dados localizados" na página anterior.

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**.

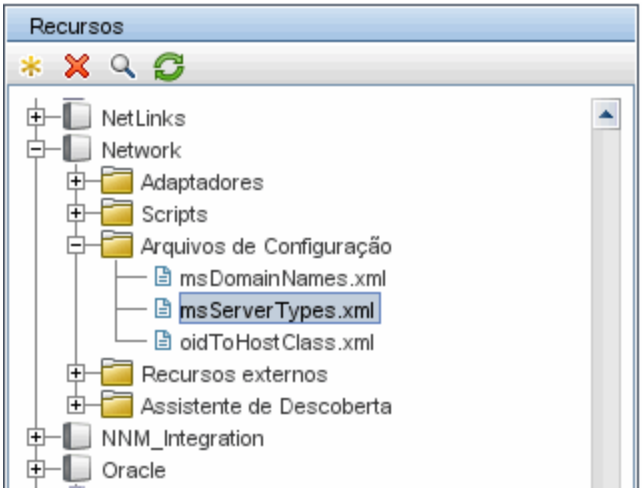
O caso de uso regular funciona da seguinte maneira:

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 anterior. 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 obter detalhes, consulte Configuration File Pane 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 cada bundle de recursos contém dados para um idioma apenas.)

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" abaixo
- "O método executeCommandAndDecode" na página seguinte
- "O método getCharsetName" na página seguinte
- "O método useCharset" na página seguinte
- "O método getLanguageBundle" na página 61
- "O campo osLanguage" na página 61

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.

Nome	Descrição
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 anterior.

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.

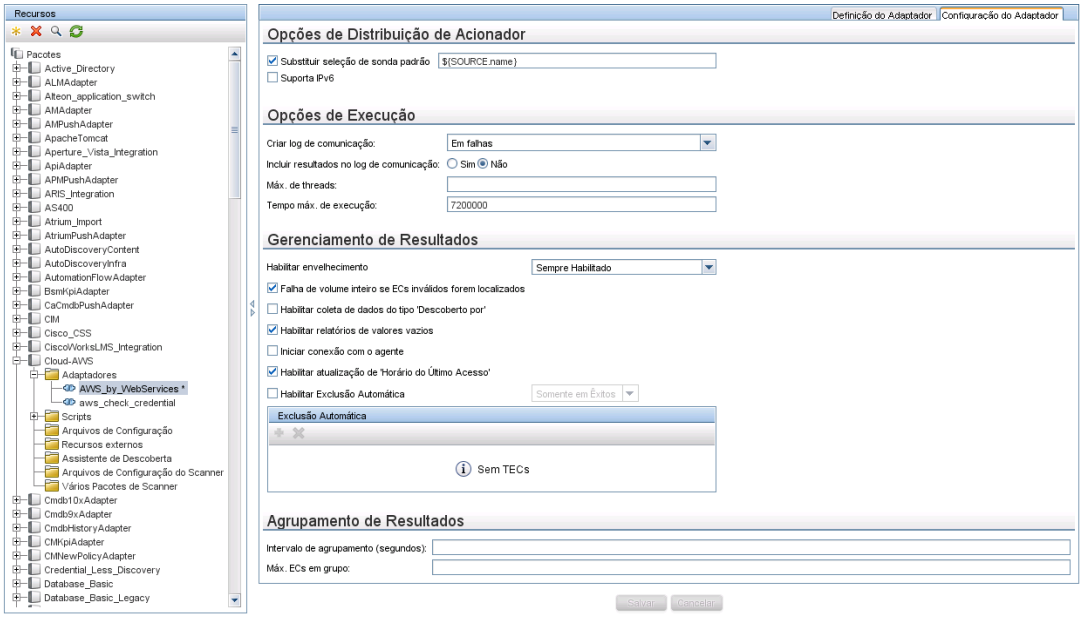
Registrar código do DFM

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:

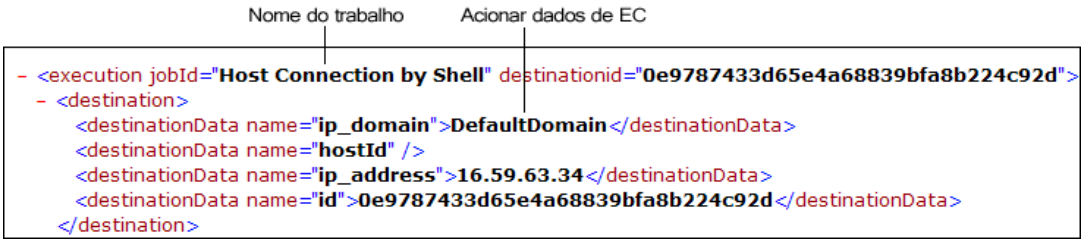
1. Acesse **Gerenciamento de Fluxo de Dados > Universal Discovery**. Clique com o botão direito do mouse no trabalho cuja execução precisa ser registrada em log e selecione **Ir para Adaptador** para abrir o aplicativo Gerenciamento do Adaptador.

2. Localize o painel **Opções de Execução** na guia **Configuração do Adaptador**, conforme exibido abaixo.

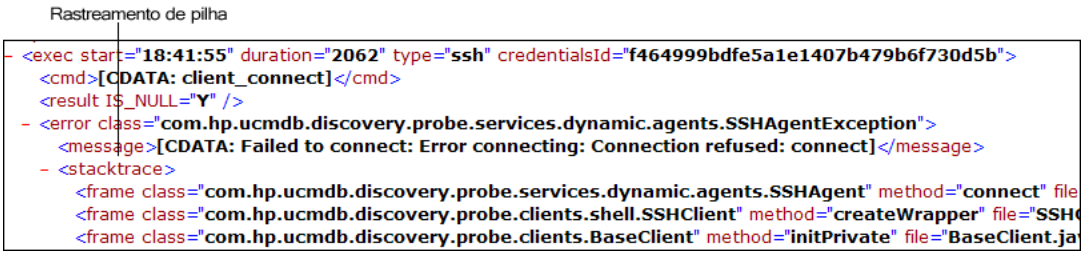


3. Altere a caixa **Criar logs de comunicação** para **Sempre**. Para ver detalhes sobre como configurar opções de registro em log, consulte "Execution Options Pane" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

   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:



Bibliotecas e utilitários Jython

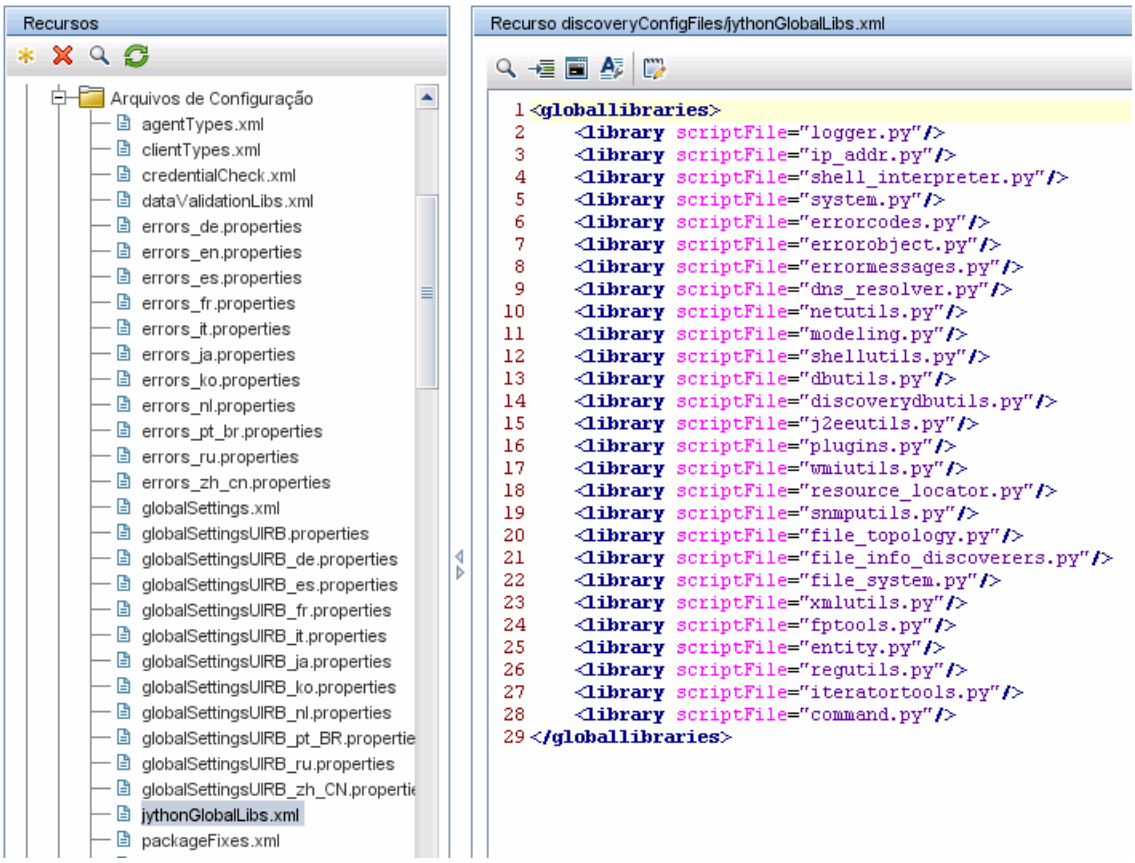
Vários scripts de utilitários são usados amplamente nos adaptadores. Esses scripts fazem parte do pacote `AutoDiscovery` e estão localizados em: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** com os outros scripts baixados para o Sonda.

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" abaixo
- "modeling.py" na página seguinte
- "netutils.py" na página seguinte
- "shellutils.py" na página seguinte

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 obter detalhes, consulte "Níveis de gravidade de erro" na página 69.

```
#########################################################################
################         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/pro
beMgr-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 = Framework.createClient(Framework.getTriggerCIData(BaseClient.CREDENT
IALS_ID), Props)
clientShUtils = shellutils.ShellUtils(ttyClient)
if (clientShUtils.isWinOs()):
        logger.debug ('discovering Windows..')
```

Capítulo 3: Mensagem de erro

Este capítulo inclui:

Visão geral de mensagens de erro 66

Convenções da criação de mensagens 66

Níveis de gravidade de erro 69

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. Esses erros são exibidos na janela do Universal Discovery sempre que o fluxo de descoberta regular não tiver êxito. You can drill down from the Trigger CI that caused the problem to view the error message itself.

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" abaixo.

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 Progresso da Descoberta e Resultados no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

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

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" abaixo
- "Arquivo de propriedade de mensagens de erro" abaixo
- "Convenções de nomenclatura de local" na página seguinte
- "Códigos de mensagem de erro" na página seguinte
- "Erros de conteúdo não classificado" na página 69
- "Mudanças na metodologia" na página 69

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
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.

Avisos

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 seja calculado incorretamente.

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

Capítulo 4: Desenvolvimento de adaptadores de banco de dados genéricos

Este capítulo inclui:

Visão geral dos adaptadores de banco de dados genéricos 72
Consultas TQL para o adaptador de banco de dados genérico 72
Reconciliação 73
Hibernar como provedor de JPA 74
Preparar criação do adaptador 76
Preparar o pacote de adaptadores 81
Configurar o adaptador - Método mínimo 84
Configurar o adaptador - Método avançado 89
Implementar um plugin 94
Implantar o adaptador 97
Editar o adaptador 97
Criar um ponto de integração 97
Criar uma visualização 98
Calcular os resultados 98
Visualizar os resultados 98
Visualizar relatórios 98
Habilitar arquivos de log 98
Usar o Eclipse para mapear entre atributos de TEC e tabelas de banco de dados 99
Arquivos de configuração do adaptador 107
Conversores prontos 132
Plug-ins 138
Exemplos de Configuração 138
Arquivos de log do adaptador 147
Referências externas 149
Solução de problemas e limitações 149

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.

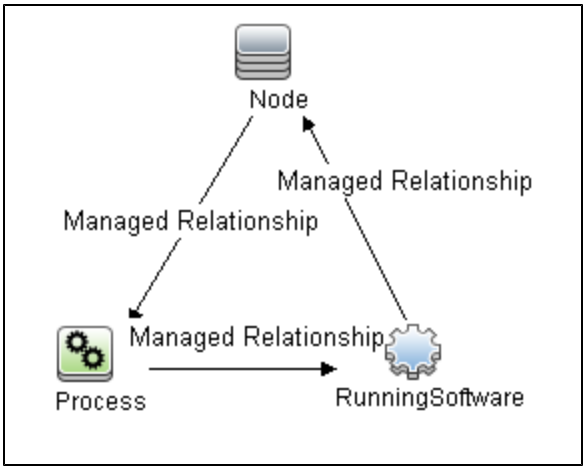
Consultas TQL para o adaptador de banco de dados genérico

Em trabalhos de população, cada layout obrigatório de um EC deve ser verificado na caixa de diálogo Configurações de Layout no Modeling Studio. Para obter detalhes, consulte Query Node/Relationship Properties Dialog Box no *Guia de Modelagem do HP Universal CMDB*. É importante observar que um EC pode exigir que um atributo seja identificado e, sem esses atributos, o EC falhará em ser adicionado ao UCMDB.

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

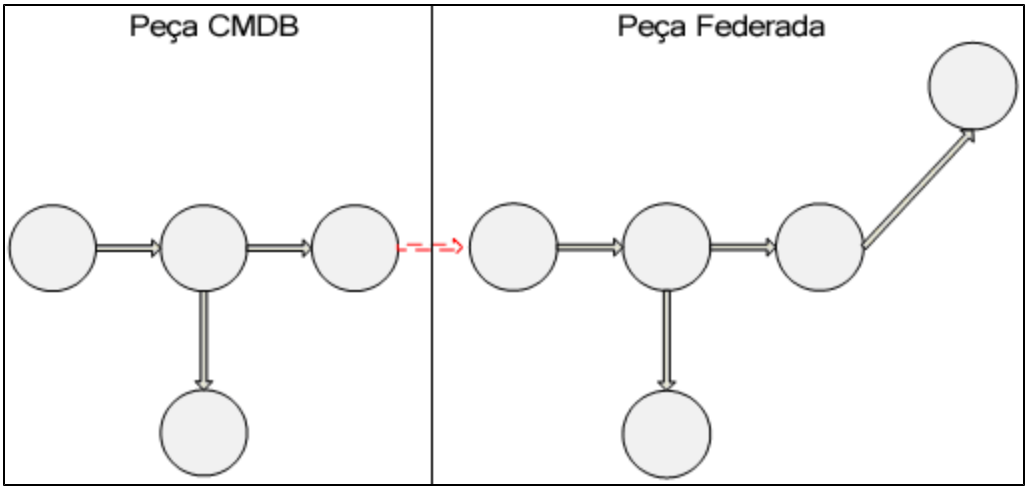
- não há suporte a subgráficos
- não há suporte a relacionamentos compostos
- não há suporte a ciclos ou partes de ciclo

  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 obter detalhes, consulte "Pré-requisitos" na página 76.
- A reconciliação com a ID do CMDB também é aceita.
- A reconciliação com a ID Global também é aceita.

Hibernar 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 obter detalhes, consulte "Limitações funcionais" na página 149.

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 obter detalhes, consulte "Associações" na página 76 a seguir.

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:

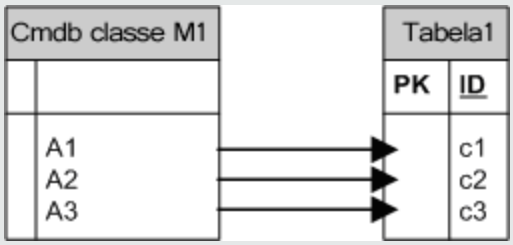
- "Exemplos de mapeamento relacional de objeto" abaixo
- "Associações" na página 76
- "Facilidade de uso" na página 76

Exemplos de mapeamento relacional de objeto

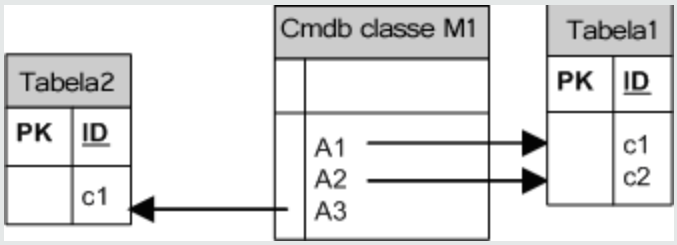
Os exemplos a seguir descrevem o mapeamento relacional de objeto:

Exemplo de uma classe do CMDB mapeada para uma tabela do 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`.

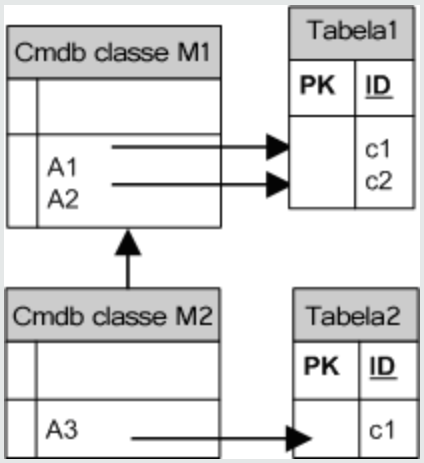


Exemplo de uma classe do CMDB mapeada para duas tabelas do banco de dados:



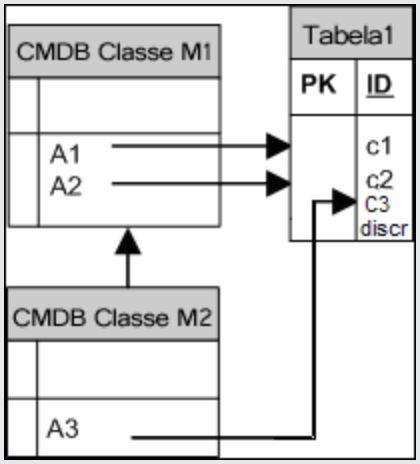
Exemplo de herança:

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.



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 a definição de associações.

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.

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" abaixo
- "Criar um tipo de EC" na página 78
- "Criar um relacionamento" na página 78

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 multinós) 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 73.

ID	NAME	IP_ADDRESS
31	BABA	16.59.33.60

ID	NAME	IP_ADDRESS
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 "Hibernar como provedor de JPA" na página 74.

  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 que representa os dados no RDBMS (a fonte de dados externa).

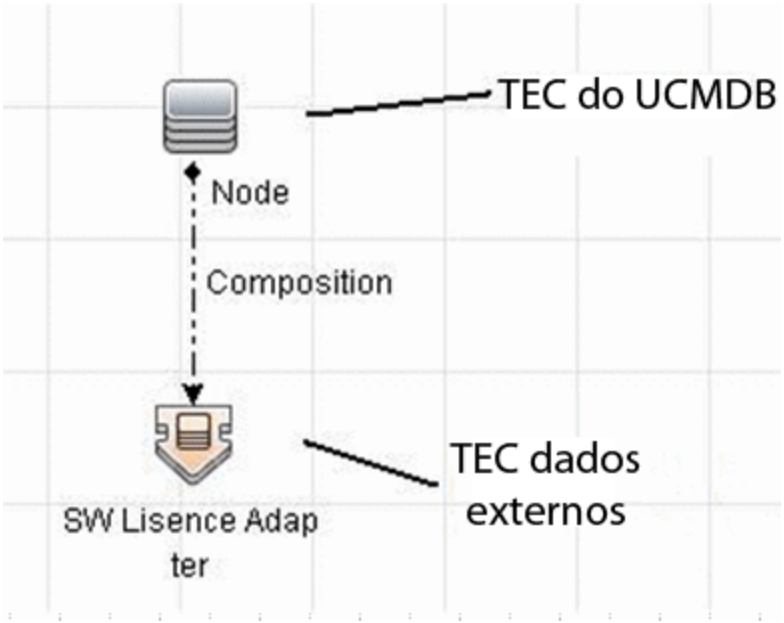
   a. No UCMDB, acesse o Gerenciador de Tipo de EC e crie um novo Tipo de EC. Para obter detalhes, consulte Create a CI Type no *Guia de Modelagem do HP Universal CMDB*.

   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 da fonte de dados externa.

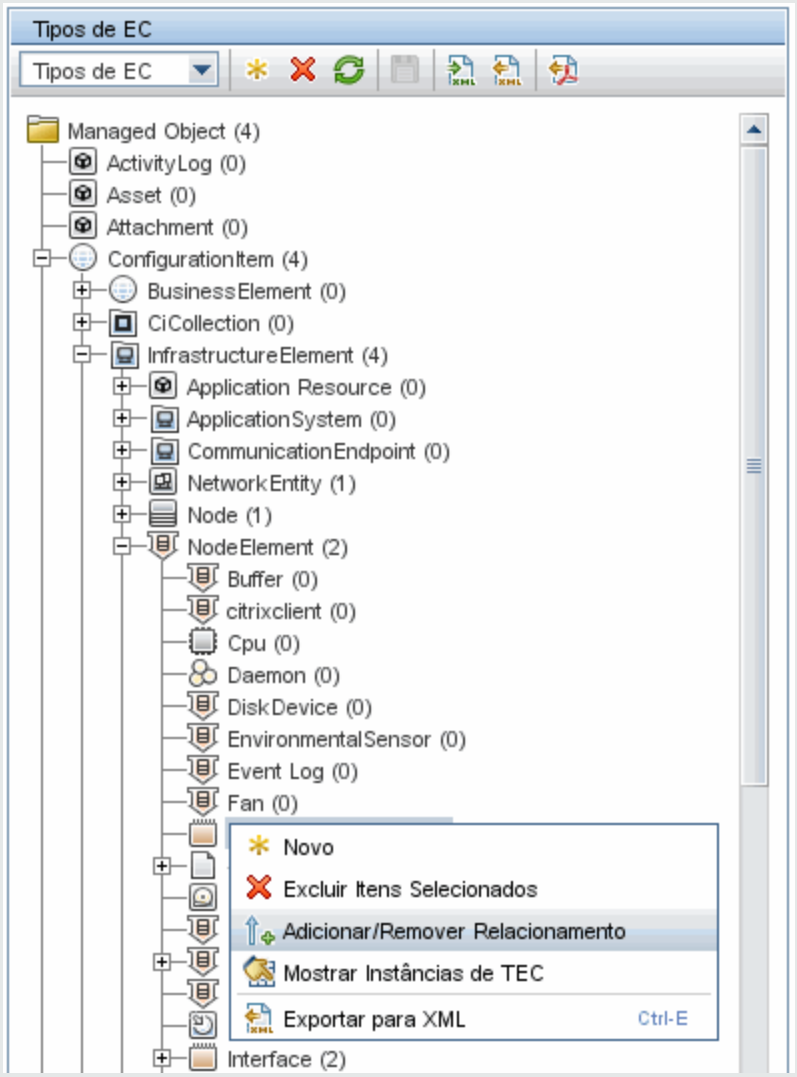
   Adicione relacionamentos válidos apropriados ao novo TEC. Para obter detalhes, consulte Add/Remove Relationship Dialog Box no *Guia de Modelagem do HP Universal CMDB*.



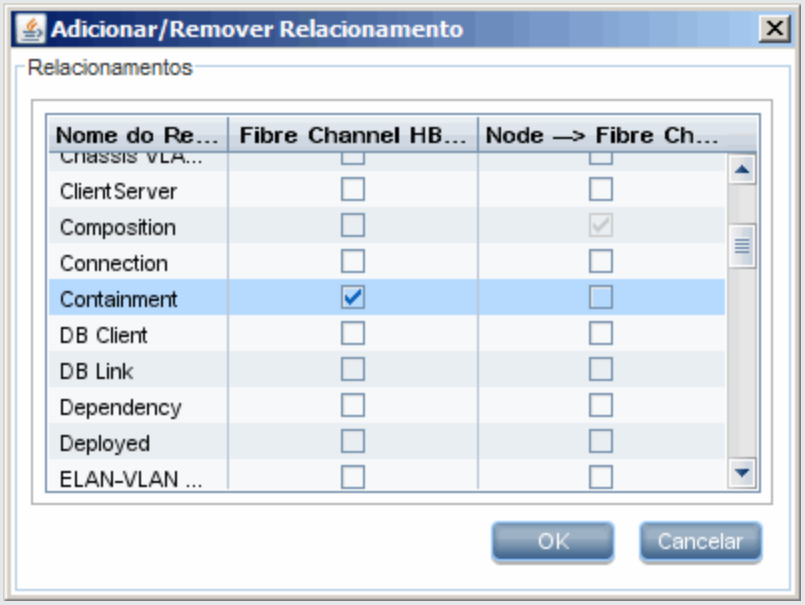
Observação: Nesta etapa, você ainda não pode visualizar os dados federados ou popular os dados externos, já que ainda não definiu o método para importar os dados.

Exemplo de criação de um relacionamento Containment:

a. No Gerenciador de TEC, selecione os dois TECs:



b. 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:noNamespaceSchem
aLocation="../../Patterns.xsd" description="Discovery Pattern Descript
ion"
         schemaVersion="9.0" xmlns:xsi="http://www.w3.org/2001/XMLSche
ma-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.

Ao criar um Adaptador de BD Genérico, não é necessário editar a marcação **changes-source** na marcação **support-replicatioin-data**. Se o plug-in **FcmdbPluginForSyncGetChangesTopology** estiver implementado, a topologia alterada da última execução será retornada. Se o plug-in não estiver implementado, toda a topologia será retornada e a exclusão automática será realizada de acordo com os ECs retornados.

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.AdapterMappingEngine</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.

- A conexão com o banco de dados pode ser descrita usando um nome de usuário (esquema), senha, tipo de banco de dados, nome da máquina de host do banco de dados e nome do banco de dados ou SID.

Para esse tipo de conexão, os parâmetros Têm os seguintes elementos na seção **parameter** do arquivo XML do adaptador:

```
<parâmetros>
    <!--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.-->
    <!--Descrição e nome de exibição podem ser substituídos por valores
I18N-->
        <parameter name="host" display-name="Hostname/IP" type="string"
description="The host name or IP address of the remote machine" mandat
ory="false" order-index="10" />
        <parameter name="port" display-name="Port" type="integer" desc
ription="The remote machine's connection port" mandatory="false" order-
index="11" />
        <parameter name="dbtype" display-name="DB Type" type="string"
description="The type of database" valid-values="Oracle;SQLServer;MySQ
L;BO" mandatory="false" order-index="13">Oracle</parameter>
        <parameter name="dbname" display-name="DB Name/SID" type="stri
ng" description="The name of the database or its SID (in case of Oracl
e)"  mandatory="false" order-index="13" />
        <parameter name="credentialsId" display-name="Credentials ID"
type="integer" description="The credentials to be used" mandatory="tru
e" order-index="12" />
</parameters>
```

Observação: Essa é a configuração padrão. Portanto, o arquivo **db_adapter.xml** já contém essa definição.

Há situações nas quais a conexão com o banco de dados não pode ser configurada dessa maneira. Por exemplo, conectar a um Oracle RAC ou conectar usando um driver do banco de dados que não seja o fornecido com o CMDB.

Para essas situações, você pode descrever a conexão usando nome de usuário (esquema), senha e uma cadeia de URL de conexão.

Para definir isso, edite a seção de parâmetros do XML do adaptador como a seguir:

```
<parâmetros>
    <!--O atributo de descrição pode ser escrito em texto simples ou em HTM
L.-->
    <!--O atributo de host é tratado como um caso especial pelo CMDBRTSM-->
```

```
    <!--e selecionará automaticamente o nome da sonda (se possível)-->
    <!--de acordo com o valor do atributo.-->
    <!--Descrição e nome de exibição podem ser substituídos por valores I18
N-->
        <parameter name="url" display-name="Connection String" type="strin
g" description="The connection string to connect to the database" mandatory
="true" order-index="10" />
        <parameter name="credentialsId" display-name="Credentials ID" type=
"integer" description="The credentials to be used" mandatory="true" order-i
ndex="12" />
</parameters>
```

Um exemplo de uma URL que conecta a um Oracle RAC usando o driver pronto para o uso Data Direct é:
jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true.

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

   Essa pasta contém a configuração do adaptador, 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 obter detalhes, consulte "Configurar o adaptador - Método mínimo" abaixo.

6. Crie um arquivo `*.zip` com o mesmo nome dado ao atributo **adapter id**, conforme descrito na etapa "Edite o arquivo XML do adaptador:" na página 81.

   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**.

Configurar o adaptador - Método mínimo

O método simplificado (mínimo) é um método para criar o arquivo de mapeamento **simplifiedConfiguration.xml** usado pelo adaptador. Este método permite uma população ou federação básica de um único TEC.

As instruções fornecidas nesta seção descrevem um método de mapear o modelo de classe para certot tipos de EC no CMDB para um RDBMS.

Todos esses arquivos de configuração mencionados nesta seção estão localizados no pacote **db-adapter.zip** da pasta **C:\hp\UCMDB\UCMDBServer\content\adapters** que você extraiu em "Preparar o pacote de adaptadores" na página 81.

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.

Você usará este método mínimo quando precisar:

- Federar/popular um único nó, como um atributo de nó.
- Demonstrar os recursos do adaptador de banco de dados genérico

Este método:

- Suporta somente federação/população de um nó
- Suporta somente relacionamentos virtuais de muitos-para-um

Configurar o arquivo adapter.conf

Para alterar as configurações no arquivo `adapter.conf` para que o adaptador use o método de configuração simplificado:

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**.

Exemplo: Preenchendo um nó e endereço IP usando o método simplificado

Esse exemplo demonstra o preenchimento de um **Nó** relacionado por um link de contenção para **Endereço IP** no UCMDB. O The RDBMS tem uma tabela chamada **simpleNode** que contém dados no nome do computador, nó do computador e o endereço IP do computador.

O conteúdo da tabela **simpleNode** é mostrado abaixo:

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

A população é realizada em três estágios, como a seguir:

1. "Criar o arquivo simplifiedConfiguration.xml" na página seguinte
2. "Crie o TQL" na página 87
3. "Criar um ponto de integração" na página 89

Criar o arquivo simplifiedConfiguration.xml

Criar o arquivo **simplifiedConfiguration.xml** como indicado a seguir:

1. Criar uma entidade **cmdb-class** como indicado a seguir:

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

   O Tipo de EC é **node** e o nome da tabela RDBMS é **simpleNode**.

2. Definir o primary-key da tabela como indicado a seguir:

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

   Essa chave principal é equivalente à id de entidade no arquivo **orm.xml**.

3. Defina a regra **reconciliation-by-two-nodes** como a seguir:

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

   Essa tag define a relação entre os tipos de EC **Node** e **IpAddress**. O tipo de relação é o link Containment. A reconciliação é executada pelos dois Tipos de EC conectados. O mapeamento de atributos do nó conectado (nesse caso IpAddress) é definido no atributo **connected-node**.

4. Adicione a condição **or** entre os atributos de reconciliação como indicado a seguir:

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

   Essa marcação define um relacionamento OR entre os atributos de reconciliação, significando que o primeiro atributo de reconciliação que é **verdadeiro** define toda a regra de reconciliação como **verdadeira**.

5. Adicione os seguintes atributos:

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

   Essa marcação define um mapeamento entre **node.name** no UCMDB para a coluna **host_name** na tabela **simpleNode**.

   Faça o mesmo com o atributo **data_note**:

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

   Adicione o atributo de nó conectado:

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

   Essa marcação define um mapeamento entre **ip_address.name** para a coluna **ip_address** na tabela **simpleNode**.

6. Feche a marcação aberta pela ordem:

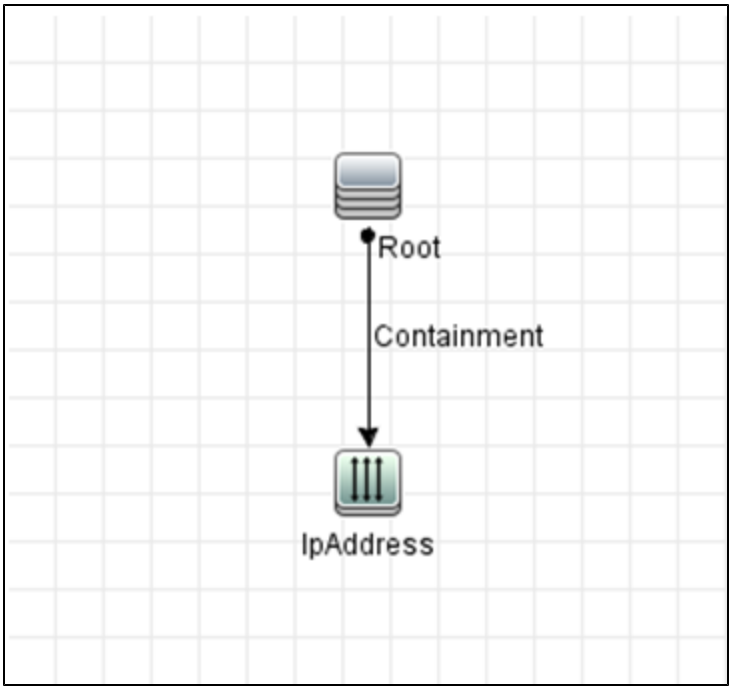
```
</or>

</reconciliation-by-two-nodes>

</cmdb-class>
```

O conteúdo do arquivo simplifiedConfiguration.xml aparece agora como a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-db-adapter-config xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfig
uration.xsd">
   <cmdb-class cmdb-class-name="node" default-table-name="simpleNode">
       <primary-key column-name="host_id"/>
   <reconciliation-by-two-nodes connected-node-cmdb-class-name="ip_address" cmd
b-link-type="containment">
       <or is-ordered="true">
       <attribute cmdb-attribute-name="name" column-name="host_name" ignore-
case="true"/>
       <attribute cmdb-attribute-name="data_note" column-name="note" ignore-
case="true"/>
          <connected-node-attribute cmdb-attribute-name="name" column-name="ip_a
ddress"/>
       </or>
       </reconciliation-by-two-nodes>
   </cmdb-class>
</generic-db-adapter-config>
```

Crie o TQL

O TQL é um **nó** conectado por um link de contenção a **ip_address**. O nó deve ser marcado como **root**, conforme mostrado abaixo.



Para criar o TQL:

1. Vá até **Modelagem > Modeling Studio**.
2. Clique no botão **Novo** e crie uma nova consulta.
3. Vá até a guia Tipos de EC e arraste o Tipo de EC Node e o Tipo de EC IpAddress para a tela do TQL.
4. Conecte **Node** e **IpAddress** com um relacionamento Containment.
5. Clique com o botão direito do mouse no elemento **Nó** e escolha Propriedades do Nó de Consulta.
6. Altere o **Nome do elemento** para **Raiz**.
7. Vá até a guia **Layout de Elemento**. Na condição de Atributos, selecione **Atributos Específicos**. Escolha **Nome** e **Nota** a partir da janela de Atributos Disponíveis e mova-os para a janela Atributos Específicos.
8. Clique com o botão direito do mouse no elemento **IpAddress** e escolha Propriedades do Nó de Consulta.
9. Vá até a guia **Layout de Elemento**. Na condição de Atributos, selecione **Atributos Específicos**. Escolha **Nome** a partir da janela de Atributos Disponíveis e mova-os para a janela Atributos Específicos.
10. Salve o TQL.

Criar um ponto de integração

Crie um ponto de integração como indicado a seguir:

1. Vá até **Gerenciamento de Fluxo de Dados > Integration Studio** e clique no botão **Novo Ponto de Integração**.
2. Insira os detalhes do Ponto de Integração e clique em **OK**.
3. Na guia População, selecione o botão **Novo Trabalho de Integração** e adicione o TQL criado anteriormente.
4. Salve o Ponto de Integração e clique no botão **Executar Sincronização Completa**.

Configurar o adaptador - Método avançado

Estes arquivos de configuração estão localizados no pacote **db-adapter.zip** da pasta **C:\hp\UCMDB\UCMDBServer\content\adapters** que você extraiu ao preparar o pacote do adaptador. Para obter detalhes, consulte "Preparar o pacote de adaptadores" na página 81.

Esta tarefa inclui as seguintes etapas:

- "Configurar o arquivo orm.xml" abaixo
- "Configurar o arquivo reconciliation_rules.txt " na página 93

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.

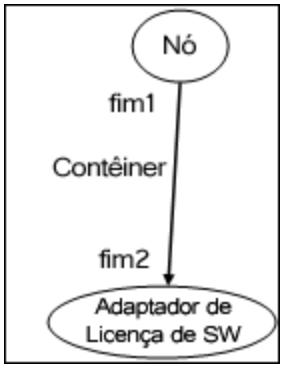
   Esse arquivo, por padrão, contém um modelo que é usado para mapear quantos TECs e relacionamentos forem necessários.

   > **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**:

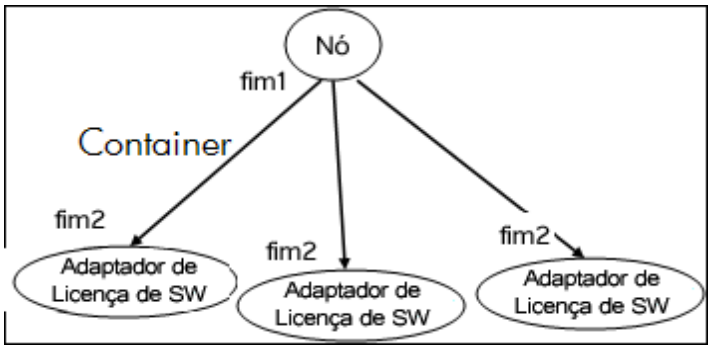
- Um-para-um:



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>
```

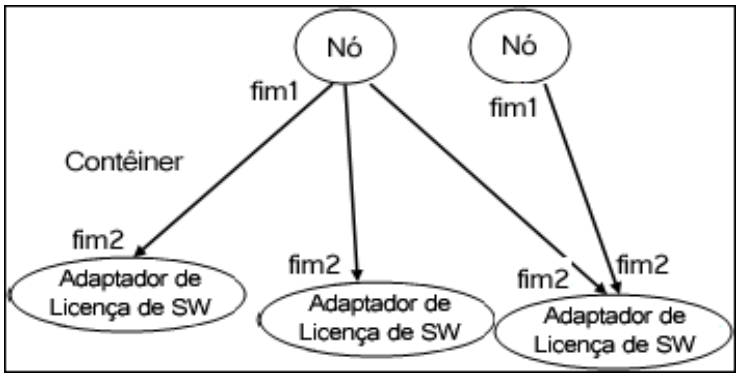
- Muitos-para-um:



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 116.

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 relacionamento 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 78:

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

- O nome da tabela correspondente no RDBMS:

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

- A identidade exclusiva no RDBMS:

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

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_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 obter detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 126.

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 81.

1. Opção 1 – Escrever um plugin baseado em Java

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

   b. 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	FcmdbPluginForSyncGetFullTopology	getFullTopology
Sincronizar alterações	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Sincronizar layout	FcmdbPluginForSyncGetLayout	getLayout
Recuperar consultas aceitas	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Alterar definições e resultados de consulta TQL	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Alterar solicitação de layout para ECs	FcmdbPluginGetCIsLayout	getCisLayout
Alterar solicitação de layout para links	FcmdbPluginGetRelationsLayout	getRelationsLayout
Devolver IDs	FcmdbPluginPushBackIds	getPushBackIdsSQL

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.

Se **FcmdbPluginForSyncGetChangesTopology** for implementado, há dois modos diferentes de reportar as mudanças:

- **Reporte toda a topologia raiz todas as vezes.** De acordo com essa topologia, a função de exclusão automática descobre quais ECs devem ser removidos. Nesse caso, a função de exclusão automática deve ser habilitada usando o seguinte:

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

- **Reporte cada instância de EC removida/atualizada.** Nesse caso, o mecanismo de exclusão automática deve ser habilitado usando o seguinte:

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

c. 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**.

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

2. Opção 2 – Escrever um plugin baseado em Groovy

   a. Crie um arquivo de código Groovy (MyPlugin.groovy) no Menu Gerenciamento do Adaptador, nos arquivos de configuração do pacote do adaptador.

   b. Na classe Groovy, implemente as interfaces apropriadas. As interfaces são definidas no arquivo db-interfaces.jar. consulte a tabela acima.

3. 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 re
sult
[getRelationsLayout]
# internal not mandatory plugin to change action on pushBackIds
```

```
[pushBackIds]
```

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.

4. 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 obter detalhes, consulte "Package Manager Page" 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 obter detalhes, consulte "Gerenciamento do Adaptador" no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

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 obter detalhes, consulte New Integration Point/Edit Integration Point Dialog Box 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 obter detalhes, consulte Federation Tab 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 obter detalhes, consulte Create a Pattern View no *Guia de Modelagem do HP Universal CMDB*.

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 obter detalhes, consulte Topology Reports Overview 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 obter detalhes, consulte "Arquivos de log do adaptador" na página 147.

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.

A versão 1.1 do plugin é compatível com a versão 9.01 do UCMDB ou posterior e Eclipse IDE for Java EE Developers, versão 1.2.2.20100217-2310 ou posterior.

Esta tarefa inclui as seguintes etapas:

- "Pré-requisitos" na página seguinte
- "Instalação" na página seguinte
- "Preparar o ambiente de trabalho" na página seguinte
- "Criar um adaptador" na página 101
- "Configure o plugin do CMDB" na página 101
- "Importar o modelo de classe do UCMDB" na página 102
- "Criar o arquivo ORM – mapear classes do UCMDB para tabelas de banco de dados" na página 102
- "IDs de Mapa" na página 103
- "Atributos de mapa" na página 103

- "Mapear um link válido" na página 103
- "Criar o arquivo ORM – usar tabelas secundárias" na página 104
- "Definir uma tabela secundária" na página 105
- "Mapear um atributo para uma tabela secundária" na página 105
- "Usar um arquivo ORM existente como base" na página 105
- "Importando um arquivo ORM existente de um adaptador" na página 105
- "Verificar a exatidão do arquivo orm.xml – verificação de exatidão interna" na página 106
- "Criar um novo ponto de integração" na página 106
- "Implantar o arquivo ORM no CMDB" na página 106
- "Executar uma consulta TQL de exemplo" na página 106

1. ## Pré-requisitos

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

2. ## Instalação

   a. Baixe e extraia **Eclipse IDE for Java EE Developers** em **http://www.eclipse.org/downloads** 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:\Program Files\Eclipse\eclipse.exe**. 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.zip** de **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** into **C:\Documents and Settings\All Users.**

> **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 > 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 28.

5. ## Configure 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 do 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 acima na etapa "Criar o arquivo ORM – mapear classes do UCMDB para tabelas de banco de dados" na página anterior 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 obter detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 126.

a. Mapeie os atributos de ID da classe de link conforme descrito em "IDs de Mapa" na página anterior. 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 103.

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 "Criar o arquivo ORM – mapear classes do UCMDB para tabelas de banco de dados" na página 102. 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.xml** 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" abaixo.

16. ## Verificar a exatidão do arquivo orm.xml – 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 obter 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 de multinós, se eles estiverem configurados no arquivo **reconciliation_rules.txt**. Para obter detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 126.

18. ## Implantar o arquivo ORM no CMDB

    Salve o arquivo **orm.xml** e implante-o no servidor UCMDB clicando em **UCMDB > 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. Definir uma consulta (não uma visualização) no Modeling Studio Para obter detalhes, consulte Modeling Studio no *Guia de Modelagem do HP Universal CMDB*.

    b. Crie um ponto de integração usando o adaptador criado na etapa "Criar um novo ponto de integração" acima. Para obter detalhes, consulte New Integration Point/Edit Integration Point Dialog Box 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 obter detalhes, consulte a etapa acima "Configure o plugin do CMDB" na página 101.

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.

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 descreve os seguintes arquivos de configuração:

- "O arquivo adapter.conf" na página 109
- "O arquivo simplifiedConfiguration.xml" na página 110
- "O arquivo orm.xml" na página 112
- "O arquivo reconciliation_types.txt" na página 126
- "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 126
- "O arquivo transformations.txt" na página 128
- "O arquivo discriminator.properties" na página 129
- "O arquivo replication_config.txt" na página 130
- "O arquivo fixed_values.txt" na página 130
- "Arquivo Persistence.xml" na página 131

Configuração geral

- **adapter.conf.** O arquivo de configuração do adaptador. Para obter detalhes, consulte "O arquivo adapter.conf" na página 109.

Configuração simples

- **simplifiedConfiguration.xml**. Arquivo de configuração que substitui os arquivos **orm.xml**, **transformations.txt** e **reconciliation_rules.txt** com menos recursos. Para obter detalhes, consulte "O arquivo simplifiedConfiguration.xml" na página 110.

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 obter detalhes, consulte "O arquivo orm.xml" na página 112.
- **reconciliation_rules.txt**. Contém as regras de reconciliação. Para obter detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 126.
- **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 obter detalhes, consulte "O arquivo transformations.txt" na página 128.
- **Discriminator.properties.** Este arquivo mapeia cada tipo de EC aceito para uma lista separada por vírgulas de possíveis valores correspondentes. Para obter detalhes, consulte "O arquivo discriminator.properties" na página 129.
- **Replication_config.txt.** Este arquivo contém uma lista separada por vírgulas de tipos de ECs e relacionamentos cujas condições de propriedade são aceitas pelo plugin de replicação. Para obter detalhes, consulte "O arquivo replication_config.txt" na página 130.
- **Fixed_values.txt.**Este arquivo permite configurar os valores fixos para atributos específicos de determinados TECs. Para obter detalhes, consulte "O arquivo fixed_values.txt" na página 130.

Configuração do Hibernate

- **persistence.xml.** Usado para substituir configurações prontas do Hibernate. Para obter detalhes, consulte "Arquivo Persistence.xml" na página 131.

Habilitando Suporte a Tabelas Temporárias para o Adaptador

Habilitar tabelas temporárias permite que o adaptador trabalhe com mais eficiência com o banco de dados remoto, reduzindo portanto o stress no banco de dados e na rede, além de melhorar o desempenho.

Para habilitar o suporte a tabelas temporárias no Adaptador de Banco de Dados Geral, as seguintes condições devem ser atendidas:

- As credenciais fornecidas para conectar ao banco de dados, incluir permissão para criar, modificar e excluir tabelas temporárias.
- Defina as seguintes configurações no arquivo de configuração adapter.conf:

  temp.tables.enabled=true

  performance.enable.single.sql=true

Observação: Tabelas temporárias apenas são suportadas para Microsoft SQL e Oracle.

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.

- **performance.memory.id.filtering=true**. Quando o GDBA executa TQLS, em alguns casos, um grande número de IDs pode ser recuperado e enviado de volta ao banco de dados usando o SQL. Para evitar esse trabalho excessivo e melhorar o desempenho, o GDBA tenta ler toda a visualização/tabela e filtra os resultados na memória.

- **id.reconciliation.cmdb.id.type=string/bytes**. Ao mapear o adaptador do banco de dados genérico usando a Reconciliação de ID, você pode mapear **cmdb_id** para uma **cadeia** ou tipo de coluna **bytes/raw** alterando a propriedade **META-INF/ adapter.conf**.

- **performance.enable.single.sql=true**. Esse é um parâmetro opcional. Se ele não for exibido no arquivo, seu valor padrão é **verdadeiro**. Quando **verdadeiro**, o adaptador de banco de dados genérico tenta gerar uma única instrução SQL para cada consulta que está sendo executada (para população ou uma consulta federada). Usar uma única instrução SQL melhora o desempenho e o consumo de memória do adaptador de banco de dados genérico. Quando **falso**, o adaptador de banco de dados genérico gera várias instruções SQL, que podem levar mais tempo e consumir mais memória que uma única. Mesmo quando esse atributo é definido como **verdadeiro**, o adaptador não gera uma única instrução SQL nos seguintes cenários:

  - O banco de dados ao qual o adaptador se conecta não está em um servidor Oracle ou SQL.
  - O TQL em execução contém uma condição de cardinalidade diferente de 0..* e 1..* (por exemplo, se houver uma condição de cardinalidade como 2..* ou 0..2).

- **in.expression.size.limit=950** (padrão). Esse parâmetro divide a expressão 'IN' do SQL executado, quando o limite de tamanho da lista de argumentos é atingido.

- **stringlist.delimiter.of.<Nome do TEC>.<Nome do Atributo>=<delimiter>**. Para mapear um atributo de lista de cadeia para uma coluna do banco de dados no adaptador de banco de

dados genérico, o atributo precisa ser mapeado para uma coluna de cadeia que contém uma lista de valores concatenados. Por exemplo, para mapear o atributo **policy_category** com o Tipo de EC **policy** e a coluna da cadeia contém uma lista de valores: value1##value2##value3 (que define uma lista de 3 valores value1, value2, value3), use a configuração: **stringlist.delimiter.of.policy.policy_category=##.**

- **temp.tables.enabled=true**. Permite o uso de tabelas temporárias para melhorar o desempenho. Disponível somente quando **performance.enable.single.sql** está ativada (somente suportado no Microsoft SQL e Oracle). Certas permissões no servidor de banco de dados podem ser necessárias.

- **temp.tables.min.value=50**. Define o número de valores de condição (ou IDs) necessárias para usar tabelas temporárias.

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, vá até **Gerenciamento do Adaptador > db-adapter > arquivos de configuração**.

Esta seção inclui os seguintes tópicos:

- "O modelo de arquivo simplifiedConfiguration.xml" abaixo
- "Limitações" na página 112

O modelo de arquivo simplifiedConfiguration.xml

- A propriedade **CMDB-class-name** é de tipo de vários nós (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-instanc
e" 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.

  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.impl. GenericEnumTransformer (generic-enum-transformer-example.xml)" to-CMDB-onverter="com.
  ```

```
mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)" />

<attribute CMDB-attribute-name="[CMDB_attribute_name]" column-name="[column_
name]" />

<attribute CMDB-attribute-name="[CMDB_attribute_name]" column-name="[column_
name]" />

 </class>

</generic-DB-adapter-config>
```

Limitações

- Pode ser usado para mapear somente consultas TQL com um 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**.

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" na página seguinte
- "Vários arquivos ORM" na página 116
- "Convenções de nomenclatura" na página 116
- "Usando instruções SQL em linha em vez de nomes de tabela" na página 117
- "O esquema orm.xml" na página 117

- "Exemplo de criação do arquivo orm.xml" na página 122
- "Configurando um orm.xml específico para cada versão de produto remoto" na página 126

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:xs
i="http://www.w3.org/2001/XMLSchema-instance" version="1.0" xsi:schemaLocati
on=
"http://java.sun.com/xml/ns/persistence/orm http://java.sun.com/xml/ns/persi
stence/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 de vários nós.

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_n
ame]" />
            </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_cl
ass_name]">
        <table name="[default_table_name]" />
        <secondary-table name="" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <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_n
ame]" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
            </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_adap
ter.node_composition_[CMDB_class_name]">
        <table name="[default_table_name]" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_n
ame]" />
                <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>
```

node attribute. Esse é um exemplo de como adicionar um atributo de nó.

```
<entity class="generic_db_adapter.host_node">
  <discriminator-value>host_node</discriminator-value>
    <attributes/>
</entity>
<entity class="generic_db_adapter.nt">
  <discriminator-value>nt</discriminator-value>
    <attributes>
      <basic name="nt_servicepack">
        <column updatable="false" insertable="false" name="specific_type_value"/>
      </basic>
    </attributes>
</entity>
```

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 f
rom
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.

Nome e caminho do elemento	Descrição	Atributos
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.	
descrição (entity-mappings)	Uma descrição de texto livre do documento de mapeamento de entidade. (Opcional)	

Nome e caminho do elemento	Descrição	Atributos
pacote (entity-mappings)	O nome do pacote Java que conterá as classes de mapeamento. Deve sempre conter o texto `generic_db_adapter`.	1. **Nome:** nome **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 necessário?:** Opcional **Tipo:** Cadeia 2. **Nome:** classe **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 necessário?:** Obrigatório **Tipo:** Cadeia

Nome e caminho do elemento	Descrição	Atributos
table (entity-mappings>entity)	Esse elemento define a tabela primária da entidade de banco de dados. Pode aparecer somente uma vez. Obrigatório.	**Nome:** nome **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 necessário?:** 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:** nome **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 pode ser qualquer instrução SELECT válida. Se essa for uma instrução SELECT, será necessário encapsulá-la com parênteses. **É necessário?:** Obrigatório **Tipo:** Cadeia
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:** nome **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 necessário?:** Opcional **Tipo:** Cadeia

Nome e caminho do elemento	Descrição	Atributos
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:** estratégia **Descrição:** Define a maneira como a herança é implementada no banco de dados. **É necessário?:** Obrigatório **Tipo:** Um dos seguintes valores: • SINGLE_TABLE: Esta 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:** nome **Descrição:** O nome da coluna do discriminador. **É necessário?:** Obrigatório **Tipo:** Cadeia
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.	

Nome e caminho do elemento	Descrição	Atributos
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:** nome **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 necessário?:** Obrigatório **Tipo:** Cadeia
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:** nome **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 necessário?:** 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.	1. **Nome:** nome **Descrição:** O nome do campo. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** tabela **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 necessário:** Opcional **Tipo:** Cadeia

Nome e caminho do elemento	Descrição	Atributos
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.	1. **Nome:** nome **Descrição:** Qual das duas extremidades este campo representa. **É necessário?:** Obrigatório **Tipo:** `end1` ou `end2` 2. **Nome:** target-entity **Descrição:** O nome da entidade à qual a extremidade se refere. **É necessário?:** 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.	1. **Nome:** nome **Descrição:** O nome do campo na tabela atual que será usado para executar a junção um-para-um. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** nome **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 necessário?:** Opcional **Tipo:** Cadeia

Exemplo de criação do arquivo orm.xml

O exemplo apresentado aqui mostra como criar o arquivo **orm.xml**. Nesse exemplo, tabelas SQL em um banco de dados remoto são mapeadas para tipos de EC no UCMDB.

Dadas as tabelas com o seguinte formato no banco de dados remoto, preencha a tabela **Hosts** com nós, a tabela **IP_Addresses** com endereços IP e crie links entre os nós e endereços IP como a seguir:

Tabela de hosts

host_name	host_id
Teste1	1
Teste2	2
Teste3	3

Tabela IP_Addresses

ip_address	ip_id
10.1.1.1	1
10.2.2.2	2
10.3.3.2	3
10.4.4.4	4

Tabela Host_IP_Link (links entre Nós e Endereços IP)

host_id	ip_id
1	1
2	2
2	3
3	4

A chave principal para a tabela **Hosts** é o campo **host_id** e a chave primária na tabela **IP_ Addresses** é o campo **ip_id**. Na tabela **Host_IP_Link**, **host_id** e **ip_id** são chaves externas da tabela **Hosts** e **IP_Addresses**.

De acordo com as tabelas acima, crie o arquivo **orm.xml** como as etapas a seguir. As entidades usadas nesse exemplo são **node**, **ip_address** e **node_containment_ip_address**.

1. Crie a entidade **node** mapeando **host_id** a partir da tabela **Hosts** como a seguir:

```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
       /orm_1_0.xsd">
    <description>test_integration</description>
    <package>generic_db_adapter</package>
```

```
    <entity class="generic_db_adapter.node">
       <table name="Hosts"/>
       <attributes>
        <id name="id1">
                <column updatable="false" insertable="false" name=
                       "host_id"/>
                <generated-value strategy="TABLE"/>
        </id>
        <basic name="name">
                <column updatable="false" insertable="false" name=
                       "host_name"/>
        </basic>
       </attributes>
    </entity>
```

A **classe** da entidade deve ser um tipo de EC que já existe no UCMDB. O **nome** da tabela é a tabela dentro do banco de dados que contém uma ID e as informações do host. O atributo ID é necessário para identificar hosts específicos e é usado posteriormente no mapeamento. Nesse exemplo, o atributo **name** dessa entidade será preenchido com a coluna **hostname** da tabela de hosts.

2. Para a próxima entidade, mapeie os endereços IP a partir da tabela interfaces:

```
<entity name="ip_address" class="generic_db_adapter.ip_address">
    <table name="IP_Addresses"/>
    <attributes>
       <id name="id1">
        <column insertable="false" updatable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
       </id>
       <basic name="name">
        <column updatable="false" insertable="false" name="ip_address"/>
       </basic>
    </attributes>
```

```
</entity>
```

3. Em seguida, o link entre o Nó e o Endereço IP devem ser criados pela tabela de mapeamento e fazer referência ao campo **ip_id** (embora possa fazer referência aos campos **host_id** e **ip_id**, se desejado).

```
<entity name="node_containment_ip_address"
        class="generic_db_adapter.node_containment_ip_address">
    <table name="Host_IP_Link"/>
    <attributes>
       <id name="id1">
        <column updatable="false" insertable="false" name="ip_id"/>
        <generated-value strategy="TABLE"/>
       </id>
       <many-to-one target-entity="node" name="end1">
        <join-column name="host_id"/>
       </many-to-one>
       <one-to-one target-entity="ip_address" name="end2">
        <join-column name="ip_id"/>
       </one-to-one>
    </attributes>
</entity>
```

O nome da entidade para o contêiner tem o formato: [end1 CIT]_[link CIT]_[end2 CIT]. Então, para esse exemplo, como o Tipo de EC de link é **containment**, o nome da entidade para o contêiner é: **node_containment_ip_address** e a classe de entidade é **generic_db_adapter.node_containment_ip_address**. A ID é necessária nesse bloco de código e, embora esse exemplo funcione com uma única ID da Interface, as duas colunas devem usar as referências id1 e id2. O código para isso seria:

```
<id name=”id1”>
    <column updatable=”false” insertable=”false” name=”ip_id”/>
    <generated-value strategy=”TABLE”/>
</id>
```

```
<id name=”id2”>
    <column updatable=”false” insertable=”false” name=”host_id”/>
    <generated-value strategy=”TABLE”/>
</id>
```

As duas extremidades desse link são ‘vários para um’ e ‘um para um’, significando que cada endereço IP será vinculado a 1 nó, mas um nó pode ser vinculado a vários endereços IP. As colunas incluídas são da tabela Links e fazem referência às tabelas Hosts e Interfaces.

Configurando um orm.xml específico para cada versão de produto remoto

É possível configurar um arquivo **orm.xml** específico para que o adaptador use um **orm.xml** específico para uma determinada versão do produto remoto. Por exemplo, se o armazenamento de dados remoto tem duas versões do produto x e y, para cada versão, pode haver um mapeamento diferente de entidades.

Para configurar um arquivo orm.xml específico por versão de produto remoto:

1. Adicione um parâmetro ao arquivo **adapter.xml** chamado **version** e especifique possíveis valores da versão como **valid-values**.
2. No pacote do adaptador, na pasta META-INF, crie uma pasta chamada **VersionOrm**.
3. Na pasta **VersionOrm**, crie um arquivo **orm.xml** para cada versão específica. O nome do arquivo deve conter o prefixo da versão. Por exemplo, se a versão for chamada **x**, o nome do arquivo deve ser **x_orm.xml**.

Observação: O arquivo **orm.xml** na pasta META-INF é carregado para qualquer versão do produto remoto, independentemente de você criar um arquivo específico **orm.xml** para uma versão do produto remoto. Ele pode ter entidades mapeadas da mesma maneira para todas as versões.

O arquivo reconciliation_types.txt

A partir do UCMDB 10.00, o arquivo **reconciliation_types.txt** não é mais relevante. Qualquer TEC pode ser usado para a reconciliação. O mecanismo de federação executa o mapeamento automaticamente.

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 multinó é preenchido com o nome de multinó (o TEC do CMDB que está conectado a um TEC de banco de dados federado na consulta 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 a consulta TQL de reconciliação contém dois nós e não só um multinó. Nesse caso, a consulta 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 a consulta TQL de reconciliação deve ter no máximo dois nós. Por exemplo, a consulta 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):

Nó de BD	
PK	node_id
	nome

Endereço_IP do BD	
PK	ip_id
	nome

Ticket do BD	
PK	ticket_id
	node_id

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.mercu
ry.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 obter detalhes, consulte "Conversores prontos" na página 132.

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 ou uma condição que corresponda a possíveis valores da coluna do discriminador.

Se uma condição for usada, use a sintaxe: `like(condition)`, onde `condition` é uma cadeia que pode conter os seguintes curingas:

- **%** (sinal de percentual) - permite que você corresponda qualquer cadeia de qualquer tamanho (incluindo uma cadeia de tamanho zero)
- **_** (sublinhado) - permite que você corresponda um único caractere

Por exemplo, `like(%unix%)` corresponderá a unix, linux, unix-aix e assim por diante. As condições like só podem ser aplicadas a colunas de cadeias.

Você também pode ter um valor de discriminador mapeado para qualquer valor que não pertença a outro discriminador informando `'all-other'`.

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:

Por exemplo, o adaptador suporta os tipos de EC `node`, `nt`, e `unix`, e o banco de dados contém uma única tabela chamada `t_nodes` que contém uma coluna chamada **type**. Se o tipo for 10001, a linha representa um nó; se o tipo for 10004, ele representa uma máquina unix e assim por diante. O arquivo **discriminator.properties** pode ser semelhante a este:

```
node=10001, 10005
nt=10002,10003
```

```
unix=2%
mainframe=all-other
```

O arquivo **orm.xml** inclui o seguinte código:

```
    <entity class="generic_db_adapter.node" >
        <table name="t_nodes" />
        ...
        <inheritance strategy="SINGLE_TABLE" />
        <discriminator-value>node</discriminator-value>
        <discriminator-column name="type" />
        ...
    </entity>
    <entity class="generic_db_adapter.nt" name="nt">
        <discriminator-value>nt        </discriminator-value>
        <attributes>
    </entity>
    <entity class="generic_db_adapter.unix" name="unix">
        <discriminator-value>unix</discriminator-value>
        <attributes>
    </entity>
```

O atributo discriminator_column é calculado da seguinte maneira:

- Se **type** contém `10002` ou `10003` para uma certa entrada, a entrada será mapeada para o TEC **nt**.
- Se **type** contém `10001` ou `10005` para uma certa entrada, a entrada será mapeada para o TEC **node**.
- Se **type** começa com `2` para uma certa entrada, a entrada será mapeada para o TEC **unix**.
- Qualquer outro valor na coluna **type** será mapeado para o TEC **mainframe**.

> **Observação:** 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 ECs e relacionamentos cujas condições de propriedade são aceitas pelo plugin de replicação. Para obter detalhes, consulte "Plug-ins" na página 138.

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]
```

O arquivo também suporta uma lista de constantes. Para definir uma lista de constantes, use a seguinte sintaxe:

```
entity[<entityName>] attribute[<attributeName>] value[{<Val1>, <Val2>, <Val3>, .
.. }]
```

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 SQL 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** (para obter detalhes, consulte http://www.hibernate.org).

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">
        <propriedades>
            <!--  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.me
rcury.
               jdbc.MercOracleDriver" />
            <!--A url de conexão"/-->
            <property name="hibernate.connection.url" value="jdbc:mercury:or
acle:
               //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>
```

Conectar a banco de dados usando autenticação do NT

É possível se conectar a um MS SQL Server que exija autenticação do NT. Para isso, um driver que pode analisar o domínio é necessário (isto é, o driver jTDS JDBC).

A autenticação é feita de acordo com os parâmetros fornecidos (domínio, nome de usuário, senha) e não com as credenciais do NT do processo em execução no momento.

1. Em **persistence.xml**, edite as seguintes propriedades como a seguir:

```
<!--Nome da classe do driver"/-->
<property name="hibernate.connection.driver_class"
value="net.sourceforge.jtds.jdbc.Driver"/>
<property name="hibernate.connection.url" value="jdbc:jtds:sqlserver://[host name]:
[port];DatabaseName=[database name];domain=[the domain]"/>
<!-- Credenciais de logon no BD"/-->
<property name="hibernate.connection.username" value="[username]"/>
<property name="hibernate.connection.password" value="[password]"/>
```

2. Coloque o arquivo do driver JDBC em: **<pasta de instalação da sonda>\lib**.

3. Reinicie a sonda.

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:

- "Conversores prontos" acima
- "O conversor SuffixTransformer" na página 136

- "O conversor PrefixTransformer" na página 136
- "O conversor BytesToStringTransformer" na página 136
- "O conversor StringDelimitedListTransformer" na página 136
- "O Conversor Personalizado" na página 137

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.

O transformador realiza uma comparação entre duas cadeias usando um método de distinção entre maiúsculas e minúsculas ou sem distinção entre maiúsculas e minúsculas. O comportamento padrão é diferenciar maiúsculas de minúsculas. Para definir como distinção entre maiúsculas e minúsculas, use: `case-sensitive="false"` no elemento enum-transformer.

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**.

XSD de arquivo de entrada:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="q
ualified" attributeFormDefault="unqualified">
  <xs:element name="enum-transformer">
    <xs:complexType>
        <xs:sequence>
           <xs:element ref="value" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="db-type" use="required">
           <xs:simpleType>
               <xs:restriction base="xs:string">
                   <xs:enumeration value="integer"/>
                   <xs:enumeration value="long"/>
                   <xs:enumeration value="float"/>
                   <xs:enumeration value="double"/>
```

```
                <xs:enumeration value="boolean"/>
                <xs:enumeration value="string"/>
                <xs:enumeration value="date"/>
                <xs:enumeration value="xml"/>
                <xs:enumeration value="bytes"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="cmdb-type" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="integer"/>
                <xs:enumeration value="long"/>
                <xs:enumeration value="float"/>
                <xs:enumeration value="double"/>
                <xs:enumeration value="boolean"/>
                <xs:enumeration value="string"/>
                <xs:enumeration value="date"/>
                <xs:enumeration value="xml"/>
                <xs:enumeration value="bytes"/>
             </xs:restriction>
        </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="non-existing-value-action" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="return-null"/>
                <xs:enumeration value="return-original"/>
                <xs:enumeration value="throw-exception"/>
            </xs:restriction>
        </xs:simpleType>
```

```
        </xs:attribute>
        <xs:attribute name="case-sensitive" use="optional">
            <xs:simpleType>
                <xs:restriction base="xs:boolean">
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="value">
    <xs:complexType>
        <xs:attribute name="cmdb-value" type="xs:string" use="required"/>
        <xs:attribute name="external-db-value" type="xs:string" use="require
d"/>
        <xs:attribute name="is-cmdb-value-null" type="xs:boolean" use="optio
nal"/>
        <xs:attribute name="is-db-value-null" type="xs:boolean" use="optiona
l"/>
    </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-act
ion="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>
```

Exemplo de conversão de um valor externo ou do CMDB para um valor nulo:

Neste exemplo, um valor de **NNN** no banco de dados remoto é transformado em um valor nulo no banco de dados do CMDB.

```
<value cmdb-value="null" is-cmdb-value-null="true" external-db-value="NNN"/>
```

Neste exemplo, o valor **OOO** no CMDB é transformado em um valor nulo no banco de dados.

```
<value cmdb-value="OOO" external-db-value="null" is-db-value-null="true"/>
```

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.

O conversor StringDelimitedListTransformer

Esse conversor é usado para transformar uma única lista de cadeias em uma lista de inteiros/cadeias no CMDB.

O conversor é: **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.StringDelimitedListTransformer.**

O Conversor Personalizado

É possível escrever seu próprio conversor personalizado (transformador) do zero. Isso permite criar qualquer conversor necessário de acordo com as suas necessidades.

Há duas maneiras de escrever um conversor personalizado:

1. Escrever um conversor Java compilado
   a. Criar um projeto Java em um Java IDE (como Eclipse, Intellij ou Netbeans).
   b. Adicione federation-api.jar e db-interfaces.jar ao caminho da sua classe.
   c. Crie uma classe Java que implemente as seguintes interfaces (de **db-interfaces.jar**):
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   d. Compile o projeto e crie um arquivo jar.
   e. Coloque o arquivo jar no pacote de adaptadores (em adapterCode\<ID do Adaptador>)
   f. Implante o pacote.
   g. Adicione o nome da classe do novo conversor ao arquivo **transformations.txt**.
2. Escreva um conversor Groovy (baseado em script)

   Um exemplo é encontrado no pacote GDBA original, **GroovyExampleTransformer.groovy**.
   a. Crie um arquivo Groovy no pacote do adaptador (em adapterCode\<ID do Adaptador>). Você pode fazer isso diretamente usando o menu Gerenciamento do Adaptador.
   b. Crie uma classe Groovy que implemente as seguintes interfaces (de **db-interfaces.jar**):
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   c. Adicione o nome da classe Groovy do novo conversor ao arquivo **transformations.txt** de acordo.

Observação: Groovy é uma linguagem de scripting que amplia o Java. Códigos Java regulares também são códigos válidos Groovy.

Plug-ins

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 sincronização de layout.
- 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.
- Um plugin opcional interno para alterar a ação de fazer push back de IDs.

Para ver detalhes sobre como implementar e implantar plugins, consulte "Implementar um plugin" na página 94.

Exemplos de Configuração

Esta seção fornece exemplos de configurações.

Esta seção inclui os seguintes tópicos:

- "Caso de uso" abaixo
- "Reconciliação de nó único" na página seguinte
- "Reconciliação de nó duplo" na página 141
- "Usando uma chave primária que contém mais de uma coluna" na página 144
- "Usando transformações" na página 146

Caso de uso

Uma consulta 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-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-single-node>
            <ou>
                <attribute CMDB-attribute-name="name" column-name="Device_Na
me" />
            </or>
        </reconciliation-by-single-node>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMD
B-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="hwCardVend
or" />
        <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 112.

Exemplo do arquivo orm.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xs
i="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_rules.txt

Para obter detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 126.

```
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.

A consulta 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-insta
nce" 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_addr
ess" CMDB-link-type="containment">
            <ou>
                <attribute CMDB-attribute-name="name" column-name="Device_Na
me" />
                <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-CMD
B-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="hwCardVend
or" />
        <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-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <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_addr
ess" CMDB-link-type="containment">
            <e>
                <attribute CMDB-attribute-name="name" column-name="Device_Na
me" />
                <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-CMD
B-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="hwCardVend
or" />
        <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-insta
nce" xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xs
d">
    <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_addr
ess" CMDB-link-type="containment">
            <ou>
                <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-CMD
B-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="hwCardVend
or" />
        <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_rules.txt

Para obter detalhes, consulte "O arquivo reconciliation_rules.txt (para compatibilidade com versões anteriores)" na página 126.

```
multinode[node] expression[ip_address.name OR node.name] end1_type[node] end
2_type[ip_address] link_type[containment]

multinode[node] expression[ip_address.name AND node.name] end1_type[node] en
d2_type[ip_address] link_type[containment]

multinode[node] expression[ip_address.name] end1_type[node] end2_type[ip_add
ress] 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-CMD
B-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="hwCardVend
or" />
        <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 obter detalhes, consulte "O arquivo orm.xml" na página 112.

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="fals
e"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false" updat
able="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column name="hwCards_Seq" insertable="false" updatable="fal
se"/>
                <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="fals
e"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false" updat
able="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column name="hwCards_Seq" insertable="false" updatable="fal
se"/>
                <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-n
ame="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:noNa
mespaceSchemaLocation="../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">
            <ou>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"
                 from-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter
.dal.
                 transform.impl.GenericEnumTransformer(generic-enum-transformer-
example.
                 xml)" to-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAda
pter.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.
GenericEnumTransformer(generic-enum-transformer-example.xml)] from_DB_class
[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(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" abaixo
- "Locais dos arquivos" abaixo

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 a consulta 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 "Hibernar como provedor de JPA" na página 74.

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

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**.

Capítulo 5: Desenvolvimento de adaptadores Java

Este capítulo inclui:

Visão geral do Federation Framework 151

Interação de adaptador e mapeamento com o Federation Framework 156

Fluxo do Federation Framework para consultas TQL federadas 157

Interações entre o Federation Framework, servidor, adaptador e mecanismo de mapeamento 158

Fluxo do Federation Framework para população 168

Interfaces de adaptador 169

Depurar recursos do adaptador 171

Adicionar um adaptador para uma nova fonte de dados externos 171

Criar um adaptador de amostra 180

Marcas e propriedades de configuração XML 181

A Interface DataAdapterEnvironment 183

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.

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" abaixo
- "Push de Dados" na página seguinte
- "População" na página 154

Federação em tempo real

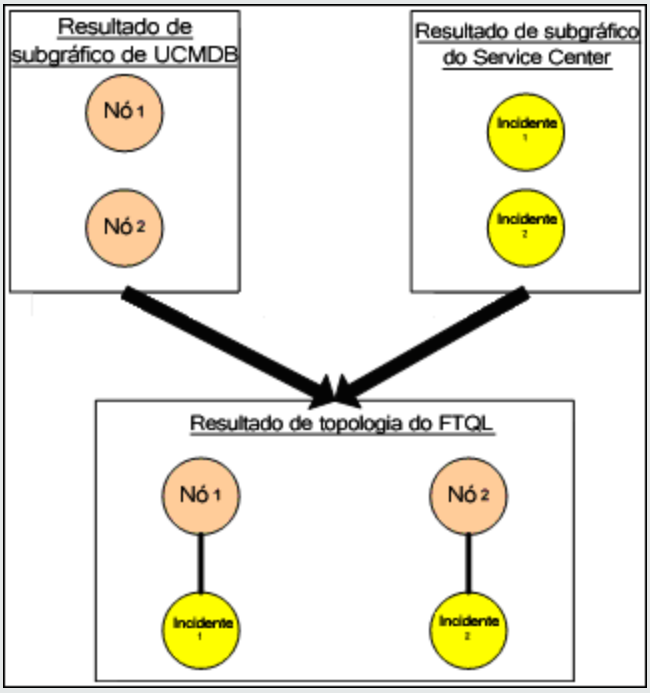
As consultas TQL federadas permitem a recuperação de dados de qualquer repositório de dados externo sem replicar seus dados.

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.

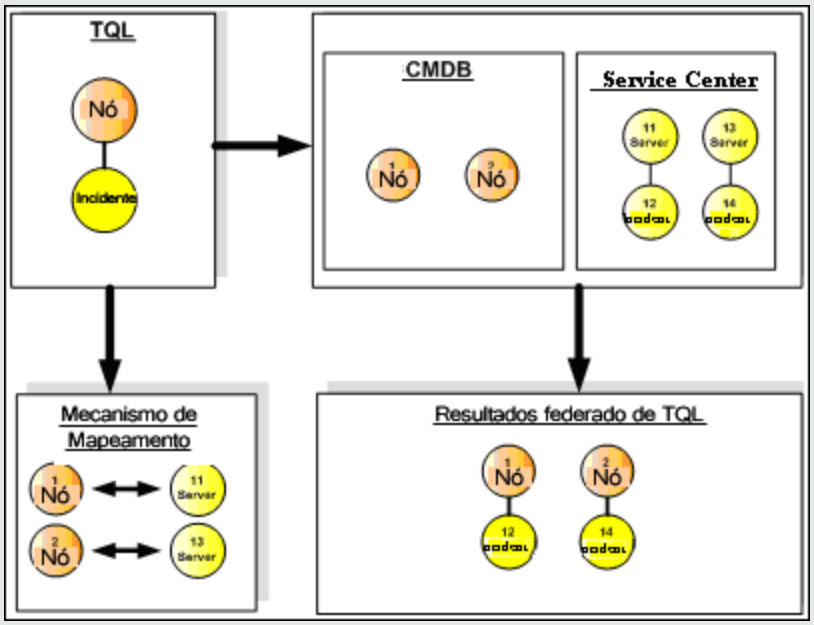
Exemplo de fluxo de federação em tempo real:

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 relacionamento virtual (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 o Data Flow Probe 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 o Data Flow Probe 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.

Fluxo de População Baseado em Instância

Se o adaptador for definido para suportar um fluxo baseado em instância (por meio da tag **<instance-based-data>**, conforme descrito em "Marcas e propriedades de configuração XML" na página 181), o mecanismo de população encontrará automaticamente ECs removidos dentro da instância e os removerá do UCMDB (supondo que a exclusão é permitida para o trabalho específico de população). Cada instância deve ter um EC Raiz, marcado na definição TQL com o nome **Raiz**. Sempre que um EC raiz é transmitido, toda a sua instância (todos os ECs conectados a ele) são comparados com a última vez em que ele foi enciado ao UCMDB e qualquer EC que foi conectado à raiz, mas não está agora conectado a ela é excluído do UCMDB. Para que o adaptador suporte corretamente o fluxo baseado em instância, qualquer alteração em qualquer EC ou atributo em toda a instância deve disparar um reenvio da instância inteira ao UCMDB.

Interação de adaptador e mapeamento com o Federation Framework

Um 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" abaixo
- "Métodos assist do adaptador" abaixo

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 no Data Flow Probe, 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 as classes externas 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" abaixo
- "Mecanismo de mapeamento" na página seguinte
- "Adaptador federado" na página seguinte

Consulte "Interações entre o Federation Framework, servidor, adaptador e mecanismo de mapeamento" na página seguinte para diagramas que ilustram as interações entre o Federation Framework, o UCMDB, o adaptador e o Mecanismo de Mapeamento.

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.

Interações entre o Federation Framework, servidor, adaptador e mecanismo de mapeamento

Os diagramas a seguir ilustram as interações 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.

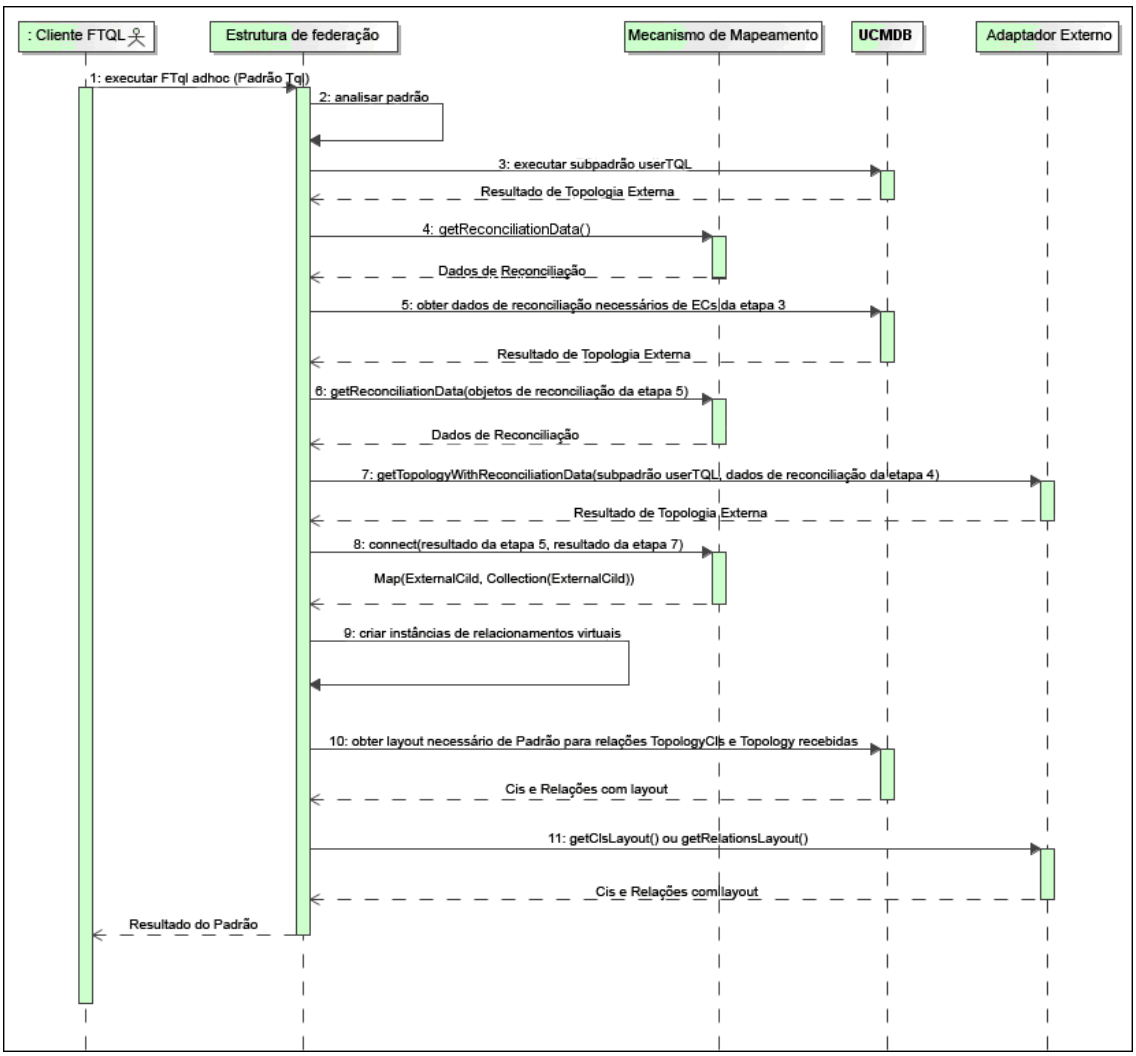
Esta seção inclui os seguintes tópicos:

- "O cálculo começa na extremidade do servidor" na página seguinte
- "O cálculo começa na extremidade do adaptador externo" na página 161
- "Exemplo do fluxo do Federation Framework para consultas TQL federadas" na página 163

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 servidor

O diagrama de sequência a seguir ilustra as interações 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.



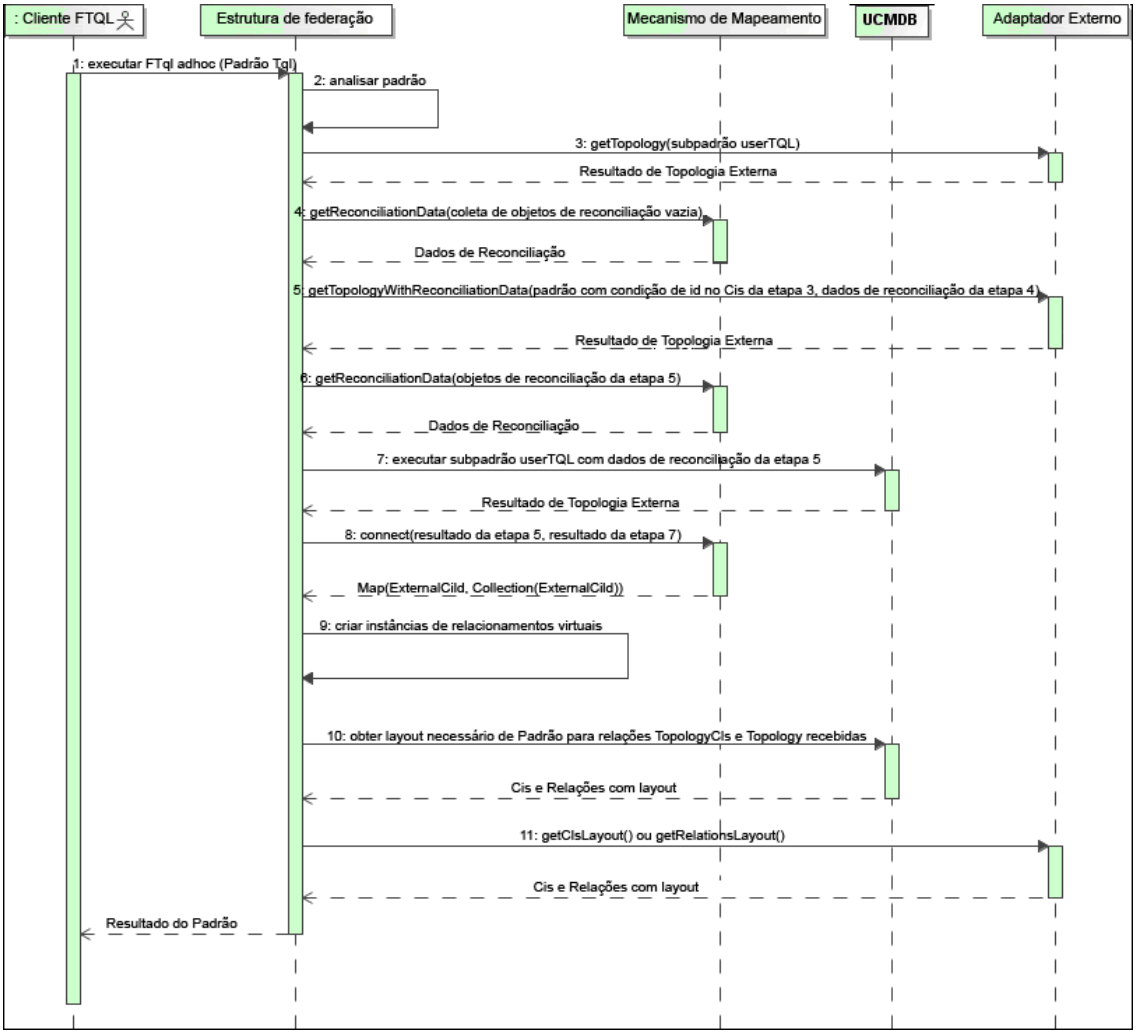
Os números nesta imagem são explicados a seguir:

Número	Explicação
1	O Federation Framework recebe uma chamada para um cálculo de TQL federado.

Número	Explicação
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" acima 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" acima), o Mecanismo de Mapeamento recebe os objetos de reconciliação da etapa "5" acima 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 do repositório de dados externo. O adaptador externo recebe os dados de reconciliação da etapa "6" acima 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" acima e o parâmetro `secondResult` é o resultado de topologia externa recebido do Adaptador Externo na etapa "7" acima. 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.

Número	Explicação
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:

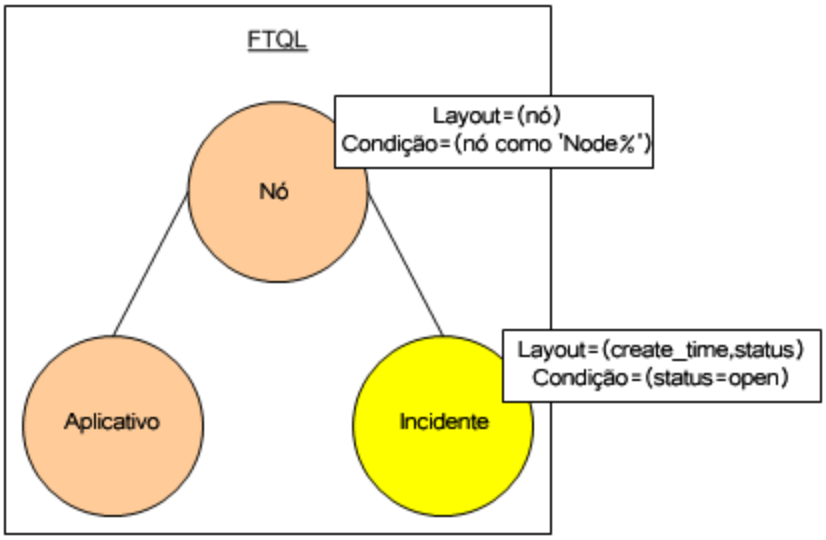
Número	Explicação
1	O Federation Framework recebe uma chamada para um cálculo de TQL federado.
2	O Federation Framework analisa o adaptador, localiza o relacionamento virtual e divide o TQL original em dois subadaptadores – um para o UCMDB e outro para o repositório de dados externo.

Número	Explicação
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.
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 etapa 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.

Número	Explicação
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.

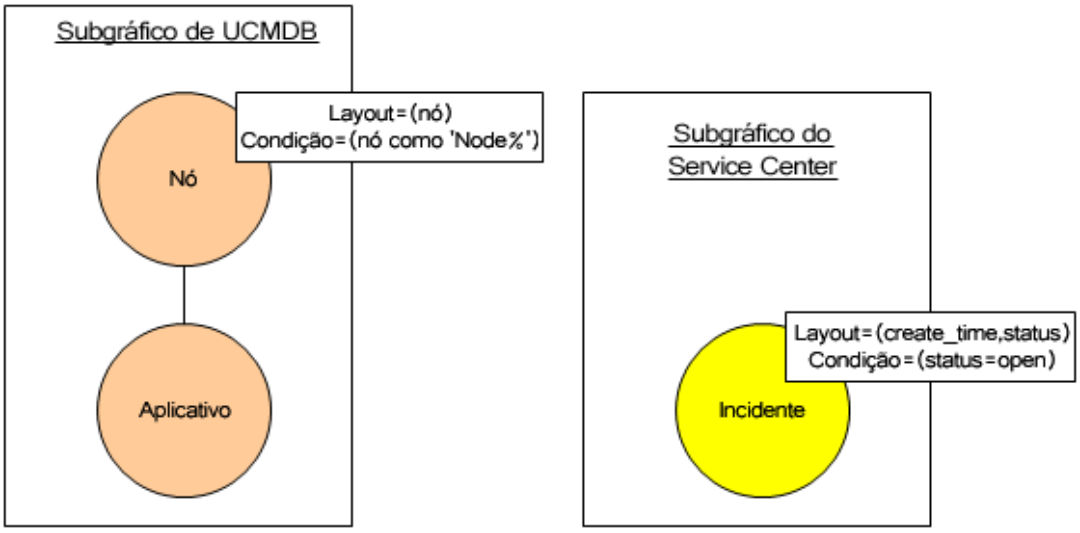
Exemplo do fluxo do Federation Framework para consultas TQL federadas

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.

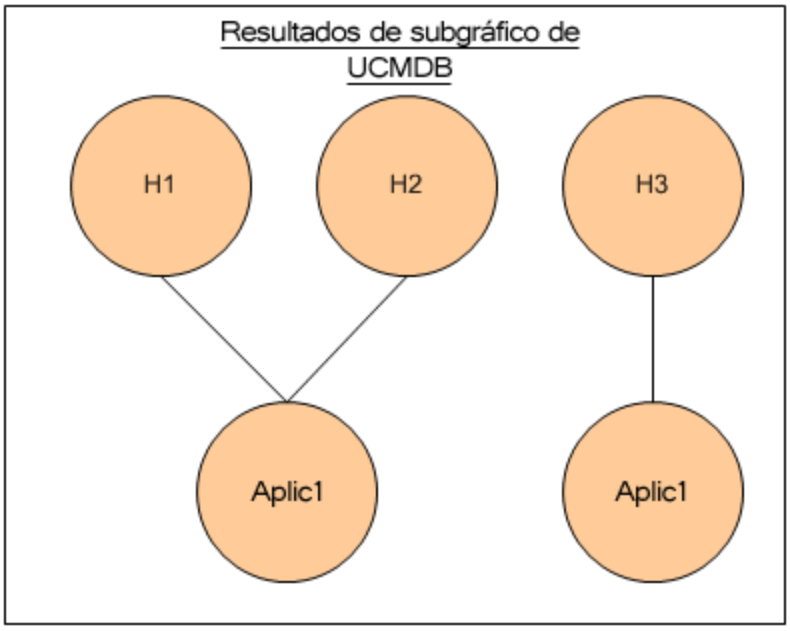


Observação: Para federação de atributos, o método **getTopology** do adaptador é chamado. Os dados de reconciliação são adaptados no TQL do usuário (nesse caso, o elemento de EC).

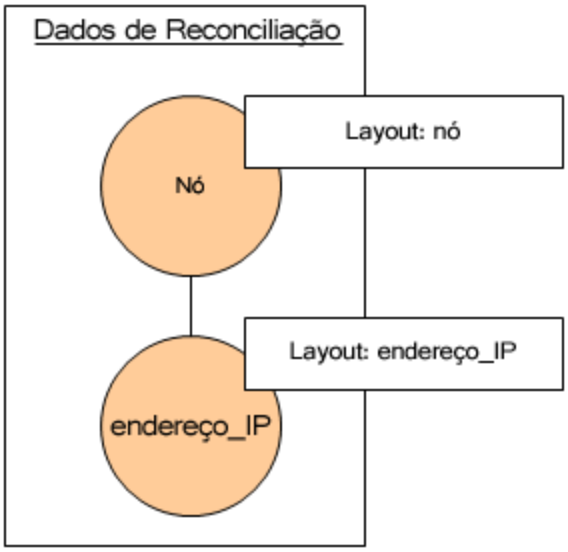
1. Depois de analisar o adaptador, o Federation Framework reconhece o relacionamento virtual entre `Node` e `Incident` e divide a consulta TQL federada em dois subgráficos.



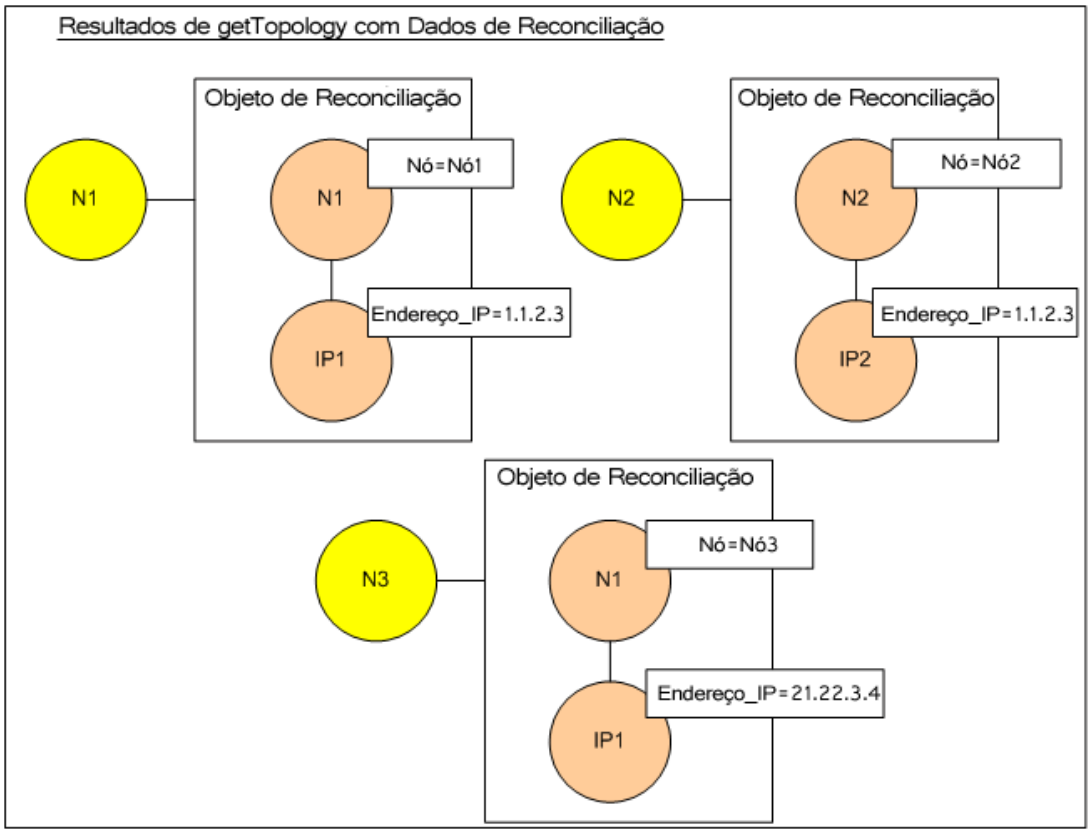
2. O Federation Framework executa o subgráfico do UCMDB para solicitar a topologia e recebe os seguintes resultados:



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:

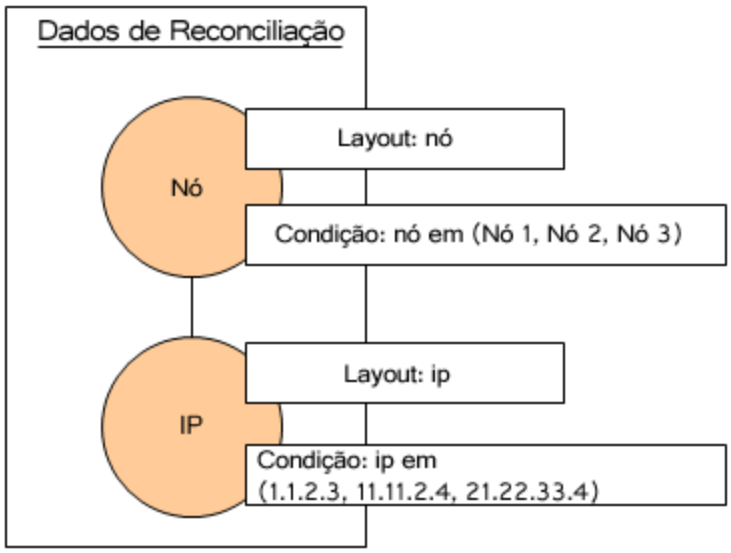


4. O Federation Framework cria uma consulta de topologia de um nó com as condições de Nó e ID nela do resultado anterior (`node` em H1, H2, H3) e executa essa consulta com os dados de reconciliação necessários no UCMDB. O resultado inclui ECs de Nó que são relevantes para a condição de ID e o objeto de reconciliação apropriado para cada EC:

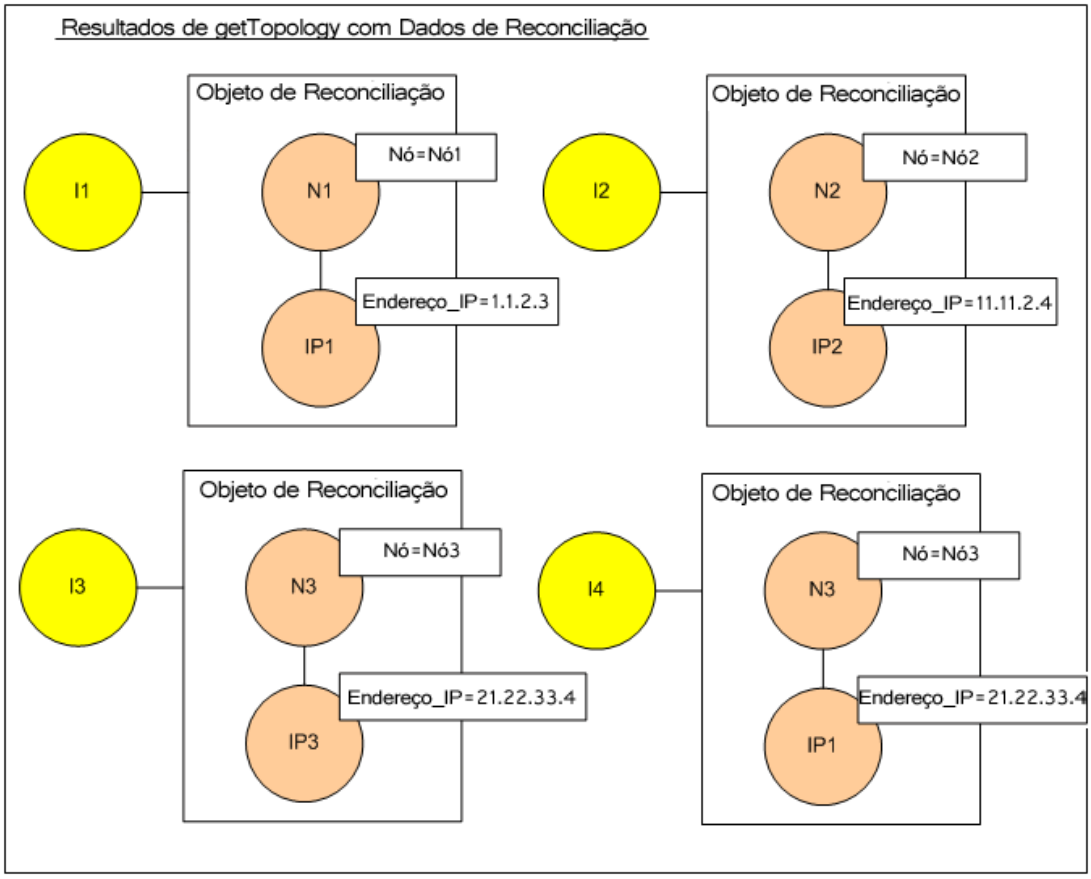


5. Os dados de reconciliação para o ServiceCenter devem conter uma condição para `node` e `ip`

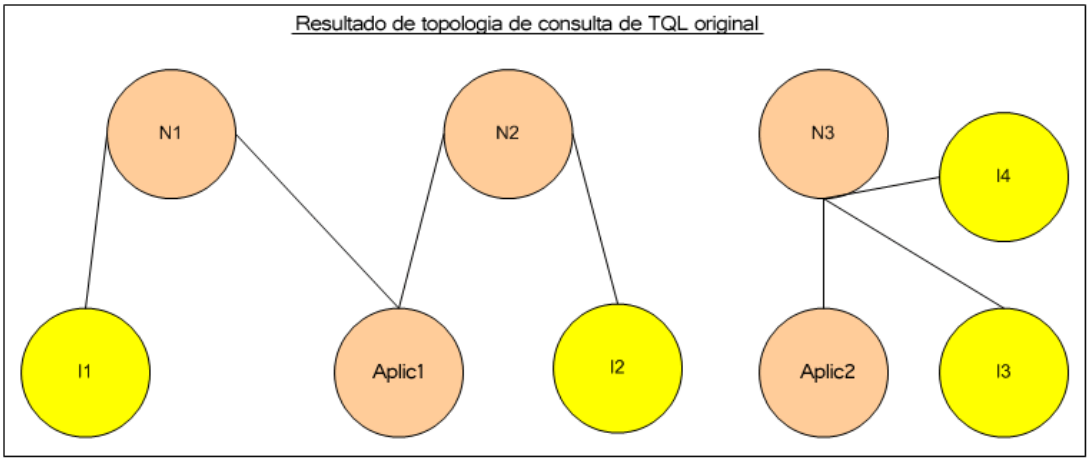
que é derivada dos objetos de reconciliação recebidos do UCMDB:



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:

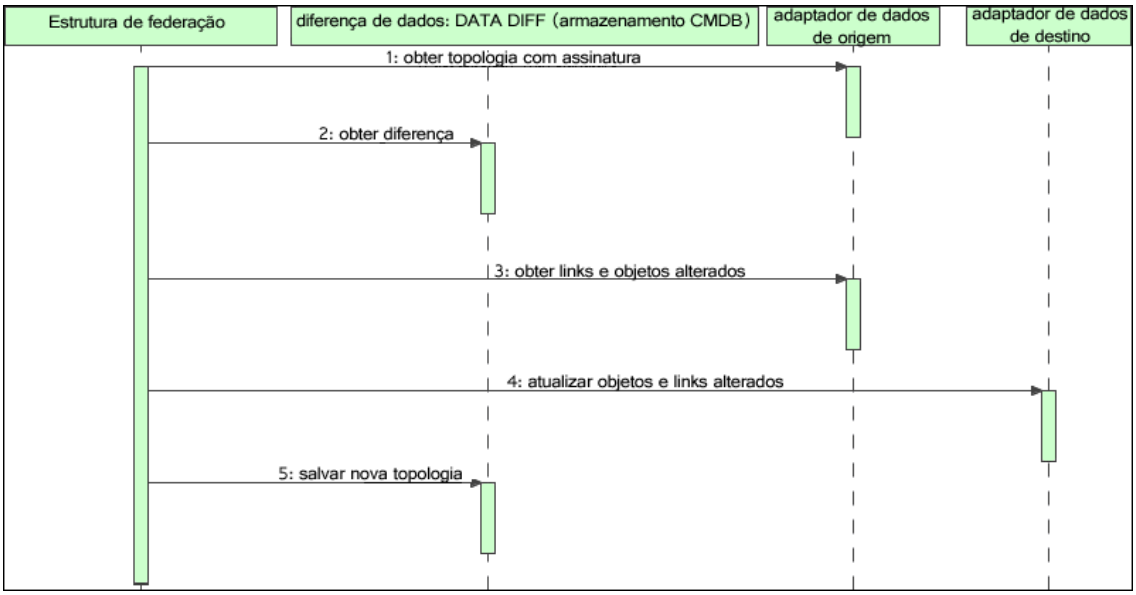
- "Definições e termos" abaixo
- "Diagrama de fluxo" abaixo

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" abaixo
- "Interfaces de adaptador para consultas TQL federadas" abaixo

Definições e termos

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 **interface de topologia de Nó Único** 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. Os dados de reconciliação necessários para concluir a operação podem ser descritos como uma complexa consulta (consulte SingleNodeFederationTopologyReconciliationAdapter a seguir).

  Todas as interfaces SingleNode são criadas para simplificar o fluxo de trabalho; para esses casos em que é necessário usar uma consulta mais extensiva, use a interface **FederationTopologyAdapter**.

- Uma **interface FederationTopologyAdapter** é usada para definir adaptadores que oferecem suporte a consultas federadas complexas. A solicitação de reconciliação nesses adaptadores faz parte do único parâmetro **QueryDefinition**.

  O mecanismo de Federação usa dados de reconciliação para conectar os dados federados aos ECs locais adequados. Os dados de reconciliação podem ser buscados em mais de uma solicitação (calculados de modo recursivo de acordo com os resultados). Nesse caso, o adaptador recebe uma solicitação com apenas dados de reconciliação.

Interfaces SingleNode

As interfaces a seguir têm diferentes tipos de dados de reconciliação:

- **SingleNodeFederationIdReconciliationAdapter.** Use se o adaptador oferecer suporte a um **TQL de nó único** e a reconciliação entre repositórios de dados for calculada pelo ID.

- **SingleNodeFederationPropertyReconciliationAdapter.** 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.

- **SingleNodeFederationTopologyReconciliationAdapter.** Use se o adaptador oferecer suporte a um **TQL de nó único** e a reconciliação entre repositórios de dados for executada pela topologia. O adaptador deve suportar o caso onde o elemento de consulta está vazio e apenas a topologia de reconciliação é solicitada.

Interfaces de adaptador de dados

- **FederationTopologyAdapter.** Use este adaptador para oferecer suporte a consultas TQL federadas complexas. Permite a maior diversidade. O adaptador deve suportar o caso onde a definição de consulta está descrevendo apenas dados de reconciliação.

- **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.

Observação: Ao desenvolver um adaptador que pode retornar grandes conjuntos de dados, é importante permitir o agrupamento implementando a interface ChunkGetter. Para obter mais informações, consulte o documento Java do adaptador específico.

Interfaces de Relatórios de Recursos

As seguintes interfaces permitem que o adaptador reporte os recursos que podem ser configurados para personalizar o comportamento do adaptador. Isso permite a edição desses recursos diretamente do Integration Studio. Essas interfaces devem ser usadas além das interfaces regulares do adaptador acima.

- **PopulationQueriesResourcesLocator.** Define quais recursos podem ser editados para cada consulta de população específica.

- **PushQueriesResourceLocator.** Define quais recursos podem ser editados para cada consulta de Push de Dados.

- **GeneralResourcesLocator.** Define quais recursos gerais podem ser editados nesse adaptador.

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.

Depurar recursos do adaptador

Esta tarefa descreve como usar o console JMX para criar, exibir e excluir recursos de estado do adaptador (qualquer recurso criado usando os métodos de manipulação de recursos na interface DataAdapterEnvironment, salvos no banco de dados do UCMDB ou da Sonda) para fins de depuração e desenvolvimento.

1. Inicie o navegador da Web e insira o endereço do servidor, da seguinte maneira:
   - Para o servidor UCMDB: `http://localhost:8080/jmx-console`
   - Para a Sonda: `http://localhost:1977`

   Você pode precisar fazer logon com um nome de usuário e senha.
2. Para abrir a página Visualização do JMX MBEAN, realize uma das seguintes opções:
   - No servidor UCMDB: clique em **UCMDB:service=FCMDB Adapter State Resource Services**
   - Na Sonda: clique em **type=AdapterStateResources**
3. Insira valores nas operações que deseja usar e clique em **Invoke**.

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 seguinte
- "Definir relacionamentos válidos para relacionamentos virtuais" na página seguinte
- "Definir uma configurações de adaptador" na página 173
- "Definir classes aceitas" na página 177

- "Implementar o adaptador" na página 178
- "Definir as regras de reconciliação ou implementar o mecanismo de mapeamento" na página 178
- "Adicionar JARs necessários para implementação no caminho de classe" na página 179
- "Implantar o adaptador" na página 179
- "Atualizar o adaptador" na página 180

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 Working with the CI Selector 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: Essa seção só é relevante 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 obter detalhes, consulte "Package Manager" 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. Vá até **Gerenciamento do Adaptador**.
   b. Clique no botão **Criar novo recurso**  e selecione **Novo Adaptador**.
   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.cor
e.
 AdapterService">

<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.cor
e.
 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.BaseDestination
Data">

<!-- check -->

<destinationData name="adapterId"
description="">${ADAPTER.adapter_id}</destinationData>

<destinationData name="attributeValues"
description="">${SOURCE.attribute_values}</destinationData>

<destinationData name="credentialsId"
description="">${SOURCE.credentials_id}</destinationData>

<destinationData name="destinationId"
description="">${SOURCE.destination_id}</destinationData>

</destinationInfo>

<resultMechanism isEnabled="true">

<autoDeleteCITs isEnabled="true">

<CIT>link</CIT>

<CIT>object</CIT>

</autoDeleteCITs>
```

```
</resultMechanism>

</taskInfo>

<adapterInfo>

<adapter-capabilities>

<support-federated-query>

<!--<classes com suporte/> <!—consulte a seção sobre classes
com suporte-->

<topologia>

<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 181.

4. **Definir classes aceitas**

Defina as classes aceitas no código do adaptador implementando o método *getSupportedClasses()* ou usando o arquivo XML de padrão.

```
<supported-classes>
    <supported-class name="HistoryChange" is-derived="false" is-reconcili
ation-supported="false" federation-not-supported="false" is-id-reconcilia
tion-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.

Se o adaptador implementa **getTopologyWithReconciliationData** e os recursos do adaptador incluem a capacidade a ser usada como ponto de partida, o adaptador também deveria suportar a solicitação de topologia com dados de reconciliação sem qualquer condições (apenas tipo). Nesse caso, o adaptador deve retornar dados completos de reconciliação dos resultados encontrados.

O suporte à reconciliação do adaptador pode ser definido de acordo com a **global_id**, sendo que, nesse caso, **global_id** deve ser definida como parte dos atributos de reconciliação nas classes suportadas do adaptador. Se o suporte à reconciliação do adaptador for definido de acordo com **global_id**, **getTopologyWithReconciliationData()** deverá retornar **global_id** como parte das propriedades do objeto de reconciliação. O UCMDB usa **global_id** para a reconciliação de resultados federados para um TEC, em vez da regra de identificação.

Parte da API da federação é a interface DataAdapterEnvironment. Essa interface representa o ambiente do adaptador de dados. Ela contém a API do ambiente necessária para que o adaptador funcione. Para obter mais informações sobre a interface DataAdapterEnvironment, consulte "A Interface DataAdapterEnvironment" na página 183.

6. **Definir as regras de reconciliação ou implementar o mecanismo de mapeamento**

Se o adaptador oferecer suporte a consultas TQL federadas, você terá duas opções para definir seu Mecanismo de Mapeamento:

- Use o mecanismo de mapeamento padrão do , que usa as regras de reconciliação interna para mapeamento do CMDB. Para usá-lo, deixe a marca XML **<default-mapping-engine/>** vazia.
- 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

   Implante o pacote de adaptadores. Para ver detalhes gerais sobre como implantar um pacote, consulte "Package Manager" 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 obter detalhes, consulte "How to Deploy a Package" no *Guia de Administração do HP Universal CMDB*.

Criar um adaptador de amostra

Este exemplo ilustra como criar um adaptador de exemplo. Esta tarefa inclui as seguintes etapas:

- "Selecionar lógica do adaptador" abaixo
- "Carregar o projeto" abaixo

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. Filtre alguns dos dados no serviço remoto e faça 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 arquivo de **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\APIs\DBAdapterFramework_JavaAPI\index.html

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" or 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.
	<start-point-adapter>	Especifica se esse adaptador é o ponto de partida para o cálculo da consulta TQL.
	<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.
	<push-back-ids>	Devolva a ID global do EC à coluna global_ id da tabela (deve ser definida no orm.xml). O comportamento pode ser substituído pela implementação do plug-in **FcmdbPluginPushBackIds**.
	<changes-source>	Esse adaptador pode ser usado para fluxos de alterações de população.
	<instance-based-data>	Essa tag define que o adaptador suporte uma instância com base no fluxo da 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
	<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.
	<adapter-settings>	A lista de configurações do adaptador.
	<list.attributes.for.set>	Determina quais atributos substituem o valor anterior (se existir).

A Interface DataAdapterEnvironment

OutputStream openResourceForWriting(String resourceName) throws FileNotFoundException;

Este método abre um recurso com um nome fornecido para escrever. Ele é usado para salvar dados persistentes para a integração. Este método deve ser usada em vez de tentar carregar arquivos usando métodos java. O usuário deve garantir que o fluxo seja fechado ao concluir a gravação no fluxo. close()/flush() salvará o recurso. Esse método cria um recurso de tempo de execução (ele pode não substituir arquivos que vieram no pacote do adaptador).

Parâmetro

- **resourceName:** O nome do recurso a recuperar. Esse nome deve ser exclusivo em todas as integrações do mesmo adaptador.

Valor de Retorno

Retorna um fluxo no qual gravar.

Exceções

- Esse método lança *FileNotFoundException* se o tipo de recurso for file e o arquivo não existir, se o recurso for um diretório, e não um arquivo regular ou por algum outro motivo o recurso não pode ser aberto para leitura.
- Esse método lança *SecurityException* se um gerenciador de segurança existir e seu método *checkRead* negar acesso ao arquivo.

InputStream openResourceForReading(String resourceName) throws FileNotFoundException;

Este método abre um recurso com um nome fornecido para leitura. Ele é usado para ler dados persistentes para a integração. Este método deve ser usado em vez de tentar carregar um arquivo usando métodos java. O usuário deve garantir que o fluxo seja fechado quando terminar de lê-lo. Primeiro ele tenta carregar arquivos que vieram no pacote do adaptador. Se não for encontrado, ele tenta carregar um recurso criado de tempo de execução de

DataAdapterEnvironment.openResourceForWriting(String). Os recursos de tempo de execução podem ser vistos usando o JMX (da sonda e do servidor de acordo).

Parâmetro

- **resourceName:** O nome do recurso a recuperar. Esse nome deve ser exclusivo em todas as integrações do mesmo adaptador.

Valor de Retorno

Retorna um fluxo para ler.

Exceções

- Esse método lança *FileNotFoundException* se o tipo de recurso for **file** e o arquivo não existir, se o recurso for um diretório, e não um arquivo regular ou por algum outro motivo o recurso não pode ser aberto para leitura.
- Esse método lança *SecurityException* se um gerenciador de segurança existir e seu método *checkRead* negar acesso de leitura ao arquivo.

Properties openResourceAsProperties(String propertiesFile) throws IOException;

Este método abre um recurso com um nome fornecido e o carrega como uma estrutura *Propriedades*. Ele é usado para ler dados persistentes para a integração. Este método deve ser usado em vez de tentar carregar os arquivos **.properties** usando métodos java. Primeiro ele tenta carregar arquivos que vieram no pacote do adaptador. Se não for encontrado, ele tenta carregar um recurso criado de tempo de execução de *DataAdapterEnvironment.openResourceForWriting (String)*. Os recursos de tempo de execução podem ser vistos usando o JMX (da sonda e do servidor de acordo).

Parâmetro

- **propertiesFile:** O nome do recurso a recuperar. Esse nome deve ser exclusivo em todas as integrações do mesmo adaptador.

Valor de Retorno

Retorna o conteúdo do arquivo representado em Propriedades.

Exceções

- Esse método lança *FileNotFoundException* se o tipo de recurso for **file** e o arquivo não existir, se o recurso for um diretório, e não um arquivo regular ou por algum outro motivo o recurso não pode ser aberto para leitura.
- Esse método lança *SecurityException* se um gerenciador de segurança existir e seu método *checkRead* negar acesso de leitura ao arquivo.
- Este método lança *IOException* se o arquivo de propriedades falhou ao converter para o Objeto *Propriedades*.

String openResourceAsString(String resourceName) throws IOException;

Este método abre um recurso com um nome fornecido e o carrega como uma cadeia. Ele é usado para ler dados persistentes para a integração. Este método deve ser usada em vez de tentar carregar arquivos usando métodos java.

Primeiro ele tenta carregar arquivos que vieram no pacote do adaptador. Se não for encontrado, ele tenta carregar um recurso criado de tempo de execução de *DataAdapterEnvironment.openResourceForWriting(String)*. Os recursos de tempo de execução podem ser vistos usando o JMX (da sonda e do servidor de acordo).

Parâmetro

- **resourceName:** O nome do recurso a recuperar. Esse nome deve ser exclusivo em todas as integrações do mesmo adaptador.

Valor de Retorno

Retorna o conteúdo do arquivo representado no formato Cadeia.

Exceções

- Esse método lança *FileNotFoundException* se o tipo de recurso for **file** e o arquivo não existir, se o recurso for um diretório, e não um arquivo regular ou por algum outro motivo o recurso não pode ser aberto para leitura.
- Esse método lança *SecurityException* se um gerenciador de segurança existir e seu método *checkRead* negar acesso de leitura ao arquivo.
- Este método lança *IOException* se um erro de E/S ocorre.

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

Este método recebe uma Cadeia e a salva como um recurso. Ele é usado para salvar dados persistentes para a integração. Este método deve ser usado em vez de tentar salvar arquivos usando métodos java. Este método converte a Cadeia em um fluxo e a salva para o recurso. Ele cria um recurso de tempo de execução, mas não pode substituir arquivos que vieram no pacote do adaptador). Os recursos de tempo de execução podem ser vistos usando o JMX (da sonda e do servidor de acordo).

Parâmetro

- **relativeFileName:** O nome do recurso a recuperar. Esse nome deve ser exclusivo em todas as integrações do mesmo adaptador.
- **value:** A Cadeia para salvar como recurso

Exceções

Este método lança IOException se um erro de E/S ocorre.

boolean resourceExists(String resourceName);

Esse método verifica se o nome do recurso fornecido existe. Ele procura arquivos que vieram no pacote do adaptador e recursos criados por tempo de execução a partir de *DataAdapterEnvironment.openResourceForWriting(String)*.

Parâmetro

- **resourceName:** O nome do recurso a recuperar. Esse nome deve ser exclusivo em todas as integrações do mesmo adaptador.

Valor de Retorno

Retorna **Verdadeiro** se *resourceName* existe.

boolean deleteResource(String resourceName);

Este método exclui o recurso fornecido de dados persistentes. Ele exclui um recurso de tempo de execução e pode não excluir arquivos que vieram no pacote do adaptador. Os recursos de tempo de execução podem ser vistos usando JMX (para a sonda e servidor de acordo).

Parâmetro

- **resourceName:** O nome do recurso a excluir. Esse nome deve ser exclusivo em todas as integrações do mesmo adaptador.

Valor de Retorno

Retorna **True** se o recurso for excluído com êxito.

Collection<String> listResourcesInPath(String path);

Esse método recupera uma lista de recursos no caminho de recurso fornecido. Ele procura arquivos que vieram no pacote do adaptador e por recursos criados por tempo de execução a partir de *DataAdapterEnvironment.openResourceForWriting(String)*. Os recursos de tempo de execução podem ser vistos usando JMX (para a sonda e servidor de acordo).

Parâmetro

- **caminho:** O caminho do recurso. Por exemplo, "META-INF/myfiles/"

Valor de Retorno

Retorna uma lista de recursos no caminho.

DataAdapterLogger getLogger();

Recupera o agente de log a ser usado pelo adaptador. Esse agente de log é usado para eventos de logs no seu adaptador.

Valor de Retorno

Retorna o agente de log usado pelo DataAdapter.

DestinationConfig getDestinationConfig();

Esse método recupera a configuração de destino da integração. Essa configuração mantém todas as configurações de conexão e execução para a integração.

Valor de Retorno

Retorna o DestinationConfig do adaptador.

int getChunkSize();

Esse método recupera o tamanho do bloco de população solicitado para essa integração.

Valor de Retorno

Retorna o tamanho do bloco de população.

int getPushChunkSize();

Esse método recupera o tamanho do bloco de push solicitado para essa integração.

Valor de Retorno

Retorna o tamanho do bloco de push.

ClassModel getLocalClassModel();

Este método recupera um modelo de classe para consultar informações sobre o modelo de classe local do UCMDB. Esse método apresenta um ClassModel atualizado. Depois que o objeto ClassModel for retornado, ele não será atualizado para nenhuma alteração de modelo de classe. Para recuperar um modelo de classe atualizado, use esse método novamente para recuperá-lo.

Valor de Retorno

Retorna o modelo de classe do UCMDB.

CustomerInformation getLocalCustomerInformation();

Esse método recupera informações do cliente para o cliente que está executando o adaptador.

Valor de Retorno

Retorna informações do cliente para o cliente que está executando o adaptador.

Objeto getSettingValue(String name);

Esse método recupera uma configuração do adaptador específico.

Parâmetro

nome: O nome da configuração.

Valor de Retorno

Retorna o valor de configuração do Objeto.

Map<String, Object> getAllSettings();

Esse método recupera todas as configurações do adaptador.

Valor de Retorno

Retorna as configurações do adaptador.

boolean isMTEnabled();

Esse método verifica se o ambiente do servidor suporta a Locação Múltipla (MT).

Valor de Retorno

Retorna **verdadeiro** se ambiente do servidor suporta MT, caso contrário retorna **falso**.

String getUcmdbServerHostName();

Esse método retorna o nome do host do servidor UCMDB local.

Valor de Retorno

Retorna o nome do host do servidor UCMDB local.

Capítulo 6: Desenvolvimento de adaptadores push

Este capítulo inclui:

Desenvolvimento e implementação de adaptadores push 189

Criar um pacote de adaptador 190

Criar Mapeamentos 194

Escrever scripts Jython 198

Suporte à sincronização diferencial 202

Consultas SQL do Adaptador Push XML Genérico 203

Adaptador Push de Serviço Web Genérico 204

Referência do arquivo de mapeamento 223

Esquema do arquivo de mapeamento 225

Esquema de resultados de mapeamento 238

Personalização 242

Desenvolvimento e implementação de adaptadores push

Os Adaptadores Push Genéricos fornecem uma plataforma comum que permite desenvolver rapidamente integrações que enviam por push dados do UCMDB para repositórios de dados externos (bancos de dados e aplicativos de terceiros). Os Adaptadores Push Genéricos são categorizados de acordo com o protocolo usado para fazer o push de dados. Para obter detalhes sobre como fazer o push via XML, usando o adaptador de push XML genérico, consulte "Consultas SQL do Adaptador Push XML Genérico" na página 203. Para obter detalhes sobre como fazer o push via Serviço Web, usando o adaptador de push de Serviço Web genérico, consulte "Adaptador Push de Serviço Web Genérico" na página 204.

O desenvolvimento de uma integração personalizada baseada no Adaptador Push Genérico exige:

- A criação de um novo pacote de adaptador a partir dos arquivos de gabarito do Adaptador Push Genérico. Para obter detalhes, consulte "Criar um pacote de adaptador" na página seguinte.
- Mapeamentos entre os tipos de vínculo de EC do UCMDB e itens de dados externos. Os mapeamentos são armazenados como XML e são personalizados para cada repositório de dados externo. Para obter detalhes, consulte "Criar Mapeamentos" na página 194.
- Um script Jython para enviar por push itens de dados ao repositório de dados externo. Para obter detalhes, consulte "Escrever scripts Jython" na página 198.

- Etapas adicionais específicas do adaptador. Por exemplo, escolher o caminho do arquivo a ser escrito para o adaptador push XML ou criar um receptor de dados para o adaptador push de Serviço Web.

Criar um pacote de adaptador

Para criar um novo adaptador de push específico de MDR, você deve fazer uma cópia do adaptador genérico e editá-lo para personalizá-lo como um adaptador para um destino push específico.

Os pacotes de adaptadores genéricos podem ser encontrados em um dos seguintes locais:

- XML Genérico adaptador push: **hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip**
- Adaptador de serviço web genérico: **hp\UCMDB\UCMDBServer\content\adapters\web-service-push-adapter.zip**

Para criar um novo adaptador push a partir do adaptador push genérico:

1. Extraia o conteúdo do arquivo zip do pacote selecionado em uma pasta de trabalho.
2. Analise os seguintes diretórios na preparação para a fase de renomear e substituir:
   - **adapterCode:** Contém o diretório implantado no **diretório C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase**. Jars implantados aqui não reiniciam automaticamente a sonda e não aparecem automaticamente no CLASSPATH da sonda.
   - **discoveryConfigFiles:** Contém as definições de mapeamentos do adaptador e pontos para o script Jython correto (**push.properties**)
   - **discoveryPatterns:** Contém a definição XML do adaptador implantado no Servidor do UCMDB
   - **discoveryScripts:** Contém os scripts Jython do adaptador pela conexão a armazenamentos de dados de terceiros feita e há o push de dados
   - **discoveryResources:** Contém **UCMDBDataReceiver.jar** contendo as classes de integração Java para o serviço Web.

   Observação: Quando você implanta esse pacote, a sonda é reiniciada para incluir esse **.jar** no CLASSPATH da sonda. Nenhuma ação é necessária além da implantação do pacote.
3. Faça as seguintes alterações na estrutura do diretório do adaptador descompactado:

a. **discoveryConfigFiles\<Your_Push_Adapter_Name>:** renomear o diretório "PushAdapter" ou "XMLtoWebService" para o nome do novo adaptador push (por exemplo, “myPushAdapter”).

b. **discoveryConfigFiles\<Your_Push_Adapter_Name>\push.properties:** No arquivo **push.properties**, faça o seguinte:

   - Atualize o nome de **jythonScript.name** para o nome do script Jython a ser usado pelo novo adaptador push (por exemplo, **pushToMyService.py**).

   - Atualize o nome do arquivo de mapeamentos a ser usado pelo novo adaptador push (por exemplo, myPushAdapter_mappings). Não adicionar a extensão **.xml**, ela é preenchida automaticamente.

c. **discoveryPatterns\<nome do adaptador push>.xml:** Renomear esse arquivo para o nome do novo arquivo XML da definição do adaptador (por exemplo, **my_push_ adapter.xml**).

d. **discoveryPatterns\<your_push_adapter>.xml:** Atualize esse arquivo da seguinte maneira:

   - Para o elemento XML **<padrão>:** defina os atributos de id e descrição de acordo. Por exemplo:

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

     alterado para:

     ```
     <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">
     ```

   - Para o elemento XML **<parâmetros>:** atualizar os elementos filhos de acordo com as necessidades do seu adaptador. Por padrão, os seguintes elementos filhos são usados para definir um adaptador push. Esses valores são atribuídos quando o ponto de integração é definido no Integration Studio após o adaptador ser configurado. Atualize a lista de parâmetros para que reflita os atributos de conexão necessários. Não remova o atributo **probeName**.

     - **host:** o nome do servidor que hospeda serviço Web

     - **porta:** porta do serviço de escuta do Receptor de Dados do UCMDB

     - **Adaptador Push de Serviço Web: uri** - o restante da URL para formar o endereço do ponto de extremidade do serviço do receptor de dados.

- **probeName:** define em qual sonda de fluxo de dados o trabalho de push é executado

- Para o elemento XML **<integração>:** atualizar o valor do elemento filho **<categoria>** para algo diferente de Genérico. Por padrão, adaptadores de integração pertencentes à categoria Genérica não são mostrados no Integration Studio. Se você está integrando com um armazenamento de dados de terceiros, defina esse valor como "Terceiros". Se você está integrando com um produto da HP BTO, defina esse valor como "Produtos HP BTO".

e. **adapterCode\PushAdapter:** renomeie essa pasta com a ID de adaptador usada na etapa anterior (por exemplo, **adapterCode\MyPushAdapter**).

f. **discoveryScripts\<your_Jython_push_script>.py:** crie um arquivo com o mesmo nome do definido na propriedade **push.properties jythonScript.name**. No arquivo **discoveryScript**, há um script que insere os ECs e vincula a um banco de dados externo Oracle. Substitua **discoveryScripts\pushScript.py** pelo script que escreveu (para obter detalhes, consulte "Escrever scripts Jython" na página 198). Se você renomear o script, a propriedade **jythonScript.name** em **adapterCode\<adapter ID>\push.properties** deverá ser atualizada de acordo.

- Adaptador Push XML: **pushScript.py**

- Adaptador Push de Serviço Web: **XMLtoWebService.py**

g. **tql\<your_integration_TQLs>:** como um pacote regular, coloque a definição XML TQL dos TQLs da sua integração nesse diretório. Todos os TQLs dessa pasta são implantados quando o pacote do adaptador é implantado.

h. **discoveryConfigFiles\<Your_Push_Adapter_Name>\mappings:** criar um arquivo de mapeamento XML por TQL que você deseja usar na sua integração. Observe que o adaptador push aplica as transformações no arquivo de mapeamento para os resultados dos TQLs de integração e, em seguida, envia esses dados em três parâmetros (**addResult**, **updateResult** e **deleteResult**) de uma tarefa ad-hoc para a sonda de fluxo de dados.

i. **adapterCode\<adapter ID>\mappings:** substitua o arquivo **mappings.xml** pelos arquivos de mapeamento que preparou (para obter detalhes, consulte "Criar Mapeamentos" na página 194).

Adaptador Push XML: Esse exemplo de mapeamento corresponde ao exemplo das tabelas criadas no ORACLE no arquivo sql_queries.

Para 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** é 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\<adapterID>\push.properties**.

4. Depois de fazer todas as alterações acima, crie um arquivo **.zip** selecionando as pastas e os arquivos especificados na etapa 3 acima (por exemplo, **my_Push_Adapter.zip**).

5. Implantar o arquivo **.zip** recém-criado no servidor do UCMDB pelo Gerenciador de Pacotes (vá até **Administração > Gerenciador de Pacotes**).

6. Crie um ponto de integração em **Gerenciamento de Fluxo de Dados > Integration Studio** e defina os TQLs de integração que o ponto de integração usa. Defina uma programação para o push de dados automático.

Solução de Problemas

O procedimento para compilar um novo adaptador push requer uma renomeação e substituição completa e correta. Qualquer erro provavelmente afetaria o adaptador. O pacote deve ser descompactado e recompactado corretamente como um pacote do UCMDB. Consulte os pacotes prontos para o uso como exemplos. Os erros comuns incluem:

- Incluir outro diretório acima dos diretórios do pacote no arquivo ZIP.
  Solução: compacte o pacote no mesmo diretório dos diretórios do pacote como **discoveryResources**, **adapterCode**, etc. Não incluir outro nível de diretório acima desse no arquivo ZIP.

- Omitir uma renomeação crítica de um diretório, um arquivo ou uma cadeia em um arquivo.
  Solução: Siga as instruções desta seção com muita atenção.

- Digitar incorretamente uma renomeação crítica de um diretório, um arquivo ou uma cadeia em um arquivo.
  Solução: Não alterar sua convenção de nomenclatura no meio do fluxo depois de começar o procedimento de renomeação. Se você perceber que precisa alterar o nome, comece completamente do zero, em vez de tentar corrigir retroativamente o nome, pois há um risco alto de erro. Além disso, use o recurso de localizar e substituir, em vez de substituir manualmente cadeias para reduzir o risco de erros.

- Implantar adaptadores com nomes de arquivos iguais de outros adaptadores, especialmente nos diretórios **discoveryResources** e **adapterCode**.
  Solução: Você pode estar usando uma versão do UCMDB com um problema conhecido que impede que arquivos de mapeamentos tenham o mesmo nome de qualquer outro adaptador no mesmo ambiente do UCMDB. Se você tentar implantar um pacote com nomes duplicados, a implantação do pacote falhará. Esse problema pode ocorrer mesmo se esses arquivos estiverem em diretórios diferentes. Além disso, esse problema pode ocorrer, mesmo se os duplicados estiverem no pacote ou com outros pacotes implantados anteriormente.

Nesse ponto, você pode criar um novo trabalho do adaptador push no Integration Studio usando o novo adaptador que você acabou de implantar.

Práticas recomendadas de TQL para adaptadores push

1. Crie uma estrutura de pasta no TQL e nas árvores de Visualização e mantenha todos os novos TQLs e visualizações lá. Usar uma convenção de nomenclatura.
2. A menos que o TQL seja pequeno, copie primeiro o TQL mais similar.
3. Faça uma alteração de cada vez. Salve, teste e visualize após cada alteração. Repita até que os resultados estejam em conformidade com seus requisitos.

Criar Mapeamentos

Os dados do resultado do TQL bruto estão na forma do esquema de modelo de classe do UCMDB. É provável que o consumidor use um modelo de dados diferente. O adaptador push fornece um mecanismo de mapeamento para transformar os dados em um formato mais adequado para consumo. Os mapeamentos realizam transformações diretas e complexas, de conversão direta, de tipo de nomenclatura, a funções de agregação pai/filho e de referência.

A especificação de mapeamento pode ser encontrada na seção "Referência do arquivo de mapeamento" na página 223. Usar a referência para criar um arquivo de mapeamento.

Observação: O arquivo de propriedades do adaptador refere-se ao nome do arquivo de mapeamento. Em arquivos de configuração do adaptador, o adaptador implementa uma estrutura de pasta usando o nome do adaptador. Renomeie essa pasta ao implementar um adaptador para manter a exclusividade necessária pelo gerenciador de pacote.

Compilar um Arquivo de Mapeamento

Para compilar um arquivo de mapeamento:

1. Inicie com um arquivo de mapeamento padrão.
2. Implante o adaptador e execute-o uma vez.
3. Observe os resultados.
4. Identifique e observe o que deve ser alterado.
5. Faça as mudanças identificadas na etapa anterior. A lista a seguir pode ajudar a orientar a ordem das alterações.
    a. Comece com a seção superior não transformadora. Verifique se o adaptador é executado após cada alteração.
    b. Altere a seção de ECs de origem para os nomes do UCMDB no resultado do TQL.
    c. Primeiro mapeie as chaves.

d. Então, adicione todos os mapeamentos diretos.

e. Adicione os mapeamentos complexos.

f. Adicione os mapeamentos de vínculo.

Repita as etapas 2 a 5 até que os dados mapeados estejam adequados para consumo. Selecione o pacote de adaptador apropriado Genérico a partir do qual deseja criar o novo adaptador push.

Os arquivos de mapeamentos trabalham da mesma maneira para todos os tipos de adaptadores push. O adaptador push XML genérico grava os resultados mapeados em um arquivo. O adaptador push so Serviço Web Genérico envia os resultados XML para um receptor de dados. Para obter mais detalhes, consulte "Adaptador Push de Serviço Web Genérico" na página 204.

Preparar os arquivos de mapeamento

Observação: Você pode recuperar todos os ECs e relacionamentos como eles são no CMDB sem mapeamento, não criando o arquivo **mappings.xml**. Isso retorna todos os ECs e relacionamentos com todos os seus atributos.

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.

  Cada arquivo de mapeamento é nomeado com o formato **<nome da consulta>.xml**.

Para obter detalhes, consulte "Esquema do arquivo de mapeamento" na página 225.

Esta tarefa inclui as seguintes etapas:

- "Criar um arquivo mappings.xml" abaixo
- "Mapear ECs" na página seguinte
- "Mapear vínculos" na página 197

1. Criar um arquivo mappings.xml

A estrutura do arquivo de mapeamento é criada como a seguir (usar um arquivo existente como gabarito):

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

```
    <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**

   Há duas maneiras de mapear tipos de EC do CMDB:

   - Mapear um tipo de EC para que ECs desse tipo e todos os tipos herdados sejam mapeados da mesma forma:

```
<source_ci_type_tree name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey>
            <pkey>name</pkey>
        </targetprimarykey
        <target_attribute name=" name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
        <!-- mais atributos de destino --->
    </target_ci_type>
</source_ci_type_tree>
```

   - Mapear um tipo de EC para que somente ECs desse tipo sejam processados. ECs de tipos herdados não serão processados, a menos que seu tipo também seja mapeado (de uma das duas maneiras a seguir):

```
<source_ci_type name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey>
            <pkey>name</pkey>
        </targetprimarykey
        <target_attribute name=" name" datatype="STRING">
```

```
            <map type="direct" source_attribute="name" >
        </target_attribute>
        <!-- mais atributos de destino --->
    </target_ci_type>
</source_ci_type>
```

Um tipo de EC mapeado indiretamente (um de seus ancestrais é mapeado usando **source_ci_type_tree**) também pode substituir o mapa de seu pai fazendo com que ele seja exibido em seu próprio **source_ci_type_tree** ou **source_ci_type**.

É recomendável usar **source_ci_type_tree** sempre que possível. Caso contrário, os ECs resultantes de um tipo de EC não exibidos nos arquivos de mapeamento não serão transferidos ao script do Jython.

3. **Mapear vínculos**

Há duas maneiras de mapear links:

- Mapear um link para que links desse tipo e todos os links herdados sejam mapeados da mesma forma:

```
<source_link_type_tree name="dependency" target_link_type="dependency"
mode="update_else_insert" source_ci_type_end1="webservice" source_ci_t
ype_end2="sap_gateway">
    <target_ci_type_end1 name="webservice" >
    <target_ci_type_end2 name="sap_gateway" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
</source_link_type_tree>
```

- Mapear um link para que somente links desse tipo sejam processados. Links de tipos herdados não serão processados, a menos que seu tipo também seja mapeado (de uma das duas maneiras 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" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
    </target_attribute>
</link>
```

Escrever scripts Jython

O script de mapeamento é um script Jython comum, e deve seguir as regras aplicáveis a scripts Jython. Para obter detalhes, consulte "Desenvolvimento de adaptadores Jython" na página 38.

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 a etapa sobre edição do arquivo **discoveryPatterns\push_adapter.xml** em "Criar um pacote de adaptador" na página 190), 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" abaixo
- "Lidando com a conexão de teste no script" na página 201

Trabalhando com resultados do mapeamento

Os adaptadores push genéricos criam 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.

No XML, o atributo `id` de um EC armazena `cmdbId` como uma id externa ou `ExternalId` desse EC se o EC tinha um `ExternalId` quando o EC foi enviado ao script. Os campos `end1Id` e `end2Id` do link mantêm para cada uma das extremidades do link `cmdbId` como uma id externaa ou `ExternalId` da extremidade desse link se o EC da extremidade do link tinha um `ExternalId` quando ele foi enviado ao script.

Ao processar os ECs no script do Jython, o valor de retorno do script é um mapeamento entre a id do CMDB do EC e a id fornecida (a id fornecida a cada EC do script). Se um CI sofreu push pela primeira vez, a id que está no XML desse EC é a id do CMDB. Se um EC não sofreu push pela primeira vez, a id do EC é a mesma id que foi fornecida a esse EC no script quando ele sofreu push pela primeira vez.

A id é recuperada do script XML do EC como a seguir:

1. A partir do elemento do EC no XML, recupere a id do atributo de id. Por exemplo: `id = objectElement.getAttributeValue('id')`.

2. Após recuperar a id do XML, restaure a id do atributo (cadeia). Por exemplo: `objectId = CmdbObjectID.Factory.restoreObjectID(id)`.

3. Verifique se o `objectId` recebido na etapa anterior é a id do CMDB. Você pode fazer isso verificando se o `objectId` tem a nova id fornecida a ele pelo script. Se tiver, a id retornada não é a ID do CMDB. Por exemplo:
   `newId = objectId.getPropertyValue(<o nome do atributo id fornecido pelo script>)`.

   Se `newId` for nulo, a id retornada no XML é uma id do CMDB.

4. Se a id for uma id do CMDB (isto é, `newId` for nulo), faça o seguinte (se a id não for uma id do CMDB, vá para a etapa 5):

   a. Crie uma propriedade para aquele EC com a nova id. Por exemplo: `propArray = [TypesFactory.createProperty('<o nome do atributo id fornecido pelo script>', '<new id>')]`.

   b. Crie um `externaId` para aquele EC. Por exemplo:
   ```
   cmdbId = extI.getPropertyValue('internal_id')
   className = extI.getType()
   externalId = ExternalIdFactory.createExternalCiId(className, propArray)
   ```

   c. Mapeie a id do CMDB para o recém-criado `externalId` (e na próxima etapa retorne esse mapeamento ao adaptador). Por exemplo: `objectMappings.put(cmdbId, externalId)`

   d. Quando todos os ECs e links forem mapeados:
   ```
   updateResult = DataPushResultsFactory.createDataPushResults
   (objectMappings, linkMappings);
   ```

```
return updateResult
```

5. Se a id for a nova id (isto é, `newId` não for nulo), `externalId` é `newId`.

Também é possível reportar o status do push para cada EC e vincular como a seguir:

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

   onde `updateStatus` é uma instância da classe `UpdateStatus` que contém status dos ECs e vínculos.

2. Adicione um status a `updateStatus` chamando o método `reportCIStatus` ou `reportRelationStatus`.

   Por exemplo:

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

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

   Onde `ERROR_CODE_CI` é o número das mensagens de erro como aparecem no arquivo do adaptador **properties.errors** (para detalhes no arquivo **properties.errors**, consulte "Convenções da criação de mensagens" na página 66) e `errorParams` contém os parâmetros a serem transmitidos para a mensagem. Consulte **ReplicationActionDataFactory** javadoc para mais detalhes.

3. Crie um resultado push com o status como o seguinte:

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

   ```
   return updateResult
   ```

Exemplo do resultado do XML

```
<root>
  <data>
    <objects>
      <Object mode="update_else_insert" name="UCMDB_UNIX" operation="add" ma
mId="0c82f591bc3a584121b0b85efd90b174" id="HiddenRmiDataSource%0Aunix%0A1%0A
internal_id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A">
        <field name="NAME" key="false" datatype="char" length="255">UNIX5</field>
        <field name="DATA_NOTE" key="false" datatype="char" length="255"></field>
      </Object>
    </objects>
    <links>
```

```
        <link targetRelationshipClass="TALK" targetParent="unix" targetChild="unix"
operation="add" mode="update_else_insert"
mamId="265e985c6ec51a8543f461b30fa58f81"
id="end1id%5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%3D41372a1
cbcaba27b214b84a2ec9eb535%0A%5D%0Aend2id%
5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%3D0c82f591bc3a584121
b0b85efd90b174%0A%5D%0AHiddenRmi
DataSource%0Atalk%0A1%0Ainternal_id%3DSTRING%3D265e985c6ec51a8543f461b30fa58
f81%0A">

        <field name="DiscoveryID1">41372a1cbcaba27b214b84a2ec9eb535</field>

        <field name="DiscoveryID2">0c82f591bc3a584121b0b85efd90b174</field>

        <field name="end1Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%
3D41372a1cbcaba27b214b84a2ec9eb535%0A</field>

        <field name="end2Id">HiddenRmiDataSource%0Aunix%0A1%0Ainternal_id%3DSTRING%
3D0c82f591bc3a584121b0b85efd90b174%0A</field>

        <field name="NAME" key="false" datatype="char" length="255">TALK4</field>

        <field name="DATA_NOTE" key="false" datatype="char" length="255"></field>

        </link>

    </links>

  </data>

</root>
```

Observação: Se `datatype="BYTE"`, o valor do resultado retornado é uma **cadeia** gerada como: `nova cadeia([o atributo de matriz de byte])`. O `objeto byte[]` pode ser reconstruído por: `<a cadeia recebida>.getBytes()`. Se houver diferenças na localidade padrão entre o servidor e a sonda, a reconstrução é realizada de acordo com a localidade padrão do servidor.

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

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. As últimas IDs são do tipo **ExternalId**.

O comando **ExternalIdUtil.restoreExternal**, que recebe a ID do EC no CMDB como um parâmetro, restaura a ID externa a partir da ID do EC no CMDB. Esse comando pode ser usado, por exemplo, ao realizar uma sincronização diferencial e um link é recebido onde uma de suas extremidades não está na massa (já foi sincronizada).

Se o método **DiscoveryMain** no script Jython no qual o adaptador push foi baseado retornar uma instância de **ObjectStateHolderVector** 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.

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 obter detalhes, consulte "Criar um pacote de adaptador" na página 190.

Essa tarefa permite que o adaptador push realize a sincronização diferencial.

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 ExternalI
dUtil
```

- Use a classe de fábrica **DataPushResultsFactory** para obter o objeto **DataPushResults** da função **DiscoveryMain**.

```
# Criar o objeto UpdateResult

updateResult = DataPushResultsFactory.createDataPushResults(objectMappings, li
nkMappings);
```

- 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):

```
externaCIlId = ExternalIdFactory.createExternalCmdbCiId(ciType, ciIDAsStrin
g)

externalRelationId = ExternalIdFactory.createExternalCmdbRelationId(linkTyp
e, 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.

Consultas SQL do Adaptador Push XML Genérico

No pacote do adaptador, o arquivo **sql_queries** localizado em **adapterCode > PushAdapter > sqlTablesCreation,** contém as consultas necessárias para criar tabelas em um novo esquema no

Oracle para testar o adaptador. As tabelas correspondem ao arquivo adapterCode\<adapter ID>\mappings\mappings.xml.

Observação: O arquivo **sql_queries** não é necessário para o adaptador. É apenas um exemplo.

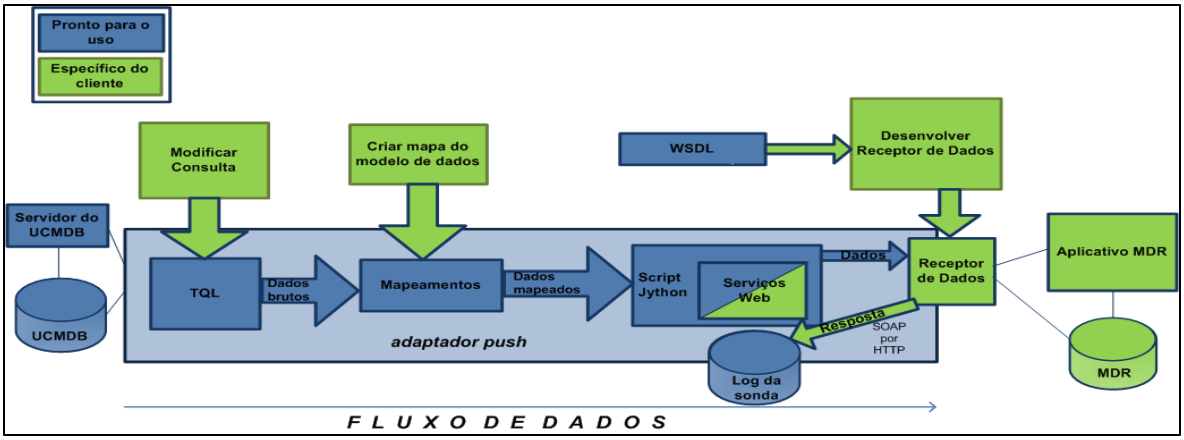
Adaptador Push de Serviço Web Genérico

O adaptador push de serviço web genérico fornece um push iniciado pelo UCMDB de mensagens SOAP contendo dados de consulta para um receptor de dados do serviço web. Os resultados mapeados são enviados em mensagens SOAP padrão pelo protocolo HTTP POST ao receptor de dados. O receptor de dados deve entender as mensagens SOAP produzidas pelo adaptador push. Para facilitar o desenvolvimento do receptor de dados adequado, um WSDL é fornecido com esse adaptador push.

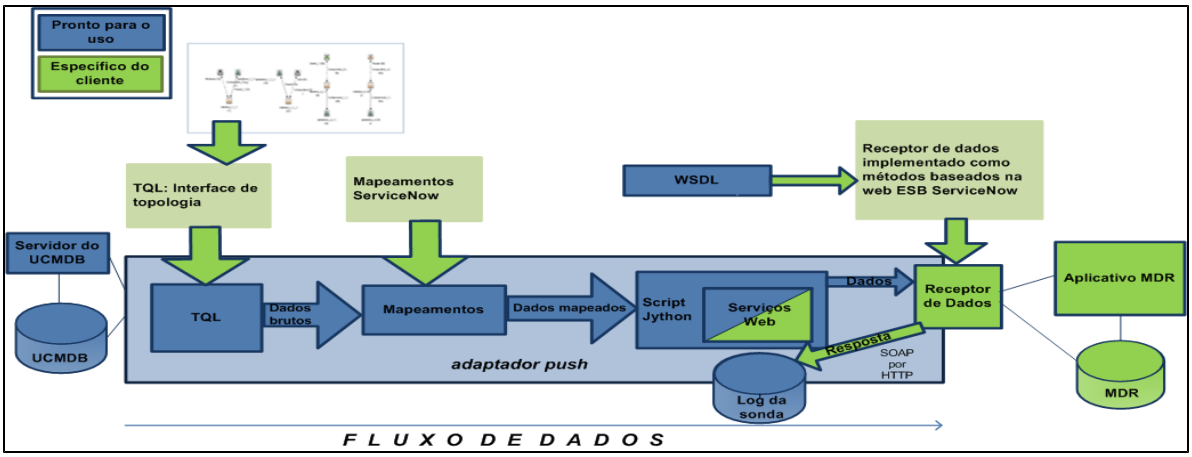
O processamento personalizado do XML de resposta da mensagem SOAP é possível no script Jython.

Para entender o formato dos dados mapeados de entrada, o desenvolvedor do receptor de dados deve se comunicar com o desenvolvedor do arquivo de mapeamentos. Um **.xsd** no momento não é fornecido com essa versão do adaptador push do serviço web, então dados devem ser processados de um modo que reflita os dados de entrada, uma combinação do TQL original e dos mapeamentos aplicados.

As funções do adaptador push do serviço web para fazer o push de dados para o cliente são mostradas abaixo. Os itens em verde são personalizados ou fornecidos pelo cliente para implementar o adaptador para um destino de push específico. Os itens em azul são componentes prontos para o uso.

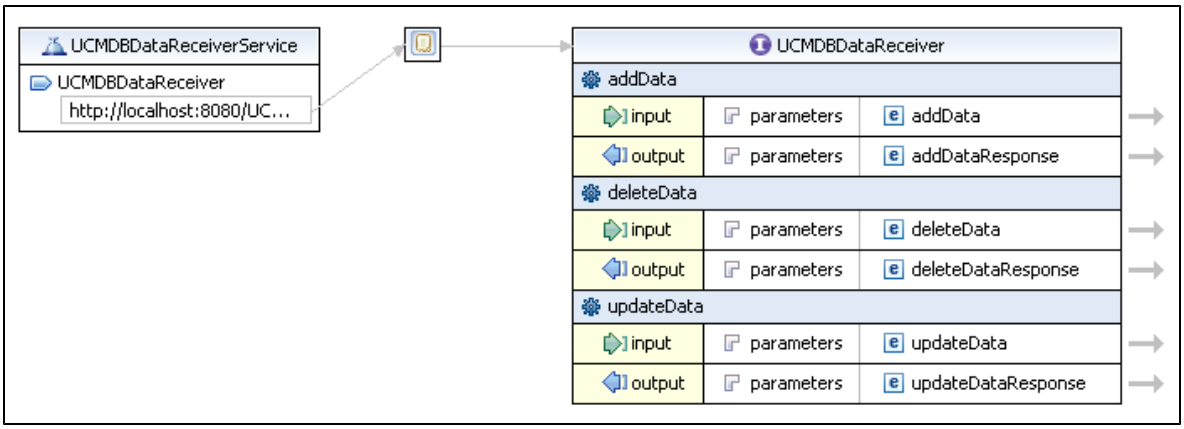


Um exemplo de uma implementação do adaptador push do serviço web genérico a um adaptador push específico do MDR que usa um Enterprise Service Bus (ESB) é mostrado aqui:



WSDL

Um WSDL é fornecido ao desenvolvedor cliente para criar um receptor de dados capaz de comunicar-se com o adaptador push do UCMDB por um serviço web. O **UCMDBDataReceiver.wsdl** descreve as mensagens SOAP usadas para comunicar dados do UCDMB para o receptor de dados. O diagrama de design do WSDL é mostrado aqui:



O receptor de dados (que é na prática um servidor ou “ponto de extremidade de serviço" na terminologia SOAP) deve implementar três métodos: **addData**, **deleteData** e **updateData**, correspondendo aos conjuntos de dados que recebem push pelo UCMDB. Os cabeçalhos HTTP contêm a palavra-chave **SoapAction** correta que indica os tipos de dados que estão sendo enviados. O receptor de dados é responsável por implementar a lógica dos negócios e processar os dados.

A URL WSDL padrão é:

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

Conforme implemented pelo Receptor de Dados, a URL pode ficar parecida com o seguinte:

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

A URL do serviço Web é a mesma da URL do WSDL sem “?wsdl” no final.

A origem do WSDL está incluída abaixo:

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

<wsdl:definitions targetNamespace="http://ucmdb.hp.com" xmlns:apachesoap="ht
tp://xml.apache.org/xml-soap" xmlns:impl="http://ucmdb.hp.com" xmlns:intf="h
ttp://ucmdb.hp.com" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdl
soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/20
01/XMLSchema">

<!--WSDL criado pela versão Axis do Apache: Compilação 1.4 de 22 de abril de
2006 (06:55:48 PDT)-->

    <wsdl:types>

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

            <element name="addData">

                <complexType>

                    <sequência>

                    <element name="xmlAdded" type="xsd:string"/>

                    </sequence>

                </complexType>

            </element>

            <element name="addDataResponse">

                <complexType/>

            </element>

            <element name="deleteData">

                <complexType>

                    <sequência>

                    <element name="xmlDeleted" type="xsd:string"/>

                    </sequence>

                </complexType>

            </element>

            <element name="deleteDataResponse">

                <complexType/>

            </element>

            <element name="updateData">

                <complexType>
```

```
                    <sequência>
                    <element name="xmlUpdate" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="updateDataResponse">
                <complexType/>
            </element>
        </schema>
    </wsdl:types>

    <wsdl:message name="addDataRequest">
        <wsdl:part element="impl:addData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="deleteDataResponse">
        <wsdl:part element="impl:deleteDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="updateDataResponse">
        <wsdl:part element="impl:updateDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="deleteDataRequest">
        <wsdl:part element="impl:deleteData" name="parameters">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="addDataResponse">
        <wsdl:part element="impl:addDataResponse" name="parameters">
        </wsdl:part>
    </wsdl:message>
```

```
<wsdl:message name="updateDataRequest">
    <wsdl:part element="impl:updateData" name="parameters">
    </wsdl:part>
</wsdl:message>
<wsdl:portType name="UCMDBDataReceiver">
    <wsdl:operation name="addData">
        <wsdlsoap:operation soapAction="addDataRequest"/>
        <wsdl:input message="impl:addDataRequest" name="addDataRequest">
        </wsdl:input>
        <wsdl:output message="impl:addDataResponse" name="addDataRespons
        e">
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="deleteData">
        <wsdlsoap:operation soapAction="deleteDataRequest"/>
        <wsdl:input message="impl:deleteDataRequest" name="deleteDataReq
        uest">
        </wsdl:input>
        <wsdl:output message="impl:deleteDataResponse" name="deleteDataR
        esponse">
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="updateData">
        <wsdlsoap:operation soapAction="updateDataRequest"/>
        <wsdl:input message="impl:updateDataRequest" name="updateDataReq
        uest">
        </wsdl:input>
        <wsdl:output message="impl:updateDataResponse" name="updateDataR
        esponse">
        </wsdl:output>
    </wsdl:operation>
</wsdl:portType>
```

```
<wsdl:binding name="UCMDBDataReceiverSoapBinding" type="impl:UCMDBDataRe
ceiver">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap
    .org/soap/http" />
    <wsdl:operation name="addData">
        <wsdl:input name="addDataRequest">
            <wsdlsoap:body use="literal" />
        </wsdl:input>
        <wsdl:output name="addDataResponse">
            <wsdlsoap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="deleteData">
        <wsdl:input name="deleteDataRequest">
            <wsdlsoap:body use="literal" />
        </wsdl:input>
        <wsdl:output name="deleteDataResponse">
            <wsdlsoap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="updateData">
        <wsdl:input name="updateDataRequest">
            <wsdlsoap:body use="literal" />
        </wsdl:input>
        <wsdl:output name="updateDataResponse">
            <wsdlsoap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="UCMDBDataReceiverService">
    <wsdl:port binding="impl:UCMDBDataReceiverSoapBinding" name="UCMDBDa
    taReceiver">
```

```
            <wsdlsoap:address location="http://localhost:8080/UCMDBDataRecei
            ver/services/
            UCMDBDataReceiver"/>

        </wsdl:port>

    </wsdl:service>

</wsdl:definitions>
```

Manuseio da resposta

O receptor de dados deve retornar uma cadeia nas estruturas **addDataResponse**, **deleteDataResponse** ou **updateDataResponse**. O adaptador passa os dados de resposta não processados ao **probeMgr-adaptersDebug.log** da sonda. O receptor pode retornar qualquer dado de cadeia e as respostas ficam encapsuladas no XML compatível com SOAP. No script Jython, você pode usar **SOAPMessage** e relacionado classes Java relacionadas para analisar as mensagens de resposta. Veja a seguir um exemplo de uma mensagem de resposta do receptor de dados:

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

<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

<intf:addDataResponse xmlns:intf="http://ucmdb.hp.com">

    <xml>&lt;result&gt;&lt;status&gt;error&lt;/status&gt;
    &lt;message&gt;Error publishing config item changes&lt;/message&gt;
    &lt;/result&gt;</xml>

</intf:addDataResponse>

</soapenv:Body>
```

A mensagem mostrada é uma mensagem de erro **<Erro ao publicar mudanças no elemento de configuração>**, mas o conteúdo pode ser qualquer coisa com a qual o receptor de dados foi projetado para responder. A resposta é uma mensagem de erro simplesmente porque essa é a intenção, porque o designer diz que isso é uma mensagem de erro e o adaptador push espera que a resposta seja alguma indicação de êxito ou falha. O conteúdo pode ser IDs de reconciliação de todas as IDs adicionadas com êxito ou mensagens de erro para ECs específicos. A personalização do GWSPA poderia incluir a análise da mensagem de resposta e a realização de ações como reenviar certos ECs ou realizar outros registros.

Testando WSDL

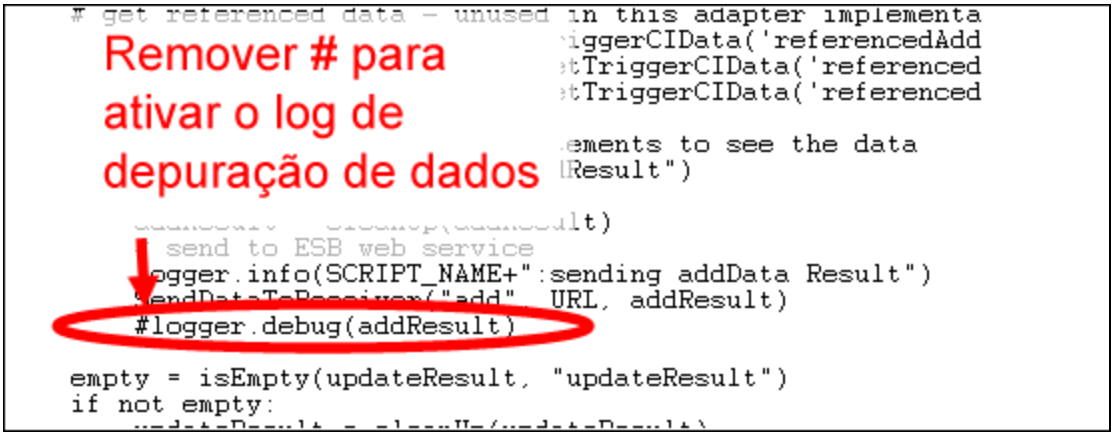
O plugin SOAPUI Eclipse é usado para testar camadas de service web durante a implantação. Você pode usar SOAPUI para ajudar na personalização de um serviço web. O SOAPUI oferece um ambiente de desenvolvimento integrado (IDE) para testar a criação, o envio e o recebimento de mensagens SOAP. Na perspectiva do SOAPUI, o WSDL nas páginas 206-210 geraram a seguinte mensagem de amostra:

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

O "?" no elemento **xmlAdded** acima é o local dos dados, fornecidos pela integração do adaptador push do serviço web.

Observando Resultados

Quando o adaptador push está funcionando normalmente, em modo de não depuração, os dados nunca são gravados em um arquivo até que o resultado final seja escrito (os resultados de TQL intermediários e os resultados dos dados mapeados não são normalmente visíveis em nenhum arquivo de log). No entanto, os resultados podem ser escritos no arquivo de depuração da sonda ao cancelar o comentário das instruções **logger.debug** (remova o caractere "#") na seção DiscoveryMain conforme mostrado aqui:



Verifique se a instrução do logger começa na mesma coluna das linhas anteriores e seguintes. O Jython é sensível a recuo e o script falhará se o recuo de todas as linhas não estiver correto.

O arquivo de log de depuração **probeMgr-adaptersDebug.log** da sonda aqui mostra o conteúdo da saída:

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

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

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

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

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

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

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

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

<root>

    <data>

        <objects>

            <Object mode="" name="u_imp_ip_switch" operation="add" mamId="9e
            8c2f6bdfe4b7d0864c79e70833902c">

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

                <field name="name" key="false" datatype="char" length="">nm
                a_09sw</field>

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

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

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

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

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

            </Object>
```

Modificando o Script Jython XMLtoWebService.py

O script Jython usado pelo adaptador push do Serviço Web é muito similar ao adaptador push do XML. O script usa **UCMDBDataReceiver.jar**, incluído no adaptador. O script implementa o método **SendDataToReceiver()**. **SendDataToReceiver()** usa três parâmetros:

1. Ação (adicionar, atualizar ou excluir)
2. A URL do Receptor de Dados
3. Os dados

Por exemplo, o bloco de adição parece com isto: **SendDataToReceiver("add", URL, addResult)**

Todas as camadas de SOAP e serviço web são encapsuladas. A URL é o endereço do ponto de extremidade do serviço do receptor de dados do UCMDB. É a mesma URL usada para obter o wsdl pelo sufixo "?wsdl".

A origem do script Jython é indicada abaixo. As linhas do encapsulador de integração do serviço web são realçadas em verde.

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

# script: XMLtoWebService.py

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

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

# and sends it to a web service. O serviço Web é chamado UCMDBDataReceiver.

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

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

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

# expressões regulares

import re

# logging

importar agente de log

# web service interface

from com.hp.ucmdb import SendDataToReceiver

from com.hp.ucmdb.SendDataToReceiver import locateService

from com.hp.ucmdb.SendDataToReceiver import SendData

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

########      VARIABLES            ##########
```

```
################################################
SCRIPT_NAME = "XMLtoWebService.py"
logger.info(SCRIPT_NAME+" started")
def cleanUp(str):

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

    # replace mamId with id
    str = re.sub("\smamId=\"", " id=\"", str)

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

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

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

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

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

    return str
```

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

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

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

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

    logger.info(SCRIPT_NAME+":ESB Push parameters:")
    logger.info("Host Name="+WShostName)
```

```
logger.info("Protocol Port="+WShostport)
logger.info("URI="+WSuri)
URL = "http://"+WShostName+":"+WShostport+"/"+WSuri
logger.info("URL="+URL)
if testConnection == 'true':
    # locate the service
    test_receiver = SendDataToReceiver()
    locator = test_receiver.locateService(URL)
    #locator = locateService(URL)
    if(locator):
        logger.info(SCRIPT_NAME+":conexão de teste foi bem-sucedida")
        return
    else:
        raise Exception, errMsg
        return
# do same thing here if not just a test connection -
receiver = SendDataToReceiver()
locator = receiver.locateService(URL)
if(locator):
    logger.info(SCRIPT_NAME+":Connected to "+URL)
else:
    logger.error(SCRIPT_NAME+":no locator")
    raise Exception, errMsg
    return

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

```
# get referenced data - unused in this adapter implementation
#addRefResult = Framework.getTriggerCIData('referencedAddResult')
#updateRefResult = Framework.getTriggerCIData('referencedUpdateResult')
#deleteRefResult = Framework.getTriggerCIData('referencedDeleteResult')
# uncomment out the logger statements to see the data
empty = isEmpty(addResult, "addResult")
se não vazio:
    addResult = cleanUp(addResult)
    # send to ESB web service
    logger.info(SCRIPT_NAME+":sending addData Result")
    rcvr = SendDataToReceiver()
    resp = rcvr.SendData("add", URL, addResult)
    logger.info(SCRIPT_NAME+":addData received response:"+resp)
    #logger.debug(addResult)
empty = isEmpty(updateResult, "updateResult")
se não vazio:
    updateResult = cleanUp(updateResult)
    # send to ESB web service
    #logger.debug(updateResult)
    logger.info(SCRIPT_NAME+":sending updateData Result")
    rcvr = SendDataToReceiver()
    resp = rcvr.SendData("update", URL, updateResult)
    logger.info(SCRIPT_NAME+":received response:"+resp)


empty = isEmpty(deleteResult, "deleteResult")
se não vazio:
    deleteResult = cleanUp(deleteResult)
    # send to ESB web service
    #logger.debug(deleteResult)
    logger.info(SCRIPT_NAME+":sending deleteData Result")
    rcvr = SendDataToReceiver()
```

```
        resp = rcvr.SendData("delete", URL, deleteResult)
        logger.info(SCRIPT_NAME+":received response:"+resp)
    logger.info(SCRIPT_NAME+" ended")
```

Personalizando o Processamento de Mensagens de Resposta

O receptor de dados deve retornar uma cadeia contendo qualquer resposta ou status desejado. O adaptador push do serviço web transmite por por padrão a resposta ao log de nível de informações da sonda. A mensagem de resposta é um XML formatado por SOAP contendo a(s) cadeia(s) de resposta retornada dentro. Quaisquer dados podem ser retornados pelo receptor, como mensagens de êxito ou erro agrupadas ou individuais. Se o processamento adicional for desejado, a resposta pode ser processada pelo script Jython do adaptador. Nenhuma programação Java é necessária.

Um exemplo de uma mensagem de resposta de retorno, enviada usando o seguinte:

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


public String addData (String xmlAdd){
System.out.println(xmlAdd); // do something with the data
// send back a response message based on what you did
String tr = new String("a test response from addData!");
return tr;
}
```

é mostrado aqui:

```
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <addDataResponse xmlns="http://ucmdb.hp.com">
        <addDataReturn>a test response from addData!</addDataReturn>
    </addDataResponse>
</soapenv:Body>
```

Modificando o Receptor de Dados

Um cliente Java pode implementar as classes contidas em **UCMDBDataReceiver.jar** e chamar o serviço web da mesma maneira que o Jython. Além disso, os métodos não encapsulados também podem ser chamados. Um Javadoc existe para as classes **UCMDBDataReceiver.jar**. O código de origem abaixo mostra como usar esses métodos essenciais para encapsular os dados em uma mensagem SOAP e enviá-los ao receptor por HTTP.

O processo é criar um objeto **UCMDBDataReceiverServiceLocator** e, em seguida, atribuir **UCMDBDataReceiverEndPointAddress** à URL do receptor de dados.

Para enviar dados, o método **getUCMDBDataReceiver** do localizador é chamado para criar um objeto **UCMDBDataReceiver**. O objeto **UCMDBDataReceiver** implementa os métodos para realmente enviar os dados de adição/alteração/exclusão. Há três blocos de códigos idênticos para processar cada tipo de requisição.

O código de origem para a classe **SendDataToReceiver** está listado abaixo. Métodos e objetos realçados são os elementos essenciais a usar.

```
/**
* Test SendData for para o Receptor de Dados do UCMDB para o Adaptador Push de Serviço Web do UCMDB
*/
pacote com.hp.ucmdb;
import com.hp.ucmdb.SendDataToReceiver;
/**
* TestSendData pode ser usado para verificar se as classes SOAP estão funcionando.
* TestSendData cria uma classe SendDataToReceiver e chama seu método SendData.
* uma Cadeia de resposta é retornada.
* A URL de teste normalmente é anexada com "?wsdl" para obter o WSDL do serviço.
*/
public class TestSendData {
    /**
    * @param args - test SOAP message.
    *  optional arguments [0] a test string [1] a service endpoint URL of a Data Receiver.
    * the default URL is sent the incoming argument as a test message.
    * the default URL is "http://localhost:8080/UCMDBDataReceiver/services/UCMDBDataReceiver".
    * If any errors are encountered, TestClient will attempt to throw exceptions.
    */
    public static void main(String[] args) {
```

```
// use test message if supplied, otherwise supply a default test str
ing

String teststring = new String("Test SOAP message from UCMDBDataRece
iver TestSendData.");

if(args.length > 0) {

    teststring = args[0];

}

// use test URL if supplied, otherwise supply the default URL

String URL = new String("");

if(args.length > 1) {

    URL = args[1];

}

// return response

String response = new String("");

// perform the tests

try{

    if(URL.equals("")) {

        UCMDBDataReceiverServiceLocator locator = new UCMDBDataRece
        iverServiceLocator();

        UCMDBDataReceiver receiver = locator.getUCMDBDataReceiver()
        ;

        URL = locator.getUCMDBDataReceiverAddress();

        System.out.println("TestClient: tested URL="+locator.getUCM
        DBDataReceiverAddress());

        System.out.println("TestClient: receiver="+receiver.toStrin
        g());

    }

    SendDataToReceiver sdtr = new SendDataToReceiver();

    // this sends a test push and gets a response message

    response = sdtr.SendData("add", URL, args[0]);

    System.out.println("Response received was:"+response);

} catch(Exception e){

    System.out.println("TestClient: Remote Error:");
```

```
            e.printStackTrace();
        }
    }
}
```

O código de origem também é incluído no arquivo **UCMDBDataReceiver.jar** para as outras classes:

- TestClient.java
- UCMDBDataReceiver.java
- UCMDBDataReceiverProxy.java
- UCMDBDataReceiverService.java
- UCMDBDataReceiverServiceLocator.java
- UCMDBDataReceiverSoapBindingStub.java

A origem foi gerada no Eclipse IDE e então modificada. Tenha cuidado ao modificar o código do UCMDB, pois muito dele é gerado automaticamente para corresponder à especificação SOAP e ao receptor de dados do UCMDB.

Javadoc

Um **javadoc** totalmente comentado é fornecido com o adaptador push de serviço web genérico. O **javadoc** está incluído na pasta docs **javadoc**. Iniciar com **index.html**. A página de visão geral oferece acesso à documentação para todas as classes e métodos no SDK.

Todas as Classes

- **SendDataToReceiver:** API para wrapper de serviço Web
- **TestClient:** testar cliente para verificar conectividade a um ponto de extremidade de serviço
- **UCMDBDataReceiver:** wrapper de serviço Web

O restante é gerado automaticamente pelo criador do serviço Web

- UCMDBDataReceiverProxy
- UCMDBDataReceiverService
- UCMDBDataReceiverServiceLocator
- UCMDBDataReceiverSoapBindingStub

Visão geral

O uso básico do SDK, incluindo exemplos de código-fonte, é explicado na documentação do pacote. Esse **javadoc** é para o adaptador push de serviço web do UCMDB. A API pode ser chamada de Jython ou Java.

O SDK fornece duas amostras de origem, **TestClient** e **SendDataToReceiver**. **TestClient** fornece um teste muito limitado do cliente local que responde. **SendDataToReceiver** é a classe principal usada para enviar dados a um serviço web.

Primeiro, use esse SDK (principalmente o WSDL inserido) para implementar um receptor de dados UCMDB para se comunicar com esse serviço web. Em seguida, use esse SDK para criar um adaptador push no UCMDB para fazer o push dos dados de resultados de TQL do UCMDB ao receptor de dados. O uso básico dessa API é descrito abaixo, com implementações Jython e Java.

Implementando SendDataToReceiver()

SendDataToReceiver() encapsula todas as funções com um único método:

- Jython: `SendDataToReceiver("add",yourURL,"Hello!")`
- Java: `SendDataToReceiver("add",yourURL,"Hello!");`

Ou, crie um objeto **SendDataToReceiver** (por exemplo, para manipular outras configurações) e chame o método **SendData** separadamente, conforme mostrado aqui:

- Jython:

```
rcvr = SendDataToReceiver()
responseMsg = rcvr.SendData(“add”, yourURL, “Hello!”)
```

- Java:

```
SendDataToReceiver rcvr = new SendDataToReceiver();
String responseMsg = rcvr.SendData(“add”, yourURL, “Hello!”);
```

Ou, se você precisar fazer uma etapa de cada vez, pode fazer o seguinte:

1. Crie um novo objeto x **UCMDBDataReceiverServiceLocator()** e defina o endereço do ponto de extremidade do objeto depois, como mostrado aqui:
   - **Jython:**

```
x = UCMDBDataReceiverServiceLocator()
x.setUCMDBDataReceiverEndPointAddress(URL)
```

   - **Java:**

```
UCMDBDataReceiverServiceLocator x = new UCMDBDataReceiverServiceLocator();
x. setUCMDBDataReceiverEndPointAddress(URL);
```

2. Em seguida, crie um UCMDBDataReceiver com

- **Jython:** `y = x.getUCMDBDataReceiver()`
- **Java:** `UCMDBDataReceiver y = x.getUCMDBDataReceiver();`

3. Em seguida, envie os dados pelo serviço web SOAP assim:
   - **Jython:**
     - `y.addData(yourData)`
     - ou `y.updateData(yourData)`
     - ou `y.deleteData(yourData)`
   - **Java:**
     - `y.addData(yourData);`
     - ou `y.updateData(yourData);`
     - ou `y.deleteData(yourData);`
4. Pode ser necessário testar a conectividade e depois, se bem-sucedido, reutilize o mesmo objeto localizador para retornar **UCDMBDataReceiver** para usar para a transferência de dados.

As classes não contêm destrutores e não realizam gerenciamento de memória.

Referência do arquivo de mapeamento

Usando Mapeamentos

Um mapeamento deve ser criado para cada atributo de destino na saída XML transformada. Os mapeamentos especificam onde e como obter os dados. Se os dados estão em outro atributo correspondente no UCMDB, um mapeamento direto é usado.

Para extrair dados de vários atributou ou atributos do filho do EC do UCMDB ou atributos do EC pai, outros mapeamentos complexos podem ser necessários. O esquema de mapeamento abaixo mostra todos os mapeamentos possíveis.

O arquivo de mapeamento é um arquivo XML que define quais tipos de ECs/relacionamentos do UCMDB são mapeados para quais tipos de ECs/relacionamentos do armazenamento de dados de destino. O formato é explicado em detalhes abaixo. O arquivo de mapeamento controla quais ECs e tipos de relacionamentos recebem push, bem como controla exatamente quais atributos recebem push.

Uma entrada de mapeamento existe para cada atributo a receber push para o MDR de destino. Cada entrada de mapeamento pode consistir em um ou mais atributos nos dados push brutos do UCMDB. As entradas de mapeamento permitem um controle completamente granular da estrutura final e a nomenclatura dos dados a receber push para o MDR de destino.

Mapeamentos Diretos

Os mapeamentos transformam um modelo de dados para outro (nesse caso, o UCMDB para o MDR de destino de push). As transformações podem ser simples, no caso de um relacionamento 1:1 entre o atributo do UCMDB e o destino, eles diferem apenas por nome e talvez tipo.

A maioria dos mapeamentos de atributos são diretos. Por exemplo, o nome do servidor "ServerX", pode ser representado no UCMDB como um EC de tipo **unix** com um nome de atributo de **primary_server_name**, de tipo **string** com um tamanho de 50. O modelo de dados do MDR de destino pode especificar a mesma entidade lógica com um tipo de EC de **linux**, com um nome de atributo de **hostname** com um tipo de **char[]** com um tamanho máximo de 250. Mapeamentos Diretos podem atingir todos esses tipos mencionados acima de tarefas de tradução.

Veja um exemplo de um mapeamento direto:

```
 <target_attribute name="dns_domain" datatype="char">
<map type="direct" source_attribute="domain_name" />
</target_attribute>
```

Esse mapeamento direto mapeia o atributo do UCMDB **dns_domain** para o atributo **domain_name** no modelo de dados de destino.

Use o tipo de dado **char** seja qual for o tipo de dado real, a menos que seja necessário usar o tipo de dado real.

Mapeamentos Complexos

Mais mapeamentos complexos permitem transformações adicionais:

- Mapear valores de atributos a partir de vários ECs para um EC de destino.
- Mapear atributos de ECs filhos (os que têm um relacionamento **container_f** ou contido) para o EC pai no armazenamento de dados de destino. Por exemplo, definir um valor chamado **Número de CPUs** em um EC de host de destino. Outro exemplo poderia ser definir o valor **Memória Total** (adicionando valores de tamanho de memória de todos os ECs de memória de um EC host no UCMDB) em um EC de host de destino.
- Mapear atributos de ECs pais (os que têm um relacionamento **container_f** ou contido) para o EC no armazenamento de dados de destino. Por exemplo, definir um valor chamado **Servidor Contido** em um atributo de destino chamado EC de **Software Instalado**, obtendo o valor do host contêiner do EC de software no UCMDB.

Veja abaixo um exemplo de um mapeamento complexo, usando dois atributos de origem separados por um caractere de vírgula, para criar o atributo de destino **os**":

```
<target_attribute name="os" datatype="char">
    <map type="compoundstring">
        <source_attribute name="discovered_os_name" />
        <valor constante="," />
        <source_attribute name="host_osinstalltype" />
```

```
    </map>
</target_attribute>
```

Revertendo Direções de Vínculos

É possível que o UCMDB contenha dados diferentes em estrutura de origem para origem. Por exemplo, o relacionamento entre um EC IpAddress e um EC Interface pode ser um **pai**, como pode ocorrer com a integração do HP Network Node Manager. Ou pode ser um vínculo **containment** link como é comumente criado pelo Universal Discovery. Além disso, a direção desses vínculos são opostas entre si.

Atualmente não é possível reverter a direção de vínculos no arquivo de mapeamento. A reversão das variáveis **_end1** e **_end2** altera a ordem dos dados no XML transformado ou o vínculo está faltando nos dados de origem.

Definir uma regra de melhoria é uma solução possível para esse problema, que funciona como a seguir:

1. A parte de TQL da melhoria é um subconjunto de um TQL usado pelo adaptador push. Esse TQL em particular seleciona todos os links que estão na direção oposta do que é desejado no xml transformado.
2. A parte de melhoria define um novo link da direção correta e tipo desejado.
3. A melhoria é ativada e em seguida cria os links corretos.
4. O TQL de trabalho de integração refere-se agora ao vínculo melhorado e não ao vínculo original.
5. Os mapeamentos de <vínculos> no adaptador push então se referem ao vínculo melhorado também e produzem um conjunto de vínculos consistentes em tipo e direção.

Esquema do arquivo de mapeamento

Nome e caminho do elemento	Descrição	Atributos
integração	Define o conteúdo de mapeamento do arquivo. Deve ser o bloco mais periférico no arquivo, exceto pela linha de início e quaisquer comentários.	

Nome e caminho do elemento	Descrição	Atributos
info (integration)	Define informações sobre os repositórios de dados sendo integrados.	
source (integration > info)	Define informações sobre o repositório de dados de origem.	1. **Nome:** tipo **Descrição:** Nome do repositório de dados de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** versões **Descrição:** Versão(ões) do repositório de dados de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia 3. **Nome:** fornecedor **Descrição:** Fornecedor do repositório de dados de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia
target (integration > info)	Define informações sobre o repositório de dados de destino.	1. **Nome:** tipo **Descrição:** Nome do repositório de dados de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** versões **Descrição:** Versão(ões) do repositório de dados de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia 3. **Nome:** fornecedor **Descrição:** Fornecedor do repositório de dados de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia
targetcis (integration)	Elemento de contêiner de todos os mapeamentos de TEC.	

Nome e caminho do elemento	Descrição	Atributos
source_ci_type_tree (integration > targetcis)	Define um TEC de origem e todos os tipos de EC herdados dele.	1. **Nome:** nome **Descrição:** Nome do TEC de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** modo **Descrição:** Tipo da atualização necessária ao tipo de EC atual. **É necessário?:** Obrigatório **Tipo:** Uma das seguintes cadeias: a. **insert:** Use apenas se o EC ainda não existir. b. **update:** Use apenas se o EC já existir. c. **update_else_insert:** Se o EC existir, atualize; caso contrário, crie um novo EC. d. **ignore:** Não faça nada com esse tipo de EC.
source_ci_type (integration > targetcis)	Define um TEC de origem sem os tipos de EC herdados dele.	1. **Nome:** nome **Descrição:** Nome do TEC de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** modo **Descrição:** Tipo da atualização necessária ao tipo de EC atual. **É necessário?:** Obrigatório **Tipo:** Uma das seguintes cadeias: a. **insert:** Use apenas se o EC ainda não existir. b. **update:** Use apenas se o EC já existir. c. **update_else_insert:** Se o EC existir, atualize; caso contrário, crie um novo EC. d. **ignore:** Não faça nada com esse tipo de EC.

Nome e caminho do elemento	Descrição	Atributos
target_ci_type (integração > targetcis > source_ci_type -OU- integração > targetcis > source_ci_type_ tree)	Define um TEC de destino.	1. **Nome:** nome **Descrição:** Nome do tipo de EC de destino. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** esquema **Descrição:** Nome do esquema que será usado para armazenar esse tipo de EC no destino. **É necessário?:**Não é obrigatório **Tipo:** Cadeia 3. **Nome:** namespace **Descrição:** Indica o namespace desse tipo de EC no destino. **É necessário?:**Não é obrigatório **Tipo:** Cadeia
targetprimarykey (integration > targetcis > source_ci_type) -OU- (integration > targetcis > source_ci_type_tree -OU- (integration > targetrelations > link) -OU- (integration > targetrelations > source_link_type_tree)	Identifica atributos de chave primária do TEC de destino.	

Nome e caminho do elemento	Descrição	Atributos
pkey (integration > targetcis> source_ci_ type > targetprimarykey -OU- integration > targetcis > source_ci_type_tree > targetprimarykey -OU- (integration > targetrelations > link > targetprimarykey) -OU- integration > targetrelations > source_link_type_tree > targetprimarykey)	Identifica um atributo de chave primária. Obrigatório somente se o modo for **update** ou **insert_else_update**.	

Nome e caminho do elemento	Descrição	Atributos
target_attribute (integration > targetcis> source_ci_type -OU- integration > targetcis > source_ci_type_tree -OU- integration > targetrelations > link -OU- integration > targetrelations > source_link_type_tree)	Define o atributo do TEC de destino.	1. **Nome:** nome **Descrição:** Nome do atributo do TEC de destino. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** datatype **Descrição:** Tipo de dados do atributo do TEC de destino. **É necessário?:** Obrigatório **Tipo:** Cadeia 3. **Nome:** tamanho **Descrição:** Para tipos de dados cadeia/caractere, esse é o tamanho inteiro do atributo de destino. **É necessário?:**Não é obrigatório **Tipo.** Inteiro 4. **Nome.** option **Descrição.** A função de conversão a ser aplicada ao valor. **É obrigatório.** Falso **Tipo.** Uma das seguintes cadeias: a. **uppercase** – Converte para maiúscula b. **lowercase** – Converte para minúscula Se esse atributo estiver vazio, nenhuma função de conversão será aplicada.

Nome e caminho do elemento	Descrição	Atributos
map (integration > targetcis > source_ci_type > target_attribute -OU- integration > targetcis > source_ci_type_tree > target_attribute) -OU- (integration > targetrelations > link > target_attribute -OU- integration > targetrelations > source_link_type_tree > target_attribute)	Especifica como obter o valor do atributo de TECs de origem.	1. **Nome.** tipo **Descrição.** O tipo de mapeamento entre os valores de origem e destino. **É obrigatório.** Obrigatório **Tipo.** Uma das seguintes cadeias: a. **direct** – Especifica um mapeamento de 1 para 1 entre o valor do atributo de origem e o valor do atributo de destino. b. **compoundstring –** Subelementos são unidos em uma única cadeia e o valor do atributo de destino é definido. c. **childattr** – Subelementos são um ou mais atributos de um TEC filho. Um TEC filho é aquele com um relacionamento do tipo **composition** ou **containment**. d. **constant** – Cadeia estática 2. **Nome.** value **Descrição.** Cadeia constante para tipo=**constant** **É obrigatório.** Necessário apenas quando tipo=**constant** **Tipo.** Cadeia 3. **Nome.** attr **Descrição.** Nome de atributo de origem para tipo=**direct** **É obrigatório.** Necessário apenas quando tipo=**direct** **Tipo.** Cadeia

Nome e caminho do elemento	Descrição	Atributos
aggregation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetcis > source_ci_type_tree > target_attribute > map -OU- (integration > targetrelations > link > target_attribute > map -OU- integration > targetrelations > source_link_type_tree > 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:** tipo **Descrição.** O tipo da função de agregação **É necessário?:** 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 a soma de todos os valores numéricos 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.

Nome e caminho do elemento	Descrição	Atributos
source_child_ci_type (integration > targetcis> source_ci_ type > target_attribute > map -OU- integration > targetcis > source_ci_type_tree > target_attribute > map -OU- (integration > targetrelations > link > target_attribute > map -OU- integration > targetrelations > source_link_type_tree > target_attribute > map) Válido apenas quando o tipo do mapa é **childattr**.	Especifica a partir de qual EC conectado o atributo filho é retirado.	1. **Nome.** name **Descrição.** O tipo do EC filho **É obrigatório.** Obrigatório **Tipo.** Cadeia 2. **Name.** source_attribute **Descrição.** O atributo do EC filho mapeado. **É obrigatório.** Necessário somente se o tipo de agregação **childAttr** (que está no mesmo caminho) não for =count. **Tipo.** Cadeia

Nome e caminho do elemento	Descrição	Atributos
validation (integration > targetcis > source_ci_type > target_attribute > map -OU- integration > targetcis > source_ci_type_tree > target_attribute > map -OU- (integration > targetrelations > link > target_attribute > map -OU- integration > targetrelations > source_link_type_tree > 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.	1. **Nome.** minlength **Descrição.** Exclui cadeias mais curtas do que o valor determinado. **É necessário?:** Não é obrigatório **Tipo.** Inteiro 2. **Nome.** maxlength **Descrição.** Exclui cadeias mais longas do que o valor determinado. **É necessário?:** Não é obrigatório **Tipo.** Inteiro 3. **Nome.** minvalue **Descrição.** Exclui números menores do que o valor determinado. **É necessário?:** Não é obrigatório **Tipo.** Numérico 4. **Nome.** maxvalue **Descrição.** Exclui números maiores do que o valor determinado. **É necessário?:** Não é obrigatório **Tipo.** Numérico
targetrelations (integration)	Elemento de contêiner de todos os mapeamentos de relacionamento. Opcional.	

Nome e caminho do elemento	Descrição	Atributos
source_link_type_tree (integration > targetrelations)	Faz o mapeamento entre um tipo de relacionamento de origem sem os tipos herdados dele para um relacionamento de destino. Obrigatório apenas se **targetrelation** estiver presente.	1. **Nome:** nome **Descrição.** Nome do relacionamento de origem. **É necessário?:**Necessário **Tipo.** Cadeia 2. **Nome:** target_link_type **Descrição.** Nome do relacionamento de destino **É necessário?:** Obrigatório **Tipo.** Cadeia 3. **Nome:** nameSpace **Descrição:** O namespace do vínculo que será criado com o destino. **É necessário?:** Não é obrigatório **Tipo:** Cadeia 4. **Nome:** modo **Descrição:** Tipo da atualização necessária para o vínculo atual. **É necessário?:** 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. 5. **Nome:** source_ci_type_end1 **Descrição:** Tipo de EC End1 do relacionamento de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia 6. **Nome:** source_ci_type_end2 **Descrição:** Tipo de EC End2 do relacionamento de origem. **É necessário?:** Obrigatório

Nome e caminho do elemento	Descrição	Atributos
		Tipo: Cadeia

Nome e caminho do elemento	Descrição	Atributos
link (integration > targetrelations)	Faz o mapeamento entre um relacionamento de origem e um relacionamento de destino. Obrigatório apenas se **targetrelation** estiver presente.	1. **Nome:** source_link_type **Descrição:** Nome do relacionamento de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** target_link_type **Descrição:** Nome do relacionamento de destino. **É necessário?:** Obrigatório **Tipo:** Cadeia 3. **Nome:** nameSpace **Descrição:** O namespace do vínculo que será criado com o destino. **É necessário?:** Não é obrigatório **Tipo:** Cadeia 4. **Nome:** modo **Descrição:** Tipo da atualização necessária para o vínculo atual. **É necessário?:** 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. 5. **Nome:** source_ci_type_end1 **Descrição:** Tipo de EC End1 do relacionamento de origem **É necessário?:** Obrigatório **Tipo:** Cadeia 6. **Nome:** source_ci_type_end2 **Descrição:** Tipo de EC End2 do relacionamento de origem. **É necessário?:** Obrigatório

Nome e caminho do elemento	Descrição	Atributos
		Tipo: Cadeia
target_ci_type_end1 (integration > targetrelations > link -OU- integration > targetrelations > source_link_type_tree)	Tipo de EC End1 do relacionamento de destino.	1. **Nome:** nome **Descrição:** Nome do tipo de EC de destino. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** superclass **Descrição:** Nome da superclasse do tipo de EC End1. **É necessário?:** Não é obrigatório **Tipo:** Cadeia
target_ci_type_end2 (integration > targetrelations > link -OU- integration > targetrelations > source_link_type_tree)	Tipo de EC End1 do relacionamento Tipo de EC End2.	1. **Nome:** nome **Descrição:** Nome do tipo de EC End2 do relacionamento de destino. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** superclass **Descrição:** Nome da superclasse do tipo de EC End2. **É necessário?:** Não é obrigatório **Tipo:** Cadeia

Esquema de resultados de mapeamento

Nome e caminho do elemento	Descrição	Atributos
root	A raiz do documento de resultado.	
dados (raiz)	A raiz do próprio dado.	
objects (root > data)	O elemento raiz dos objetos a atualizar.	

Nome e caminho do elemento	Descrição	Atributos
Object (root > data > objects)	Descreve a operação de atualização para um único objeto e todos os seus atributos.	1. **Nome:** nome **Descrição:** O nome do tipo de EC **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** modo **Descrição:** Tipo da atualização necessária ao tipo de EC atual. **É necessário?:** Obrigatório **Tipo:** Uma das seguintes cadeias: a. **insert** – Use apenas se o EC ainda não existir. b. **update** – Use apenas se o EC já existir. c. **update_else_insert** – Se o EC existir, atualize; caso contrário, crie um novo EC. d. **ignore** – Não faça nada com esse tipo de EC. 3. **Nome:** operação **Descrição:** A operação a ser realizada com esse EC. **É necessário:** Obrigatório **Tipo:** Uma das seguintes cadeias: a. **add** – O EC deve ser adicionado b. **update** – O EC deve ser atualizado c. **delete** – O EC deve ser excluído Se nenhum valor estiver definido, o valor padrão de **add** será usado. 4. **Nome:** mamId **Descrição:** O ID do objeto no CMDB de origem. **É necessário?:** Obrigatório **Tipo:** Cadeia

Nome e caminho do elemento	Descrição	Atributos
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 contém um link, o valor é a ID de uma das extremidades. Cada ID de extremidade aparece como um objeto (em <objetos>).	1. **Nome:** nome **Descrição:** O nome do campo. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** chave **Descrição:** Especifica se esse campo é uma uma chave do objeto. **É necessário?:** Obrigatório **Tipo:** Booliano 3. **Nome:** datatype **Descrição:** O tipo do campo. **É necessário?:** Obrigatório **Tipo:** Cadeia 4. **Nome:** tamanho **Descrição:** Para tipos de dados cadeia/caractere, esse é o tamanho do inteiro do atributo de destino. **É necessário?:**Não é obrigatório **Tipo:** Inteiro

Nome e caminho do elemento	Descrição	Atributos
links (root > data)	O elemento raiz dos links para atualizar.	1. **Nome:** targetRelationshipClass **Descrição:** O nome do relacionamento (vínculo) no sistema de destino. **É necessário?:** Obrigatório **Tipo:** Cadeia 2. **Nome:** targetParent **Descrição:** O tipo da primeira extremidade do vínculo (pai). **É necessário?:** Obrigatório **Tipo:** Cadeia 3. **Nome:** targetChild **Descrição:** O tipo da segunda extremidade do vínculo (filho). **É necessário?:** Obrigatório **Tipo:** Cadeia 4. **Nome:** modo **Descrição:** Tipo da atualização necessária ao tipo de EC atual. **É necessário?:** Obrigatório **Tipo:** Uma das seguintes cadeias: a. **insert** – Use apenas se o EC ainda não existir. b. **update** – Use apenas se o EC já existir. c. **update_else_insert** – Se o EC existir, atualize; caso contrário, crie um novo EC. d. **ignore** – Não faça nada com esse tipo de EC. 5. **Nome:** operação **Descrição:** A operação a ser realizada com esse EC. **É necessário?:** Obrigatório **Tipo:** Uma das seguintes cadeias: a. add – O EC deve ser adicionado b. update – O EC deve ser atualizado c. delete – O EC deve ser excluído Se nenhum valor estiver definido, o valor padrão de **add** será usado. 6. **Nome:** mamId **Descrição:** O ID do objeto no CMDB de

Nome e caminho do elemento	Descrição	Atributos
		origem. **É necessário?:** Obrigatório **Tipo:** Cadeia

Personalização

Esta seção explica alguns procedimentos básicos para tipos comuns de personalização para adaptadores push.

Adicionando um Atributo

1. Verifique se o atributo está incluído no resultado do TQL.
2. Adicione o mapeamento de atributo ao arquivo de mapeamentos na seção de mapeamento de EC correta.
3. Verifique se o receptor de dados está preparado para receber o atributo adicional nos dados.

Removendo um Atributo

Para remover um atributo, remova-o do arquivo de mapeamento. Você também deve remover o atributo do TQL se ele não for mais usado no resultado ou como um nó condicional.

Adicionando um Tipo de EC

1. Adicionar o tipo de EC à TQL.
2. Verifique se o tipo de EC e seus dados de atributo aparecem no resultado de TQL (usar calcular e visualizar).
3. Adicione o mapeamento de tipo de EC no arquivo de mapeamento. Copiar outros mapeamentos de tipo de EC para criar rapidamente um novo tipo de EC.
4. Modifique o nome XML copiado e os mapeamentos de atributos para corresponder ao novo tipo de EC e seus atributos. Consulte "Referência do arquivo de mapeamento" na página 223 para os tipos disponíveis de mapeamentos.

Removendo um Tipo de EC

1. Remover o Tipo de EC da TQL
2. Remova a seção de mapeamento para esse tipo de EC no arquivo de mapeamento.

Adicionando links

1. Verifique se os dois ECs finais estão presentes nos dados.
2. Verifique se o vínculo que você precisa adicionar é de fato um vínculo válido (verifique no gerenciador de tipo de EC).
3. Adicione os elementos do vínculo na seção de relacionamentos do xml de mapeamento.

Removendo Vínculos

1. Remova a seção de vínculos do vínculo que deseja remover do arquivo de mapeamentos.
2. Se possível, remova o vínculo do TQL (a menos que ele afete a eficiência ou função do TQL).

Capítulo 7: Desenvolvendo adaptadores push genéricos aprimorados

Este capítulo inclui:

Visão geral 244

O arquivo de mapeamento 244

O viajante do Groovy 247

Escrever scripts Groovy 250

Implementar interface PushConnector 251

Criar um pacote de adaptador 252

Esquema do arquivo de mapeamento 253

Visão geral

Um adaptador push aprimorado é uma estrutura de dados que representa o resultado da consulta TQL. Cada adaptador criado com base no adaptador push aprimorado tratará dessa estrutura de dados e a aplicará ao seu destino obrigatório.

A estrutura de dados é chamada **ResultTreeNode (RTN)**. O RTN é criado de acordo com o arquivo de mapeamento do adaptador e os resultados da consulta TQL. As consultas usadas para o adaptador push aprimorado devem ser baseadas na raiz, isto é, a consulta deve conter um nó de consulta com o nome do elemento **root** ou um ou mais elementos de relacionamento que começam com o prefixo **root**. Esse EC ou relacionamento serve como o elemento raiz da consulta. Para obter detalhes, consulte Data Push no *Guia de Gerenciamento de Fluxo de Dados do HP Universal CMDB*.

Há duas etapas básicas envolvidas no desenvolvimento dos adaptadores push aprimorados:

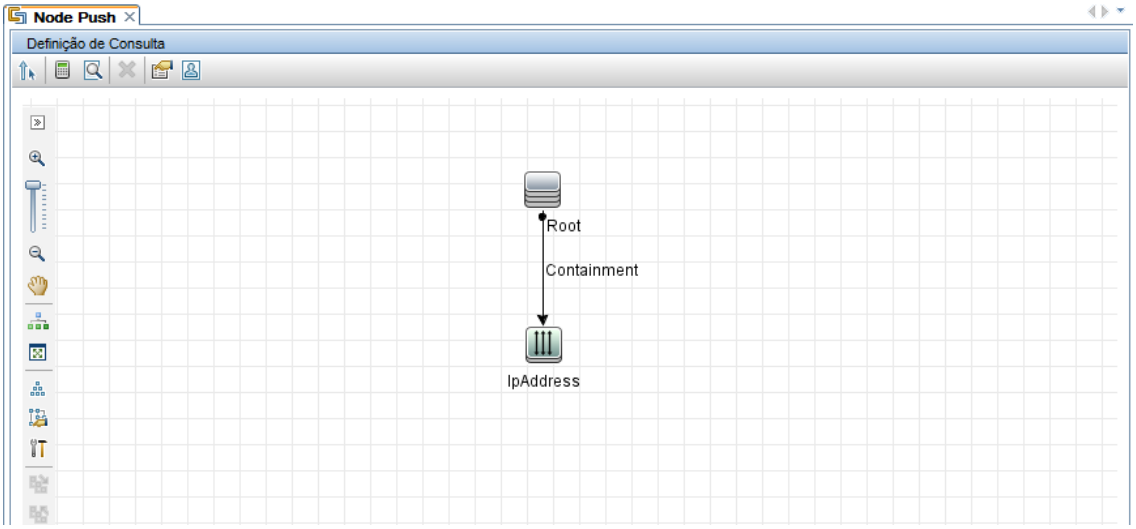
1. Implementar a interface PushConnector – essa interface recebe os dados a adicionar, atualizar e excluir como uma lista de RTNs e executa o push no destino.

2. Criar o arquivo de mapeamento – o arquivo de mapeamento determina a criação da estrutura do RTN, mapeando ECs e atributos do resultado da TQL.

O arquivo de mapeamento

O exemplo a seguir demonstra como criar o arquivo de mapeamento.

Nesse exemplo, simularemos um push de um nó e um endereço IP. Criaremos uma consulta TQL chamada: **Node Push**, como a seguir:

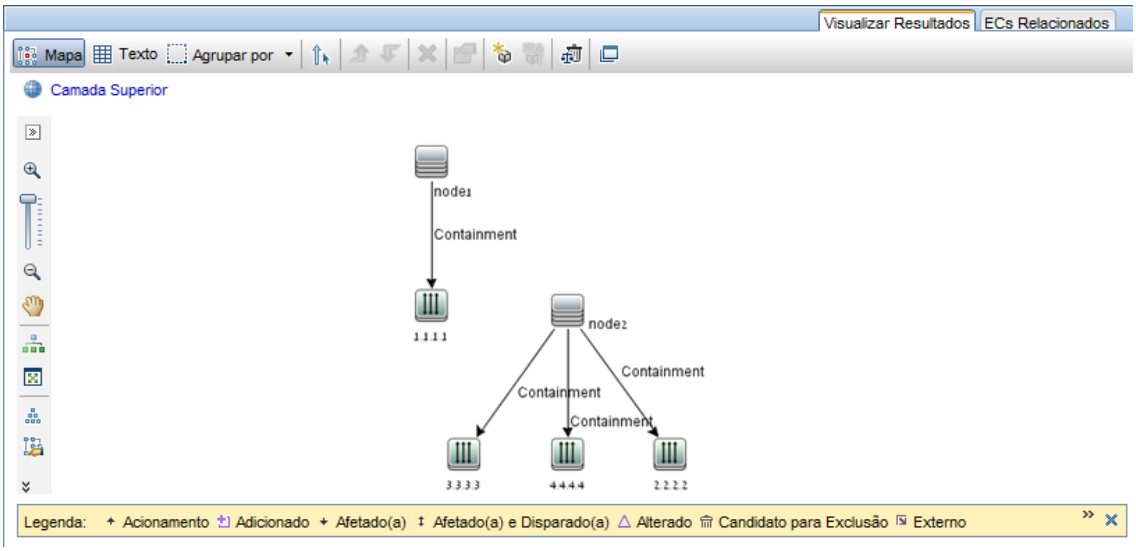


No arquivo de mapeamento, criamos dois tipos de EC de destino: **Computer** e **IP**. O computador tem uma variável e dois atributos. O IP tem um atributo.

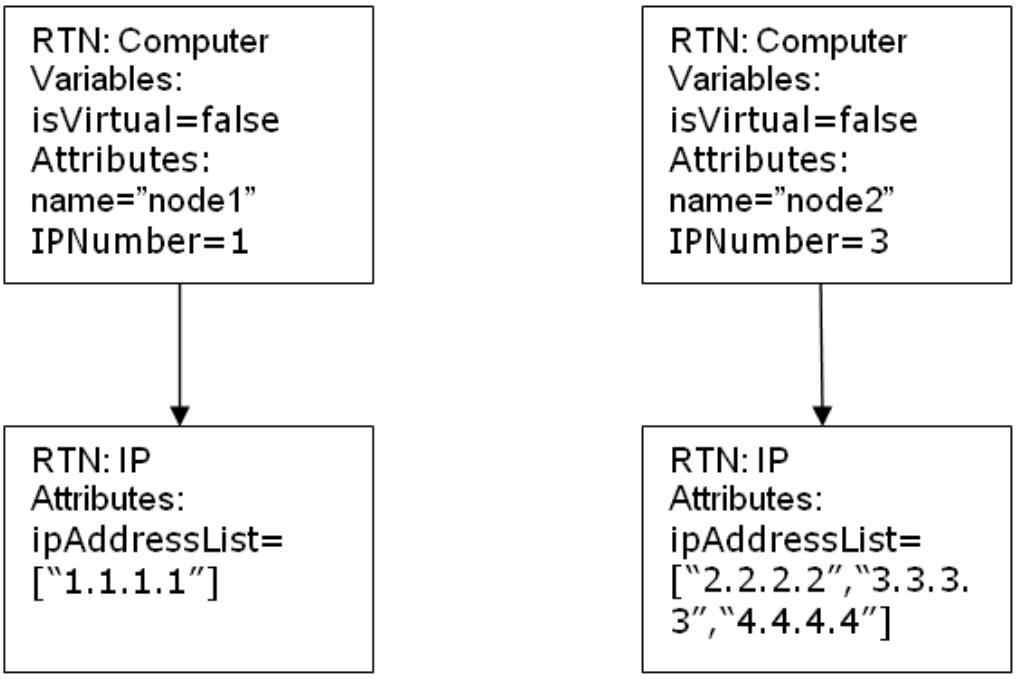
Veja a seguir o arquivo XML de mapeamento:

```
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://schemas.hp.com/ucmdb/1/pushAdap
ter">
        <info>
            <source name="UCMDB" versions="10.0" vendor="HP" />
            <target name="PushProduct" versions="9.3" vendor="HP" />
        </info>
        <import>
            <!—imports the Groovy script file. -->
            <scriptFile path="mappings.scripts.PushFunctions" />
        </import>
<targetcis>
<source_instance_type query-name="Node Push" root-element-name="Root">
 <target_ci_type name="Computer" is-
    valid="(Root['root_iscandidatefordeletion']==null) ? true :
    !Root['root_iscandidatefordeletion']">
  <variable name="isVirtual" datatype="BOOLEAN"
    value="PushFunctions.isVirtual(Root['root_class']) " />

  <target_mapping name="name" datatype="String" value="Root[`name’] " />
  <target_mapping name="ipNumber" datatype="INTEGER"
    value="Root.IpAddress.size()" />
  <target_mapping name="description" datatype="STRING" value=" PushFunctions
    .getDescription(isVirtual)" />
  <target_ci_type name="IP">
    <target_mapping name="ipAddressList" datatype="STRING_LIST"
    value="Root.IpAddress*.getAt(`name’) " />
</target_ci_type>
</target_ci_type>
</targetcis>
</integration>
```

Os resultados da consulta aparecem da seguinte forma:



A lista do RTN que será criada de acordo com esse arquivo de mapeamento será parecida com:



Cada instância raiz é mapeada separadamente usando o arquivo de mapeamento. Portanto, nesse exemplo, o PushConnector recebe uma lista de duas raízes RTN.

Observação: O adaptador push anterior tinha a capacidade de criar um mapeamento geral para um tipo de EC. O novo mapeamento de adaptador push é por consulta TQL. Enquanto estiver executando um trabalho de push que usa uma consulta chamada **x**, o adaptador procura o arquivo de mapeamento relevante (o que tem o atributo: query-name=x).

Você pode calcular os valores no arquivo de mapeamento usando a linguagem de script groovy. Para obter detalhes, consulte "O viajante do Groovy" abaixo.

O viajante do Groovy

Acessar os resultados da consulta TQL da seguinte maneira:

- **Root[attr]** retorna o atributo **attr** do elemento raiz.
- **Root.Query_Element_Name** retorna uma lista das instâncias do EC nomeadas na TQL Query_Element_Name que estão vinculadas ao EC raiz atual.
- **Root.Query_Element_Name[2][attr]** retorna o atributo **attr** do terceiro Query_Element_Name vinculado ao EC raiz atual.
- **Root.Query_Element_Name*.getAt(attr)** retorna uma lista dos atributos **attr** das instâncias do EC chamadas Query_Element_Name na TQL que estão vinculadas ao EC raiz atual.

Esses são atributos adicionais que podem ser acessados pelo viajante do groovy:

- **cmdb_id** – retorna a ID do UCMDB do EC ou relacionamento como uma cadeia.
- **external_cmdb_id** – retorna a ID externa do EC ou relacionamento como uma cadeia.
- **Element_type** – retorna o tipo de elemento do EC ou relacionamento como uma cadeia.

A cadeia de importação:

```
<import>
<scriptFile path="mappings.scripts.PushFunctions"/>
</import>
```

Isso significa que estamos declarando uma importação para todos os scripts do groovy no arquivo de mapeamento. Nesse exemplo, **PushFunctions** é um arquivo de script do groovy que contém algumas funções estáticas, e podemos acessá-lo durante o mapeamento (isto é, value=" PushFunctions.foo()")

source_instance_type

O mapeamento é feito por TQL, o valor do nome da consulta é a TQL relacionada do mapeamento atual. O '*' significa que esse arquivo de mapeamento é associado a todas as consultas TQL que começam com o prefixo: **Node Push**.

```
<source_instance_type query-name="Node Push*" root-element-name="Root">
```

A tag source_instance_type designa o elemento raiz que estamos mapeando.

root-element-name deve ser exatamente igual ao nome da raiz na TQL.

target_ci_type

Essa tag é usada para a criação do RTN.

O atributo de nome representa o nome target_ci_type: name=Computer

O atributo **is-valid** é um valor booliano calculado durante o mapeamento e determina se o target_ci atual é válido. target_ci_types inválidos não são adicionados ao RTN. Neste exemplo, não queremos criar uma instância target_ci_type para a qual o atributo **root_iscandidatefordeletion** no UCMDB é verdadeiro.

O target_ci_type pode ter variáveis calculadas durante o mapeamento:

```
<variable name="vSerialNo" datatype="STRING" value="Root['serial_number']"/>
```

A variável **vSerialNo** obtém o valor do **serial_number** da raiz atual.

O atributo do RTN é criado pela tag **target_mapping**. O resultado da execução do script do groovy no campo **value** é atribuído ao atributo do RTN.

```
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
```

SerialNo atribui o valor da variável **vSerialNo**.

É possível definir um target_ci_type como filho de outro target_ci_type, como indicado a seguir:

```
<target_ci_type name="Portfolio">
<variable name="vSerialNo" datatype="STRING" value="Root['global_id']"/>
<target_mapping name="CMDBId" datatype="STRING" value="globalId"/>
<target_ci_type name=Asset">
<target_mapping name="SerialNo" datatype="STRING" value="vSerialNo"/>
</target_ci_type>
</target_ci_type>
```

O **Portfólio** do RTN terá um RTN filho chamado **Asset**.

for-each-source-ci

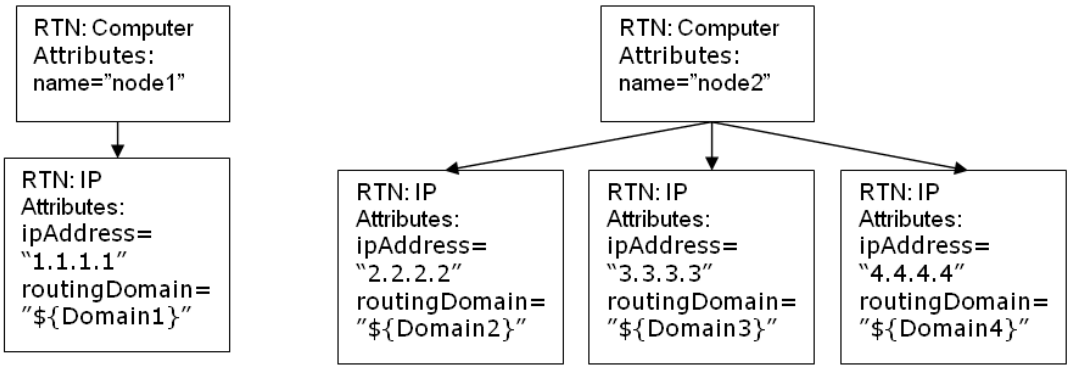
Essa tag lista os ECs específicos da instância raiz. Ela tem os seguintes campos:

- source-cis=" " – a lista de ECs para as quais um EC de destino é criada. Esta lista é definida pelo viajante do Groovy no campo **Root.IpAddress**.
- count-index=" " – uma variável que mantém o índice do EC na iteração atual do loop.
- var-name=" " – o nome do EC na iteração atual do loop.

Vamos modificar nosso arquivo de mapeamento de exemplo:

```
<target_ci_type name="Computer">
  <target_mapping name="name" datatype="String" value="Root[`name'] " />
   <for-each-source-ci count-index="i" var-name="currIP" source-cis="Root.IpAddress" >
     <target_ci_type name="IP">
         <target_mapping name="ipAddress" datatype="STRING"
     value="Root.IpAddress[i][`name']"/>
         <target_mapping name="routingDomain" datatype="STRING"
     value="currIP['routing_domain']"/>
  </target_ci_type>
 </for-each-source-ci>
 </target_ci_type>
```

A lista do RTN que será criada de acordo com esse arquivo de mapeamento será parecida com:



dynamic_mapping

Essa tag adiciona a capacidade de criar um mapeamento de dados a partir do armazenamento de dados de destino durante a criação da estrutura do RTN.

Exemplo: Suponha que o destino seja um banco de dados com uma tabela chamada **Computador** com uma coluna **id** e uma coluna **nome** correlacionada a **Node.name** no UCMDB. As duas colunas são únicas. O banco de dados também tem uma tabela chamada **IP** que tem uma chave com referência a **parentID** na tabela Computador. O 'dynamic_mapping' pode criar um mapa que armazena o nome e a id como <nome,id>. Com base nesse mapa, o adaptador pode corresponder ids com computadores e aplicar o valor correto ao atributo parentID na tabela IP. Você pode usar esse mapa para atribuir um valor ao atributo parentID ao criar o RTN.

O mapeamento é determinado pelo **map_property**. O dynamic_mapping é executado uma vez para cada bloco.

```
<dynamic_mapping name="IdByName " keys-unique="true">
```

O atributo **name** representa o nome do mapa. O atributo **keys-unique** indica se as chaves são únicas (cada chave é mapeada para um valor ou para um conjunto de valores).

O nome do mapa neste exemplo é **IdByName** e tem chaves exclusivas. Para acessar o mapa no script, execute o seguinte comando:

```
DynamicMapHolder.getMap('IdByName')
```

Ele retorna uma referência para aquele mapa.

A marcação **map_property** cria a propriedade na qual o mapeamento se baseia.

Exemplo:

```
<map_property property-name="SQLQuery" datatype="STRING"
property-value="SELECT name, id FROM Computer"/>
```

Nesse exemplo, o nome da propriedade é **SQLQuery** e seu valor é uma instrução SQL que cria o mapa. A implementação dos métodos **retrieveUniqueMapping** e **retrieveNonUniqueMapping** para a interface PushConnector determinará o conteúdo real do mapa retornado.

Variáveis globais

As seguintes variáveis globais são acessíveis para o script groovy no arquivo de mapeamento:

- **Topology** – Tipo: Topology. Uma instância da topologia do bloco atual.
- **QueryDefinition** - Tipo: QueryDefinition. Uma instância da definição de consulta da TQL atual.
- **OutputCI** – Tipo: ResultTreeNode. O RTN do elemento raiz no mapeamento de árvore atual.
- **ClassModel** – Tipo: ClassModel. Uma instância do modelo de classe.
- **CustomerInformation** – Tipo: CustomerInformation. Informações sobre o cliente que executa o trabalho.
- **Logger** – Tipo: DataAdapterLogger. Esse logger está disponível no adaptador para escrever logs na estrutura de logs do UCMDB.

Escrever scripts Groovy

Nesta seção, criamos o arquivo **PushFunctions.groovy**. Este arquivo conterá funções estáticas usadas durante o mapeamento da instância raiz.

```
package mappings.scripts

public class PushFunctions {

	public static boolean isVirtual(def nodeRole){
		return isListContainsOne(def list, "MY_VM", "MY_SIMULATOR");
	}
```

```
public static String getDescription(boolean isVirtual){
               if(isVirtual){
                       return "This is a VM";
               }
               else{
                       return "This is physical machine";
               }
       }



private static boolean isListContainsOne(def list, ...stringList){
               //returns true if the list contains one of the values.
       }
}
```

Implementar interface PushConnector

A implementação deve suportar as seguintes etapas básicas:

```
public class PushExampleAdapter implements PushAdapterConnector

{

public UpdateResult pushTreeNodes(PushConnectorInput input) lança DataAccess
Exception{

// 1. build an UpdateResult instance - the UpdateResult is used to return ma
ppings between the sent ids to the actual ids that entered the data store.
// Também tem um status de atualização que permite passar o status dos dados
que realmente tiveram push, relatórios de status detalhados sobre IDs com fa
lhas e ações realizadas realmente em ids bem-sucedidas.
// 2. handle the data:
// a. handle data to add. Can be retrieved by: input.getResultTreeNodes.getD
ataToAdd();
//  b. handle data to update.
//  c. handle data to delete.
// 3. Return the Update result.
       }

public void start(PushDataAdapterEnvironment env) throws DataAccessException{
       // this method is called when the integration point created,
or when the adapter is reloaded
       //(i.e after changing one of the mapping files
       // and pressing ‘save’).

}
```

```
public void testConnection(PushDataAdapterEnvironment env) throws DataAccess
Exception {
               // this method is called when pressing the 'test
connection' button in the
               //creation of the integration point.
               // For example if we push data to RDBMS this method
can create a connection
               //to the database and will run a dummy SQL statement.
               // If it fails it writes an error message to the log
and throws an exception.
       }



Map<Object, Object> retrieveUniqueMapping(MappingQuery mappingQuery){
//This method will create the map according to the given mappingQuery. It wi
ll be called in the
// mapping stage of the adapter execution, before the ‘UpdateResult pushTree
Nodes’ method.
// This method is called when the ‘keys-unique’ attribute of the ‘dynamic_ma
pping’ tag is true.
}
Map<Object, Set<Object>> retrieveNonUniqueMapping(MappingQuery mappingQuery){
 // This method is called when the ‘keys-unique’ attribute of the ‘dynamic_m
apping’ tag is false.
//  In this case a key can be mapped to several values.
}
}
```

Criar um pacote de adaptador

Verifique se o pacote do adaptador contém as seguintes pastas:

- **adapterCode.** Nessa pasta, crie uma pasta chamada **PushExampleAdapter**, que conterá o arquivo .jar que criamos a partir de PushExampleAdapter.java. Ela também conterá uma pasta chamada **mappings**, onde você poderá colocar o arquivo de mapeamento criado anteriormente, **computerIPMapping.xml**. Ela também deve conter outra pasta chamada **scripts** que contém o arquivo **PushFunctions.groovy**.
- **discoveryConfigFiles.** Conter arquivos de configuração como os códigos de erro usados ao reportar um erro usando UpdateResult. Neste exemplo, a pasta está vazia.
- **discoveryPatterns.** Conter **push_example_adapter.xml**.

- **tql.** Conter a consulta TQL criada para o exemplo. Essa pasta é opcional, mas quando o pacote é implantado, o TQL é criado automaticamente.

Esquema do arquivo de mapeamento

Nome e caminho do elemento	Descrição	Atributos
Integração	Define o mapeamento conteúdo do arquivo. Deve ser o bloco mais periférico no arquivo, exceto pela linha de início e quaisquer comentários.	
info (integração)	Define informações sobre os repositórios de dados que estão sendo integrados.	
source (info)	O produto de origem	**Nome:** nome **Descrição:** o nome do produto **É necessário?:** necessário **Tipo:** Cadeia
		Nome: fornecedor **Descrição:** o fornecedor do produto **É necessário?:** necessário **Tipo:** Cadeia
		Nome: versões **Descrição:** a versão do produto **É necessário?:** necessário **Tipo:** decimal

Nome e caminho do elemento	Descrição	Atributos
target (info)	O produto de destino	**Nome:** nome **Descrição:** o nome do produto **É necessário?:** necessário **Tipo:** Cadeia
		Nome: fornecedor **Descrição:** o fornecedor do produto **É necessário?:** necessário **Tipo:** Cadeia
		Nome: versões **Descrição:** a versão do produto **É necessário?:** necessário **Tipo:** decimal
Importar (integração)	Um elemento de contêiner para arquivos de script importados.	
scriptFile (integração>importação)	Define o arquivo de script groovy a ser importado.	**Nome:** path **Descrição:** o caminho do arquivo de script **É necessário?:** necessário **Tipo:** cadeia
Targetcis (integração)	Um elemento de contêiner para tipos de EC de Destino.	

Nome e caminho do elemento	Descrição	Atributos
Source_instance_type (integration > targetcis)	Define o tipo de instância do EC de origem. Deve ser o elemento raiz conforme definido no TQL.	**Nome:** query-name. **Descrição:** O nome do TQL relevante **É necessário?:** necessário **Tipo:** Cadeia
		Nome: nome **Descrição:** O elemento raiz conforme definido no TQL. **É necessário?:** necessário **Tipo:** Cadeia
dynamic_mapping (integração > targetcis > source_instance_type)	Define um mapa criado uma vez por bloco.	**Nome:** nome **Descrição:** o nome do mapa **É necessário?:** necessário **Tipo:** Cadeia
		Nome: keys-unique **Descrição:** Declara se as chaves são exclusivas ou não. **É necessário?:** necessário **Tipo:** booliano

Nome e caminho do elemento	Descrição	Atributos
target_ci_type (integração > targetcis > source_instance_type) -OU- (integração > targetcis > source_instance_type> for-each-source-ci)	Define o tipo de EC de destino a ser adicionado ao RTN.	**Nome:** nome **Descrição:** o nome do tipo de EC de destino **É necessário?:** necessário **Tipo:** Cadeia
		Nome: is-valid **Descrição:** verifica se o EC de destino atual é válido de acordo com o script fornecido. **É necessário?:** não é necessário **Tipo:** Cadeia (script groovy)

Nome e caminho do elemento	Descrição	Atributos
Variável (target_ci_type)	Define uma variável para o tipo de EC de destino.	**Nome:** nome **Descrição:** o nome da variável **É necessário?:** necessário **Tipo:** Cadeia
		Nome: datatype **Descrição:** o tipo de dado da variável. **É necessário?:** necessário **Tipo:** type-enum (pode ser um dos seguintes: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Nome: valor **Descrição:** O valor para atribuir à variável **É necessário?:** necessário **Tipo:** script groovy

Nome e caminho do elemento	Descrição	Atributos
target_mapping (target_ci_type)	Define um atributo para o tipo de EC de destino.	**Nome:** nome **Descrição:** o nome do atributo **É necessário?:** necessário **Tipo:** Cadeia
		Nome: datatype **Descrição:** o tipo de dado da variável. **É necessário?:** necessário **Tipo:** type-enum (pode ser um dos seguintes: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Nome: valor **Descrição:** O valor para atribuir à variável **É necessário?:** necessário **Tipo:** script groovy
		Nome: ignore-on-null **Descrição:** Se o valor de mapeamento de destino for nulo e esse atributo for TRUE, ignore o atributo **É necessário?:** não é necessário **Tipo:** Booliano (script groovy)

Nome e caminho do elemento	Descrição	Atributos
before-mapping (target_ci_type)	Um script groovy que é executado antes do mapeamento do tipo de EC de destino.	
after-mapping (target_ci_type)	Um script groovy que é executado após o mapeamento do tipo de EC de destino.	
for-each-source-ci (target_ci_type)	Define uma iteração de ECs específicos da instância raiz.	**Nome:** count-index **Descrição:** o índice do EC iterado atual **É necessário?:** necessário **Tipo:** Cadeia
		Nome: var-name **Descrição:** a variável que se refere ao EC iterado atual. **É necessário?:** não é necessário **Tipo:** Cadeia
		Nome: source-cis **Descrição:** O nome do EC na consulta que deve ser iterado **É necessário?:** necessário **Tipo:** Cadeia (script groovy)

Usando APIs

Capítulo 8: Introdução a APIs

Este capítulo inclui:

Visão geral sobre APIs 261

Visão geral sobre APIs

As seguintes APIs estão incluídas com o HP Universal CMDB:

- **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 "API do HP Universal CMDB" na página 262.
- **API de Serviço Web do UCMDB.** Permite a criação de definições de itens de configuração e seus relacionamentos topológicos com o UCMDB, além da pesquisa de informações usando consultas TQL e ad hoc. Para obter detalhes, consulte "API de serviço Web do HP Universal CMDB" na página 271.
- **API Java do Gerenciamento de Fluxo de Dados.** Permite o gerenciamento de sondas, trabalhos, acionadores e credenciais para o Gerenciamento de Fluxo de Dados. Para obter detalhes, consulte "API Java do Gerenciamento de Fluxo de Dados" na página 306.
- **API do serviço Web do Gerenciamento de Fluxo de Dados** Permite o gerenciamento de sondas, trabalhos, acionadores e credenciais para o Gerenciamento de Fluxo de Dados. Para obter detalhes, consulte "API do Serviço Web do Gerenciamento de Fluxo de Dados" na página 308.

Observação: Para obter o valor total da documentação da API, é recomendável acessar a documentação online. A versão em PDF não tem os links para a documentação da API gerada em formato HTML.

Capítulo 9: API do HP Universal CMDB

Este capítulo inclui:

Convenções 262

Usando a API do HP Universal CMDB 262

Estrutura geral de um aplicativo 263

Colocar o arquivo Jar da API no classpath 266

Criar um usuário de integração 266

Casos de uso da API do UCMDB 268

Exemplos 269

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.

Para ver a documentação completa sobre as APIs disponíveis, consulte HP UCMDB API Reference.

Esses arquivos estão localizados na seguinte pasta:

\\<diretório raiz do UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Usando a API do HP Universal CMDB

Observação: 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
- Recuperar informações do histórico do UCMDB

- 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 obter 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" abaixo
- "Permissões" abaixo

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 da API do UCMDB" na página 268.

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.

Além disso, o usuário deve ter a permissão de ação geral **Acesso ao SDK** para fazer logon.

> **Cuidado:** O cliente da API também pode funcionar com usuários regulares desde que eles tenham permissão de autenticação na API. No entanto, essa opção não é recomendada.

Para obter detalhes, consulte "Criar um usuário de integração" na página 266.

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 ucmdbService =
   provider.connect(provider.createCredentials(USERNAME,
   PASSWORD), provider.createClientContext("Test"));
TopologyQueryService queryService = ucmdbService.getTopologyQueryService(
);
TopologyQueryFactory factory = queryService.getFactory();
QueryDefinition queryDefinition = factory.createQueryDefinition("Test Que
ry");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() + " hosts i
n uCMDB");
```

Os serviços disponíveis no **UcmdbService** são:

Métodos de serviço	Uso
getAuthorizationModelService	Realizar operações de autenticação (criar usuários e usuários, atribuir funções a usuários e grupos e assim por diante).
getClassModelService	informação sobre tipos de ECs e relações
getConfigurationService	Gerenciamento de configurações de infraestrutura, para configuração do servidor
getDataStoreMgmtService	Dados de consulta armazenam informações incluindo quais ECs e atributos serão federados.
getDDMConfigurationService	Configurar o sistema do Gerenciamento de Fluxo de Dados
getDDMManagementService	Analisar e exibir o andamento, resultados e erros do sistema de Gerenciamento de Fluxo de Dados
getDDMZoneService	Importar e exportar zonas de gerenciamento (com suas atividades).
getHistoryService	Informações sobre o histórico de ECs monitorados (alterações, remoções e assim por diante)

Métodos de serviço	Uso
getImpactAnalysisService	Executar o cenário de análise de impacto (também conhecido como **correlação**).
getLicensingService	Informações de consulta sobre licenças instaladas no sistema.
getMultipleCMDBService	Converter entre IDs globais e IDs do UCMDB.
getMultiTenancyService	Criar, ler, atualizar e excluir locatários.
getPersistencyService	Persistir dados binários em pares de valor de chave.
getQueryManagementService	Gerenciar acesso a consultas - salvar, excluir, listar existente. Fornece também validação de consulta e a descoberta de dependências de consultas.
getReconciliationService	Fornece recursos de identificação e mesclagem.
getResourceBundleManagementService	Marcação de recurso (serviços de "agrupamento"). Permite a criação explícita de novas marcas e a remoção de marcas de todos os recursos marcados.
getResourceManagementService	Implantar pacotes de recursos (de consultas TQL, visualizações, usuários e assim por diante) ao sistema.
getSecurityService	Verificar se as credenciais são válidas.
getServerService	Informações genéricas de consulta sobre o sistema.
getSnapshotService	Fornecer serviços para gerenciamento de instantâneos (obter, salvar, comparar e assim por diante)
getSoftwareSignatureService	Definir itens de software a serem descobertos pelo sistema de Gerenciamento de Fluxo de Dados
getStateService	Fornecer serviços para gerenciamento de estados (listar, adicionar, remover e assim por diante)
getSystemHealthService	Fornecer serviços de integridade do sistema (indicadores básicos de desempenho do sistema, capacidade e métricas de disponibilidade)
getTopologyQueryService	Obter informações sobre o universo de TI
getTopologyUpdateService	Alterar informações no universo de TI
getUcmdbVersion	Consultar versões do UCMDB e de pacote de conteúdo e informações sobre compilações.

Métodos de serviço	Uso
getViewArchiveService	Exibir resultados de estatísticas. Permite salvar o resultado da exibição atual e recuperar resultados salvos anteriormente.
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.

O cliente comunica-se com o servidor usando HTTP(S).

Colocar o arquivo Jar da API no classpath

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 onde `localhost` é a máquina onde o UCMDB está instalado 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.

Observação: O uso do Jar da API Java do UCMDB requer que você tenha o JRE versão 6 ou posterior instalado.

Criar um usuário de integração

Você pode criar um usuário dedicado para integrações entre outros produtos e o UCMDBUCMDB. 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: Também é possível conectar a um usuário comum do UCMDB, (por exemplo, admin). No entanto, essa opção não é recomendada. Para conectar a um usuário do UCMDB, você deve conceder ao usuário permissão de autenticação da API.

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.

2. No `UCMDB`, clique em **service=UCMDB Authorization Services**.

3. Localizar a operação **createUser**. Esse método aceita os seguintes parâmetros:

- **customerId**. ID do cliente.
- **username**. O nome do usuário de integração.
- **userDisplayName**. O nome de exibição do usuário de integração.
- **userLoginName**. O nome de logon do usuário de integração.
- **password**. A senha do usuário de integração.

4. Clique em **Invoke**.
5. Em um ambiente de um único locatário, localize o método **setRolesForUser**e insira os seguintes parâmetros:
   - **userName**. O nome do usuário de integração.
   - **funções.** SuperAdmin.

   Clique em **Invoke**.
6. Em um ambiente de locação múltipla, localize o método **grantRolesToUserForAllTenants** e insira os seguintes parâmetros para atribuir a função em conexão com todos os locatários:
   - **userName**. O nome do usuário de integração.
   - **funções.** SuperAdmin.

   Clique em **Invoke**.

   Como alternativa, para atribuir a função em conexão com locatários específicos, chame o método **grantRolesToUserForTenants**, usando os mesmos valores de parâmetro de funções e userName. Para o parâmetro **tenantNames**, insira os locatários necessários.
7. Crie mais usuários ou feche o console JMX.
8. Faça logon no UCMDB como administrador.
9. Na guia **Administração** , execute o **Gerenciador de Pacotes**.
10. Clique no ícone **Criar pacote personalizado**.
11. Digite um nome para o novo pacote e clique em **Avançar**.
12. Na guia Seleção de Recurso, em **Configurações**, clique em **Usuários**.
13. Selecione um ou mais usuários que tenha criado usando o console JMX.
14. Clique em **Avançar** e em **Concluir**. Seu novo pacote aparecerá na lista Nome do Pacote no Gerenciador de Pacotes.

15. Implemente o pacote aos usuários que executarão os aplicativos API.

    Para obter detalhes, consulte a seção "Como 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 **true**. Use os métodos **systemUser** (**removeUser**, **resetPassword**, **UserAuthenticate** e assim por diante).

    Há dois usuários do sistema prontos para o uso. É recomendável alterar suas senhas após a instalação usando o método **resetPassword**.

    - **sysadmin**/**sysadmin**
    - **UISysadmin**/**UISysadmin** (esse usuário também é o **SuperIntegrationUser**).

      Se você alterar a senha UISysadmin usando **resetPassword**, você deverá fazer o seguinte:

      i. No Console JMX, localize o serviço **UCMDB-UI:name=UCMDB Integration**.

      ii. Execute **setCMDBSuperIntegrationUser** com o nome de usuário e a nova senha do usuário de integração.

Casos de uso da API do UCMDB

Os casos de uso listados nesta seção supõem 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 " abaixo
- "Consultando o CMDB " na página seguinte
- "Consultando o modelo de classe" na página seguinte
- "Analisando o impacto das alterações " na página seguinte

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

Veja as amostras de código a seguir:

- Create a Connection
- Create and Execute an Ad Hoc Query

- Create and Execute a View
- Add and Delete Data
- Execute an Impact Analysis
- Query the Class Model
- Query a History Sample

Esses arquivos estão localizados no seguinte diretório:

\\<diretório raiz do UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples

Capítulo 10: API de serviço Web do HP Universal CMDB

Este capítulo inclui:

Convenções 271

Visão geral da API de serviço Web do HP Universal CMDB 272

Chamar o serviço Web 275

Consultar o CMDB 275

Atualizar o UCMDB 279

Consultar o modelo de classe do UCMDB 280

Consulta para análise de impacto 282

Parâmetros gerais do UCMDB 282

Parâmetros de saída do UCMDB 285

Métodos de consulta do UCMDB 286

Métodos de atualização do UCMDB 297

Métodos de análise de impacto do UCMDB 300

API de serviço Web de estado real 302

Casos de uso da API de Serviço Web do UCMDB 304

Exemplos 305

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.

Para ver a documentação completa sobre as estruturas de solicitação e resposta, consulte HP UCMDB Web Service API Reference. Esses arquivos estão localizados na seguinte pasta:

<Diretório Raiz do UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

Visão geral da API de serviço Web do HP Universal CMDB

Observação: Use este capítulo junto com a documentação de esquema do UCMDB, disponível na Biblioteca de Documentação online.

A API de serviço Web do HP Universal CMDB é usada para integrar aplicativos ao HP Universal CMDB (UCMDB). A API fornece métodos para:

- Adicionar, remover e atualizar ECs 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 obter 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 seguinte
- "Permissões" na página seguinte

Usos da API

A API de Serviços Web do UCMDB 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

Para acessar o arquivo WSDL para o serviço Web, vá até: **http://localhost:8080/axis2/services/UcmdbService?wsdl**. Você precisará fornecer ao administrador do servidor credenciais de usuário para exibir o arquivo WSDL.

O usuário deve ter a permissão de ação geral **Executar API Herdada** para fazer logon.

A tabela a seguir exibe as permissões necessárias adicionais para cada comando da API do serviço Web:

Comando da API do serviço Web	Permissões Necessárias
addCIsAndRelations deleteCIsAndRelations updateCIsAndRelations	Ações Gerais: **Atualização de Dados**
executeTopologyQueryByName(AdHoc) executeTopologyQueryByNameWithParameters(AdHoc) executeTopologyQueryWithParameters(AdHoc)	Ações Gerais: **Executar Consulta por Definição** Para cada consulta: Permissão de **Exibição**

Comando da API do serviço Web	Permissões Necessárias
getTopologyQueryExistingResultByName getTopologyQueryResultCountByName releaseChunks pullTopologyMapChunks getCINeighbours getFilteredCIsByType getCIsByID getCIsByType getRelationsById	Ações Gerais: **Exibir ECs** Para cada consulta: Permissão de **Exibição**
getQueryNameOfView	Ações Gerais: **Exibir ECs** Para cada exibição: Permissão de **Exibição**
getChangedCIs	Ações Gerais: **Exibir Histórico**, **Exibir ECs**
calculateImpact getImpactPath getImpactRulesByGroupName getImpactRulesByNamePrefix	Ações Gerais: **Executar Análise de Impacto**
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Nenhum

Observação: Quando o contexto-raiz tiver sido alterado no UCMDB, siga estas etapas para ativar o acesso à API do Serviço Web:

1. Abra o arquivo de configuração **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\web.xml** e localize a seguinte seção:

   ```
   <servlet-class>
   org.apache.axis2.transport.http.AxisServlet
   </servlet-class>
   ```

   Adicione as seguintes linhas após isso:

   ```
   <init-param>
   <param-name>axis2.find.context</param-name>
   <param-value>false</param-value>
   </init-param>
   ```

2. Abra o arquivo de configuração **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\conf\axis2.xml** e localize a seguinte linha:

   ```
   <parameter name="enableSwA" locked="false">false</parameter>
   ```

Adicione a seguinte linha após isso:

```
<parameter name="contextRoot" locked="false">test1/setup1/axis2
</parameter>
```

onde **test1/setup1** é seu contexto-raiz.

(Para remover o contexto-raiz, remova o texto adicionado ao caminho.)

3. Reinicie o servidor do UCMDB.

Chamar o serviço Web

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, o 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 da API de Serviço Web do UCMDB" na página 304.

Consultar o CMDB

O CMDB é consultado usando as APIs descritas em " Métodos de consulta do UCMDB" na página 286. As consultas e os elementos CMDB retornados sempre contêm IDs UCMDB reais. Para ver exemplos de uso dos métodos de consulta, consulte Query Example.

Esta seção inclui os seguintes tópicos:

- "Cálculo de resposta JIT (Just In Time)" na página seguinte
- "Processando respostas extensas" na página seguinte
- "Especificando propriedades para retornar" na página seguinte
- "Propriedades concretas" na página 277

- "Propriedades derivadas" na página seguinte
- "Propriedades de nomenclatura" na página 278
- "Outros elementos de especificação de propriedades" na página 278

Cálculo de resposta JIT (Just In Time)

Para todos os métodos de consulta, o servidor UCMDB 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 295 e "releaseChunks" na página 297.

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.** Usar isso 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 297.

Esta tarefa inclui as seguintes etapas:

- "Parâmetros de atualização do UCMDB" abaixo
- "Uso dos tipos de ID com métodos de atualização" abaixo

Parâmetros de atualização do UCMDB

Este tópico descreve os parâmetros usados somente pelos métodos de atualização do serviço.

- **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 obter detalhes, consulte referencedCIs em "CIsAndRelationsUpdates" na página anterior.

- **Inserindo novos itens 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 298 e " Métodos de consulta do UCMDB" na página 286.

- **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 obter detalhes, consulte CI Type Manager no *Guia de Modelagem do HP Universal CMDB*.

Esta seção fornece informações sobre os seguintes métodos que retornam informações sobre TECs e relacionamentos:

- "getClassAncestors" abaixo
- "getAllClassesHierarchy" na página seguinte
- "getCmdbClassDefinition" na página seguinte

getClassAncestors

O método `getClassAncestors` recupera o caminho entre o TEC específico e sua raiz, além da própria raiz.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
className	O nome do tipo. Para obter detalhes, consulte "Nome do tipo" na página 284.

Saída

Parâmetro	Comentário
classHierarchy	Uma coleção de pares de nomes de classe e nome de classe pai.
comentários	Somente para uso interno.

getAllClassesHierarchy

O método `getAllClassesHierarchy` recupera toda a árvore de modelo de classe.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página seguinte.

Saída

Parâmetro	Comentário
classesHierarchy	Uma coleção de pares de nome de classe e nome de classe pai.
comentários	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 obter detalhes, consulte "CmdbContext" na página seguinte.
className	O nome do tipo. Para obter detalhes, consulte "Parâmetros gerais do UCMDB " na página seguinte.

Saída

Parâmetro	Comentário
cmdbClass	A definição de classe, consistindo em `name`, `classType`, `displayLabel`, `description`, `parentName`, qualificadores e atributos.
comentários	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 Impact Analysis Example.

Parâmetros gerais do UCMDB

Esta seção descreve os parâmetros mais comuns dos métodos do serviço.

Esta seção inclui os seguintes tópicos:

- "CmdbContext" abaixo
- "ID" abaixo
- "Atributos-chave" na página seguinte
- "Tipos de ID" na página seguinte
- "CIProperties" na página seguinte
- "Nome do tipo" na página 284
- "Elemento de configuração (EC)" na página 284
- "Relacionamento" na página 284

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 uma ID do CMDB. 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 obter detalhes, consulte Add/Edit Attribute Dialog Box 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 281.

Tipos de ID

Um elemento `ID` pode conter uma ID real ou uma ID temporária.

Uma 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 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.

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 seguinte e "Relacionamento" na página seguinte.

As coleções de propriedades são:

- `dateProps` - 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 do 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 `EC`, 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, a `ID` pode ficar vazia. Os " Métodos de consulta do UCMDB" obtêm as `ID`s reais como parâmetros de entrada e retornam `ID`s reais nos resultados de consulta.

O `type` pode ser qualquer nome de tipo definido no Gerenciador de Tipo de EC. Para obter detalhes, consulte CI Type Manager no Guia de Modelagem do HP Universal CMDB.

O elemento `props` é uma coleção CIProperties. Para obter detalhes, consulte "Parâmetros gerais do UCMDB " na página 282.

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 os " Métodos de atualização do UCMDB" para atualizar uma `Relação`, o valor da ID de `Relação` pode ser uma ID do CMDB real ou uma ID temporária. Ao excluir um elemento, a `ID` pode ficar vazia. Os " Métodos de consulta do UCMDB" obtêm as `ID`s reais como parâmetros de entrada e retornam `ID`s 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 relacionamentos definidos no CMDB. Para ver mais informações sobre classes ou tipos, consulte "Consultar o modelo de classe do UCMDB " na página 280.

Para obter detalhes, consulte CI Type Manager 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 obter detalhes, consulte "CIProperties" na página anterior.

Parâmetros de saída do UCMDB

Esta seção descreve os parâmetros mais comuns dos métodos do serviço. Para obter mais detalhes, consulte o online schema documentation.

Esta seção inclui os seguintes tópicos:

- "CIs" abaixo
- "ShallowRelation" abaixo
- "Topology" abaixo
- "CINode" abaixo
- "RelationNode" abaixo
- "TopologyMap" abaixo
- "ChunkInfo" na página seguinte

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 relacionamentos 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 anterior).
- `relationNodes`. Este consiste em um ou mais `RelationNode` (consulte "RelationNode" na página anterior).

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 obter mais informações, consulte "Processando respostas extensas" na página 276.

Métodos de consulta do UCMDB

Esta seção fornece informações sobre os seguintes métodos:

- "executeTopologyQueryByNameWithParameters" abaixo
- "executeTopologyQueryWithParameters " na página seguinte
- "getChangedCIs" na página 288
- "getCINeighbours" na página 289
- "getCIsByID" na página 289
- "getCIsByType" na página 290
- "getFilteredCIsByType " na página 290
- "getQueryNameOfView" na página 294
- "getTopologyQueryExistingResultByName" na página 295
- "getTopologyQueryResultCountByName" na página 295
- "pullTopologyMapChunks" na página 295
- "releaseChunks" na página 297

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 obter detalhes, consulte "CmdbContext" na página 282.
queryName	O nome da TQL parametrizada no CMDB para o qual o mapa será obtido.
parameterizedNodeList	As condições que cada nó precisa atender para ser incluído nos resultados da consulta.
queryTypedProperties	Uma coleção de conjuntos de propriedades que serão recuperadas para elementos de um Tipo de Elemento de Configuração específico.

Saída

Parâmetro	Comentário
topologyMap	Para obter detalhes, consulte "TopologyMap" na página 285.
ChunkInfo	Para obter detalhes, consulte "ChunkInfo" na página anterior e "Processando respostas extensas" na página 276.

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 `parameterizedNodeList`. 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.

Para usar uma TQL exportada como entrada desse método, faça o seguinte:

1. Inicie o navegador da Web e insira o seguinte endereço: **http://localhost:8080/jmx-console**.

   Você pode precisar fazer logon com um nome de usuário e senha.

2. Clique em **UCMDB:service=TQL Services**.

3. Localize a operação **exportTql**.
   - Na caixa de parâmetro **customerId**, insira **1** (o padrão).
   - Na caixa de parâmetro **patternName**, insira um nome TQL válido.
4. Clique em **Invoke**.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
queryXML	Uma cadeia de caracteres XML que representa um TQL sem marcas de recursos.
parameterizedNodeList	As condições que cada nó precisa atender para ser incluído nos resultados da consulta.

Saída

Parâmetro	Comentário
topologyMap	Para obter detalhes, consulte "TopologyMap" na página 285.
ChunkInfo	Para obter detalhes, consulte "ChunkInfo" na página 286 e "Processando respostas extensas" na página 276.

getChangedCIs

O método `getChangedCIs` retorna os dados de alteração para todos os ECs relacionados aos ECs especificados.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
ids	A lista dos IDs dos ECs raiz cujos ECs relacionados são verificados para ver se há alterações. Somente IDs do CMDB reais são válidas nessa coleção.
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
getChangedCIsResponseList	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 obter detalhes, consulte "CmdbContext" na página 282.
ID	A 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 obter detalhes, consulte "Nome do tipo" na página 284.
CIProperties	Os dados que serão retornados em cada elemento de configuração, chamados Layout de Consulta na interface do usuário. Para obter detalhes, consulte "TypedProperties" na página 278.
relationProperties	Os dados que serão retornados em cada relacionamento (chamados Layout de Consulta na interface do usuário). Para obter detalhes, consulte "TypedProperties" na página 278.

Saída

Parâmetro	Comentário
topologia	Para obter detalhes, consulte "Topology" na página 285.
comentários	Somente para uso interno.

getCIsByID

O método `getCIsByID` recupera os elementos de configuração por seus IDs do CMDB.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
CIsTypedProperties	Uma coleção de propriedades de tipos. Para obter detalhes, consulte "Outros elementos de especificação de propriedades" na página 278.
IDs	Somente IDs do CMDB reais são válidas nessa coleção.

Saída

Parâmetro	Comentário
CIs	Uma coleção de elementos EC.
ChunkInfo	Para obter detalhes, consulte "ChunkInfo" na página 286 e "Processando respostas extensas" na página 276.

getClsByType

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 obter detalhes, consulte "CmdbContext" na página 282.
type	O nome de classe. Para obter detalhes, consulte "Nome do tipo" na página 284.
propriedades	Os dados que serão retornados em cada elemento de configuração. Para obter detalhes, consulte "CustomProperties" na página 278.

Saída

Parâmetro	Comentário
CIs	Uma coleção de elementos EC.
ChunkInfo	Para ver detalhes, consulte: "ChunkInfo" na página 286 e "Processando respostas extensas" na página 276.

getFilteredClsByType

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.
Igual	Cadeia de caracteres e numérica
EqualIgnoreCase	Cadeia
Maior	Numérico
GreaterEqual	Numérico

Operador	Tipo de condição/comentários
Em	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.
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.
Menor	Numérico
LessEqual	Numérico
Como	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

Operador	Tipo de condição/comentários
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(11f);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.FloatOperator.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.setPropertyNames(Arrays.asList("name","size"));
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties);
```

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.

Parâmetro	Comentário
type	O nome de classe. Para obter detalhes, consulte "Nome do tipo" na página 284. O tipo pode ser qualquer um dos tipos definidos usando o Gerenciador de Tipo de EC. Para obter detalhes, consulte CI Type Manager no *Guia de Modelagem do HP Universal CMDB*.
propriedades	Os dados que serão retornados em cada EC (chamados Layout de Consulta na interface do usuário). Para obter detalhes, consulte "CustomProperties" na página 278.
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 obter detalhes, consulte "ChunkInfo" na página 286 e "Processando respostas extensas" na página 276.

getQueryNameOfView

O método `getQueryNameOfView` recupera o nome do TQL que serve de base para a visualização especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
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 obter detalhes, consulte "CmdbContext" na página 282.
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
topologyMap	Para obter detalhes, consulte "TopologyMap" na página 285.
ChunkInfo	Para ver detalhes, consulte "ChunkInfo" na página 286 e "Processando respostas extensas" na página 276.

getTopologyQueryResultCountByName

O método `getTopologyQueryResultCountByName` recupera o número de instâncias de cada nó que corresponde à consulta especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
queryName	O nome de um TQL.
countInvisible	Se for verdadeiro, a saída incluirá ECs definidos como invisíveis na consulta.

Saída

Parâmetro	Comentário
getTopologyQueryResultCountByNameResponse	O número de instâncias correspondentes à consulta.

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 obter detalhes, consulte "ChunkInfo" na página 286 e "Consultar o CMDB" na página 275.

O aplicativo cliente precisa poder manipular os mapas parciais.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
ChunkRequest	O número da parte a ser recuperada e o `ChunkInfo` que é retornado pelo método de consulta.
queryTypedProperties	Uma coleção de conjuntos de propriedades que serão recuperadas para elementos de um Tipo de Elemento de Configuração específico.

Saída

Parâmetro	Comentário
topologyMap	Para obter detalhes, consulte "TopologyMap" na página 285.
comentários	Somente para uso interno.

Exemplo de processamento de blocos:

```
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,chunkRe
quest);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().sizeCINodeList() ;
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINode(m).getCIs();
            for(int i=0 ; i < cis.sizeCIList() ; i++) {
                // your code to process the CIs
            }
```

```
        }
}

GetCIsByType request =
     new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
     ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1 ; j <= response.getChunkInfo().getNumberOfChunks() ; j++) {
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req = new PullTopologyMapChunks(cmdbContext, chunk
Request);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().getCINodes().size();
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINodes().get(m).getCIs();
            for(int i=0 ; i < cis.getCIs().size(); i++) {
                // your code to process the CIs
            }
        }
}
```

releaseChunks

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 obter detalhes, consulte "CmdbContext" na página 282.
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" abaixo
- "addCustomer" na página seguinte
- "deleteCIsAndRelations" na página seguinte
- "removeCustomer" na página seguinte
- "updateCIsAndRelations" na página 300

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 obter detalhes, consulte "CmdbContext" na página 282.
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.
CIsAndRelationsUpdates	Os elementos que serão atualizados ou criados. Para obter detalhes, consulte "CIsAndRelationsUpdates" na página 279.

Parâmetro	Comentário
ignoreValidation	Se for verdadeiro, nenhuma verificação será executada antes de atualizar o CMDB.
dataStore	Alterar informações.

Saída

Parâmetro	Comentário
createdIDsMapList	A lista de IDs do cliente mapeadas para IDs do CMDB. Para obter detalhes, consulte a descrição acima.
comentários	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 obter detalhes, consulte "CmdbContext" na página 282.
CIsAndRelationsUpdates	Os itens que serão excluídos. Para obter detalhes, consulte "CIsAndRelationsUpdates" na página 279
dataStore	Alterar informações.

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 obter detalhes, consulte "CmdbContext" na página 282.
CIsAndRelationsUpdates	Os itens que serão atualizados. Para obter detalhes, consulte "CIsAndRelationsUpdates" na página 279.
ignoreValidation	Se for verdadeiro, nenhuma verificação será executada antes de atualizar o CMDB.
dataStore	Alterar informações.

Saída

Parâmetro	Comentário
createdIDsMapList	A lista de IDs do cliente mapeadas para IDs do CMDB. Para obter detalhes, consulte "addCIsAndRelations" na página 298.

Métodos de análise de impacto do UCMDB

Esta seção fornece informações sobre os seguintes métodos:

- "calculateImpact" abaixo
- "getImpactPath" na página seguinte
- "getImpactRulesByNamePrefix" na página 302

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" abaixo.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
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 obter detalhes, consulte "Topology" na página 285.
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" na página anterior é usada como o argumento de entrada `identifier` de `getImpactPath`.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
identifier	A chave da resposta do servidor que foi retornada por calculateImpact.
relation	Uma relação que se baseia em um dos "ShallowRelation" retornados por calculateImpact no elemento `impactTopology`.

Saída

Parâmetro	Comentário
impactPathTopology	Uma coleção de `ECs` e uma coleção `ImpactRelations`.
comentários	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 obter detalhes, consulte "CmdbContext" na página 282.
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`.

API de serviço Web de estado real

A API do serviço Web de estado real é usada principalmente pelo Service Manager para recuperar informações de estado real para uma ID específica do CMDB ou uma ID global e uma ID de cliente específica. A API encontra uma consulta correspondente na pasta **Integração/Consulta do SM** e executa o TQL com a ID do CMDB ou a ID Global como uma condição e retorna a saída da consulta.

URL do serviço Web: http://[machine_name]:8080/axis2/services/ucmdbSMService

Esquema do serviço Web: http://[machine_name]:8080/axis2/services/ucmdbSMService?xsd=xsd0

Fluxo

Quando o método da API é chamado, ele tenta encontrar uma consulta apropriada na pasta **Integração/Consulta do SM**. Ele tenta corresponder o tipo da ID do CMDB/ID Global solicitada com uma das consultas na pasta anterior procurando primeiro um **QueryElement** com o nome **Root** e, se não for encontrado, ele tenta usar qualquer **QueryNode** do mesmo tipo como a ID do CMDB/ID Global solicitada. Depois que uma Consulta e QueryNode forem encontrados, ele coloca CMDBID/GlobalID como uma condição em QueryNode e executa Query. O resultado é retornado ao chamador da API.

Manipulando o resultado usando transformações

Em alguns casos, convém aplicar mais transformações ao XML resultante (por exemplo, para

somar os tamanhos de todos os discos e adicionar essa soma como um atributo adicional ao EC). Para adicionar mais transformações aos resultados do TQL, coloque um recurso chamado **[tql_ name].xslt** na configuração do adaptador, como a seguir: **Gerenciamento do Adaptador > ServiceDeskAdapter7-1 > Arquivos de Configuração > [tql_name].xslt.**

Logs para a API de serviço Web de estado real

A configuração do log para o UCMDB reside em: **UCMDBServer/Conf/log** nos vários arquivos ***.properties**.

Para exibir logs do fluxo de estado real do SM:

1. Abra o arquivo **cmdb_soaapi.properties** e mude o nível de log para depurar da seguinte forma: **loglevel=DEBUG**.
2. Abra o arquivo **fcmdb.properties** e mude o nível de log para depurar da seguinte forma: **loglevel=DEBUG**.
3. Aguarde 1 minuto até que o servidor recupere as mudanças.
4. Execute o estado real a partir do SM.
5. Veja os seguintes arquivos de log em **UCMDBServer/Runtime/log**:
   - cmdb.soaapi.log
   - fcmdb.log

Habilitando o estado real de ECs replicados após alterar o contexto raiz

Se você tiver alterado o contexto-raiz usado para acessar o UCMDB, você deverá fazer as seguintes alterações de configuração para habilitar o estado real de ECs replicados:

1. Em **UCMDBServer\deploy\axis2\WEB-INF**, abra o arquivo **web.xml**.
2. Adicione o seguinte parâmetro **servlet init** a AxisServlet (cole essas quatro linhas após a linha 28):

   ```
   <init-param>
   <param-name>axis2.find.context</param-name>
   <param-value>false</param-value>
   </init-param>
   ```

   Essa configuração impede que o Axis2 tente calcular a raiz de contexto e informa a ele para procurar explicitamente em **axis2.xml**.
3. Em **UCMDBServer\deploy\axis2\WEB-INF\conf**, abra o arquivo **axis2.xml**.
4. Na linha 58, remova os comentários do parâmetro **contextRoot** e edite-o como indicado a seguir:

```
<parameter name="contextRoot" locked="false">test/axis2</parameter>
```

(em que **test** é o novo contexto-raiz em **cmdb.xml**).

> **Observação:** Não há nenhuma barra no início de **test/axis2**.

Casos de uso da API de Serviço Web do UCMDB

Os seguintes casos de uso pressupõem dois sistemas:

- HP Universal CMDB
- Um sistema de terceiros que contém um repositório dos elementos de configuração

Esta seção inclui os seguintes tópicos:

- "Populando o CMDB" abaixo
- "Consultando o CMDB" abaixo
- "Consultando o modelo de classe" na página seguinte
- "Analisando o impacto das alterações" na página seguinte

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

Veja as amostras de código a seguir:

- The Example Base Class
- Query Example
- Update Example
- Class Model Example
- Impact Analysis Example

Esses arquivos estão localizados no seguinte diretório:

\\<diretório raiz do UCMDB>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\WebServiceAPI_Samples

Capítulo 11: API Java do Gerenciamento de Fluxo de Dados

Este capítulo inclui:

Usando a API Java do Gerenciamento de Fluxo de Dados 306

Usando a API Java do Gerenciamento de Fluxo de Dados

Observação: Use este capítulo junto com a documentação da API DFM, disponível na Biblioteca de Documentação online.

Este capítulo explica como ferramentas de terceiros ou personalizadas podem usar a API Java do Gerenciamento de Fluxo de Dados da HP para gerenciar o Fluxo de Dados. A API fornece métodos para:

- **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**. Dados sobre domínios e sondas.

Os seguintes serviços estão disponíveis no pacote Serviços de Descoberta:

- **DDMConfigurationService.** Serviços para configurar as Sondas de Fluxo de Dados, clusters, Intervalos de IP e Credenciais. O servidor do Universal Discovery pode ser configurado com um arquivo XML ou pelo Data Flow Probe.
- **DDMManagementService.** Serviços para analisar e exibir o andamento, resultados e erros da execução do Universal Discovery.
- **DDMSoftwareSignatureService.** Serviços para definir itens de software a serem descobertos pelos componentes do Data Flow Probe. As definição são em todo o sistema. Se mais de um componente da sonda de Fluxo de Dados for definido, as definições serão aplicadas a todos eles.
- **DDMZoneService.** Serviços para gerenciar descoberta baseada em zona.

Além desses serviços, há APIs de cliente de Gerenciamento de Fluxo de Dados, que são usadas na criação de adaptadores Jython. Para obter detalhes, consulte "Desenvolvimento de adaptadores Jython" na página 38.

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.

Além disso, o usuário deve ter a permissão de ação geral **Acesso ao SDK** para fazer logon.

> **Cuidado:** O cliente da API também pode funcionar com usuários regulares desde que eles tenham permissão de autenticação na API. No entanto, essa opção não é recomendada.

Para obter detalhes, consulte "Criar um usuário de integração" na página 266.

Capítulo 12: API do Serviço Web do Gerenciamento de Fluxo de Dados

Este capítulo inclui:

Visão Geral da API do Serviço Web do Gerenciamento de Fluxo de Dados 308

Convenções 309

Chamando o serviço Web 309

Métodos de Gerenciamento de Fluxo de Dados e Estruturas de Dados 309

Amostra de código 320

Exemplo de adição de credenciais 323

Visão Geral da API do Serviço Web do Gerenciamento de Fluxo de Dados

Este capítulo explica como ferramentas de terceiros ou personalizadas podem usar a API do serviço Web do Gerenciamento de Fluxo de Dados da HP para gerenciar o Fluxo de Dados.

A API do serviço Web do Gerenciamento de Fluxo de Dados da HP é usada para integrar aplicativo com o HP Universal CMDB. A API fornece métodos para:

- **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**. Dados sobre domínios e sondas.

Os usuários do serviço Web do Gerenciamento de Fluxo de Dados da HP devem estar familiarizados com:

- A especificação SOAP
- Uma linguagem de programação orientada a objeto como C++, C# ou Java
- HP Universal CMDB
- Gerenciamento de Fluxo de Dados

O usuário deve ter a permissão de ação geral **Executar API Herdada** para fazer logon.

Para obter a documentação completa sobre as operações disponíveis, consulte *Referência de Esquema do HP Discovery and Dependency*. Esses arquivos estão localizados na seguinte pasta:

<Diretório Raiz do UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_Schema\webframe.html

Convenções

Este capítulo usa as seguintes convenções:

- Esse `Element` de estilo indica que um item é uma entidade no banco de dados ou um elemento definido no esquema, incluindo estruturas transmitidas ou retornadas por métodos. Um texto simples indica que o item está sendo discutido em um contexto geral.
- Os elementos e argumentos de método do Gerenciamento de Fluxo de Dados são escritos levando em conta maiúsculas e minúsculas do mesmo modo como são especificados no esquema. Isso normalmente significa que um nome de classe ou uma referência genérica a uma instância da classe é capitalizada. Um elemento ou argumento de um método não tem inicial maiúscula. Por exemplo, `credential` é um elemento de tipo `Credential` repassado para um método.

Chamando o serviço Web

A API do serviço Web do Gerenciamento de Fluxo de Dados da HP permite que sejam chamados métodos do servidor usando técnicas de programação SOAP padrão. 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, o serviço 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.

Para chamar o serviço, use:

- `Protocolo: http ou https (depende da configuração do servidor)`
- `URL: <Servidor do UCMDB>:8080/axis2/services/DiscoveryService`
- `Senha padrão: "admin"`
- `Nome de usuário padrão: "admin"`

Os programadores SOAP podem acessar o WSDL em:

- `axis2/services/DiscoveryService?wsdl`

Métodos de Gerenciamento de Fluxo de Dados e Estruturas de Dados

Esta seção lista os métodos e as estruturas de dados da API do Serviço Web do Gerenciamento de Fluxo de Dados e fornece um breve resumo de seus usos. Para ver a documentação completa

da solicitação e resposta de cada operação, consulte *Referência de Esquema do HP Universal Discovery*.

Esta seção inclui os seguintes tópicos:

- "Estruturas de dados" abaixo
- "Gerenciando métodos de trabalho de descoberta" na página seguinte
- "Gerenciando métodos de acionador" na página 312
- "Métodos de dados de sonda e domínio" na página 314
- "Métodos de dados de credenciais" na página 316
- "Métodos de atualização de dados" na página 318

Estruturas de dados

Algumas das estruturas de dados usadas na API do serviço Web do Gerenciamento de Fluxo de Dados.

CIProperties

`CIProperties` é uma coleção de coleções. Cada coleção contém propriedades de um tipo de dado diferente. Por exemplo, pode haver uma coleção `dateProps`, `strListProps`, `xmlProps` e assim por diante.

Cada coleção de tipo contém propriedades individuais do tipo fornecido. Os nomes dos elementos dessas propriedades são os mesmos do contêiner, mas no singular. Por exemplo, `dateProps` contém elementos `dateProp`. Cada propriedade é um par nome-valor.

Consulte CIProperties na *Referência de Esquema do HP Universal Discovery*.

IPList

Uma lista de elementos `IP`, sendo que cada um contém um endereço IPv4 ou IPv6.

Consulte IPList na *Referência de Esquema do HP Universal Discovery*.

IPRange

Um `IPRange` tem dois elementos, `Start` e `End`. Cada elemento contém um elemento `Address` que é um endereço IPv4 ou IPv6.

Consulte IPRange na *Referência de Esquema do HP Universal Discovery*.

Scope

Dois `IPRanges`. `Exclude` é uma coleção de `IPRanges` a excluir do trabalho. `Include` é uma coleção de `IPRanges` a incluir no trabalho.

Consulte Scope na *Referência de Esquema do HP Universal Discovery*.

Gerenciando métodos de trabalho de descoberta

activateJob

Ativa o trabalho especificado.

Consulte " Amostra de código" na página 320.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	O nome do trabalho.

deactivateJob

Desativa o trabalho especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	O nome do trabalho.

dispatchAdHocJob

Distribui um trabalho na sonda ad-hoc. O trabalho precisa estar ativo e conter o EC acionador especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	O nome do trabalho.
CIID	A ID do EC acionador.
ProbeName	O nome da sonda.
Tempo limite	Em milissegundos

getDiscoveryJobsNames

Retorna a lista de nomes de trabalho.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.

Saída

Parâmetro	Comentário
strList	A lista de nomes de trabalho.

isJobActive

Verifica se o trabalho está ativo.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	O nome do trabalho a verificar.

Saída

Parâmetro	Comentário
JobState	Verdadeiro se o trabalho está ativo.

Gerenciando métodos de acionador

addTriggerCI

Adiciona um novo EC acionador ao trabalho especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	O nome do trabalho.
CIID	A ID do EC acionador.

addTriggerTQL

Adiciona um novo TQL acionador ao trabalho especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	O nome do trabalho.
TqlName	O nome do TQL a adicionar.

disableTriggerTQL

Impede o TQL de acionar o trabalho, mas não o remove permanentemente da lista de consultas que acionam o trabalho.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	O nome do trabalho.

removeTriggerCI

Remove o EC especificado da lista de ECs que acionam o trabalho.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	Nome do trabalho.
CIID	A ID do EC acionador.

removeTriggerTQL

Remove o TQL especificado da lista de consultas que acionam o trabalho.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	Coleção de nomes de trabalhos a verificar.
CIID	A ID do TQL a remover.

setTriggerTQLProbesLimit

Restringe as sondas em que o TQL está ativo no trabalho à lista especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
JobName	O nome do trabalho.
tqlName	O nome do TQL.
probesLimit	A lista de sondas para as quais o TQL está ativo.

Métodos de dados de sonda e domínio

getDomainType

Retorna o tipo de domínio.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O nome do domínio.

Saída

Parâmetro	Comentário
domainType	O tipo de domínio.

getDomainsNames

Retorna os nomes dos domínios atuais.

Consulte " Amostra de código" na página 320.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.

Saída

Parâmetro	Comentário
domainNames	A lista de nomes de domínios.

getProbeIPs

Retorna os endereços IP da sonda especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio a verificar.
probeName	O nome da sonda usada nesse domínio.

Saída

Parâmetro	Comentário
probeIPs	A "IPList" dos endereços na sonda.

getProbesNames

Retorna os nomes das sondas no domínio especificado.

Consulte " Amostra de código" na página 320.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio a verificar.

Saída

Parâmetro	Comentário
probesName	A lista de sondas no domínio.

getProbeScope

Retorna a definição de escopo da sonda especificada.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio a verificar.
probeName	O nome da sonda.

Saída

Parâmetro	Comentário
probeScope	O "Scope" da sonda.

isProbeConnected

Verifica se a sonda especificada está conectada.

Consulte " Amostra de código" na página 320.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio a verificar.
probeName	A sonda a ser verificada

Saída

Parâmetro	Comentário
isConnected	Verdadeiro se a sonda está conectada.

updateProbeScope

Define o escopo da sonda especificada, substituindo o escopo existente.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio.
probeName	A sonda a ser atualizada.
newScope	O "Scope" a definir para a sonda.

Métodos de dados de credenciais

addCredentialsEntry

Adiciona uma entrada de credenciais ao protocolo especificado para o domínio especificado.

Consulte " Amostra de código" na página 320.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio a atualizar.

Parâmetro	Comentário
protocolName	O nome do protocolo.
credentialsEntryParameters	A coleção "CIProperties" das novas credenciais.

Saída

Parâmetro	Comentário
credentialsEntryID	A ID do EC da nova entrada de credencial.

getCredentialsEntriesIDs

Retorna os IDs das credenciais definidas para o protocolo especificado.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio para o qual obter as credenciais.
protocolName	O nome de um protocolo usado nesse domínio.

Saída

Parâmetro	Comentário
credentialsEntryIDs	A lista de IDs de credenciais para o protocolo no domínio.

getCredentialsEntry

Retorna as credenciais definidas para o protocolo especificado. Os atributos criptografados são retornados vazios.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio para o qual obter as credenciais.
protocolName	O nome de um protocolo usado nesse domínio.
credentialsEntryID	A ID da credencial a obter.

Saída

Parâmetro	Comentário
credentialsEntryParameters	A coleção "CIProperties" das credenciais.

removeCredentialsEntry

Retorna as credenciais especificadas do protocolo.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio.
protocolName	O nome de um protocolo usado no domínio.
credentialsEntryID	A ID da credencial a remover.

updateCredentialsEntry

Define novos valores para propriedades da entrada de credenciais especificada.

As propriedades existentes são excluídas e essas propriedades são definidas. Qualquer propriedade cujo valor não está definido nessa chamada permanece indefinido.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
domainName	O domínio no qual atualizar credenciais.
protocolName	O nome de um protocolo usado no domínio.
credentialsEntryID	A ID da credencial a atualizar.
credentialsEntryParameters	A coleção "CIProperties" a definir como propriedades para as credenciais.

Métodos de atualização de dados

rediscoverCIs

Localiza os acionadores que descobriram os objetos de EC especificados e executa novamente esses acionadores. **rediscoverCIs** é executado de modo assíncrono. Chame **checkDiscoveryProgress** para determinar quando a redescoberta estará concluída.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
CmdbIDs	Coleção de IDs dos objetos a redescobrir.

Saída

Parâmetro	Comentário
isSucceed	Verdadeiro de a redescoberta de ECs foi bem-sucedida.

checkDiscoveryProgress

Retorna o andamento da chamada **rediscoverCIs** mais recente nos IDs especificados. A resposta é um valor entre 0 e 1. Quando a resposta for 1, a chamada rediscoverCIs terá sido concluída.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
CmdbIDs	Coleção de IDs dos objetos na chamada de redescoberta a rastrear.

Saída

Parâmetro	Comentário
andamento	Um trabalho concluído tem um andamento de 1. Trabalhos não concluídos têm uma fração menor que 1.

rediscoverViewCIs

Localiza os acionadores que criaram os dados para popular a visualização especificada e executa novamente esses acionadores. **rediscoverViewCIs** é executado de modo assíncrono. Chame **checkViewDiscoveryProgress** para determinar quando a redescoberta estará concluída.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
viewName	As visualizações a verificar.

Saída

Parâmetro	Comentário
isSucceed	Verdadeiro se a redescoberta de ECs foi bem-sucedida.

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.

Entrada

Parâmetro	Comentário
cmdbContext	Para obter detalhes, consulte "CmdbContext" na página 282.
viewName	A coleção de visualizações a verificar.

Saída

Parâmetro	Comentário
andamento	Um trabalho concluído tem um andamento de 1. Trabalhos não concluídos têm uma fração menor que 1.

Amostra de código

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.*;
import com.hp.ucmdb.generated.types.*;
public class test {
    static final String HOST_NAME = "<my_hostname>";
    static final int PORT = 8080;
    private static final String PROTOCOL = "http";
    private static final String FILE = "/axis2/services/DiscoveryService";

    private static final String PASSWORD = "<my_password>";
    private static final String USERNAME = "<my_username>";

    private static CmdbContext cmdbContext = new CmdbContext("ws tests");


    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest(
            "Range IPs by ICMP", cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protcol
        addNTCMDCredentialsEntry();
    }
```

```
    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(
                new GetDomainsNamesRequest(cmdbContext)).
                getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credential
s");
            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", newCredsPropertie
s);
        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, C
IProperties 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 Ge
tDomainsNamesRequest(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=" + isConn
ected);
            }
        }
    }


    private static DiscoveryService getDiscoveryService() throws Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub =
                new DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth =
                new HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
```

```
            serviceStub._getServiceClient().getOptions().setProperty(
                HTTPConstants.AUTHENTICATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }
        return discoveryService;
    }
}
```

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 addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(new GetDomainsNamesRequest(cmdbCont
ext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credential
s");
            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", newCredsPropertie
s);
        setStringProperties("ntadminprotocol_ntdomain", "test doamin", newCredsP
roperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(new AddCredentialsEntryRequest(doma
inName,  "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, C
IProperties 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 Ge
tDomainsNamesRequest(cmdbContext ));
        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName = result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(new GetProbesNamesRequest(domain
Name, 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 IsProbeConnectedReques
t(domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected();
                // Do something ...
                System.out.println("probe " + probeName + " isconnect=" + isConn
ected);
            }
        }
    }

    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.toSt
ring());

            // Authenticate info
            HttpTransportProperties.Authenticator auth = new HttpTransportProper
ties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty(HTTPConstan
ts.AUTHENTICATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }
        return discoveryService;
```

```
    }
} // End class
```

Agradecemos seu feedback!

Se tiver comentários sobre este documento, entre em contato com a equipe de documentação por e-mail. Se um cliente de e-mail estiver configurado nesse sistema, clique no link acima e uma janela de e-mail será aberta com as seguintes informações na linha de assunto:

Feedback sobre Guia de Referência do Desenvolvedor (Universal CMDB 10.10)

Adicione seu feedback ao e-mail e clique em Enviar.

Se nenhum cliente de e-mail estiver disponível, copie as informações acima para uma nova mensagem em um cliente de e-mail da Web e envie seu feedback para SW-Doc@hp.com.