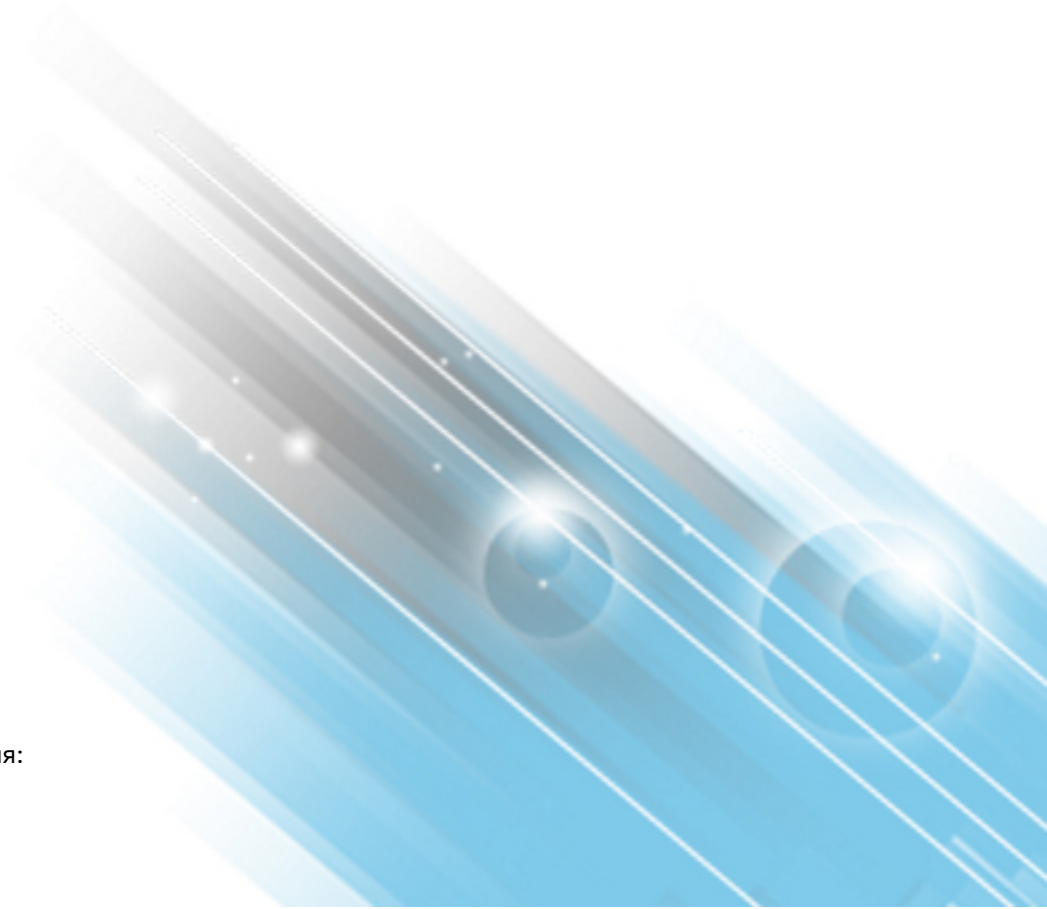
HP Universal CMDB

Версия программного обеспечения: 10.20

Справочное руководство для разработчиков



Дата выпуска документа: Январь 2015 г.
Дата выпуска программного обеспечения: Январь 2015 г.

Правовые уведомления

Гарантийные обязательства

Гарантии на продукты и услуги компании HP формулируются только в заявлениях о прямой гарантии, сопровождающих эти продукты и услуги. В них нет ничего, что может быть истолковано как дополнительная гарантия. Компания HP не несет ответственности за содержащиеся в них технические или редакционные ошибки.

Приводимые в них сведения могут быть изменены без какого-либо уведомления.

Ограничение прав

Конфиденциальное компьютерное ПО. Для обладания, использования или копирования необходима действующая лицензия от компании HP. Согласно FAR 12.211 и 12.212, выдача лицензий на коммерческое компьютерное ПО, документацию на компьютерное ПО и технические данные для коммерческих элементов правительству США производится на условиях стандартной коммерческой лицензии поставщика.

Заявление об авторских правах

© 2002 - 2015 Hewlett-Packard Development Company, L.P.

Информация о товарных знаках

Adobe™ является товарным знаком компании Adobe Systems Incorporated.

Microsoft® и Windows® являются зарегистрированными в США товарными знаками корпорации Microsoft.

UNIX® является зарегистрированным товарным знаком группы The Open Group.

Этот продукт содержит интерфейс библиотеки сжатия данных общего назначения zlib. © Жан-Лу Гайи (Jean-loup Gailly) и Марк Адлер (Mark Adler), 1995-2002.

Обновление документации

На титульном листе настоящего документа приведены следующие идентификационные данные:

- Номер версии программного обеспечения.
- Дата выпуска документа, которая меняется при каждом обновлении документа.
- Дата выпуска ПО, которая указывает дату выпуска текущей версии программного обеспечения.

Чтобы проверить наличие обновлений или убедиться в том, что используется последняя редакция документа, откройте веб-сайт **https://softwaresupport.hp.com**

Чтобы воспользоваться этим сайтом, необходимо зарегистрировать идентификатор HP Passport и войти в систему. Регистрация HP Passport ID производится на сайте **https://hpp12.passport.hp.com/hppcf/createuser.do**

Либо нажмите ссылку **Зарегистрировать** в верхней части страницы поддержки HP Software.

Оформление подписки в службе поддержки соответствующего продукта также позволит получать обновленные и новые редакции. Обратитесь в торговое представительство компании HP для получения подробной информации.

Поддержка

Посетите веб-сайт технической поддержки компании HP Software по адресу **https://softwaresupport.hp.com**

Этот веб-сайт содержит контактную информацию и дополнительные сведения о продуктах, услугах и поддержке, которые предоставляет HP Software.

Веб-сайт технической поддержки программного обеспечения компании HP предоставляет возможности самостоятельного решения проблем. Это позволяет быстро и эффективно получить доступ к интерактивным инструментам технической поддержки, необходимым для управления компанией. Каждый клиент службы поддержки может пользоваться следующими функциями веб-сайта технической поддержки:

- поиск документов базы знаний;
- отправка и отслеживание обращений и запросов на расширение возможностей;
- загрузка исправлений программного обеспечения;
- управление договорами поддержки;
- поиск контактов технической поддержки HP;
- просмотр сведений о доступных услугах;
- участие в обсуждениях различных вопросов с другими заказчиками ПО;
- поиск курсов обучения по программному обеспечению и регистрация для участия в них.

Для получения доступа к большинству разделов поддержки сначала необходимо зарегистрироваться в качестве пользователя службы HP Passport, а затем войти в систему. Использование некоторых из них также требует наличия договора на оказание поддержки. Чтобы зарегистрироваться для получения идентификатора HP Passport ID, перейдите на веб-сайт

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

Дополнительные сведения об уровнях доступа представлены по адресу

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

HP Software Solutions Now обеспечивает доступ к веб-сайту HPSW Solution and Integration Portal. Да этом веб-сайте можно узнать, какие продукты и решения HP подойдут для ваших деловых задач, ознакомиться с полным списком интеграций между продуктами HP, а также найти перечень процессов ITIL. Адрес веб-сайта: **http://h20230.www2.hp.com/sc/solutions/index.jsp**

О PDF-версии интерактивной справки

Этот документ является PDF-версией интерактивной справки. PDF-файл предоставляется для удобства печати нескольких разделов справочных сведений и чтения интерактивной справки в формате PDF. Поскольку содержимое этого документа было изначально рассчитано на просмотр в виде интерактивной справки через веб-браузер, некоторые главы могут быть неверно отформатированы. Некоторые интерактивные главы могут отсутствовать в PDF-версии. Эти главы можно распечатать из интерактивной справки.

Содержание

Часть I: Создание адаптеров обнаружения и интеграции 11
Глава 1: Разработка и написание адаптеров 12
Обзор разработки и написания адаптеров 12
Создание содержимого 12
Цикл разработки адаптеров 13
Управление потоком данных и интеграция 16
Связывание ценности для бизнеса и разработки адаптеров обнаружения 17
Изучение требований к интеграции 18
Разработка содержимого интеграции 20
Разработка содержимого обнаружения 23
Адаптеры обнаружения и связанные компоненты 23
Разделение адаптеров 24
Внедрение адаптера обнаружения 25
Шаг 1: Создание адаптера 28
Шаг 2: Назначение задания адаптеру 34
Шаг 3: Создание кода Jython 35
Настройка удаленного выполнения процессов 35
Глава 2: Разработка адаптеров Jython 37
Справка по API-интерфейсу Управления потоком данных 37
Создание кода Jython 37
Использование внешних JAR-файлов Java в Jython 38
Выполнение кода 38
Изменение встроенных сценариев 38
Структура файла Jython 39
Импорт 40
Главная функция — DiscoveryMain 40
Определение функций 40
Формирование результатов сценарием Jython 42
Синтаксис ObjectStateHolder 42
Отправка больших объемов данных 43
Экземпляр Framework 44
Поиск правильных учетных данных (для адаптеров подключения) 48
Обработка исключений Java 50
Устранение неполадок при обновлении Jython с версии 2.1 до 2.5.3 50
Поддержка локализации в адаптерах Jython 52
Добавление поддержки нового языка 52
Изменение языка по умолчанию 54
Определение набора символов для кодировки 54
Настройка нового задания для работы с локализованными данными 55
Декодирование команд без ключевых слов 56
Работа с пакетами ресурсов 56

Справка по API-интерфейсам 57
Запись кода DFM 59
Средства и библиотеки Jython 61
Глава 3: Сообщения об ошибках 65
Сообщения об ошибках — Обзор 65
Правила сообщений об ошибках 65
Уровни серьезности ошибок 68
Глава 4: Сопоставление зависимостей потребитель-поставщик 70
Обнаружение зависимостей — Обзор 70
Поставщики и потребители 71
Сигнатуры зависимостей 71
Поток сопоставления зависимостей 72
Файлы сигнатуры зависимости 72
Структура файла сигнатуры зависимости 72
Переменные и понятия 73
Значения по умолчанию 76
Типы переменных IP-адресов 77
Определение дескриптора потребителя 77
Определение зависимостей 79
Составление выражения поиска 79
Использование значений переменных по умолчанию 82
Указание путей к документам конфигурации 84
Переопределение документа конфигурации 85
Зависимости, определенные в нескольких документах 87
Документы конфигурации PROPERTIES 90
Документы конфигурации XML 94
Текстовые документы конфигурации 97
Указание области поиска 99
Определение TQL-запросов 102
Упаковка и развертывание нескольких файлов сигнатуры зависимости 103
Ошибки компиляции 104
Адаптеры поиска зависимостей 105
Создание адаптера поиска зависимостей 106
Определение адаптера потребитель-поставщик 107
Определение входного TQL-запроса и целевых данных 108
Указание значений для переменных 109
Указание значений для переменных понятий 109
Запись сценария Jython 112
Ограничения адаптера 114
Полный пример 114
Рабочий процесс разработки 115
Разработка сигнатур зависимостей 116
Разработка адаптера 118
Глава 5: Разработка общих адаптеров БД 122
Обзор общего адаптера БД 122
TQL-запросы для общего адаптера БД 123

Выверка 124
Hibernate как поставщик JPA 124
Подготовка к созданию адаптера 127
Подготовка пакета адаптера 131
Настройка адаптера — Минимальный метод 134
Настройка файла adapter.conf. 134
Пример. Заполнение данных узла и IP-адреса упрощенным методом 135
Настройка адаптера — Расширенный метод 138
Реализация подключаемого модуля 142
Развертывание адаптера 145
Изменение адаптера 145
Создание точки интеграции 145
Создание представления 146
Вычисление результатов 146
Просмотр результатов 146
Просмотр отчетов 146
Активация файлов журнала 146
Использование Eclipse для сопоставления атрибутов ЭК и таблиц базы данных 147
Файлы конфигурации адаптеров 154
Файл adapter.conf 155
Файл simplifiedConfiguration.xml 156
Файл orm.xml 158
Файл reconciliation_types.txt 172
Файл reconciliation_rules.txt (для обратной совместимости) 172
Файл transformations.txt 174
Файл discriminator.properties 175
Файл replication_config.txt 176
Файл fixed_values.txt 176
Файл persistence.xml 176
Подключение к базе данных при помощи проверки подлинности NT 177
Настройка в файле persistence.xml использования в интеграции SCCM аутентификации NTLM 178
Встроенные конвертеры 179
Подключаемые модули 184
Примеры конфигурации 185
Файлы журнала адаптера 193
Внешние ссылки 195
Устранение неполадок и ограничения — Разработка общих адаптеров БД 195
Глава 6: Разработка адаптеров Java 197
Обзор Federation Framework 197
Взаимодействие адаптера и сопоставления в Federation Framework 202
Поток Federation Framework для объединенных TQL-запросов 203
Взаимодействие между Federation Framework, сервером, адаптером и системой сопоставления 204
Поток Federation Framework для заполнения 213

Интерфейсы адаптера 214
Устранение неполадок в ресурсах адаптеров 215
Добавление адаптера для нового внешнего источника данных 216
Создание примера адаптера 223
Теги конфигурации и свойства XML 224
Интерфейс DataAdapterEnvironment 226
OutputStream openResourceForWriting(String resourceName) throws FileNotFoundException; 226
InputStream openResourceForReading(String resourceName) throws FileNotFoundException; 226
Properties openResourceAsProperties(String propertiesFile) throws IOException; 227
String openResourceAsString(String resourceName) throws IOException; 227
public void saveResourceFromString(String relativeFileName, String value) throws IOException; 228
boolean resourceExists(String resourceName); 228
boolean deleteResource(String resourceName); 229
Collection<String> listResourcesInPath(String path); 229
DataAdapterLogger getLogger(); 229
DestinationConfig getDestinationConfig(); 229
int getChunkSize(); 229
int getPushChunkSize(); 230
ClassModel getLocalClassModel(); 230
CustomerInformation getLocalCustomerInformation(); 230
Object getSettingValue(String name); 230
Map<String, Object> getAllSettings(); 230
boolean isMTEnabled(); 230
String getUcmdbServerHostName(); 231
Глава 7: Разработка адаптеров принудительной отправки 232
Разработка и развертывание адаптеров принудительной отправки данных 232
Создание пакета адаптера 233
Устранение неполадок 235
Оптимальные методы создания TQL-запросов для адаптеров принудительной отправки данных 236
Создание сопоставлений 236
Создание файла сопоставления 237
Подготовка файлов сопоставления 237
Создание сценариев Jython 240
Поддержка разностной синхронизации 244
SQL-запросы общего адаптера принудительной отправки в формате XML 245
Общий адаптер принудительной отправки данных веб-служб 246
Справочные сведения о файлах сопоставления 265
Схема файла сопоставления 267
Схема результатов сопоставления 276
Настройка 278
Глава 8: Разработка общих адаптеров 280
Синхронизация экземпляров 280

Принудительная отправка данных с помощью общего адаптера 280
Принудительная отправка — Обзор 281
Файл сопоставления 281
The Groovy Traveler 284
Создание сценариев Groovy 287
Реализация интерфейса PushAdapterConnector 288
Принудительная отправка данных с помощью общего адаптера 289
Архитектура Population Framework 289
Основные артефакты, включенные в заполнение 290
TQL-запросы заполнения 291
Файлы сопоставления заполнением 291
Автоматическое заполнение связей 294
Заполнение связей вручную 294
Соединитель заполнения 296
Входные данные запроса заполнения 298
Выходные данные запроса заполнения 301
Режимы адаптера заполнения 302
Явное сопоставление внешних ID 303
Принудительная обратная отправка глобальных идентификаторов 303
Объединение данных с помощью общего адаптера 304
Сопоставительный подход к объединению 305
API объединения общего адаптера 305
Интерфейс соединителя общего адаптера для объединения 307
Поддерживаемые запросы объединения 308
Настройка объединения 308
Настройка параметров адаптера 309
Настройка запросов объединения из файлов журнала 309
Пример настройки объединения 313
Выверка 318
API общего адаптера 318
API-интерфейсы указателей ресурсов 319
Создание пакета общего адаптера 319
Создание пакета адаптера 321
TQL-запросы заполнения 321
Пример пакета 323
Различия в сопоставлении принудительной отправкой и заполнением 325
Файлы журнала общего адаптера 325
Адаптеры, использующие Generic Adapter Framework 326
Справочные материалы по XML-схеме общего адаптера 326

Часть II: Использование API-интерфейсов 327
Глава 9: Введение в API-интерфейсы 328
Обзор API-интерфейсов 328
Глава 10: API-интерфейс HP Universal CMDB 329
Обозначения 329
Использование API-интерфейса HP Universal CMDB 329

Общая структура приложения ... 330
Копирование JAR-файла API-интерфейса в каталог Classpath ... 333
Создание пользователя интеграции ... 333
API-интерфейс UCMDB — сценарии использования ... 335
Примеры ... 336
Глава 11: API-интерфейс веб-служб HP Universal CMDB ... 337
Правила ... 337
Обзор API-интерфейса веб-службы HP Universal CMDB ... 338
Вызов веб-службы HP Universal CMDB ... 341
Запрос в CMDB ... 341
Обновление UCMDB ... 344
Запросы к модели классов UCMDB ... 345
getClassAncestors ... 346
getAllClassesHierarchy ... 346
getCmdbClassDefinition ... 346
Запрос анализа влияния ... 347
Общие параметры UCMDB ... 347
Выходные параметры UCMDB ... 350
Методы запросов UCMDB ... 351
executeTopologyQueryByNameWithParameters ... 351
executeTopologyQueryWithParameters ... 352
getChangedCIs ... 353
getCINeighbours ... 354
getCIsByID ... 354
getCIsByType ... 355
getFilteredCIsByType ... 356
getQueryNameOfView ... 359
getTopologyQueryExistingResultByName ... 359
getTopologyQueryResultCountByName ... 360
pullTopologyMapChunks ... 360
releaseChunks ... 362
Методы обновления UCMDB ... 362
addCIsAndRelations ... 363
addCustomer ... 364
deleteCIsAndRelations ... 364
removeCustomer ... 364
updateCIsAndRelations ... 364
Методы анализа влияния в UCMDB ... 365
calculateImpact ... 365
getImpactPath ... 366
getImpactRulesByNamePrefix ... 367
API-интерфейс веб-службы фактического состояния. ... 367
API-интерфейс веб-службы UCMDB — сценарии использования ... 369
Примеры ... 370
Глава 12: Управление потоком данных — API Java ... 371
Управление потоком данных — использование API Java ... 371

Глава 13: API веб-службы управления потоком данных 373
API веб-службы управления потоком данных — обзор 373
Обозначения 374
Вызов веб-службы HP Data Flow Management 374
Структуры данных и методы управления потоком данных 374
Структуры данных 375
Методы управления заданиями обнаружения 375
Управление методами триггеров 377
Методы обработки данных зонда и домена 379
Методы учетных данных 381
Методы обновления данных 383
Пример кода 385
Пример добавления учетных данных 388

Отправить отзыв о документации 391

Часть I: Создание адаптеров обнаружения и интеграции

Глава 1: Разработка и написание адаптеров

Данная глава включает:

- Обзор разработки и написания адаптеров 12
- Создание содержимого 12
- Разработка содержимого интеграции 20
- Разработка содержимого обнаружения 23
- Внедрение адаптера обнаружения 25
- Шаг 1: Создание адаптера 28
- Шаг 2: Назначение задания адаптеру 34
- Шаг 3: Создание кода Jython 35
- Настройка удаленного выполнения процессов 35

Обзор разработки и написания адаптеров

Перед началом фактического планирования разработки новых адаптеров важно понять процессы и принципы взаимодействия, которые обычно связаны с такой разработкой.

Следующие разделы помогут администраторам понять действия, которые необходимо выполнить для успешного администрирования и выполнения проекта по разработке адаптера обнаружения.

В этой главе:

- Предполагается, что читатели умеют работать с HP Universal CMDB и обладают базовыми знаниями элементов системы. Настоящий документ призван помочь в процессе обучения и не содержит исчерпывающих инструкций.
- Приводится описание этапов планирования, исследования и внедрения нового содержимого обнаружения HP Universal CMDB, а также рекомендации и соображения, которые следует учитывать.
- Представлены сведения об основных API-интерфейсах платформы Управления потоком данных. См. полную документацию по доступным API-интерфейсам в документе *HP Universal CMDB Data Flow Management API Reference*. (Другие неформальные API-интерфейсы существуют и используются для встроенных адаптеров, но могут меняться.)

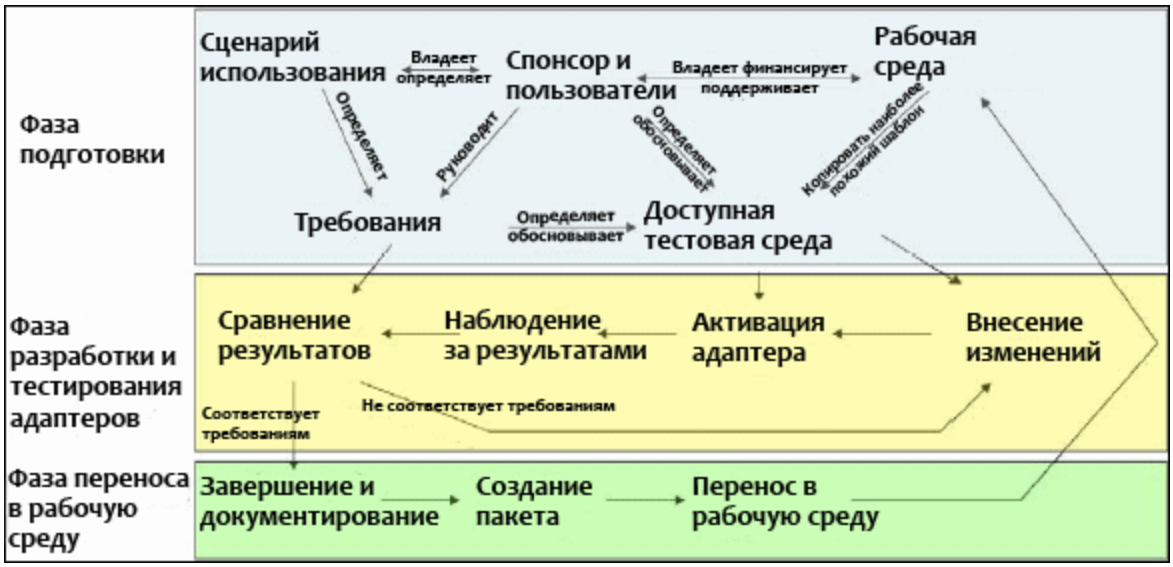
Создание содержимого

Этот раздел охватывает следующие темы:

- "Цикл разработки адаптеров" ниже
- "Управление потоком данных и интеграция" на странице 16
- "Связывание ценности для бизнеса и разработки адаптеров обнаружения" на странице 17
- "Изучение требований к интеграции" на странице 18

Цикл разработки адаптеров

На следующем рисунке приводится диаграмма процесса создания адаптера. Большая часть времени затрачивается на средний раздел, который представляет собой итеративный цикл разработки и тестирования.



Каждый этап разработки адаптера основывается на предыдущем этапе.

Когда внешний вид и работа адаптера устроят разработчика, можно приступить к его упаковке. Воспользуйтесь диспетчером пакетов UCMDB или выполните ручной экспорт компонентов, чтобы создать ZIP-файл пакета. Рекомендуется развернуть и протестировать пакет в другой системе UCMDB, прежде чем выпускать его в производственную среду. Это поможет убедиться, что все компоненты учтены и успешно упакованы. Подробные сведения об упаковке см. в разделе "Диспетчер пакетов" в документе *Руководство по администрированию HP Universal CMDB*.

В следующих разделах приводится подробное описание каждого из пунктов с наиболее важными этапами и рекомендациями:

- "Этап исследования и подготовки" на следующей странице
- "Разработка и тестирование адаптеров" на следующей странице
- "Упаковка и коммерческое внедрение адаптера " на странице 15

Этап исследования и подготовки

Этап исследования и подготовки основывается на ключевых потребностях бизнеса и сценариях использования и подразумевает резервирование ресурсов, необходимых для разработки и тестирования адаптера.

1. При планировании изменения существующего адаптера первым техническим шагом будет создание резервной копии этого адаптера и проверка возможности его возврата в исходное состояние. Если вы планируете создать новый адаптер, скопируйте наиболее близкий адаптер и сохраните его с подходящим именем. Подробнее см. в разделе "Панель "Ресурсы" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.
2. Изучите метод, используемый адаптером для сбора данных:
   - Использование внешних инструментов и протоколов для получения данных
   - Разработка методов создания ЭК на основе данных с помощью адаптера
   - Теперь вы знаете, как должен выглядеть аналогичный адаптер
3. Определите наиболее близкий адаптер, исходя из следующих характеристик:
   - Создание аналогичных ЭК
   - Использование аналогичных протоколов (SNMP)
   - Работа с аналогичными типами целевых объектов (типы ОС, версии и др.)
4. Скопируйте весь пакет.
5. Распакуйте файлы в рабочую область и переименуйте файлы адаптера (XML) и Jython (PY).

Разработка и тестирование адаптеров

Этап разработки и тестирования адаптеров подразумевает большое количество итераций. Когда адаптер начнет принимать окончательную форму, вы приступите его тестированию на соответствие окончательным сценариям использования, внесете

изменения, выполните повторное тестирование и будете повторять процесс, пока адаптера не будет соответствовать всем требованиям.

Запуск и подготовка копии

- Измените XML-части адаптера: Name (id) в строке 1, Created CI Types и Called Jython script name.
- Запустите копию, получив результаты, идентичные результатам исходного адаптера.
- Добавьте комментарии к большей части строк кода, особенно к важным командам, которые выдают результаты.

Разработка и тестирование

- Используйте другие примеры кода для разработки изменений
- Протестируйте адаптер, запустив его
- Воспользуйтесь выделенным представлением, чтобы проверить сложные результаты, и поиском для проверки простых результатов

Упаковка и коммерческое внедрение адаптера

Этап **Упаковка и коммерческое внедрение адаптера** — это последний этап разработки. Рекомендуется выполнить последний проход для устранения кода, оставшегося после отладки, очистки документов и комментариев, анализа безопасности и т. д. перед переходом к упаковке. Всегда должен быть доступен хотя бы один файл сведений с описанием внутренних принципов работы адаптера. Кому-то (возможно, это будете вы) может потребоваться анализ этого адаптера в будущем. В этом случае поможет даже ограниченная документация.

Очистка и документирование

- Удаление данных отладки
- Добавление комментариев ко всем функциям, а также вводных комментариев в главный раздел
- Создание примера TQL-запроса и его тестирование пользователем

Создание пакета

- Экспортируйте адаптеры, TQL-запросы и другие материалы с помощью диспетчера пакетов. Подробнее см. в разделе "Диспетчер пакетов" в документе *Руководство по администрированию HP Universal CMDB*.
- Проверьте зависимости пакета от других пакетов — например, являются ли ЭК, созданные другими пакетами, входными для ЭК в данном адаптере.
- Воспользуйтесь диспетчером пакетов для создания ZIP-файла пакета. Подробнее см. в разделе "Диспетчер пакетов" в документе *Руководство по администрированию HP Universal CMDB*.
- Протестируйте развертывание, удалив части нового содержимого и повторно развернув пакет или развернув его в другой тестовой системе.

Управление потоком данных и интеграция

Адаптеры DFM поддерживают интеграцию с другими продуктами. Следует учесть следующие определения:

- DFM собирает содержимое с множества целевых объектов.
- Модуль интеграции собирает содержимое разных типов из одной системы.

Обратите внимание, что в этих определениях методы сбора не различаются. Это относится и к DFM. Процесс разработки нового адаптера не отличается от разработки новой интеграции. Проводится такое же исследование, аналогичный выбор вариантов между новым и существующими адаптерами, выполняется аналогичный процесс написания адаптеров и др. Существуют следующие различия:

- Планирование окончательного адаптера. Адаптеры интеграции могут выполняться чаще, чем адаптеры обнаружения, но это зависит от сценариев использования.
- Входные ЭК:
  - Интеграция: триггер, не связанный с ЭК, для выполнения без входных данных: имя файла или источник передается с помощью параметра адаптера.
  - Обнаружение: использует обычные ЭК CMDB для ввода данных.

Для проектов по интеграции в большинстве случаев следует применять существующий адаптер. Направление интеграции (из HP Universal CMDB в другой продукт или из другого продукта в HP Universal CMDB) может повлиять на методику разработки. Существуют внешние пакеты, которые можно скопировать для решения ваших задач с использованием проверенных методов.

Из HP Universal CMDB в другой проект:

- Создайте TQL-запрос, который создает ЭК и связи для экспорта.
- Воспользуйтесь стандартным адаптером оболочки, чтобы выполнить TQL-запрос и записать результаты в XML-файл для чтения внешним продуктом.

> **Примечание.** Для получения примеров внешних пакетов обратитесь в службу Поддержка ПО HP.

При интеграции других продуктов с HP Universal CMDB, в зависимости от того, как другой продукт предоставляет доступ к своим данным, адаптер интеграции будет действовать по разному:

Тип интеграции	Пример, доступный для использования
Прямой доступ к базе данных продукта	HP ED
Чтение CSV- или XML-файла, созданного при экспорте	HP ServiceCenter
Доступ к API-интерфейсу продукта	BMC Atrium/Remedy

Связывание ценности для бизнеса и разработки адаптеров обнаружения

Сценарий использования для разработки нового содержимого обнаружения должен основываться на бизнес-обосновании и плане реализации ценности для бизнеса. Таким образом, целью сопоставления компонентов системы с ЭК и их добавления в CMDB является ценность для бизнеса.

Содержимое может и не использоваться для составления карты приложений, но это стандартный промежуточный шаг во многих сценариях использования. Независимо от конечного использования содержимого, план должен учитывать следующие вопросы:

- Кто потребитель? Как потребитель должен обрабатывать информацию, предоставленную ЭК (и связи между ними)? В каком бизнес-контексте следует рассматривать ЭК и связи между ними? Потребителем ЭК является только пользователь, только продукт или пользователь и продукт одновременно?
- После достижения идеального сочетания ЭК и связей в CMDB как они будут использоваться для реализации ценности для бизнеса?
- Как должна выглядеть идеальная карта?
  - Какой термин наиболее полно описывает связи между ЭК?
  - Какие типы добавляемых ЭК наиболее важны?
  - Каким будет конечный сценарий использования и конечный пользователь карты?
- Каким будет идеальный макет отчета?

Следующий шаг после формулировки бизнес-обоснования — документирование ценности для бизнеса. Это означает построение идеальной карты с помощью графического редактора и анализ влияния и зависимостей между ЭК и отчетами, метода отслеживания изменений и определение их важности, мониторинг, обеспечение соответствия нормативам и дополнительной ценности для бизнеса в соответствии со сценариями использования.

Этот чертеж (или модель) называется **схемой**.

Например, если для приложения важно знать, когда изменен определенный файл конфигурации, файл должен быть привязан к определенному ЭК (к которому относится) на построенной карте.

Работайте с экспертом, который является конечным пользователем разработанного содержимого. Эксперт должен указать важные объекты (ЭК с атрибутами и связями), которые должны присутствовать в CMDB для достижения целей бизнеса.

Один из возможных методов — передача анкеты владельцу приложения (который также является экспертом в нашем случае). Владелец тоже сможет указать цели, перечисленные выше, и составить схемы. Как минимум владелец должен предоставить текущую архитектуру приложения.

В схему должны быть включены только важные данные, незначительную информацию следует исключить: адаптер можно будет дополнить позднее. Целью разработки должна

быть настройка ограниченного модуля обнаружения, который работает и приносит пользу. Добавление больших объемов данных позволяет создать впечатляющие схемы, но может запутывать аудиторию и требовать длительной разработки.

После однозначного определения модели и ценности для бизнеса можно переходить к следующему этапу. К этому этапу можно вернуться после получения более точной информации на последующих этапах.

Изучение требований к интеграции

Для этого этапа необходима **схема** ЭК и связей, включающая атрибуты, которые должны быть обнаружены DFM. Дополнительные сведения см. в разделе "Обзор разработки и написания адаптеров" на странице 12.

Этот раздел охватывает следующие темы:

- "Изменение существующего адаптера" ниже
- "Создание нового адаптера" ниже
- "Исследование модели" на следующей странице
- "Исследование технологии" на следующей странице
- "Рекомендации по выбору способов доступа к данным" на следующей странице
- "Сводка" на странице 20

Изменение существующего адаптера

Существующий адаптер можно изменять, если существует встроенный адаптер или внешний адаптер, который:

- Не обнаруживает необходимые атрибуты;
- имеет определенный тип ОС, который не обнаруживается или обнаруживается некорректно;
- Не обнаруживает или не создает определенную связь.

Если существующий адаптер решает не все задачи, но некоторые из них, следует начать с оценки существующих адаптеров для выявления адаптера, который почти соответствует поставленным целям. Если такой адаптер существует, его можно изменить.

Кроме того, нужно определить, доступен ли внешний адаптер. Внешние адаптеры — это адаптеры обнаружения, которые доступны для использования, но не входят в конфигурацию продукта по умолчанию. Обратитесь в службу Поддержка ПО HP, чтобы получить актуальный список адаптеров полей.

Создание нового адаптера

Необходимость в разработке нового адаптера возникает в следующих случаях:

- Если создание нового адаптера будет быстрее, чем ручная вставка информации в CMDB (обычно от 50 до 100 ЭК и связей), или не является разовым проектом.
- Если потребности оправдывают трудозатраты.
- Если встроенные адаптеры или внешние адаптеры недоступны.

- Если результаты должны использоваться многократно.
- Когда целевая среда или ее данные доступны (нельзя обнаружить то, что недоступно).

Исследование модели

- Просмотрите модель классов UCMDB (диспетчер типов ЭК) и сопоставьте объекты и связи из **схемы** с существующими типами ЭК. Для предотвращения осложнений при обновлении версий рекомендуется следовать текущей модели. Если существующую модель необходимо расширить, создайте новые типы ЭК, поскольку обновление может привести к перезаписи встроенных типов ЭК.
- Если объекты, связи и атрибуты отсутствуют в текущей модели, их следует создать. Мы рекомендуем создать пакет с типами ЭК (который также будет содержать все данные обнаружения, представления и другие артефакты, связанные с этим пакетом), поскольку эти типы ЭК нужно будет развертывать при каждой установке HP Universal CMDB.

Исследование технологии

Убедившись, что CMDB содержит необходимые ЭК, переходите к следующему этапу — определение способа извлечения данных из соответствующих систем.

Извлечение данных обычно подразумевает использование протокола для доступа к управляющим компонентам приложения, его фактическим данным, файлам конфигурации или базам данных. Ценен любой источник данных, предоставляющий сведения о системе. Исследование технологии требует всесторонних знаний системы, а иногда и творческого подхода.

Для приложений собственной разработки может быть полезно передать форму анкеты владельцу приложения. В этой форме владелец должен перечислить все области приложения, которые могут предоставить сведения, необходимые для создания схемы и реализации ценности для бизнеса. Эта информация должна включать (без ограничений) базы данных управления, файлы конфигурации, файлы журналов, интерфейсы управления, средства администрирования, веб-службы, отправленные сообщения или события и др.

Для стандартных продуктов следует сосредоточиться на документации, форумах и службе поддержки продукта. Ищите руководства по администрированию, руководства по управлению, подключаемым модулям и интеграции и т. п. Если данные из интерфейсов управления все еще отсутствуют, ознакомьтесь с информацией о файлах конфигурации приложения, параметрах реестра, журналах сетевых событий и любых артефактах приложения, используемых для контроля над его эксплуатацией.

Рекомендации по выбору способов доступа к данным

Релевантность: Выбирайте источники или сочетания источников, предоставляющие максимальный объем данных. Если один источник предоставляет большую часть данных, но другие данные разрозненны и к ним сложно получить доступ, попробуйте оценить ценность других данных с точки зрения риска и трудозатрат на их получение. Иногда может быть целесообразно уменьшить размер схемы, если ее ценность не окупит вложенные усилия.

Многократное использование: Если HP Universal CMDB поддерживает тот или иной протокол подключения, рекомендуется использовать его. Это значит, что платформа DFM предоставит готовый клиент и конфигурацию подключения. В противном случае могут потребоваться инвестиции в разработку инфраструктуры. Поддерживаемые протоколы подключения HP Universal CMDB можно посмотреть в разделе меню **Управление потоком данных > Настройка зонда Data Flow Probe > Домены и зонды**. Подробнее о каждом протоколе см. раздел *Руководство по пакету обнаружения и интеграции HP UCMDB*.

Новые протоколы можно добавить путем добавления новых ЭК в модель. Для получения дополнительных сведений обратитесь в службу Поддержка ПО HP.

Примечание. Для доступа к данным реестра Windows можно использовать WMI или NTCmd.

Безопасность: Доступ к информации обычно требует учетных данных (имени пользователя и пароля), которые вводятся в CMDB и безопасно хранятся для всех компонентов продукта. Если это возможно и увеличение уровня безопасности не конфликтует с другими установленными принципами, выберите наиболее защищенный набор учетных данных или протокол, из вариантов, которые отвечают предъявленным требованиям. Например, если информация доступна через JMX (стандартный интерфейс администрирования, ограниченные возможности) и Telnet, лучше использовать интерфейс JMX, поскольку он предлагает встроенное ограничение доступа и, как правило, не предоставляет доступ к базовой платформе.

Удобство: Некоторые интерфейсы управления могут включать расширенные возможности. Например, может быть удобнее создавать запросы (SQL, WMI), чем вручную переходить по деревьям данных или создавать регулярные выражения для их разбора.

Разработчики: Люди, которые в конечном итоге будут разрабатывать адаптеры, могут предпочитать определенную технологию. Это следует учитывать, если две технологии предоставляют одинаковые данные по одинаковой цене, на ряду с другими факторами.

Сводка

Результат этого шага — документ, описывающий методы доступа и соответствующие данные, которые могут быть извлечены с помощью каждого метода. Кроме того, документ должен содержать сопоставление каждого источника с соответствующими данными в схеме.

Каждый метод доступа должен быть отмечен в соответствии с инструкциями выше. И наконец, следует спланировать ресурсы, которые должны быть обнаружены, и сведения, которые должны быть извлечены из каждого источника в модель схемы (к этому моменту они должны быть сопоставлены с соответствующей моделью UCMDB).

Разработка содержимого интеграции

Перед созданием нового адаптера интеграции необходимо проанализировать требования к нему:

- Должен ли адаптер интеграция копировать данные в CMDB? Должны ли данные отслеживаться в журнале? Является ли источник ненадежным?

Если вы ответили "Да" на эти вопросы, необходимо использовать **Заполнение**.

- Должен ли адаптер интеграции объединять данные в оперативном режиме для представлений и TQL-запросов? Важна ли точность изменения данных? Объем данных слишком велик для копирования в CMDB, но запрошенный объем данных обычно мал?

  Если вы ответили "Да" на эти вопросы, необходимо использовать **Объединение**.

- Должен ли адаптер интеграции принудительно отправлять данные в удаленные источники данных?

  Если вы ответили "Да", необходима **Принудительная отправка данных**.

- Длина идентификатора любого ЭК составляет больше 60 символов?

  Если ответ на этот вопрос положительный, следует **уменьшить** длину идентификаторов всех связанных ЭК, чтобы она не превышала максимальное значение в 60 символов.

Примечание. Потоки объединения и отправки данных могут быть настроены для одного адаптера интеграции для максимальной гибкости.

Подробные сведения о различных типах интеграции см. в разделе "Студия интеграции в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

Для создания адаптеров интеграции доступно несколько вариантов:

1. Адаптер Jython:
   - Классический шаблон обнаружения
   - Создается в Jython
   - Используется для наполнения

   Подробнее см. в разделе "Разработка адаптеров Jython" на странице 37.
2. Адаптер Java:
   - Адаптер, который реализует один из интерфейсов адаптеров, с помощью Federation SDK Framework.
   - Может использоваться для одного или нескольких процессов объединения, наполнения или о правки данных (в зависимости от необходимой реализации).
   - Разрабатывается в Java с нуля, что обеспечивает создание кода для соединения любого источника с целевым объектом.
   - Подходит для заданий, которые подразумевают подключение к одному источнику или целевому объекту.

   Подробнее см. в разделе "Разработка адаптеров Java" на странице 197.
3. Общий адаптер DB:
   - Абстрактный адаптер, основанный на адаптере Java и использующий платформу Federation SDK Framework.

- Обеспечивает создание адаптеров для подключения внешних репозиториев данных.
- Поддерживает объединение и заполнение (если подключаемый модуль Java реализован для поддержки изменений).
- Этот адаптер относительно удобен для настройки, так как основывается в основном на XML и файлах конфигурации свойств.
- Основная конфигурация основывается на файле **orm.xml**, который связывает классы UCMDB и столбцы базы данных.
- Подходит для заданий, которые подразумевают подключение к одному источнику данных.

Подробнее см. в разделе "Разработка общих адаптеров БД " на странице 122.

4. Общий адаптер принудительной отправки данных:
   - Абстрактный адаптер, основанный на адаптере Java (Federation SDK Framework) и адаптере Jython.
   - Обеспечивает создание адаптеров для принудительной отправки данных в удаленные целевые объекты.
   - Этот адаптер относительно прост в настройке, поскольку необходимо настроить только сопоставление между классами UCMDB и XML, а также сценарий Jython, который выполняет принудительную отправку данных в целевой объект.
   - Подходит для заданий, которые подразумевают подключение к одному целевому объекту.
   - Используется для принудительной отправки данных.

   Подробнее см. в разделе "Разработка адаптеров принудительной отправки" на странице 232.
5. Расширенный адаптер принудительной отправки:
   - Все вышеперечисленные функции общего адаптера принудительной отправки
   - Адаптер на основе корневого элемента
   - Сопоставление древовидной структуры данных UCMDB с древовидной структурой данных на целевом объекте

   Подробнее см. в разделе "Принудительная отправка данных с помощью общего адаптера" на странице 280.

В таблице ниже приводятся возможности каждого адаптера:

Поток/Адаптер	Адаптер Jython	Адаптер Java	Общий адаптер БД	Общий адаптер принудительной отправки	Расширенный адаптер принудительной

					отправки
Заполнение	✓	✓	✓	×	×
Объединение	×	✓	✓	×	×
Принудительная отправка данных	×	✓	×	✓	✓

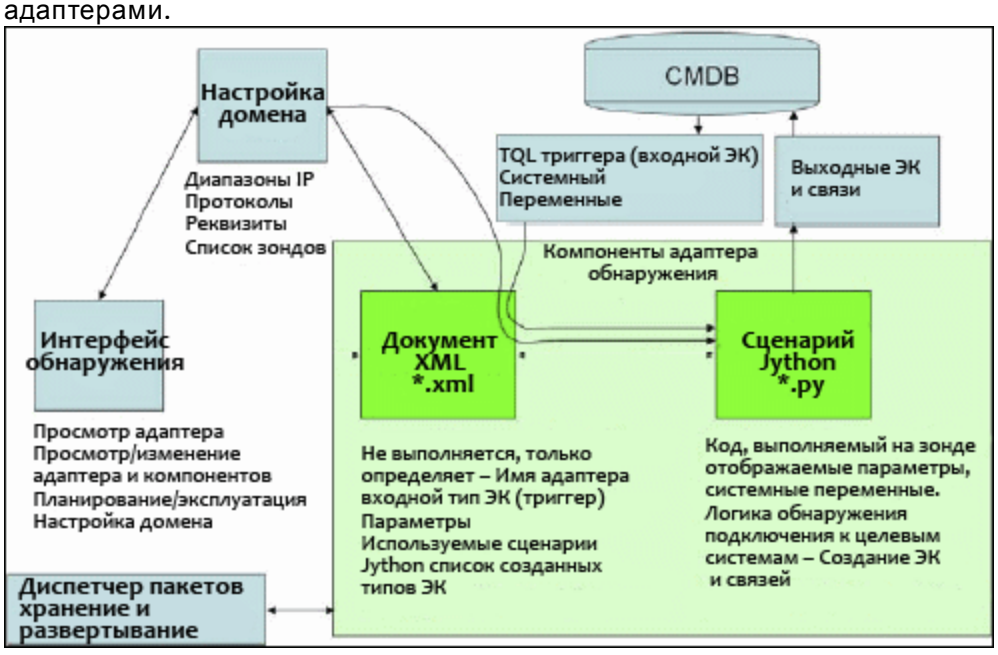
Разработка содержимого обнаружения

Этот раздел охватывает следующие темы:

- "Адаптеры обнаружения и связанные компоненты " ниже
- "Разделение адаптеров" на следующей странице

Адаптеры обнаружения и связанные компоненты

В следующей схеме приводятся компоненты адаптера и компоненты, с которыми они взаимодействуют при обнаружении. Компоненты, отмеченные зеленым цветом, — это фактические адаптеры, а компоненты, отмеченные синим, взаимодействуют с адаптерами.



Обратите внимание, что адаптер представляет собой два файла: XML-документ и сценарий Jython. Платформа обнаружения, включая входные ЭК, учетные данные и пользовательские библиотеки, предоставляются адаптеру во время выполнения. Оба компонента адаптера обнаружения администрируются через Управление потоком данных. В штатном режиме они хранятся в самой базе CMDB. Внешний пакет остается в

системе, но не используется при эксплуатации. Диспетчер пакетов обеспечивает сохранение нового содержимого обнаружения и интеграции.

Входные ЭК для адаптера предоставляются TQL-запросам и доступны сценарию адаптера в системных переменных. Параметры адаптеров также предоставляются в качестве данных целевого объекта, поэтому адаптер можно настроить в соответствии с его функцией.

Приложение DFM используется для создания и тестирования новых адаптеров. При этом используются страницы Universal Discovery, «Управление адаптерами» и «Настройка зонда Data Flow Probe».

Адаптеры хранятся и передаются в виде пакетов. Диспетчер пакетов и консоль JMX используются для создания пакетов из только что созданных адаптеров и развертывания адаптеров на новых системах.

Разделение адаптеров

Все процессы обнаружения можно определить в одном адаптере. Но требования к качественному проектированию требуют разделения сложных систем на более простые и управляемые компоненты.

Ниже приводятся правила и рекомендации по процессу разделения адаптеров:

- Обнаружение должно выполняться поэтапно. Каждый этап должен быть представлен адаптером, который составляет схему области или уровня системы. Адаптеры должны использовать предыдущий уровень или этап для следующего этапа обнаружения системы. Например, адаптер A вызывается результатом TQL-запроса сервера приложений и сопоставляет уровень сервера приложений. В рамках этого процесса выполняется сопоставление JDBC-подключения. Адаптер B регистрирует компонент JDBC-подключения в качестве триггера TQL и использует результат адаптера A для доступа к уровню базы данных (например, с помощью атрибута URL JDBC) и составляет схему этого уровня.
- **Двухэтапная парадигма подключения**: Большинство систем требуют учетных данных для доступа к данным. Это значит, что сочетание имени пользователя и пароля должно быть применено к этим системам. Администратор DFM вводит учетные данные в системе, используя безопасный метод, и может указать несколько наборов данных с различным уровнем приоритета. Это называется **словарем протоколов**. Если система недоступна (по той или иной причине), выполнять дальнейшее обнаружение не следует. Если подключение выполнено успешно, необходим способ указать, какой набор учетных данных был применен успешно для дальнейшего процесса обнаружения.

  Эти два этапа обеспечивают разделение двух адаптеров в следующих случаях:

  - **Адаптер подключения**: Это адаптер, который принимает первоначальный триггер и определяет наличие удаленного агента на этом триггере. Для этого применяется перебор всех значений в словаре протоколов, соответствующих типу агента. В случае успеха операции адаптер передает ЭК удаленного агента в качестве результата (SNMP, WMI и др.), который также указывает на правильную запись в словаре протоколов для будущих подключений. Затем этот ЭК агента становится частью триггера адаптера содержимого.

  - **Адаптер содержимого**: Предварительное условие этого адаптера — успешное подключение предыдущего адаптера (предварительные условия определяются TQL-запросами). Адаптерам этих типов больше не нужно перебирать весь словарь протоколов, поскольку они могут получить правильные учетные данные из ЭК удаленного агента и воспользоваться ими для входа в обнаруженную систему.

- Различные особенности планирования также могут повлиять на разделение обнаружения. Например, система может опрашиваться только в нерабочее время. Поэтому, хотя было бы целесообразно объединить адаптер с таким же адаптером, предназначенным для обнаружения другой системы, различие в расписании потребует создания двух адаптеров.
- Обнаружение различных интерфейсов управления или технологий в одной системе должно проводиться с помощью раздельных адаптеров. Это позволяет активировать соответствующий метод доступа для каждой системы или организации. Например, некоторые организации используют доступ к компьютерам через WMI, но агенты SNMP не установлены на этих компьютерах.

Внедрение адаптера обнаружения

Задача DFM заключается в получении доступа к удаленным (или локальным) системам, моделировании извлеченных данных как ЭК и сохранения ЭК в CMDB. Задача включает следующие шаги:

1. **Создание адаптера.**

   Настройка файла адаптера, содержащего контекст, параметры и типы результатов путем выбора сценариев, которые будут входить в адаптер. Дополнительные сведения см. в разделе "Шаг 1: Создание адаптера" на странице 28.

2. **Создание задания обнаружения.**

   Настройка задания с данными планирования и запроса-триггера. Дополнительные сведения см. в разделе "Шаг 2: Назначение задания адаптеру" на странице 34.

3. **Изменение кода обнаружения.**

   Изменение кода Jython или Java, который содержится в файлах адаптера и ссылается на платформу DFM. Дополнительные сведения см. в разделе "Шаг 3: Создание кода Jython" на странице 35.

Для создания новых адаптеров создаются все компоненты, описанные выше, и каждый из них автоматически привязывается к компонентам в предыдущем шаге. Например, после создания задания и выбора соответствующего адаптера файл адаптера привязывается к заданию.

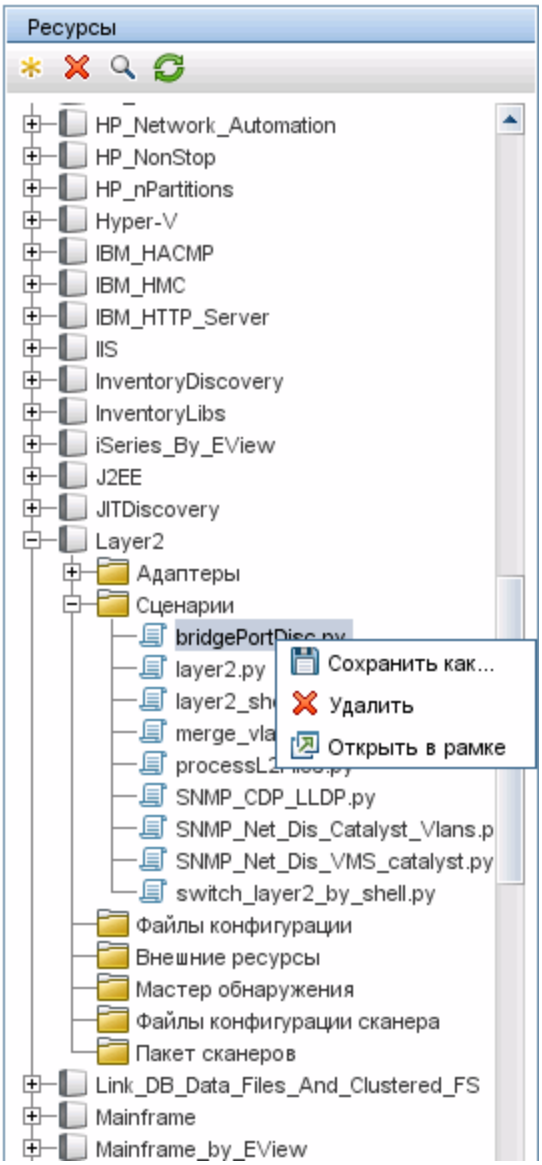
Код адаптера

Фактическая реализация подключения к удаленной системе, запроса ее данных и их сопоставление с данными CMDB основывается на коде Jython. Например, код содержит логику подключения к базе данных и извлечения данных из нее. В этом случае код ожидает URL-адрес JDBC, имя пользователя, пароль и т. д. Эти параметры относятся к каждому экземпляру базы данных, который отвечает на TQL-запрос. Эти переменные настраиваются в адаптере (в данных ЭК-триггера) и при выполнении задания эти данные

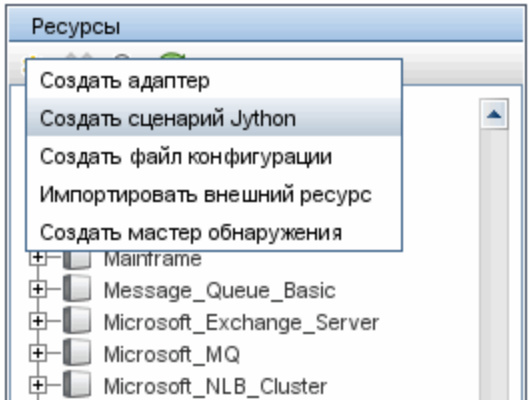
передаются в код для выполнения.

Адаптер может обращаться к этому коду по имени класса Java или имени сценария Jython. В этом разделе мы рассмотрим создание кода DFM в виде сценариев Jython.

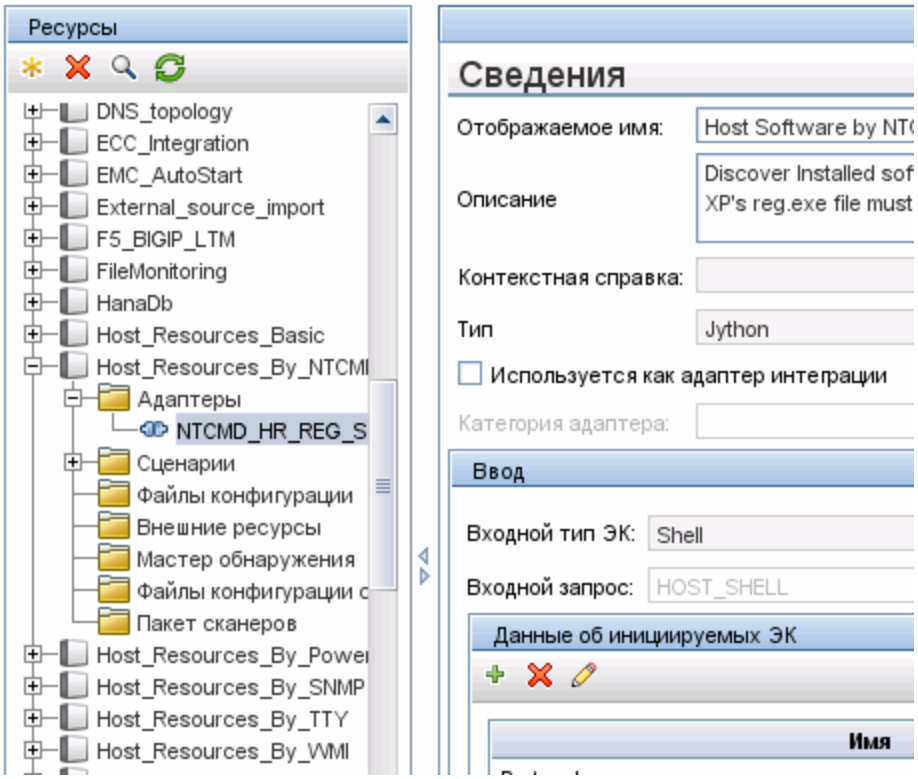
Адаптер может содержать список сценариев, которые будут использоваться при выполнении обнаружения. При создании адаптера обычно создается новый сценарий, который назначается адаптеру. Новый сценарий включает базовые шаблоны, но вы можете использовать один из прочих сценариев в качестве шаблона, щелкнув его правой кнопкой мыши и выбрав **Сохранить как**:



См. дополнительные сведения о создании новых сценариев Jython в "Шаг 3: Создание кода Jython" на странице 35. Сценарии добавляются с помощью панели «Ресурсы»:



Сценарии в списке выполняются последовательно в порядке, указанном в адаптере:



Примечание. Сценарий должен быть указан, даже если используется исключительно как библиотека для другого сценария. В этом случае сценарий библиотеки должен быть указан перед его использованием в другом сценарии. В этом примере сценарий `processdbutils.py` — это библиотека, используемая последним сценарием `host_processes.py`. Библиотеки отличаются от обычных выполняемых сценариев отсутствием функции `DiscoveryMain()`.

Шаг 1: Создание адаптера

Адаптер может считаться определением функции. Эта функция задает определение входных параметров, выполняет логику ввода, определяет вывод и предоставляет результат.

Каждый адаптер определяет входные и выходные данные: Входные и выходные данные представляют собой ЭК триггера, явно заданные для адаптера. Адаптер извлекает данные из входного ЭК-триггера и передает их в код в виде параметров. Время от времени данные из связанных ЭК также передаются в код. Подробнее см. в разделе "Окно "Связанные ЭК" в документе *Руководство по управлению потоками данных в HP Universal CMDB*. Код адаптера является стандартным за исключением этих входных параметров ЭК-триггера, которые передаются в код.

Подробные сведения о входных компонентах см. в разделе "Принципы управления потоком данных" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

Этот раздел охватывает следующие темы:

- "Определение входных данных адаптера (тип ЭК-триггера и входной запрос)" ниже
- "Определение выходных данных адаптера" на странице 30
- "Переопределение параметров адаптера" на странице 32
- " Переопределение выбора зонда - необязательно" на странице 32
- "Настройте каталог classpath для удаленного процесса (необязательно)." на странице 34

1. ## Определение входных данных адаптера (тип ЭК-триггера и входной запрос)

   Компоненты «Тип ЭК-триггера» и «Входной запрос» используются для настройки определенных ЭК в качестве входных данных адаптера:

   - Тип ЭК-триггера определяет тип ЭК, который используется в качестве входных данных адаптера. Например, для адаптера, обнаруживающего IP-адреса, входным типом ЭК будет Network.

   - Входной запрос — это обычный редактируемый запрос к CMDB. Входной тип запроса определяет дополнительные ограничения типа ЭК (например, если задача требует атрибут `hostID` или `application_ip`) и может определять дополнительные данные ЭК, если это нужно адаптеру.

     Если адаптер требует дополнительных данных от ЭК, связанных с ЭК-триггером, можно добавить дополнительные узлы во входной TQL-запрос. См. дополнительные сведения в разделе "Добавление узлов запросов и связей в TQL-запрос" (*Руководство по моделированию в HP Universal CMDB*).

   - ЭК триггера содержит все необходимые данные об ЭК триггера, а также информацию из других узлов входного TQL-запроса, если они определены. DFM использует переменные для извлечения данных из этих ЭК. После загрузки задачи в зонд переменные данных ЭК-триггера заменяются фактическими

значениями из атрибутов реальных экземпляров ЭК.

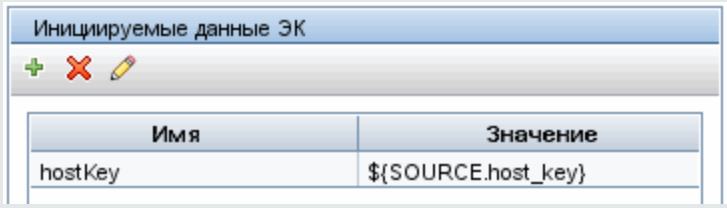
- Если в качестве значения целевых данных представлен список, можно указать число элементов из списка, отправляемых на зонд. Для этого добавьте двоеточие после значения по умолчанию и укажите число элементов. Если значение по умолчанию отсутствует, добавьте два двоеточия.

  Например, если указаны следующие целевые данные: `name=portId, value= ${PHYSICALPORT.root_id:NA:1}` or `name=portId, value= ${PHYSICALPORT.root_id::1}`, на зонд отправляется только первый порт из списка.

Пример замены фактических переменных на фактические данные:

В этом примере переменные заменяют данные ЭК **IpAddress** фактическими значениями в реальных экземплярах ЭК **IpAddress** в системе.

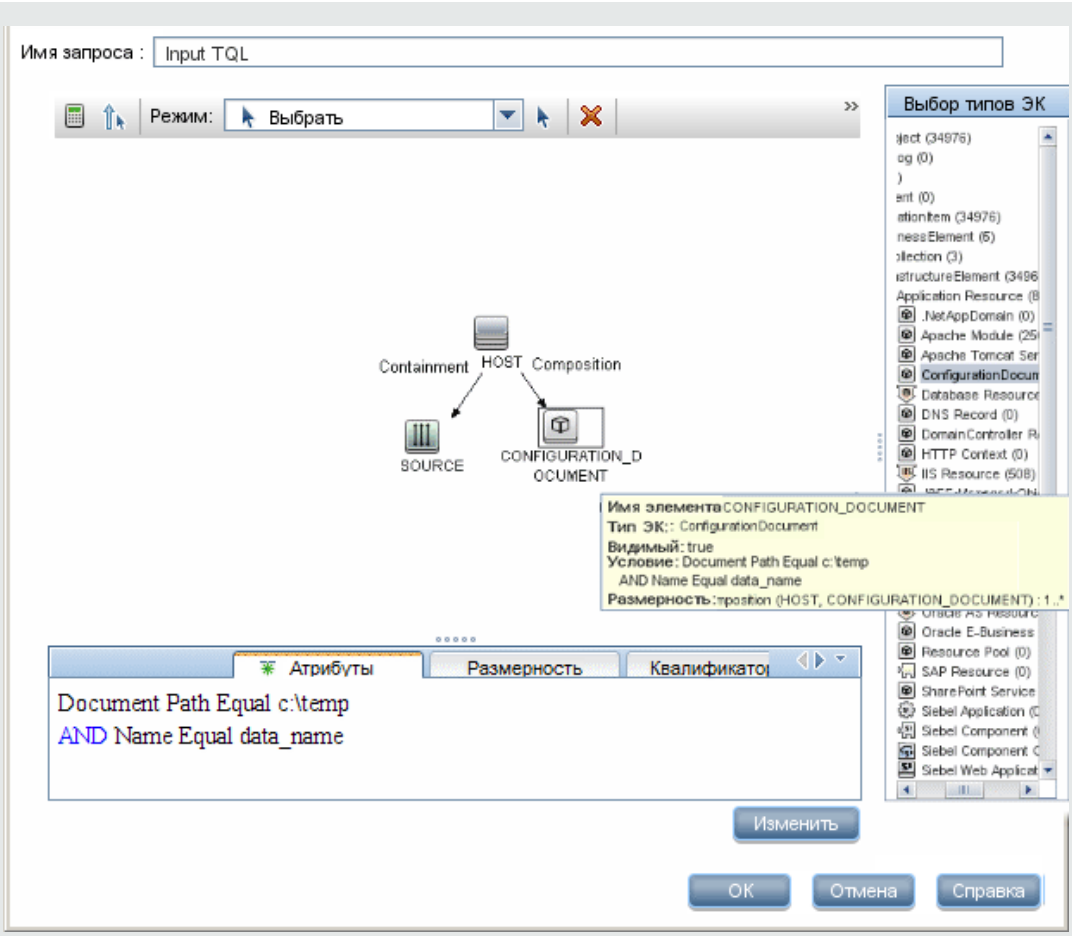
Инициируемые данные ЭК **IpAddress** включают переменную `fileName`. Эта переменная обеспечивает замену узла **CONFIGURATION_DOCUMENT** во входном TQL-запросе фактическими значениями файла конфигурации на узле:



Инициируемые данные ЭК передаются зонду, причем все переменные заменяются фактическими значениями. Сценарий адаптера содержит команду, которая позволяет извлечь фактические значения указанных переменных с помощью DFM Framework:

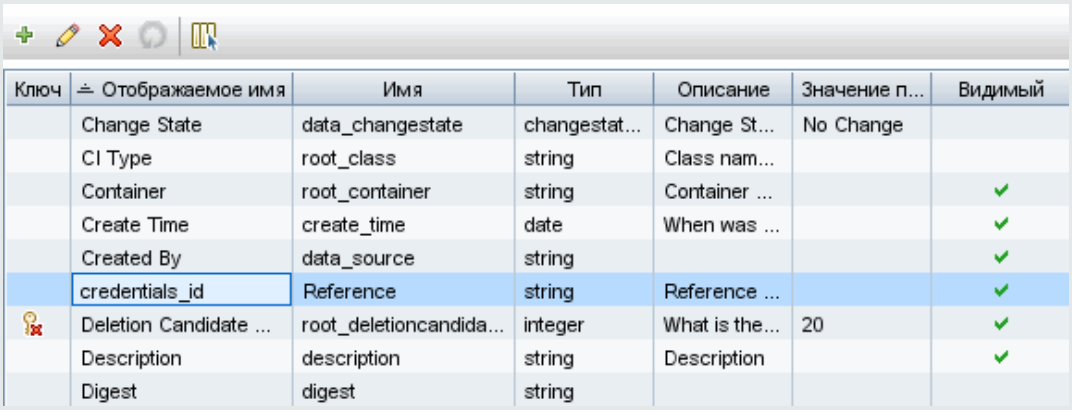
```
Framework.getTriggerCIData ('ip_address')
```

Переменные `fileName` и `path` используют атрибуты `data_name` и `document_path` из узла **CONFIGURATION_DOCUMENT** (созданного в Редакторе входных запросов, см. предыдущий пример).



Нажмите на эскиз для просмотра изображения в полном размере.

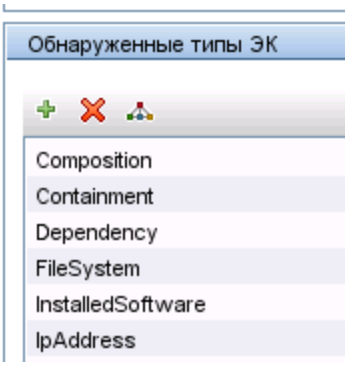
Переменные `Protocol`, `credentialsId` и `ip_address` используют атрибуты `root_class`, `credentials_id` и `application_ip`:



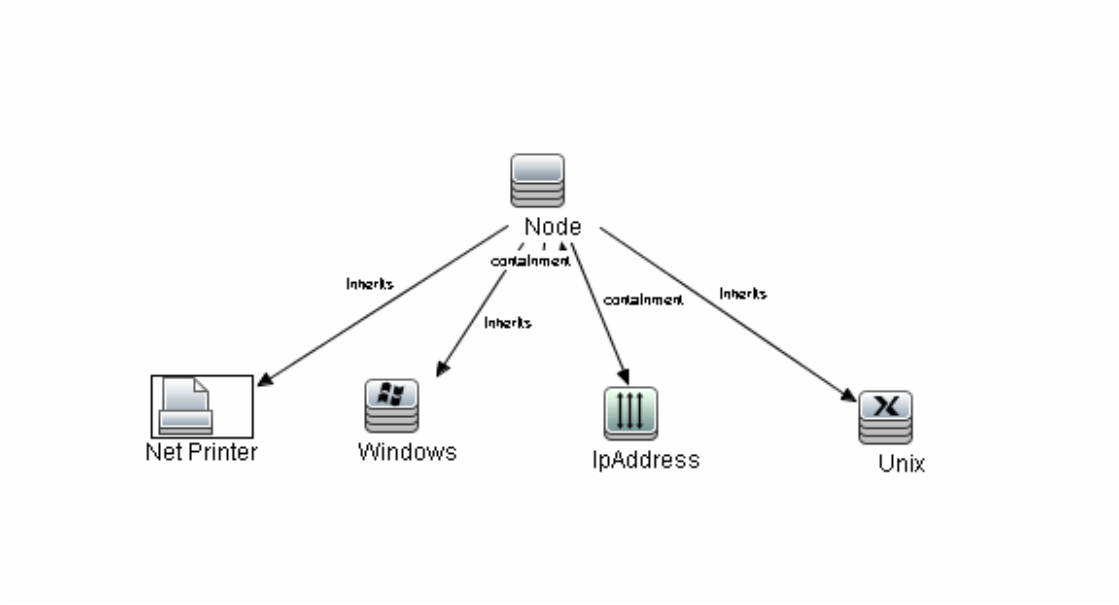
Ключ	Отображаемое имя	Имя	Тип	Описание	Значение п...	Видимый
	Change State	data_changestate	changestat...	Change St...	No Change	
	CI Type	root_class	string	Class nam...		
	Container	root_container	string	Container ...		✔
	Create Time	create_time	date	When was ...		✔
	Created By	data_source	string			✔
	credentials_id	Reference	string	Reference ...		✔
	Deletion Candidate ...	root_deletioncandida...	integer	What is the...	20	✔
	Description	description	string	Description		✔
	Digest	digest	string			

2. Определение выходных данных адаптера

   Выходные данные адаптера представляют собой список обнаруженных ЭК (**Управление потоком данных > Управление адаптерами > Определение адаптера > Обнаруженные типы ЭК**) и связей между ними:



Кроме того, типы ЭК можно просмотреть в виде топологической схемы, на которой изображены компоненты и связи между ними (нажмите кнопку **Просмотреть обнаруженные типы ЭК в виде карты**):



Обнаруженные ЭК возвращаются кодом DFM (сценарием Jython) в в формате UCMDB `ObjectStateHolderVector`. Подробнее см. в разделе "Формирование результатов сценарием Jython" на странице 42.

Пример выходных данных адаптера:

В этом примере будут настроены типы ЭК, входящие в выходные данные ЭК IP.

a. Откройте раздел **Управление потоком данных > Управление адаптерами**.
b. На панели ресурсов выберите **Сеть > Адаптеры > NSLOOKUP_on_Probe**.
c. Найдите панель «Обнаруженные типы ЭК» на вкладке «Определение адаптера».
d. Здесь перечислены типы ЭК, которые должны быть включены в выходные данные адаптеры. Добавьте типы ЭК в список или удалите их. Подробнее см.

в разделе "Вкладка "Определение адаптера" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

3. Переопределение параметров адаптера

Чтобы настроить адаптер для выполнения нескольких заданий, можно переопределить параметры адаптера. Например, адаптер `SQL_NET_Dis_Connection` используется для заданий `MSSQL Connection by SQL` и `Oracle Connection by SQL`.

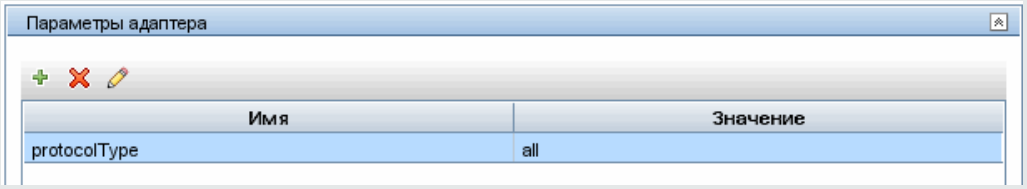
Пример переопределения параметров адаптера:

В этом примере описывается переопределение параметров адаптера, которое позволит использовать этот адаптер для обнаружения баз данных Microsoft SQL Server и Oracle.

a. Откройте раздел **Управление потоком данных > Управление адаптерами**.

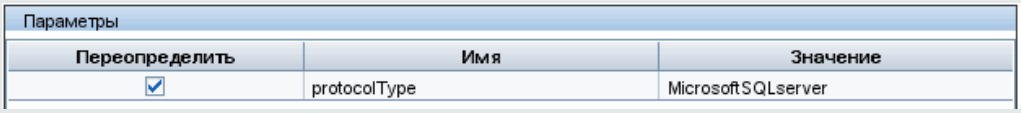
b. На панели ресурсов выберите **Database_Basic > Адаптеры > SQL_NET_Dis_Connection**.

c. Во вкладке «Определение адаптера» найдите панель **Параметры адаптера**. Параметр `protocolType` имеет значение **all**:



d. Щелкните адаптер **SQL_NET_Dis_Connection_MsSql** правой кнопкой мыши и выберите **Перейти к заданию обнаружения > MSSQL Connection by SQL**.

e. Откройте вкладку **Свойства**. Найдите панель «Параметры»:



Значение `all` заменено на значение `MicrosoftSQLServer`.

Примечание: Задание **Oracle Connection by SQL** включает этот параметр, но его значение заменено на Oracle.

Подробные сведения о добавлении, удалении и изменении параметров на панели "Параметры адаптера" см. в разделе "Вкладка "Определение адаптера" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

DFM начнет поиск экземпляров Microsoft SQL Server в соответствии с этим параметром.

4. Переопределение выбора зонда - необязательно

На сервере UCMDB предусмотрен механизм диспетчеризации, принимающий ЭК-триггеры, полученные UCMDB, и автоматически выбирающий зонд, который будет выполнять задание для каждого ЭК-триггера, согласно одному из следующих

вариантов.

- **Для типа ЭК IpAddress:** Использовать зонд, указанный для данного IP-адреса.
- **Для ЭК типа running software:** Использовать атрибуты **application_ip** и **application_ip_domain** и выбрать зонд, указанный для IP-адреса в соответствующем домене.
- **Для других типов ЭК:** Использовать IP-адрес узла, связанного с ЭК (если он есть).

Автоматический выбор зонда выполняется для узла, связанного с ЭК. После получения связанного ЭК узла механизм диспетчеризации выбирает один из IP-адресов узла, а затем выбирает зонд в соответствии с определениями охвата сети зонда.

В следующих случаях необходимо указать зонд вручную, а не использовать механизм автоматической диспетчеризации:

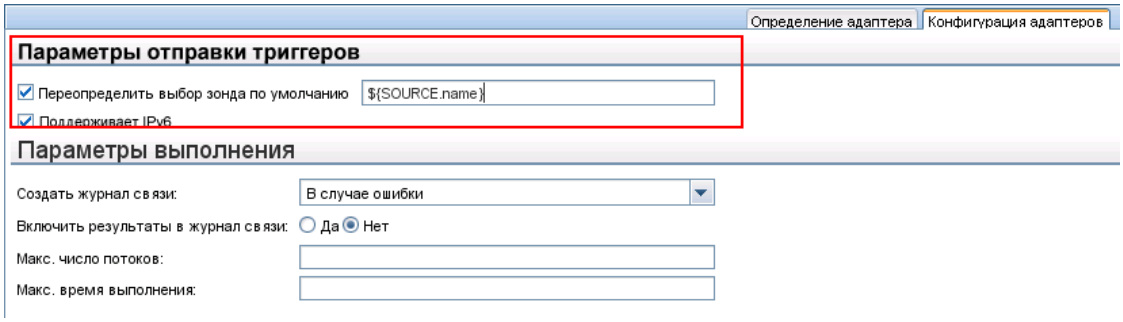
- Уже известно, какой зонд необходимо запустить для данного адаптера, и автоматический выбор зонда нежелателен (например, если ЭК-триггер является шлюзом зонда).
- Автоматический выбор зонда может завершиться неудачно. Это может произойти в следующих ситуациях:
  - У ЭК-триггера отсутствует связанный узел (такой, как тип ЭК network).
  - У узла ЭК-триггера имеется несколько IP-адресов, принадлежащих различным зондам.

Для разрешения этих проблем можно указать, какой зонд следует использовать с адаптером, следующим образом:

- Выберите адаптер и щелкните вкладку **Управление адаптерами**.
- В разделе **Параметры отправки триггеров** выберите параметр **Переопределить выбор зонда по умолчанию**.
- В поле укажите зонд в одном из следующих форматов:

Имя зонда	Имя зонда
IP-адрес	IP-адрес зонда можно задать в формате IPv4 или IPv6
IP-адрес, домен	Формат IPv4: **16.59.63.86,DefaultDomain** Формат IPv6: **2001:0:9d46:953c:34a9:1e6b:f2ff:fffe,CustomDomain**
Имя домена	Домен, в котором необходимо выбрать зонд.

Пример.



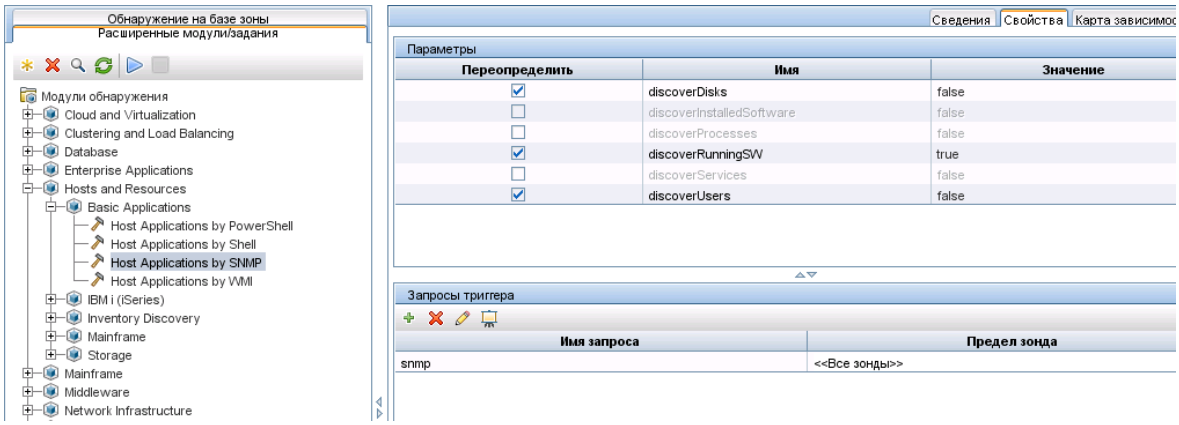
5. Настройте каталог classpath для удаленного процесса (необязательно).

   Подробнее см. в разделе "Настройка удаленного выполнения процессов" на следующей странице.

Шаг 2: Назначение задания адаптеру

С каждым адаптером связано одно или несколько заданий, определяющих политику выполнения. Задания обеспечивают планирование одного адаптера для различных наборов инициируемых ЭК и предоставление различных параметров для каждого набора.

Задания отображаются в дереве модулей обнаружения и представляют собой объект, который активирует пользователь, как показано на рисунке ниже.



Выбор TQL-запроса триггера

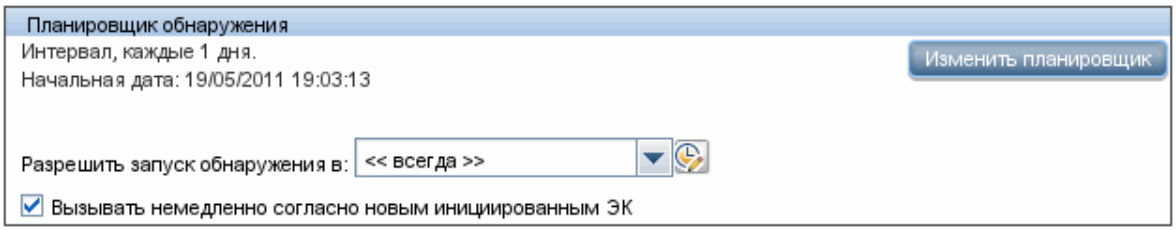
Каждое задание связывается с TQL-запросом триггера. Эти TQL-запросы триггера публикуют результаты, которые используются как входные ЭК-триггеры для этого задания.

TQL-запрос триггера может добавлять ограничения к TQL-запросу ввода. Например, если результаты TQL-запроса ввода представляют собой IP-адреса, связанные с SNMP, результатами TQL-запроса триггера могут быть IP-адреса, подключенные к SNMP, в диапазоне 195.0.0.0-195.0.0.10.

Примечание. TQL-запрос триггера должен ссылаться на те же объекты, на которые ссылается входной TQL-запрос. Например, если TQL-запрос ввода применяется к IP-адресам с использованием SNMP, вы не можете указать TQL-запрос триггера (для того же задания) для применения к IP-адресам, подключенным к хосту, поскольку некоторые из этих IP-адресов могут быть не подключены к SNMP-объекту, который требуется для входного TQL-запроса.

Ввод сведений о расписании

Сведения о планировании для зонда определяют время выполнения кода для ЭК-триггера. Если флажок **Вызывать немедленно согласно новым инициированным ЭК** установлен, код будет выполняться один раз для каждого ЭК-триггера при соединении с зондом, независимо от будущих параметров планирования.



Для каждого запланированного выполнения задания зонд выполняет код для всех ЭК-триггеров, накопленных для задания. Подробнее см. в разделе Discovery Scheduler Dialog Box в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

Переопределение параметров адаптера

При настройке задания можно переопределить параметры адаптера. Дополнительные сведения см. в разделе "Переопределение параметров адаптера" на странице 32.

Шаг 3: Создание кода Jython

HP Universal CMDB использует сценарии Jython для создания адаптеров. Например, сценарий `SNMP_Connection.py` используется адаптером `SNMP_NET_Dis_Connection` для подключения к компьютерам через SNMP. Jython — это язык, основанный на Python и использующий технологии Java.

См. дополнительные сведения о работе с Jython на следующих веб-сайтах:

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

Дополнительные сведения см. в разделе "Создание кода Jython" на странице 37.

Настройка удаленного выполнения процессов

Обнаружение может выполняться в отдельном процессе от зонда потока данных.

Например, задание можно запустить в отдельном удаленном процессе, если оно использует библиотеки **.jar**, версия которых отличается от версии библиотек зонда и несовместима с ними.

Кроме того, отдельный удаленный процесс будет полезен в случае, если задание потенциально может потреблять слишком много памяти (возвращает много данных), и необходимо изолировать зонд от возможной **нехватки памяти**.

Чтобы запускать задание в виде удаленного процесса, задайте следующие параметры в файле конфигурации соответствующего адаптера:

Параметр	Описание
remoteJVMArgs	Параметры JVM для удаленного процесса Java.
runInSeparateProcess	Если установлено значение **true**, задание обнаружения выполняется в отдельном процессе.

remoteJVMClasspath	(Необязательно) Позволяет изменять каталог classpath для удаленного процесса, переопределяя classpath по умолчанию, установленный для зонда. Это полезно в ситуациях, когда вероятна несовместимость версий jar-библиотек зонда и библиотек, необходимых для настроенного пользователем обнаружения.
	
	Если параметр **remoteJVMClasspath** не задан, используется classpath по умолчанию, указанный для зонда.
	
	Если при разработке задания обнаружения необходимо предотвратить конфликт между jar-библиотеками зонда и задания, следует как минимум использовать минимальный classpath, необходимый для простейшего обнаружения. Минимальный classpath задан в файле **DataFlowProbe.properties** в параметре **basic_discovery_minimal_classpath**.
	
	Примеры настройки **remoteJVMClasspath**:
	• Для добавления пользовательских jar-библиотек в начало или конец classpath зонда (в начало или конец) измените параметр **remoteJVMClasspath** следующим образом:
	`custom1.jar;%classpath%;custom2.jar -`
	В данном случае **custom1.jar** помещается перед каталогом classpath зонда, а **custom2.jar** — после него.
	• Чтобы использовать минимальный classpath, измените параметр **remoteJVMClasspath** следующим образом:
	`custom1.jar;%minimal_classpath%;custom2.jar`

Глава 2: Разработка адаптеров Jython

Данная глава включает:

- Справка по API-интерфейсу Управления потоком данных 37
- Создание кода Jython 37
- Поддержка локализации в адаптерах Jython 52
- Запись кода DFM 59
- Средства и библиотеки Jython 61

Справка по API-интерфейсу Управления потоком данных

См. полную документацию по доступным API-интерфейсам в документе *HP Universal CMDB Data Flow Management API Reference*. Эти файлы находятся по следующему пути:

<директория установки UCMDB>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\DDM_JavaDoc\index.html

Создание кода Jython

HP Universal CMDB использует сценарии Jython для создания адаптеров. Например, сценарий **SNMP_Connection.py** используется адаптером **SNMP_NET_Dis_Connection** для подключения к компьютерам через SNMP. Jython — это язык, основанный на Python и использующий технологии Java.

См. дополнительные сведения о работе с Jython на следующих веб-сайтах:

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

В следующем разделе описывается фактический процесс создания кода Jython с помощью DFM Framework. В разделе описываются точки взаимодействия между сценарием Jython и компонентом Framework, который он вызывает, а также библиотеки и средства Jython, которые следует использовать, когда это возможно.

Примечание.

- Сценарии, созданные для Universal Discovery, должны быть совместимы с Jython версии 2.5.3.
- Полную документацию по доступным API-интерфейсам см. в документе *HP Universal CMDB Data Flow Management API Reference*.

Этот раздел охватывает следующие темы:

- "Использование внешних JAR-файлов Java в Jython" ниже
- "Выполнение кода" ниже
- "Изменение встроенных сценариев" ниже
- "Структура файла Jython" на следующей странице
- "Формирование результатов сценарием Jython" на странице 42
- "Экземпляр Framework" на странице 44
- "Поиск правильных учетных данных (для адаптеров подключения)" на странице 48
- "Обработка исключений Java" на странице 50

Использование внешних JAR-файлов Java в Jython

При разработке новых сценариев Jython могут потребоваться внешние библиотеки Java (JAR-файлы) или сторонние исполняемые файлы, например, архивы средств Java, архивы подключения (такие как JAR-файлы драйверов JDBC) или исполняемые файлы (например, для обнаружения без учетных данных используется **nmap.exe**).

Эти ресурсы должны быть упакованы в пакет в папке **External Resources**. Любой ресурс в этой папке автоматически отправляется любому зонду, подключающемуся к серверу HP Universal CMDB.

Кроме того, при запуске обнаружения все ресурсы JAR-файлов загружаются в каталог classpath Jython, что делает все классы в ресурсе доступными для импорта и использования.

Выполнение кода

После активации задания выполняется передача на зонд задач со всеми необходимыми данными.

Зонд начинает выполнять код DFM, используя данные из задания.

Выполнение потока кода Jython начинается с главной записи сценария, затем запускается код для обнаружения ЭК и возвращаются результаты в виде вектора обнаруженных ЭК.

Изменение встроенных сценариев

Изменения во встроенных сценариях должны быть минимальными, все необходимые методы должны быть помещены во внешний сценарий. Таким образом можно более эффективно отслеживать изменения, и код не будет перезаписан при установке новой версии HP Universal CMDB.

Например, следующая строка кода во встроенном сценарии вызывает метод, который вычисляет имя веб-сервера в соответствии с приложением:

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

Ниже представлена более сложная логика, которая определяет способ расчета имени, содержащегося во внешнем сценарии:

```
# implement customer specific processing for 'servername' attribute of httpplugin
#
def PlugInProcessing(servername, transportHN, mam_utils_handle):
        # support application-specific HTTP plug-in naming
        if servername == "appsrv_instance":
                # servername is supposed to match up with the j2ee server name,
however some groups do strange things with their
                # iPlanet plug-in files. this is the best work-around we could
find. this join can't be done with IP address:port
                # because multiple apps on a web server share the same IP:port for
multiple websphere applications
                logger.debug('httpcontext_webapplicationserver attribute has been
changed from [' + servername + '] to [' + transportHN[:5] + '] to facilitate
websphere enrichment')
                servername = transportHN[:5]
        return servername
```

Сохраните внешний сценарий в папке External Resources. Подробнее см. в разделе "Панель "Ресурсы" в документе *Руководство по управлению потоками данных в HP Universal CMDB*. После добавления сценария в пакет его также можно будет использовать для других заданий. Подробные сведения об использовании диспетчера пакетов см. в разделе "Диспетчер пакетов" в документе *Руководство по администрированию HP Universal CMDB*.

Во время обновления изменения, внесенные в строку кода, перезаписываются новой версией готового сценария. Таким образом, вы должны будете заменить строку. Однако внешний сценарий перезаписан не будет.

Структура файла Jython

Файл Jython состоит из трех частей, которые следуют в определенном порядке:

1. Импорт
2. Главная функция — DiscoveryMain
3. Определения функций (необязательно)

Ниже приведен пример сценария 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
```

Импорт

Классы Jython распределены по иерархическим пространствам имен. В версии 7.0 и более поздних версиях (в отличии от предыдущих версий) подразумеваемый импорт отсутствует, поэтому каждый класс должен быть импортирован явно. (Это изменение внесено по соображениям производительности и чтобы сделать сценарий боле понятным Jython за счет отображения всех важных сведений.)

- Чтобы импортировать сценарий Jython:

  ```
  import logger
  ```

- Чтобы импортировать класс Java:

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

Главная функция — DiscoveryMain

Каждый исполняемый файл сценария Jython включает главную функцию: DiscoveryMain.

Функция `DiscoveryMain` — это главная точка входа в сценарий, первая функция, которую он выполняет. Главная функция может вызывать другие функции, указанные в сценариях:

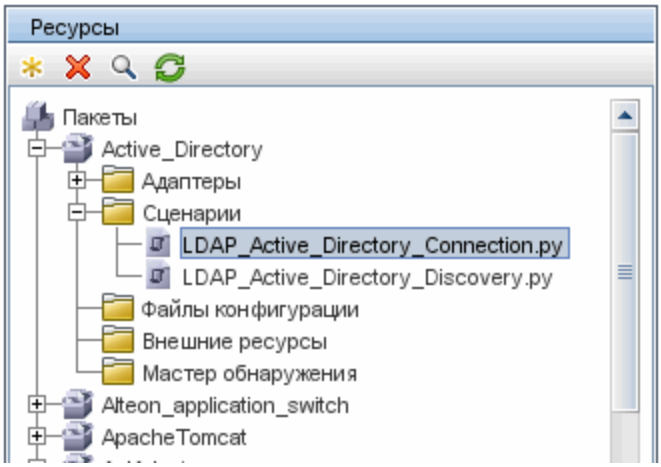
```
def DiscoveryMain(Framework):
```

Аргумент Framework должен быть указан в определении главной функции. Этот аргумент используется главной функцией для получения информации, необходимой для выполнения сценариев (например, информации об ЭК-триггерах и параметрах) и также может использоваться для создания отчетов по ошибкам, возникшим в ходе выполнения сценария.

Вы можете создать сценарий Jython без главного метода. Такие сценарии используются как сценарии библиотек и вызываются из других сценариев.

Определение функций

Каждый сценарий может содержать дополнительные функции, которые вызываются из главного кода. Каждая из таких функций может вызывать другую функцию, входящую в текущий сценарий или другой сценарий (с помощью инструкции `import`). Обратите внимание, что для использования другого сценария необходимо добавить его в раздел Scripts пакета:



Пример функции, вызывающей другую функцию:

В следующем примере главный код вызывает метод `doQueryOSUsers(..)`, который вызывает внутренний метод `doOSUserOSH(..)`:

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

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

Если сценарий является глобальной библиотекой, которая связана с большим числом адаптеров, его можно добавить в список сценариев в файле конфигурации `jythonGlobalLibs.xml`, не добавляя сценарий к каждому адаптеру (**Управление адаптерами > Панель ресурсов > AutoDiscoveryContent > Файлы конфигурации**).

Формирование результатов сценарием Jython

Каждый сценарий Jython выполняется для определенного ЭК-триггера и завершается результатами, возвращенными функцией `DiscoveryMain`.

Фактически, сценарий представляет собой группу ЭК и связей, которые должны быть вставлены или обновлены в CMDB. Сценарий возвращает группу ЭК и связей в формате `ObjectStateHolderVector`.

Класс `ObjectStateHolder` — это способ представления объекта или связи, заданных в CMDB. Объект `ObjectStateHolder` содержит имя типа ЭК и список атрибутов и их значений. `ObjectStateHolderVector` — это вектор экземпляров `ObjectStateHolder`.

Синтаксис ObjectStateHolder

В этом разделе описывается построение результатов DFM в модели CMDB.

Пример установки атрибутов ЭК:

Класс ObjectStateHolder описывает граф результатов DFM. Все ЭК и связи (отношения) помещаются в экземпляр класса `ObjectStateHolder` как в следующем примере кода Jython:
siebel application server 1 appServerOSH = ObjectStateHolder('siebelappserver') 2 appServerOSH.setStringAttribute('data_name', sblsvrName) 3 appServerOSH.setStringAttribute ('application_ip', ip) 4 appServerOSH.setContainer(appServerHostOSH)

- Строка 1 создает ЭК с типом **siebelappserver**.
- Строка 2 создает атрибут **data_name** со значением **sblsvrName**, которое представляет собой переменную Jython с именем обнаруженного сервера в качестве значения.
- Строка 3 устанавливает неключевой атрибут, который обновляется в CMDB.
- Строка 4 заключается в построении включения (результата графа). Она указывает, что сервер приложений находится в хосте (другой класс `ObjectStateHolder` в области).

Примечание: Каждый ЭК, передаваемый сценарием Jython, должен включать значения всех ключевых атрибутов типа ЭК.

Пример отношений (связей):

В следующем примере связи поясняется представление графа:
1 linkOSH = ObjectStateHolder('route') 2 linkOSH.setAttribute('link_end1', gatewayOSH) 3 linkOSH.setAttribute('link_end2', appServerOSH)

- Строка 1 создает связь (которая также присутствует в классе `ObjectStateHolder`. Единственное отличие заключается в том, что `route` — это тип ЭК связи).
- Строки 2 и 3 определяют узлы на каждой стороне каждой связи. Это реализуется с помощью атрибутов связи **end1** и **end2**, которые должны быть заданы (поскольку являются минимальными ключевыми атрибутами каждой связи). Значения атрибутов — экземпляры `ObjectStateHolder`. Подробнее об End 1 и End 2 см. в разделе "Ссылка" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

Внимание. Связь имеет направление. Вы должны убедиться, что узлы End 1 и End 2 соответствуют правильным типам ЭК на каждой стороне. Если узлы неверны, объект результата не пройдет проверку и не будет передан должным образом. Подробнее см. в разделе Связи типов ЭК в документе *Руководство по моделированию в HP Universal CMDB*.

Пример вектора (сбор ЭК):

После создания объектов с атрибутами и связей с объектами можно объединить их в группу. Для этого просто добавьте их в экземпляр `ObjectStateHolderVector` следующим образом:

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

Подробные сведения о передаче этого составного результата в компонент Framework для отправки на сервер CMDB см. в описании метода sendObjects.

После сборки графа результатов в экземпляре `ObjectStateHolderVector` его необходимо вернуть в DFM Framework для вставки в CMDB. Это реализуется путем возврата экземпляра `ObjectStateHolderVector` как результата функции `DiscoveryMain()`.

Примечание: Дополнительные сведения о создании **OSH** для общих типов ЭК см. в разделе "modeling.py" на странице 63.

Отправка больших объемов данных

В UCMDB отправка больших объемов данных (более 20 КБ) может вызвать трудности при обработке. В этом случае перед отправкой в UCMDB необходимо разбить данные на более мелкие блоки. В целях обеспечения корректной вставки блоков данных в UCMDB каждый блок должен содержать идентификационные данные обо всех ЭК в блоке. Далее приведен стандартный сценарий разработки интеграции на базе Jython. Для отправки блоков данных используется метод sendObjects. Если сценарий Jython отправляет большое количество результатов (значение по умолчанию — 20 000 — можно изменить при помощи ключа **appilog.agent.local.maxTaskResultSize** в файле DataFlowProbe.properties), разделение данных на блоки будет происходить в соответствии с их топологией. При этом будут использоваться правила идентификации для корректной вставки результатов в UCMDB. Если сценарий Jython не осуществляет разбиение данных,

это происходит на стороне зонда; однако при большом объеме данных это приведет к падению производительности.

Примечание. Следует использовать разбиение на блоки при работе с адаптерами Jython, а не при выполнении обычных заданий обнаружения. Задания обнаружения обычно выполняются в соответствии с определенным триггером и не занимаются отправкой больших объемов информации. В интеграциях на базе Jython триггер интеграции запускает задания обнаружения, которые возвращают большое количество сведений.

Однако существует возможность разбиения на блоки даже небольших объемов данных. В таких случаях между ЭК в различных блоках образовываются связи, и разработчик сценария Jython оказывается перед выбором:

- отправлять ЭК и все сведения о его идентификации в каждом блоке, который содержит ссылку на этот ЭК
- или использовать идентификатор ЭК, используемый в UCMDB. При этом сценарий Jython может получить идентификаторы UCMDB только после обработки каждого блока на сервере UCMDB. Для включения этого режима (режима синхронной отправки результатов) необходимо добавить в адаптер тег SendJythonResultsSynchronously. Этот тег отвечает за отправку идентификаторов ЭК при получении каждого блока данных. Разработчик адаптера может использовать эти идентификаторы для создания следующего блока. Для этого используют API Framework getIdMapping.

Пример использования getIdMapping

В первом блоке отправляются сведения об узлах. Во втором блоке отправляются сведения о процессах. Корневой контейнер процесса является узлом. Вместо отправки целого objectStateHolder узла в атрибуте процесса root_container, можно при помощи getIdMapping API получить его идентификатор и отправить только ID узла в атрибуте root_container, что существенно уменьшит объем блока.

Экземпляр Framework

Экземпляр Framework — это единственный аргумент главной функции сценария Jython. Этот интерфейс можно использовать для получения информации, необходимой для выполнения сценариев (например, информации об ЭК-триггерах и параметрах адаптеров). Он также может использоваться для создания отчетов по ошибкам, возникшим в ходе выполнения сценария. Подробнее см. в разделе о "Справка по API-интерфейсу Управления потоком данных" на странице 37.

Правильное применение экземпляра Framework — передача его в качестве аргумента каждому из использующих его методов.

Пример:

```
def DiscoveryMain(Framework):
         OSHVResult = helperMethod (Framework)
            return OSHVResult
def helperMethod (Framework):
```

```
....
 probe_name    = Framework.getDestinationAttribute('probe_name')
 ...
 return result
```

В этом разделе описываются важные сценарии использования Framework:

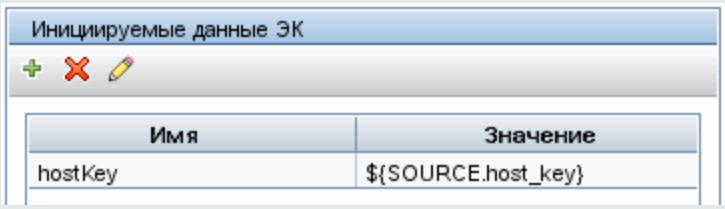
- "Framework.getTriggerCIData(String attributeName)" ниже
- "Framework.createClient(credentialsId, props)" ниже
- "Framework.getParameter (String parameterName)" на странице 47
- "Framework.reportError(String message) и Framework.reportWarning(String message)" на странице 47

Framework.getTriggerCIData(String attributeName)

Этот API-интерфейс представляет собой промежуточный этап между данными ЭК-триггера в адаптере и сценарием.

Пример получения учетных данных:

Вы запрашиваете следующие данные ЭК-триггера:



Для получения учетных данных из задачи воспользуйтесь следующим API-интерфейсом:

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

Framework.createClient(credentialsId, props)

Вы устанавливаете подключение к удаленному компьютеру, создав объект клиента и выполнив команды для этого клиента. Чтобы создать клиента, получите класс `ClientFactory`. Метод getClientFactory() получает тип запрошенного клиентского протокола. Константы протокола указаны в классе ClientsConsts. См. дополнительные сведения об учетных данных и поддерживаемых протоколах в разделе *Руководство по пакету обнаружения и интеграции HP UCMDB*.

Пример создания экземпляра клиента для идентификатора учетных данных:

Чтобы создать экземпляр `Client` для идентификатора учетных данных:

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

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

Теперь можно использовать экземпляр `Client` для подключения к нужному компьютеру и приложению.

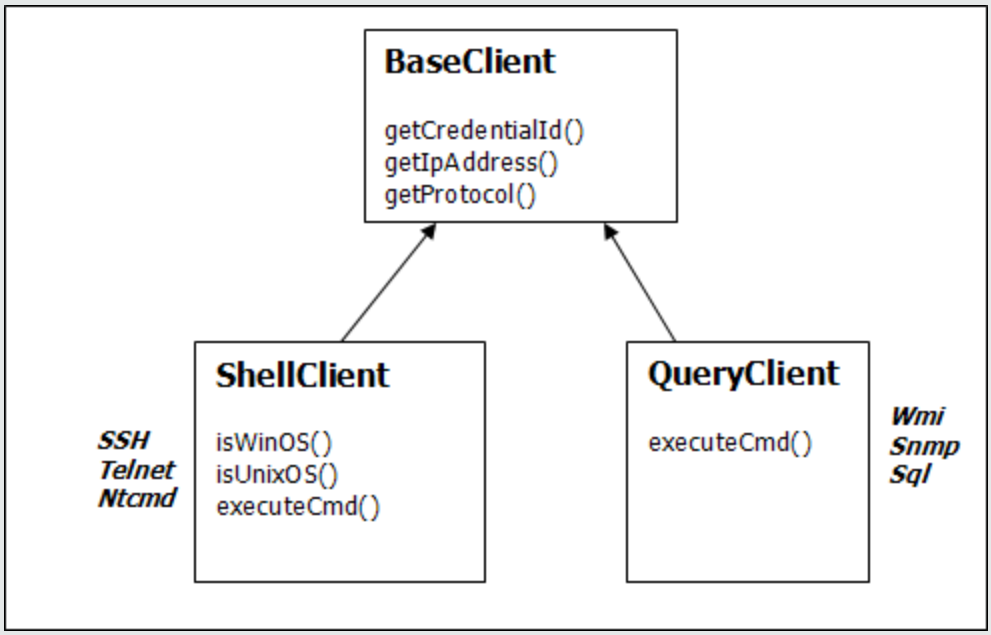
Пример создания клиента WMI и выполнения запроса WMI:

Чтобы создать клиент WMI и выполнить запрос WMI с помощью клиента:

```
wmiClient = Framework.createClient(credential)
resultSet = wmiClient. executeQuery("SELECT TotalPhysicalMemory
FROM    Win32_LogicalMemoryConfiguration")
```

Примечание: Для работы API-интерфейса `createClient()` добавьте следующий параметр к параметрам данных ЭК-триггера: **credentialsId = ${SOURCE.credentials_id}** на панели данных ЭК-триггера. Кроме того, можно вручную добавить идентификатор учетных данных при вызове функции:
wmiClient = clientFactory().createClient(credentials_id).

На следующей схеме представлена иерархия клиентов с общими поддерживаемыми API-интерфейсами:

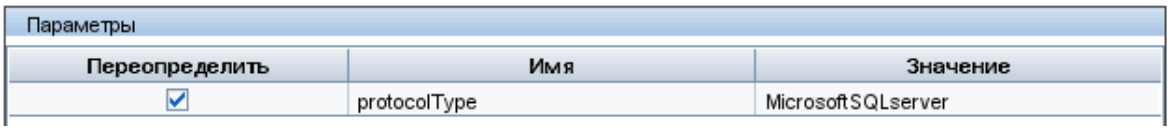


Подробные сведения о клиентах и поддерживаемых API-интерфейсах см. в разделах BaseClient, ShellClient и QueryClient (DFM Framework). Эти файлы находятся по следующему пути:

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

Framework.getParameter (String parameterName)

В дополнение к получению информации об ЭК-триггере часто требуется получить значения параметров адаптера. Пример:



Пример получения значения адаптера protocolType:

Чтобы получить значение параметра `protocolType` из сценария Jython, воспользуйтесь следующим API-интерфейсом:

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

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

Некоторые ошибки (например, ошибка подключения, неполадки оборудования, время ожидания) могут возникать при выполнении сценария. При обнаружении таких ошибок компонент Framework может сообщить о проблеме. Переданное сообщение достигает сервера и отображается для конечного пользователя.

Пример отчета об ошибке и сообщения:

В следующем примере демонстрируется использование API-интерфейса `reportError (<Error Msg>)`:

```
попробуйте:
    client = Framework.createClient(Framework.getTriggerCIData
(BaseClient.CREDENTIALS_ID))

except:
    strException = str(sys.exc_info()[1]).strip()
    Framework. reportError ('Connection failed: %s' % strException)
```

Для сообщений об ошибках можно использовать один из следующих API-интерфейсов: F`Framework.reportError(String message)`, `Framework.reportWarning(String message)` Различие двух API-интерфейсов заключается том, что при отправке сообщения об ошибке зонд сохраняет журнал обмена данными со всеми параметрами сеанса в файловой системе. Таким образом можно отслеживать сеансы и лучше понять ошибку.

См. дополнительные сведения о сообщениях об ошибках в разделе "Сообщения об ошибках" на странице 65.

Поиск правильных учетных данных (для адаптеров подключения)

Адаптер, который подключается к удаленной системе, должен перебрать все возможные учетные данные. Один из параметров, необходимых для создания клиента, — это идентификатор учетных данных. Сценарий подключения получает доступ к имеющимся наборам учетных данных и применяет их один за другим с помощью метода `Framework.getAvailableProtocols()`. Если учетные данные применяются успешно, адаптер передает объект подключения ЭК на хосте этого ЭК-триггера (с идентификатором учетных данных, соответствующих IP-адресу) в базу CMDB. Последующие адаптеры могут использовать ЭК объекта подключения напрямую для подключения к набору учетных данных (т.е. адаптеры не должны снова перебирать все доступные учетные данные).

Примечание. Доступ к конфиденциальным данным (паролям, закрытым ключам и т.д.) для следующих типов протоколов заблокирован:

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

Указанные протоколы следует использовать с выделенными клиентами.

В примере ниже представлено получения всех значений протокола SNMP. Обратите внимание, что IP-адрес получен из данных ЭК-триггера (`# Get the Trigger CI data values`).

Сценарий подключения запрашивает все возможные учетные данные протокола (`# Go over all the protocol credentials`) и выполняет цикл их перебора, пока один из наборов не срабатывает (`resultVector`). См. дополнительные сведения в подразделе **Двухэтапная парадигма подключения** раздела "Разделение адаптеров" на странице 24.

Пример

```
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')
# Получить значения данных ЭК-триггеров
ip_address = Framework.getDestinationAttribute('ip_address')

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

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

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

                    client.close()
client = None
```

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

          return (_vector, error)
```

Обработка исключений Java

Некоторые классы Java создают исключение при ошибках. Рекомендуется обработать исключение, в противном случае оно приведет к непредвиденному завершению работы адаптера.

При обработке известного исключения в большинстве случаев необходимо напечатать трассировку его стека и выдать соответствующее сообщение в интерфейсе пользователя.

Примечание. Важно импортировать базовый класс исключения в Java, как показано в примере ниже, поскольку в Python существует базовый класс исключений с таким же именем.

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

Если исключение не является неустранимым и выполнение сценария может быть продолжено, необходимо пропустить вызов метода `reportError()` и разрешить продолжение выполнения сценария.

Устранение неполадок при обновлении Jython с версии 2.1 до 2.5.3

В настоящее время в Universal Discovery используется Jython версии 2.5.3. Соответственно были изменены все стандартные сценарии. При использовании в Discovery сценариев

Jython, созданных до этого обновления, возможно появление ошибок, которые следует исправить, как указано ниже.

Примечание. Для внесения таких изменений необходимо иметь большой опыт разработки сценариев Jython.

Форматирование строк

- **Сообщение об ошибке:** `TypeError: int argument required`
- **Возможная причина:** Преобразование строчной переменной, содержащей целое число, в десятичное целое число.
- **Проблемный код Jython 2.1:**

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

- **Правильный вариант кода Jython 2.5.3:**

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

или

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

Проверка типа строки

Приведенный ниже код может работать некорректно при вводе данных в кодировке Unicode:

- **Проблемный код Jython 2.1:** `isinstance(unicodeStringVariable,'')`
- **Правильный вариант кода Jython 2.5.3:** `isinstance(unicodeStringVariable,basestring)`

  Для проверки соответствия объекта экземпляру `str` или unicode сравнение следует проводить при помощи `basestring`.

Наличие в файле знаков не из набора ASCII

- **Сообщение об ошибке:**
  `SyntaxError: Non-ASCII character in file 'x', , but no encoding declared; see http://www.python.org/peps/pep-0263.html for details`
- **Правильный вариант кода Jython 2.5.3:** (добавляется в первую строку файла)

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

Импорт подпакетов

- **Сообщение об ошибке:**
  `AttributeError: 'module' object has no attribute 'sub_package_name'`
- **Возможная причина:** Импорт подпакета без указания имени в операторе импорта.
- **Проблемный код Jython 2.1:**

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

Импорт подпакета происходит некорректно.

- **Правильный вариант кода Jython 2.5.3:**

```
import a.b
```

или

```
from a import b
```

Изменение итератора

Начиная с Jython версии 2.2, метод **__iter__** используется для создания цикла по коллекции в зоне охвата блока **for-in**. Для итератора следует вызывать метод **next**, который возвращает требуемый элемент или выдает ошибку **StopIteration**, если достигнут конец коллекции. Если метод **__iter__** не применяется, вместо него используется метод **getitem**.

Вызов исключений

- **Метод вызова исключений, использовавшийся в Jython версии 2.1, устарел:**
  `raise Exception, 'Failed getting contents of file'`
- **Рекомендуемый метод вызова исключений в Jython версии 2.5.3:**
  `raise Exception('Failed getting contents of file')`

Поддержка локализации в адаптерах Jython

Поддержка нескольких языков позволяет DFM работать с различными языками ОС и применять различные настройки во время выполнения.

Этот раздел охватывает следующие темы:

- "Добавление поддержки нового языка" ниже
- "Изменение языка по умолчанию" на странице 54
- "Определение набора символов для кодировки" на странице 54
- "Настройка нового задания для работы с локализованными данными" на странице 55
- "Декодирование команд без ключевых слов" на странице 56
- "Работа с пакетами ресурсов" на странице 56
- "Справка по API-интерфейсам" на странице 57

Добавление поддержки нового языка

В этом разделе описывается добавление поддержки нового языка.

Эта задача включает следующие шаги:

- "Добавление пакета ресурсов (PROPERTIES-файлов)" ниже
- "Объявление и регистрация объекта языка" на следующей странице

1. Добавление пакета ресурсов (PROPERTIES-файлов)

   Добавление пакета ресурсов в соответствии с выполняемым заданием. В

следующей таблице перечислены задания DFM и пакеты ресурсов, которые используются для них.

Задание	Базовое имя пакета ресурсов
File Monitor by Shell	langFileMonitoring
Host Resources and Applications by Shell	langHost_Resources_By_TTY, langTCP
Hosts by Shell using NSLOOKUP in DNS Server	langNetwork
Host Connection by Shell	langNetwork
Collect Network Data by Shell or SNMP	langTCP
Host Resources and Applications by SNMP	langTCP
Microsoft Exchange Connection by NTCMD, Microsoft Exchange Topology by NTCMD	msExchange
MS Cluster by NTCMD	langMsCluster

См. дополнительные сведения о пакетах в разделе "Работа с пакетами ресурсов" на странице 56.

2. Объявление и регистрация объекта языка

Чтобы указать новый язык, добавьте следующие две строки кода в сценарий **shellutils.py**, который содержит список всех поддерживаемых языков. Сценарий будет включен в пакет `AutoDiscoveryContent`. Для просмотра сценария откройте окно «Управление адаптерами». Подробнее см. в разделе "Окно "Управление адаптерами" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

a. Объявите язык следующим образом:

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

См. дополнительные сведения о языке класса в разделе "Справка по API-интерфейсам" на странице 57. Подробнее об объекте Class Locale см. по адресу http://java.sun.com/j2se/1.5.0/docs/api/java/util/Locale.html. Можно воспользоваться существующим языком или указать новый.

b. Зарегистрируйте язык, добавив его в следующую коллекцию:

```
LANGUAGES = (LANG_ENGLISH, LANG_GERMAN, LANG_SPANISH, LANG_RUSSIAN, LANG_
JAPANESE)
```

Изменение языка по умолчанию

Если определение языка ОС невозможно, используется язык по умолчанию. Язык по умолчанию указывается в файле **shellutils.py**.

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

Чтобы изменить язык по умолчанию, инициализируйте переменную DEFAULT_LANGUAGE с другим языком. Дополнительные сведения см. в разделе "Добавление поддержки нового языка" на странице 52.

Определение набора символов для кодировки

Подходящий набор символов для вывода команды декодирования определяется во время выполнения. Многоязычное решение основывается на следующих фактах и предположениях:

1. Существует возможность определить язык ОС способом, независимым от региональных параметров, например, запустив команду **chcp** в Windows или команду **locale** в Linux.
2. Кодировка языка связей хорошо известна и может быть указана статически. Например, в русском языке используется две распространенные кодировки: `Cp866` и `Windows-1251`.
3. Один набор символов для каждого языка является предпочтительным, например предпочтительная кодировка для русского языка: `Cp866`. Это значит, что большинство команд будут выводить данные в этой кодировке.
4. Кодировку вывода следующей команды предсказать невозможно, но это будет одна из возможных кодировок на используемом языке. Например, при использовании компьютера Windows с русским языком система выдает результаты команды **ver** в кодировке Cp866, но для вывода команды **ipconfig** будет использоваться Windows-1251.
5. Вывод известных команд содержит известные ключевые слова. Например, команда **ipconfig** содержит переведенную форму строки **IP-Address**. Таким образом, вывод команды **ipconfig** содержит **IP-Address** для английской ОС, IP-Адрес для русской ОС, **IP-Adresse** для немецкой ОС и т.д.

После определения языка вывода команды (# 1) количество возможных кодировок будет ограниченно одной или двумя (# 2). Более того, нам известно, какие ключевые слова содержатся в выводе (# 5).

Таким образом, решением задачи будет декодирование вывода команды с помощью одной из возможных кодировок и поиск ключевых слов в результате. Если ключевое слово найдено, текущий набор символов считается верным.

Настройка нового задания для работы с локализованными данными

В этой задаче описывается создание нового задания, которое может работать с локализованными данными.

Обычно сценарии Jython выполняют команды и обрабатывают их вывод. Для получения вывода команды, декодированного как свойства, используется API-интерфейс для класса **ShellUtils**. Дополнительные сведения см. в разделе " Обзор API-интерфейса веб-службы HP Universal CMDB" на странице 338.

Обычно этот код принимает следующий вид:

```
client = Framework.createClient(protocol, properties)
shellUtils = shellutils.ShellUtils(client)
languageBundle = shellutils.getLanguageBundle ('langNetwork',
shellUtils.osLanguage, Framework)
strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_address')
ipconfigOutput = shellUtils.executeCommandAndDecode('ipconfig /all',
strWindowsIPAddress)
#Do work with output here
```

1. Создание клиента:

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

2. Создайте экземпляр класса **ShellUtils** и добавьте в него язык ОС. Если язык не добавлен, используется язык по умолчанию (обычно английский):

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

   Во время инициализации объекта DFM автоматически обнаруживает язык компьютера и устанавливает предпочтительную кодировку из объекта Language. Предпочтительная кодировка — это первая кодировка в списке.

3. Получите соответствующий ресурс пакета из **shellclient** с помощью метода **getLanguageBundle**:

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

4. Получите ключевое слово из пакета ресурсов, подходящего для определенной команды:

   ```
   strWindowsIPAddress = languageBundle.getString('windows_ipconfig_str_ip_
   address')
   ```

5. Вызовите метод **executeCommandAndDecode** и передайте в него ключевое слово для объекта **ShellUtils**:

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

```
strWindowsIPAddress)
```

Объект ShellUtils object также используется для предоставления пользователю доступа к справочнику по API-интерфейсу (где приводится подробное описание метода).

6. Обработайте вывод обычным способом.

Декодирование команд без ключевых слов

Текущий подход к локализации подразумевает использование ключевого слова для декодирования всего вывода команды. Дополнительные сведения см. в описании шага о получении ключевого слова из пакета ресурсов ("Настройка нового задания для работы с локализованными данными" на предыдущей странице).

Однако в другом подходе используется ключевое слово для декодирования только вывода первой команды, а последующие команды декодируются с помощью набора символов, использованного для первой команды. Для этого используются методы **getCharsetName** и **useCharset** объекта **ShellUtils**.

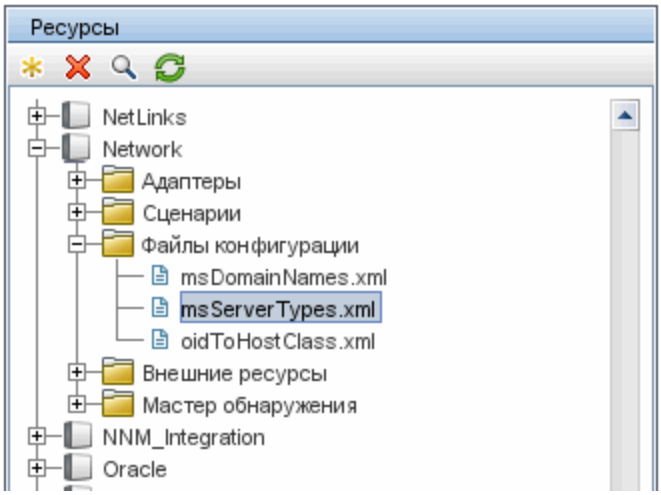
Обычный сценарий использования описывается далее.

1. Вызовите метод **executeCommandAndDecode** один раз.
2. Получите имя последнего использованного набора символов с помощью метода **getCharsetName**.
3. Настройте **shellUtils** для использования этого набора символов по умолчанию, вызвав метод **useCharset** в объекте **ShellUtils**.
4. Вызовите метод **execCmd** из **ShellUtils** один или несколько раз. В возвращенном выводе будет использоваться кодировка, указанная в предыдущем шаге. Дополнительные операции декодирования выполняться не будут.

Работа с пакетами ресурсов

Пакет ресурсов — это файл с расширением properties (***.properties**). Файл properties можно считать словарем, в котором хранятся данные в формате `ключ = значение`. Каждая строка файла properties содержит одно сопоставление `ключ = значение`. Главная функция пакета ресурсов — возврат значения по ключу.

Пакеты ресурсов находятся на компьютере зонда: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryConfigFiles**. Они загружаются с сервера UCMDB так же, как любые другие файлы конфигурации. Их можно редактировать, добавлять и удалять в окне «Ресурсы». Подробнее см. в разделе "Панель "Файл конфигурации" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.



При обнаружении места назначения DFM обычно требуется обработать текст из вывода команды или содержимого файла. Эта обработка часто основывается на регулярном выражении. Разные языки требуют разных регулярных выражений для обработки. Чтобы создать код для всех языков, необходимо извлечь все данные, зависящие от языка, в пакеты ресурсов. Для каждого языка существует пакет ресурсов. (Добавление нескольких языков в один пакет ресурсов возможно, но в DFM одному пакету ресурсов всегда соответствует один язык.)

Сам сценарий Jython не включает данные для определенных языков с жестким кодированием (например, регулярные выражения для определенных языков). Сценарий определяет язык удаленной системы, загружает необходимый пакет ресурсов и получает данные для определенного языка по указанному ключу.

В DFM для пакетов ресурсов применяются имена в определенном формате: `<base_name>_<language_identifier>.properties`, например `langNetwork_spa.properties`. (Пакет ресурсов по умолчанию имеет следующий формат имени: `<base_name>.properties`, for example, `langNetwork.properties`.)

Формат `base_name` соответствует задаче пакета. Например, **langMsCluster** означает, что пакет ресурсов содержит ресурсы для определенных языков, используемых заданием «Кластер MS».

`language_identifier` — это трехбуквенный идентификатор языка. Например, `rus` обозначает русский язык, а `ger` — немецкий. Этот идентификатор языка входит в объявление объекта `Language`.

Справка по API-интерфейсам

Этот раздел охватывает следующие темы:

- "Класс Language" на следующей странице
- "Метод executeCommandAndDecode" на следующей странице
- "Метод getCharsetName" на странице 59
- "Метод useCharset" на странице 59

- "Метод getLanguageBundle" на следующей странице
- "Поле osLanguage" на следующей странице

Класс Language

Этот класс включает информацию о языке, например постфикс пакета, возможную кодировку и т.п.

Поля

Имя	Описание
locale	Java-объект, представляющий язык.
bundlePostfix	Постфикс пакета ресурсов. Этот постфикс используется в именах файлов пакетов ресурсов для идентификации языка. Например, пакет **langNetwork_ger.properties** включает постфикс **ger**.
charsets	Наборы символов, используемые для кодирования этого языка. В каждом языке может применяться несколько наборов символов. Например, для русского языка обычно используются кодировки `Cp866` и `Windows-1251`.
wmiCodes	Список кодов WMI, используемых ОС Microsoft Windows для идентификации языка. Все доступные коды перечислены по адресу http://msdn.microsoft.com/en-us/library/aa394239(VS.85).aspx (раздел OSLanguage). Один из методов идентификации языка ОС заключается в запросе ОС класса WMI для свойства `OSLanguage`.
codepage	Кодовая страница, используемая с указанным языком. Например, `866` используется для русских компьютеров, а `437` — для английских компьютеров. Один из методов идентификации языка ОС заключается в получении ее кодовой страницы по умолчанию (например, с помощью команды `chcp`).

Метод executeCommandAndDecode

Этот метод предназначен для сценариев бизнес-логики Jython. Он инкапсулирует операцию декодирования и возвращает декодированный вывод команды.

Аргументы

Имя	Описание
cmd	Фактическая команда для выполнения.
keyword	Ключевое слово, используемое для операции декодирования.
framework	Объект Framework передается для каждого исполняемого сценария Jython в DFM.
timeout	Время ожидания команды.

Имя	Описание
waitForTimeout	Указывает, должен ли клиент ждать до окончания времени ожидания.
useSudo	Включает или отключает использование sudo (относится только к клиентским компьютерам UNIX).
language	Активирует ввод языка напрямую вместо автоматического обнаружения.

Метод getCharsetName

Этот метод возвращает имя недавно использованного набора символов.

Метод useCharset

Этот метод устанавливает набор символов для экземпляра `ShellUtils`, который использует этот набор символов для первичного декодирования данных.

Аргументы

Имя	Описание
charsetName	Имя набора символов, например `windows-1251` или `UTF-8`.

См. также раздел "Метод getCharsetName" выши.

Метод getLanguageBundle

Этот метод используется для получения правильного пакета ресурсов. Он заменяет следующий API-интерфейс:

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

Аргументы

Имя	Описание
baseName	Имя пакета без языкового суффикса, например `langNetwork`.
language	Объект language. Здесь передается `ShellUtils.osLanguage`.
framework	Framework, общий объект, который передается для каждого исполняемого сценария Jython в DFM.

Поле osLanguage

Это поле содержит объект, который представляет язык.

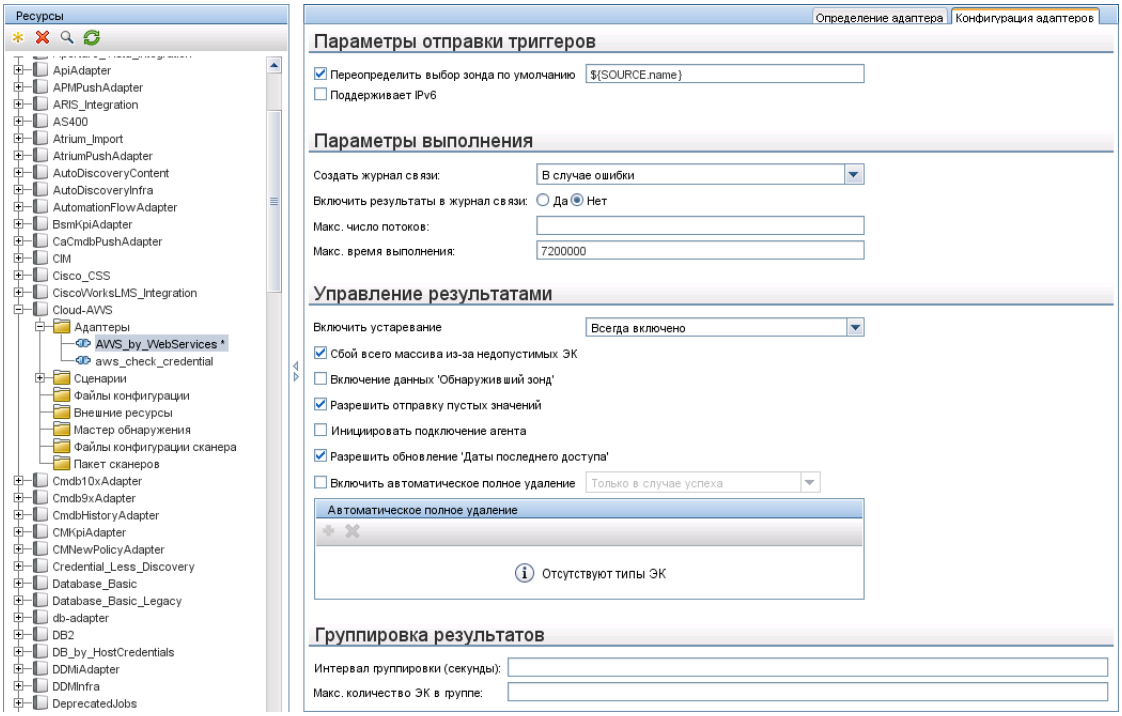
Запись кода DFM

Иногда может быть полезно записать выполнение сценария со всеми параметрами, например при отладке и тестировании кода. В этой задаче описывается запись

выполнения со всеми необходимыми переменными. Более того, вы можете просмотреть дополнительные данные отладки, которые обычно не заносятся в данные журналов даже на уровне отладки.

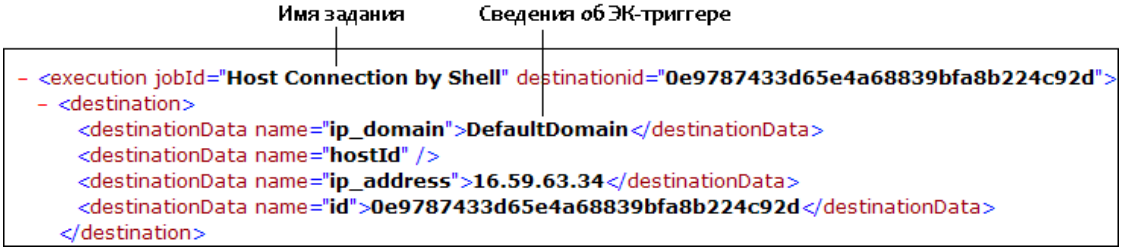
Для записи кода DFM:

1. Откройте раздел **Управление потоком данных > Управление адаптерами**. Щелкните правой кнопкой мыши задание, выполнение которого необходимо записать, и выберите **Перейти к адаптеру**, чтобы открыть приложение «Управление адаптерами».
2. Найдите панель **Параметры выполнения** на вкладке **Конфигурация адаптеров**, как показано ниже.

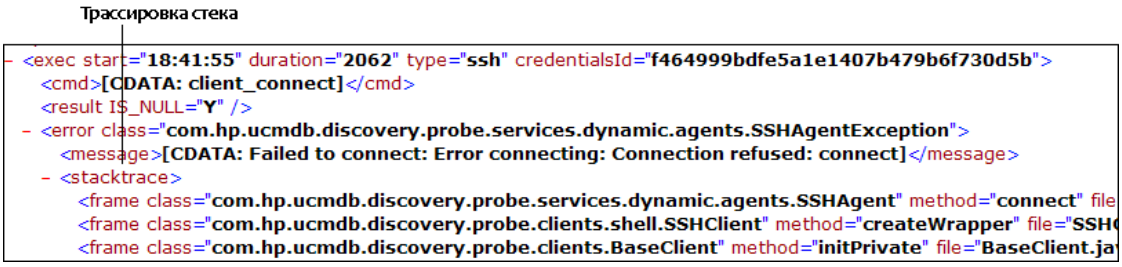


3. Измените значения поля **Создать журнал связи** на **Всегда**. Подробные сведения о настройке параметров журнала см. в разделе "Панель "Параметры выполнения" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

   Ниже приводится XML-файл журнала, созданный при выполнении задания **Host Connection by Shell** со значением параметра **Create communication logs Always** или **On Failure**:



В следующем примере представлены сообщения и параметры трассировки стека:



Средства и библиотеки Jython

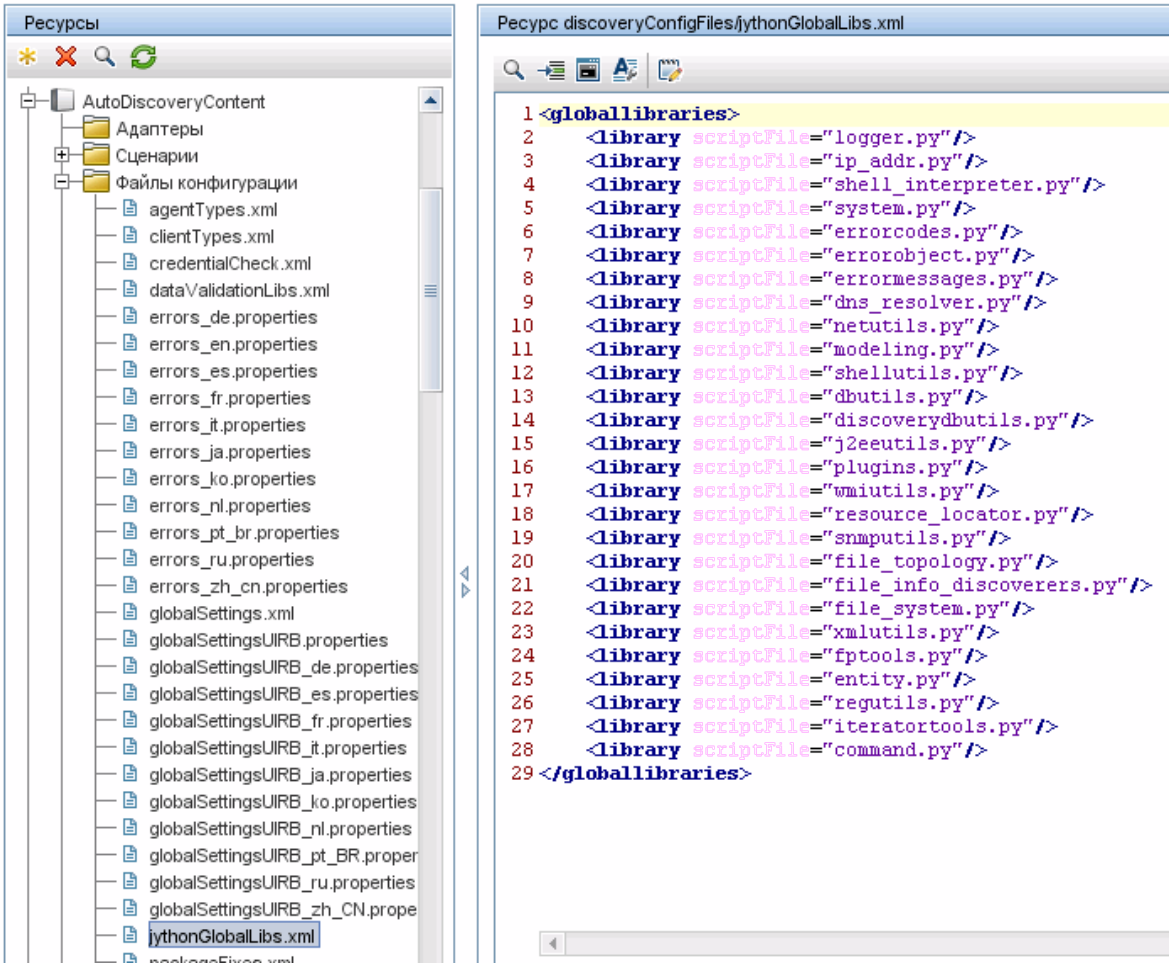
Несколько служебных сценариев широко используются в адаптерах. Эти сценарии являются частью пакета `AutoDiscovery` и находятся по следующему пути: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** вместе с другими сценариями, загруженными в зонд.

Примечание. Папка `discoveryScript` создается динамически, когда зонд начинает работу.

Чтобы воспользоваться одним из служебных сценариев, добавьте следующий файл импорта в разделе import сценария:

```
import <script name>
```

Библиотека Python AutoDiscovery содержит служебные сценарии Jython. Эти сценарии библиотеки считаются внешней библиотекой DFM. Они определены в файле `jythonGlobalLibs.xml` (в папке **Configuration Files**).



Все сценарии, указанные в файле `jythonGlobalLibs.xml`, загружаются по умолчанию при запуске зонда, поэтому их явное указание в определении адаптера не требуется.

Этот раздел охватывает следующие темы:

- "logger.py" ниже
- "modeling.py" на следующей странице
- "netutils.py" на следующей странице
- "shellutils.py" на странице 64

logger.py

Сценарий **logger.py** содержит средства журналов и вспомогательные функции для регистрации ошибок. Их можно назвать API-интерфейсами отладки, информации и ошибок для записи в файлы журнала. Сообщения журналов записываются в **C:\hp\UCMDB\DataFlowProbe\runtime\log**.

Сообщения вводятся в файле журнала в соответствии с уровнем отладки, указанном в дополнении PATTERNS_DEBUG файла **C:\hp\UCMDB\DataFlowProbe\conf\log\probeMgrLog4j.properties**. (По умолчанию используется

уровень DEBUG.) Дополнительные сведения см. в разделе "Уровни серьезности ошибок" на странице 68.

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

Информационные сообщения и сообщения об ошибках также отображаются в консоли командной строки.

Существует два набора API-интерфейсов:

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

Первый набор выполняет слияние всех атрибутов строки на соответствующем уровне журнала, а второй набор выполняет слияние и трассировку стека последнего созданного исключения для предоставления дополнительной информации, например:

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

modeling.py

Сценарий **modeling.py** содержит API-интерфейсы для создания хостов, API-интерфейсов, ЭК процессов и др. Эти API-интерфейсы обеспечивают создание общих объектов и делают код более понятным. Пример:

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

netutils.py

Библиотека **netutils.py** используется для получения информации о сети и TCP, например имен ОС, проверки допустимости MAC-адреса и IP-адреса и др. Пример:

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

shellutils.py

Библиотека **shellutils.py** предоставляет API-интерфейс для запуска команд оболочки и получения конечного статуса выполненной команды и обеспечивает выполнение нескольких команд на основании этого статуса. Библиотека инициализируется в клиенте оболочки и использует клиент для выполнения команд и получения результатов. Пример:

```
ttyClient = Framework.createClient(Framework.getTriggerCIData
(BaseClient.CREDENTIALS_ID), Props)
clientShUtils = shellutils.ShellUtils(ttyClient)
if (clientShUtils.isWinOs()):
        logger.debug ('discovering Windows..')
```

Глава 3: Сообщения об ошибках

Данная глава включает:

- Сообщения об ошибках — Обзор 65
- Правила сообщений об ошибках 65
- Уровни серьезности ошибок 68

Сообщения об ошибках — Обзор

В процессе обнаружения можно выявить большое количество ошибок: проблем при подключении, аппаратных проблем, исключений, случаев истечения времени ожидания и т.д. Эти ошибки отображаются на панели Universal Discovery при неудачном завершении стандартного потока обнаружения. Пользователь может перейти от ЭК-триггера, который стал причиной проблемы, к просмотру сообщения об ошибке.

DFM разделяет ошибки, которые могут быть пропущены в некоторых случаях (например, хост недоступен) и ошибки, которые требуют устранения (например, проблемы учетных данных или отсутствие файлов конфигурации или DLL-файлов). Более того, DFM сообщает об ошибке один раз, даже если она повторяется при последующих выполнениях, и регистрирует даже ошибки, которые происходят всего один раз.

При создании пакета можно добавить нужные сообщения об ошибках в качестве ресурсов в этот пакет. Во время развертывания пакета сообщения также будут развернуты по соответствующему пути. Сообщения должны отвечать правилам, описанным в разделе "Правила сообщений об ошибках" ниже.

DFM поддерживает сообщения об ошибках на нескольких языках. Создаваемые сообщения об ошибках можно локализовать, чтобы они отображались на других языках.

Подробные сведения о поиске ошибок см. в разделе "Ход и результат обнаружения" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

Подробные сведения о настройке журналов связи см. в разделе "Панель "Параметры выполнения" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

Правила сообщений об ошибках

- Каждая ошибка идентифицируется кодом сообщения об ошибке и массивом аргументов (**int, String[]**). Сочетание кода сообщения и массива аргументов идентифицирует определенную ошибку. Массив параметров может иметь значение null.
- Каждый код ошибки привязывается к **краткому сообщению**, которое представляет собой фиксированную строку, и **подробному сообщению**, которое является шаблоном,

содержащим ноль или более аргументов. Предполагается соответствие между числом аргументов в шаблоне и фактическим числом параметров.

Пример кода ошибки:

10234 может представлять ошибку с кратким сообщением:

```
Ошибка подключения
```

и подробным сообщением:

```
Не удалось подключиться по протоколу {0} из-за истечения времени ожидания {1} мс
```

где

{0} = первый аргумент: имя протокола

{1} = второй аргумент: время ожидания в мс

Данный раздел также включает следующие подразделы.

- "Содержимое файла свойств" ниже
- "Файл свойств сообщений об ошибках" ниже
- "Правила именования языков" ниже
- "Коды сообщений об ошибке" на следующей странице
- "Неклассифицированные ошибки содержимого" на следующей странице
- "Изменения платформы" на странице 68

Содержимое файла свойств

Файл свойств должен содержать два ключа для каждого кода сообщения об ошибке. Например, для ошибки 45:

- **DDM_ERROR_MESSAGE_SHORT_45**. Краткое описание ошибки.
- **DDM_ERROR_MESSAGE_LONG_45**. Длинное описание ошибки (может содержать параметры, например {**0**},{**1**}).

Файл свойств сообщений об ошибках

Файл свойств содержит сопоставление кода сообщения об ошибке и двух сообщений (краткого и подробного).

После развертывания файла свойств его данные объединяются с существующими данными, т. е. новые коды сообщений добавляются, а старые заменяются.

Файлы свойств инфраструктуры являются частью пакета **AutoDiscoveryInfra**.

Правила именования языков

- Для языка по умолчанию: **<имя файла>.properties.errors**
- Для определенного языка: **<имя файла>_xx.properties.errors**

где **xx** — это язык (например, **infraerr_fr.properties.errors** или **infraerr_en_us.properties.errors**).

Коды сообщений об ошибке

Следующие коды входят в конфигурацию HP Universal CMDB по умолчанию. Пользователь может добавить собственные сообщения об ошибках в этот список.

Имя ошибки	Код ошибки	Описание
Внутренние	100-199	В основном разрешаются из исключений, созданных при выполнении сценариев Jython
Подключение	200-299	Ошибка подключения, нет агента на целевом компьютере, целевая система недоступна и др.
Связанные с учетными данными	300-399	Разрешение отклонено, попытка подключения отклонена из-за отсутствия учетных данных
Время ожидания	400-499	Время ожидания истекло при подключении или вводе команды
Непредвиденное или недопустимое поведение	500-599	Отсутствие файлов конфигурации, непредвиденные прерывания и др.
Получение информации	600-699	Отсутствие информации на целевых компьютерах, ошибки запроса информации у агента и др.
Связанные с ресурсами	700-799	Ошибки, связанные с нехваткой памяти, или клиентами, не отключенными должным образом
Обработка	800-899	Ошибка обработки текста
Кодировка	900	Ошибка ввода, неподдерживаемая кодировка
Связанные с SQL	901-903, 924	Ошибка, полученные от операторов SQL
Связанные с HTTP	904-909	Ошибки, созданные при HTTP-подключений, полученные в ходе анализа кодов ошибок HTTP.
Определенное приложение	910-923	Ошибки, возникшие из-за проблем, связанных с приложениями, например неверной версией LSOF, отсутствием диспетчеров запросов и др.

Неклассифицированные ошибки содержимого

Для поддержки старого содержимого без регрессии приложение и соответствующие методы SDK обрабатывают код сообщений 100 (неклассифицированная ошибка сценария) по-разному.

Эти ошибки не группируются (то есть не считаются ошибками одного типа) по коду приложения, но группируются по содержимому сообщения. То есть если сценарий сообщает об ошибке посредством старых методов (со строкой сообщения, но без кода ошибки), все сообщения получают одинаковый код, но в приложении или соответствующих методах SDK разные сообщения отображаются как разные ошибки.

Изменения платформы

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

Следующие методы добавлены в интерфейс.

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

Следующие старые методы отмечены как устаревшие, но поддерживаются для обратной совместимости:

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

Уровни серьезности ошибок

Когда адаптер завершает выполнение для ЭК-триггера, он возвращает статус. Если ошибки и предупреждения отсутствуют, возвращается статус **Success**.

Ниже перечислены уровни серьезности (от минимального до максимального):

Неустранимые ошибки

Этот уровень включает серьезные ошибки, такие как проблемы инфраструктуры, отсутствие DLL-файлов и исключения:

- Не удалось создать задачу (зонд не найден, переменные не найдены и т. д.)
- Выполнение сценария невозможно
- Обработка результатов на сервере заканчивается неудачей и данные не записываются в CMDB

Ошибки

На этом уровне представлены ошибки, которые не позволяют DFM получить данные. Эти ошибки следует просматривать, так как обычно они требуют определенных действий (таких как увеличение времени ожидания, изменение диапазона, изменение параметра или добавления другого набора учетных данных пользователя).

- Если вмешательство пользователя может помочь, сообщение будет содержать данные о проблеме учетных данных или сети, требующей дальнейшего анализа. (Это ошибки конфигурации, а не обнаружения.)

- Внутренняя ошибка, обычно вызвана непредвиденным поведением на обнаруженном компьютере или в приложении, например отсутствием файлов конфигурации и т. д.

Предупреждения

Если выполнение успешно, но могут существовать незначительные проблемы, о которых следует знать пользователю, DFM отмечает их как **Warning**. Пользователю следует просмотреть соответствующие ЭК перед началом более детального сеанса отладки. **Warning** может содержать сообщения об отсутствии установленного агента на удаленном хосте, а также о том, что недопустимые данные привели к некорректному расчету атрибута.

- Отсутствие агента подключения (SNMP, WMI)
- Обнаружение выполнено, но не все доступные сведения обнаружены

Глава 4: Сопоставление зависимостей потребитель-поставщик

Данная глава включает:

- Обнаружение зависимостей — Обзор 70
- Файлы сигнатуры зависимости 72
- Адаптеры поиска зависимостей 105
- Полный пример 114

Обнаружение зависимостей — Обзор

Сопоставление зависимостей является удобным методом обнаружения связей между развертываемыми компонентами или запущенным программным обеспечением. Этот метод позволяет применять настраиваемые правила сопоставления зависимостей (с простым программным синтаксисом), которые используются процессом Universal Discovery для автоматического обнаружения зависимостей.

Службы делятся на два типа: бизнес-службы и IT-службы. Бизнес-служба — это служба, которую один бизнес предлагает другому бизнесу (B2B), или которую одна организация предлагает другой в рамках одного бизнеса (например, обработка платежей). IT-служба — это бизнес служба, которую IT-организация предлагает для поддержки бизнес-служб или собственных операций IT.

Развертываемый компонент — это компонент программного обеспечения, который развертывается внутри запущенного программного обеспечения, например, сервер приложений или веб-сервер. Примерами развертываемых компонентов могут служить компоненты JEE EAR или схема внутри базы данных Oracle. В контексте обнаружения зависимостей запущенное программное обеспечение считается развертываемым компонентом.

Развертываемый компонент **поставщика** предоставляет службу и указывает, каким образом другие развертываемые компоненты могут использовать эту службу. Развертываемый компонент **потребителя** "потребляет" службу, предоставляемую развертываемым компонентом поставщика. Зависимость между этими развертываемыми компонентами является зависимостью типа **потребитель-поставщик**.

Примечание.

- Чтобы иметь возможность создавать адаптеры зависимости потребитель-поставщик и использовать платформу сопоставления зависимостей, необходимо выбрать параметр **Включить поиск** при настройке схемы UCMDB в процессе установки.

- Адаптеры зависимости "потребитель-поставщик" могут выполняться в зондах потока данных только в объединенном режиме.

Дополнительные сведения см. в разделе:

- "Поставщики и потребители" ниже
- "Сигнатуры зависимостей" ниже
- "Поток сопоставления зависимостей" на следующей странице

Поставщики и потребители

Подключение к поставщикам выполняется при помощи строк подключения. Например, если поставщиком является база данных Oracle, для подключения к ее службам потребуется следующее:

- IP-адрес компьютера
- SID
- порт TCP

Эти три типа данных будут входить в строки подключения, которые необходимы потребителю для подключения в службе, предоставляемой поставщиком. К примеру, строка подключения к Oracle может содержать следующую информацию:

- IP-адреса: 1.1.1.1, 2.2.2.2
- Порт: 1521
- SID: abcd

Потребитель знает минимум об одной строке подключения к поставщику, и эта строка находится в известном расположении, например, в документе конфигурации, таблице БД , реестре Windows и т.д. В результате поиска в этих расположениях могут быть обнаружены зависимости между потребителями и поставщиками.

Если строки подключения поставщика найдены в определенном документе конфигурации, поставщик и контейнер этого документа конфигурации объединяются связью типа "потребитель-поставщик".

В результате процесс обнаружения зависимостей потребитель-поставщик становится прямолинейным: Строки подключения поставщика разыскиваются в документах конфигурации потребителя, и результаты поиска содержат все документы конфигурации, которыми владеют потребители указанного поставщика.

Подробнее см. в разделе "Определение зависимостей" на странице 79.

Сигнатуры зависимостей

Для каждого документа конфигурации и типа поставщика могут использоваться различные поисковые запросы. Эти поисковые запросы указываются в файле сигнатуры зависимости.

Сигнатура зависимости — это правило, которое определяет факт существования зависимости потребитель-поставщик между конкретным поставщиком и конкретным

потребителем при помощи строки подключения поставщика и документов конфигурации потребителя.

Сигнатура зависимости составляется из выражений поиска. Эти выражения поиска зависят от определений строк подключения, а не от фактических значений для конкретного поставщика. Кроме того, выражения поиска также зависят от имени, расположения и формата документа конфигурации потребителя, но не от содержимого самих файлов. Когда строки подключения поставщика становятся известны, они вставляются в выражения поиска, что приводит к образованию конкретных выражений поиска. Затем конкретные выражения поиска проверяются при помощи документов конфигурации потребителя. Выражения поиска возвращают значение true, только если строки подключения поставщика каким-то особым образом существуют в документах конфигурации потребителя.

Подробнее см. в разделе "Файлы сигнатуры зависимости" ниже.

Поток сопоставления зависимостей

В этом разделе представлен краткий обзор основного процесса, который происходит во время сопоставления зависимостей:

1. Обнаруживаются развертываемые компоненты и их строки подключения.
2. Каждый тип развертываемого компонента поставщика инициирует определенное задание сопоставления зависимостей. Адаптер каждого задания знает, как извлекать требуемую строку подключения для определенного типа развертываемого компонента. Адаптер ищет другие развертываемые компоненты, которые используют службу.
3. Создается связь потребитель-поставщик между каждым найденным развертываемым компонентом и его триггером (поставщиком).

Файлы сигнатуры зависимости

Данный раздел включает следующие темы:

- Структура файла сигнатуры зависимости 72
- Упаковка и развертывание нескольких файлов сигнатуры зависимости 103
- Ошибки компиляции 104

Структура файла сигнатуры зависимости

Файл сигнатуры зависимости определяет одну или несколько зависимостей между развертываемыми компонентами.

Потребительская часть зависимости определяется элементом `<Deployable>`. Каждый элемент `<Deployable>` может содержать один или несколько элементов `<Dependency>`. Каждый такой элемент `<Dependency>` содержит условия, для которых существует зависимость между потребителем (элемент `<Deployable>`) и поставщиком.

Элемент <Dependency> может рассматриваться как логическая функция, которая в качестве входных данных использует ЭК поставщика и документы конфигурации потребителя, и возвращает значение true, только если существует зависимость между этими двумя ЭК — от потребителя к поставщику.

В элементе <Dependency> поставщик идентифицируется в файле только при помощи его типа ЭК. Каждая зависимость может использоваться для одного типа ЭК поставщика. В следующем примере представлена функция зависимости между любым потребителем с типом "Running Software" и поставщиком с типом "Running Software":

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

Переменные и понятия

Переменная может содержать разные значения во время выполнения программы. При сопоставлении зависимостей переменная может содержать разные значения для различных выполнений поиска зависимостей. Эти переменные используются при указании выражений поиска для определения наличия или отсутствия строки подключения в документе конфигурации. Поскольку строки подключения разнятся от поставщика к поставщику, переменные позволяют задавать общие выражения поиска независимо от конкретных значений строк подключения.

Переменные

Примечание. Термин "переменная" относится как к самим переменным, так и к переменным понятий.

Синтаксис использования переменной — ${VARIABLE_NAME}. Значением каждой переменной должна быть строка или список строк. Например, частью строки подключения может быть IP-адрес (или адреса) поставщика. Для поиска IP-адреса в документе конфигурации можно задать переменную с именем IP_ADDRESS и использовать ее следующим образом: ${IP_ADDRESS}.

Сначала переменные должны быть заданы в области всего файла сигнатуры зависимости (глобальной области) или в области конкретной зависимости (локальной области). Определение переменной позволяет в дальнейшем использовать эту переменную.

Переменные, которые не были заданы, но используются, генерируют ошибку при развертывании файла сигнатуры.

Глобальная область

Переменная глобальной области может использоваться любой зависимостью в файле, в котором она задана. Чтобы задать переменную глобальной области:

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

В приведенном выше коде задаются две переменных глобальной области — IPADDRESS и PROTOCOL. Как и другие переменные, они могут содержать строку или список строк.

Значения глобальных переменных могут быть заданы только целевыми данными триггера.

Локальная область

Переменная локальной области может использоваться только зависимостью, для которой она задана. Эта переменная невидима для любой другой зависимости, как внутри одного развертываемого компонента, так и в разных компонентах.

Можно задать несколько переменных локальной области с одинаковыми именами в разных зависимостях. Каждая из этих переменных является уникальной, и ее значение(я) может использоваться только в той области, для которой оно задано.

Примечание. Для переменной локальной области можно задать имя, которое будет отличаться от имени переменной глобальной области, либо задать одно и то же имя для двух переменных локальной области внутри одной зависимости.

Чтобы задать переменную локальной области, используйте следующий синтаксис:

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

Значения для переменных локальной области могут присваиваться только при помощи операторов вставки. Существуют различные специальные операторы вставки, зависящие от типа файла, из которого извлекается значение. Оператор вставки может присутствовать в документе конфигурации в двух местах.

- В отдельном разделе `<Variables>`. Операторы вставки в разделе `<Variables>` проверяются, только если выражение поиска всего файла выполнено со значением true.
- В разделе `<Variables>` как часть условия поиска. В этом случае переменная будет вставлена, только если условие поиска будет выполнено со значением true. При использовании альтернативных выражений это не поддерживается. Подробнее см. в разделе "Использование значений переменных по умолчанию" на странице 82.

Подробнее о назначении переменных см. в разделе:

- "Документы конфигурации PROPERTIES" на странице 90
- "Документы конфигурации XML" на странице 94
- "Текстовые документы конфигурации" на странице 97

Понятия

Условия поиска применяются к каждому соответствующему документу конфигурации. Некоторые значения в строках подключения тесно связаны друг с другом, и их следует использовать в конкретных выражениях поиска только в такой связке. Каждый такой набор связанных строк подключения должен иметь уникальное имя, и он считается понятием.

Например, к развертываемому компоненту поставщика можно получить доступ с адреса `1.1.1.1:8080` или `2.2.2.2:85`. Понятно, что каждый потребитель (по отношению к поставщику) в своих файлах конфигурации также должен иметь адрес `1.1.1.1:8080` или `2.2.2.2:85`. Однако вряд ли в документе конфигурации какого-либо потребителя данного поставщика можно найти адрес `1.1.1.1:85`, поскольку этот поставщик не выполняет прослушивание адреса `1.1.1.1:85`. Даже если адрес `1.1.1.1:85` присутствует в одном из документов конфигурации потребителя, он отображает не зависимость от данного поставщика, а зависимость от какого-то другого развертываемого компонента, запущенного на том же узле, что и поставщик, и выполняющего прослушивание адреса `1.1.1.1:85`.

Операция поиска должна найти корректный IP-адрес или порт, в противном случае ею будут найдены ложные положительные зависимости.

Кроме того, в процессе поиска также должны искаться правильные совпадения имени IP-адреса и порта, поскольку каждый IP-адрес может иметь несколько имен (например, одно полномочное имя DNS и несколько псевдонимов).

Поэтому велика вероятность нахождения следующих совпадений или пар в документах конфигурации потребителей данного поставщика: `XYZ:8080`, `FOO:8080`, или `ABC:85` (где `XYZ`, `FOO` и `ABC` — это другие имена для адреса поставщика), и эти совпадения отображают зависимости от этого поставщика. Однако совпадение `ABC:8080` не отображает зависимости от этого поставщика, и поэтому оно не должно разыскиваться.

В этих целях используются понятия. Каждый атрибут строки подключения, который должен использоваться в связке с другим атрибутом, следует определить внутри одного и того же понятия. Каждый атрибут строки подключения должен быть переменной в этом понятии.

Понятия могут определяться только в рамках глобальной области. Каждый экземпляр понятия, представляющий набор тесно связанных между собой строк подключения, должен иметь значение, заданное адаптером (также, как и для переменных).

Каждое понятие состоит из одной переменной key и дополнительных переменных. Каждая из этих переменных (key и дополнительных) используется для хранения тесно связанных между собой строк подключения. Переменная key используется для различия разных экземпляров одного и того же понятия. Это значит, что не может существовать двух экземпляров одного понятия, имеющих одинаковые значения переменной key. Подробнее см. в разделе "Указание значений для переменных понятий" на странице 109.

Примечание. Переменная key может использоваться точно так же, как любая другая переменная понятия. Ее важность заключается в том, каким образом создаются экземпляры понятия в ходе выполнения триггера.

Для определения понятия и его переменных используйте следующий синтаксис:

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

Синтаксис использования переменной понятия в выражении поиска:

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

Для каждого понятия, используемого в выражении поиска, должен присутствовать логический оператор, содержащий все относящиеся к понятию переменные и не содержащий переменных иного понятия.

Далее приведены примеры допустимых выражений поиска, содержащих понятия:

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

А вот пример недопустимого выражения поиска:

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

Значения по умолчанию

Глобальные переменные и переменные понятий, при необходимости, могут иметь значение по умолчанию. Если переменная вставляется из адаптера со значением, совпадающим со значением по умолчанию, можно задать альтернативные выражения

поиска. Подробнее об альтернативных выражениях поиска см. в разделе "Составление выражения поиска" на странице 79.

Для указания значения по умолчанию используется следующий синтаксис:

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

Типы переменных IP-адресов

IP-адрес является важным компонентом строки подключения; при этом адрес поставщика не всегда может указываться в файлах конфигурации.

Типичной ситуацией для этого может стать расположение потребителя и поставщика на одном и том же компьютере-хосте. В таком случае вместо адреса поставщика может отображаться строка вроде "localhost" или "127.0.01", либо адрес будет отсутствовать вовсе.

Выделяя переменную с типом **IP Address**, платформа будет автоматически пытаться найти локальные зависимости, игнорируя переменные IP-адресов и находя поставщиков с остальными строками подключения, которые расположены на том же хосте.

Выделение переменной как IP-адреса

- Для глобальной переменной

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

- Для переменной понятия

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

Примечание. Локальные переменные не могут иметь тип "IP Address".

Определение дескриптора потребителя

Чтобы определить зависимость потребитель-поставщик, адаптер поиска ищет потребителей с документами конфигурации, которые используют строку подключения поставщика. При этом иногда стоить ограничить развертываемые компоненты, которые

могут быть потребителями для конкретной службы, даже если документы конфигурации потребителя содержат строку подключения или имеют различные выходные переменные в зависимости от свойств развертываемого компонента.

В этой связи развертываемый компонент можно описать с большей степенью детализации, нежели как просто тип ЭК, при помощи элемента `<Descriptor>`. Развертываемый компонент описывается следующими элементами:

- Тип ЭК — зависимости относятся только к развертываемому компоненту с типом "J2EE Application". (Обязательно)
- Условия для атрибутов его строк; например, для развертываемых компонентов с типом "J2EE Application", там где имя приложения — "MyShop". Это условие может тестировать только на наличие равенства. (Необязательно)
- Обязательная связь типа Composition с другим ЭК. Если связь Composition указана в дескрипторе, развертываемый компонент должен иметь такую связь с ЭК заданного типа. (Необязательно)
- Условия для атрибутов строк связанного ЭК, если он был указан (как в предыдущем варианте). (Необязательно)
- Путь (TQL-запрос) в узел, содержащий развертываемый компонент. Имя запроса указывается атрибутом **nodeToDeployableQuery**. Если узел связан непосредственно с развертываемым компонентом связью Composition, нет необходимости упоминать TQL-запрос в пути. Однако, если узел используется для описания развертываемого компонента (с условиями или без), его все же следует упомянуть при помощи тега `<ConnectedCICondition>`. Если у развертываемого компонента отсутствует родительский узел в иерархии, это следует указать при помощи атрибута `hasContainingNode = "false"`. При указании пути также следует добавить атрибут `hasContainingNode = "true"`. Подробнее см. в разделе "Определение TQL-запросов" на странице 102. (Необязательно)

Примечание. Только тип атрибута STRING поддерживается для развертываемых дескрипторов или подключаемых ЭК. Атрибут не может быть статическим и не может содержать вычисляемые атрибуты.

Пример дескриптора, который указывает только тип ЭК:

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

Пример дескриптора с атрибутом строки:

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

Пример дескриптора с подключенным ЭК для тестирования равенства без учета регистра:

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

Примечание. Для ConnectedCiCondition можно использовать только linkType=”composition” и isDirectionForward=”true”.

Определение зависимостей

Для каждого потребителя можно определить несколько зависимостей. Каждая зависимость ассоциируется с определенным типом ЭК поставщика. Во время проверки сигнатуры зависимости для ЭК поставщика оцениваются все зависимости, связанные с типом ЭК поставщика. Каждая зависимость имеет выражение поиска из одного или нескольких документов конфигурации. Если выражение поиска получает значение true, значит существует зависимость (связь "потребитель-поставщик") между потребителем и поставщиком. Даже если существует несколько зависимостей между одними и теми же типами ЭК потребителя и поставщика, и эти зависимости получают значение true, создается только одна связь.

Пример синтаксиса зависимости:

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

Каждая зависимость также имеет свою область. Область — это отдельные потребители из общего числа соответствующих дескриптору, которые относятся к данной конкретной зависимости. Подробнее см. в разделе "Определение дескриптора потребителя" на странице 77.

Составление выражения поиска

Примечание. Только тип атрибута STRING поддерживается для развертываемых дескрипторов или подключаемых ЭК. Атрибут не может быть статическим и не может содержать вычисляемые атрибуты.

Выражение поиска составляется из логических операторов и условий. Поддерживаемые логические операторы — "And" и "Or".

Условия — это выражения, которые проверяются при помощи строк подключения поставщика и документов конфигурации потребителя. Условия различаются в зависимости от типа файла, например, документы конфигурации PROPERTIES имеют условия, отличные от документов XML.

Ниже приведен пример выражения поиска для документа конфигурации PROPERTIES: Создание зависимости между поставщиком и потребителем (только если IP-адрес поставщика существует в документе конфигурации потребителя **MyConfig.properties** под ключом IPADDRESS). В документе конфигурации зависимости будет записано следующее:

```
<PropertiesConfigurationDocument name="MyConfig.properties"> (Тип и имя
документа конфигурации)
<Condition> (Начало выражения поиска)
<Operator type="and"> (Оператор)
<KeyCondition key="IPADDRESS"> (Начните условие и укажите его тип)
<Values>
<Value>${IP_ADDRESS}</Value> (Используйте переменную, указывающую IP-адрес
поставщика)
</Values>
</KeyCondition>
</Operator>
</Condition>
</PropertiesConfigurationDocument>
```

Примечание. В элементе <Condition> должен присутствовать хотя бы один элемент <Operator>, даже если в этом нет логической необходимости, поскольку существует только одно условие поиска, как показано на примере выше.

Давайте рассмотрим более сложное выражение поиска: IP-адрес поставщика существует в документе конфигурации потребителя **MyConfig.properties** под ключом IPADDRESS, либо имя хоста поставщика существует в том же файле под ключом HOST:

```
<PropertiesConfigurationDocument name="MyConfig.properties"> (Тип и имя
документа конфигурации)
<Condition> (Начало выражения поиска)
<Operator type="or"> (Оператор)
<KeyCondition key="IPADDRESS"> (Начните условие и укажите его тип)
<Values>
<Value>${IP_ADDRESS}</Value> (Используйте переменную, указывающую IP-адрес
поставщика)
</Values>
</KeyCondition>
<KeyCondition key="HOST"> (Начните еще одно условие под тем же оператором)
<Values>
<Value>${HOSTNAME}</Value> (Используйте переменную, указывающую имя хоста
```

```
поставщика)
</Values>
</KeyCondition>
</Operator>
</Condition>
</PropertiesConfigurationDocument>
```

Логические операторы могут вкладываться для создания таких условий, как `(C1 AND (C2 OR C3))`, и XML будет выглядеть следующим образом:

```
<PropertiesConfigurationDocument name="MyConfig.properties"> (Тип и имя
документа конфигурации)
<Condition>
<Operator type="and">
C1
<Operator type="or">
C2
C3
</Operator>
</Operator>
</Condition>
</PropertiesConfigurationDocument>
```

где `C1`, `C2` и `C3` — это XML-фрагменты требуемых условий поиска.

Существует три поддерживаемых типа файлов, и каждый из них имеет свое выражение поиска:

- PropertiesConfigurationDocument – файл, в котором каждая строка имеет формат `key=value`. Подробнее см. в разделе "Документы конфигурации PROPERTIES" на странице 90.

  KeyCondition — условие для значения определенного ключа
- XmlConfigurationDocument — XML-файл. Подробнее см. в разделе "Документы конфигурации XML" на странице 94.

  XPathCondition — проверка запроса XPath
- TextConfigurationDocument — любой текстовый документ конфигурации. Подробнее см. в разделе "Текстовые документы конфигурации" на странице 97.

  RegExp — проверка регулярного выражения

Подробнее о значениях по умолчанию см. в разделе "Использование значений переменных по умолчанию" на следующей странице.

Устранение неполадок

- Наличие различных понятий в одном и том же узле дерева условий не допускается. Точно так же не допускается наличие двух конечных узлов с различными понятиями под одним и тем же оператором.

Например, следующее определение условия является недопустимым:

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

Использование значений переменных по умолчанию

В некоторых случаях файлы конфигурации могут не содержать значение определенной строки подключения, если это значение эквивалентно какому-либо значению по умолчанию. Например, при определении URL-адреса HTTP номер порта 80 может не указываться отдельно в URL-адресе. Это значит, что файл конфигурации скорее будет содержать адрес http://www.hp.com/, нежели http://www.hp.com:80/, хотя оба адреса будут верными. Это может создать проблему при создании условия поиска, так как если переменная PORT содержит значение 80, и условие требует, чтобы такое значение существовало, тестирование URL-адреса закончится сбоем, если значение не будет указано.

Для решения этой проблемы каждая переменная, при необходимости, может иметь значение по умолчанию. Выражения поиска можно изменять в случае, если значение переменной совпадает со значением по умолчанию.

Существует два способа изменения выражения поиска:

- Игнорирование условия, если значение переменной совпадает со значением по умолчанию

  Используйте этот вариант, чтобы полностью игнорировать условие в случае, если у одной или нескольких переменных в условии поиска значение совпадает со значением по умолчанию. Если условие игнорируется, оно не возвращает значений true или false, и результат логического оператора зависит от оператора и других операндов. Пример:

  - В случае, если условие имеет форму **A и B**, где **A** и **B** являются разными выражениями, если **A** игнорируется, результаты будут совпадать с результатами **B**.
  - Для выражения: **A и B и C**, если **A** игнорируется, результат будет тот же, что и для **B и C**.
  - Для выражения: **A или B или C**, если **A** игнорируется, результат будет тот же, что и для **B или C**.

В тех редких случаях, когда все дочерние условия игнорируются, а значит игнорируется все условие целиком, результатом условия документа конфигурации будет false.

Чтобы игнорировать условие, добавьте в него атрибут **ignoreIfDefaultValue** вместе со списком переменных, которые будут инициировать игнорирование этого условия. Пример:

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

- Использование другого условия, если значение переменной совпадает со значением по умолчанию

  Продолжим рассматривать пример с портом 80 и URL-адресом. Используйте следующее регулярное выражение для тестирования URL-адреса:

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

  Понятно, что это выражение никогда не получит значения true для таких строк, как http://www.hp.com/, поскольку в строке отсутствует двоеточие (:). Поэтому нам также понадобится протестировать следующее регулярное выражение, если переменная PORT имеет значение **80**:

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

  В подобных случаях можно воспользоваться альтернативными выражениями. Чтобы задать альтернативное выражение, задайте несколько выражений поиска с одними и теми же условиями, и укажите, которое из них будет использоваться, если переменная имеет значение по умолчанию. Пример:

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

  Использование атрибута **alternativeFor** указывает на то, что если переменная имеет значение по умолчанию, вместо исходного выражения будет проверяться альтернативное выражение. Исходным является выражение без атрибута **alternativeFor** или с пустым атрибутом **alternativeFor**.

  Для всех переменных, которые присутствуют в выражении и имеют значения по умолчанию, следует задать альтернативное выражение, которое бы охватывало любые комбинации таких переменных. В противном случае может возникнуть ошибка компиляции. Если имеется только одна переменная, потребуется всего одно альтернативное выражение (как в примере выше). Если имеется несколько переменных со значениями по умолчанию, потребуется несколько альтернативных выражений.

  К примеру, если переменная DOMAIN также имеет значение по умолчанию, использование оператора **alternativeFor** означает наличие следующих альтернативных условий:

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

Только одна из этих комбинаций будет проверяться в ходе выполнения.

Подробнее об указании значений по умолчанию см. в разделе "Значения по умолчанию" на странице 76.

Указание путей к документам конфигурации

Путь к документу конфигурации — это не путь к файлу в файловой системе хоста, а топологический путь между развертываемым потребителем и документом конфигурации.

По умолчанию, если путь не задан, считается, что документ конфигурации соединен с развертываемым компонентом связью типа Composition. Другими словами, считается, что ЭК развертываемого компонента владеет ЭК документа конфигурации.

Чтобы указать другой путь:

1. Создайте TQL-запрос, как указано в разделе "Определение TQL-запросов" на странице 102.
2. Создайте ссылку на TQL-запрос из документа конфигурации при помощи элемента `<DocumentCILocation>` следующим образом:

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

В данном примере YourQueryName является ссылкой на имя запроса в разделе `<Queries>`.

При создании TQL-запроса для указания пути:

- TQL-запрос должен определять путь между развертываемым компонентом и документом конфигурации. Путь должен быть простым (без циклов).
- TQL-запрос должен содержать следующие два конечных узла с особыми именами (с учетом регистра):
  - Deployable — ЭК развертываемого компонента в пути
  - Configuration_document — ЭК в пути, указывающий на документ конфигурации

- Условия для узлов Deployable и Configuration_document не допускаются.
- Размерности между всеми узлами должны быть вида 1..1.
- Поддерживается только обычный тип связи. Составные связи, связи объединения и подграфы не допускаются.

Переопределение документа конфигурации

В некоторых ситуациях документ конфигурации может переопределять другие документы, либо сам может быть переопределен ими. Например, документ конфигурации, существующий на хосте, может переопределить документы конфигурации, существующие в кластере, к которому принадлежит хост. В таких случаях значения ключей должны проверить все файлы, которые могут переопределять значения, и выбрать один из них с самым высоким приоритетом. В данном примере локальный документ конфигурации хоста имеет более высокий приоритет, нежели документ кластера.

Для переопределения файлов в сигнатуре зависимости необходимо использовать следующий синтаксис, позволяющий добавить несколько путей с приоритетом в документ конфигурации:

```
<PropertiesConfigurationDocument name="resources.xml">
<DocumentCILocation>
<ReferenceLocation priority="2">websphereas_resource_
configfiles</ReferenceLocation>
<ReferenceLocation priority="3">j2ee_cluster_configfiles</ReferenceLocation>
<ReferenceLocation priority="1">j2eeapplication_configfiles</ReferenceLocation>
</DocumentCILocation>
…
</PropertiesConfigurationDocument
```

Имя файла (**resources.xml** в примере выше) должно быть одним и тем же во всех связанных расположениях.

Не забывайте, что связанное расположение — это ссылка на TQL-запрос, которая указывает топологический путь от развертываемого компонента потребителя к документу конфигурации. Это также означает, что такой путь должен существовать между развертываемым компонентом и всеми другими расположениями. Подробнее о <DocumentCILocation> см. в разделе "Указание путей к документам конфигурации" на предыдущей странице.

При добавлении нескольких путей с приоритетами само условие не изменяется. В ходе выполнения (при проверке выражения поиска) используются корректные значения в соответствии с приоритетами. Например, в файлах свойств, если Ключ К существует в приоритете 1 и приоритете 2 (с условием для его значения), условие будет проверяться только для документа с приоритетом 1. Если ключ К существует только в приоритете 2, условие будет проверяться только для документа с приоритетом 2.

Подробнее об условиях свойств см. в разделе "Документы конфигурации PROPERTIES" на странице 90.

В документах XML все XPath считаются ключами. К примеру, `\Root\Element\@Attribute` означает, что атрибут "Attribute" с таким путем является ключом, и его значение может быть переопределено в разных файлах.

Если у XPath также имеется константное условие (условие, в котором нет переменных), такое условие будет частью ключа. Пример: `\Root\Element[@name = 'name']\@Attribute`.

При этом, если у XPath имеется неконстантное условие, такие условия будут извлекаться из ключа и считаться частью значения. Таким образом, для пути `\Root\Element[@name = ${NAME}]\@Attribute` ключом будет путь `\Root\Element\@Attribute`, и условие `Element [@name = 'name']` будет проверяться только для документа с самым высоким приоритетом, в котором такой ключ существует.

Если документ XML использует приоритеты, важно, чтобы условия XPath создавались так, как описано выше, во избежание недопустимого и непредвиденного поведения.

Подробнее об условиях XPath см. в разделе "Документы конфигурации XML" на странице 94.

Примечание. Наличие нескольких связанных расположений с приоритетами допускается только для документов конфигурации PROPERTIES и XML. Для текстовых документов они не допускаются.

Несколько документов конфигурации с одинаковым приоритетом

Существует возможность назначения одного и того же приоритета различным элементам `<ReferenceLocation>`. В этом случае при проверке условия платформа будет изучать все файлы с одинаковым приоритетом и пытаться сопоставить их между собой.

Условие будет выполнено со значением true если хотя бы указанное количество файлов будет иметь значение true. Количество файлов определяется атрибутом **samePriorityMatchAtLeast**.

Пример:

```
<PropertiesConfigurationDocument name="resources.xml">
<DocumentCILocation samePriorityMatchAtLeast="1">
<ReferenceLocation priority="1">webspheras_resource_
configfiles</ReferenceLocation>
<ReferenceLocation priority="1">j2ee_cluster_configfiles</ReferenceLocation>
<ReferenceLocation priority="1">j2eeapplication_configfiles</ReferenceLocation>
</DocumentCILocation>
…
</PropertiesConfigurationDocument>
```

Это означает, что при проверке условия хотя бы одно связанное расположение ведет к файлу, в котором условие выполняется со значением true.

Примечание. В настоящий момент для атрибута **samePriorityMatchAtLeast** поддерживается только значение **1**.

Зависимости, определенные в нескольких документах

В многих случаях для определения того, предоставляет ли потребитель службу, требуется поиск всех строк подключения поставщика в нескольких документах конфигурации.

Поиск в нескольких несвязанных документах конфигурации

Документы конфигурации могут быть не связаны друг с другом, при этом некоторые строки подключения могут быть в одних документах, а некоторые — в других. Например, у потребителя могут быть два документа конфигурации, **A.conf** и **B.conf**. Строки подключения поставщика — C1, заданная в **A.conf**, и C2, заданная в **B.conf**. Обе строки, C1 и C2, необходимы для определения того, что между потребителем и поставщиком действительно существует зависимость. В этом случае имеются два обязательных выражения поиска, по одному в каждом документе. Тег <Dependency> будет иметь следующую структуру:

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

Примечание.

- Элемент <Dependency> может содержать неограниченное количество файлов.
- Каждый из файлов может иметь отдельный тип (PROPERTIES, XML или текстовый файл).
- Порядок размещения файлов в элементе <Dependency> игнорируется. Тестирование файлов осуществляется в произвольном порядке.

- Для существования зависимости выражения поиска из всех файлов должны возвращать значение true.
- Поскольку строки подключения распределены по нескольким файлам, выражение поиска области должно быть достаточно широким для того, чтобы как минимум один требуемый файл мог быть возвращен из области. Подробнее см. в разделе "Указание области поиска" на странице 99.

Суммируя весь процесс поиска в нескольких файлах:

1. Область фильтрации определяется в соответствии с описанием в разделе "Указание области поиска" на странице 99. Если не возвращено ни одного требуемого файла, зависимость между потребителем и поставщиком не существует.
2. Если фильтром возвращен хотя бы один из файлов, потребитель, подключенный к этому файлу, и оставшиеся требуемые конфигурации загружаются в зонд потока данных.
3. Условия из каждого файла оцениваются в произвольном порядке.
4. Если все условия возвращают значение true, зависимость существует, в противном случае — нет.

Поиск в нескольких документах конфигурации с зависимостями

В других случаях документы конфигурации потребителя связаны друг с другом. Например, файл **DBConnections.conf** определяет несколько строк подключения к нескольким базам данных и схемам, и каждой такой строке присваивается имя. В файле **MyApp.conf** используется одно из этих имен подключения, и оно указывает на использование MyApp определенной базы данных или схемы.

Переменные локальной области используются в документах конфигурации с зависимостями. Значения переменных локальной области вставляются (при помощи оператора вставки) в один из документов конфигурации, после чего значение переменной (например, имя строки подключения) будет существовать в этом документе. Затем эта переменная может использоваться в условиях другого документа конфигурации (например, использование имени подключения в файле **MyApp.conf** для того, чтобы MyApp использовало поставщика (БД)). Подробнее см. в разделе "Локальная область" на странице 74.

Используйте операторы вставки для вставки значений в переменные. Для каждого типа документа существует свой оператор:

- Файлы PROPERTIES — KeyVariable и KeyRegExpVariable. Подробнее см. в разделе "Документы конфигурации PROPERTIES" на странице 90.
- XML-файлы — XPathVariable. Подробнее см. в разделе "Документы конфигурации XML" на странице 94.
- Текстовые файлы — RegExpVariable. Подробнее см. в разделе "Текстовые документы конфигурации" на странице 97.

В этом случае имеются два обязательных выражения поиска, по одному в каждом файле. Поэтому тег <Dependency> будет иметь следующую структуру:

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

Примечание.

- Используйте элемент `<VariableDeclarations>`, чтобы задать одну или несколько локальных переменных для использования в зависимости.
- Элемент `<Variables>` используется для вставки значений в локальные переменные. Раздел выполняется, только если элемент `<Condition>` возвращает значение true.
- В данном примере `<KeyVariable>` используется для вставки значения ключа "reference_name" в переменную REFERENCE_NAME.
- Переменная `${REFERENCE_NAME}` используется в условии **MyApp.conf**. Эта переменная зависит от файла **DBConnections.conf**, она будет проверяться только после файла **DBConnections.conf**, и только если выражение поиска возвращает значение true.
- Циклические зависимости между файлами не допускаются.
- При проверке документа **MyApp.conf** переменная `${REFERENCE_NAME}` уже содержит значение из **DBConnections.conf**.

- Поскольку строки подключения распределены по нескольким файлам, выражение поиска области должно быть достаточно широким для того, чтобы файлы без зависимостей могли быть возвращены из области. Подробнее см. в разделе "Указание области поиска" на странице 99.

Суммируя весь процесс поиска в нескольких файлах с взаимными зависимостями:

1. Область фильтрации определяется в соответствии с описанием в разделе "Указание области поиска" на странице 99. Если не возвращено ни одного требуемого файла, или ни один из файлов не зависит от других файлов, зависимость между потребителем и поставщиком не существует.
2. Хотя бы один файл без зависимостей возвращается фильтром.
3. Потребитель, подключенный к этому файлу, и оставшиеся требуемые конфигурации загружаются в зонд потока данных.

Оператор вставки может не вернуть значение, например, если ключ, на который он ссылается, не существует. В этом случае переменная получит пустое значение. Однако, такое развитие событий может быть нежелательным. Если требуемое значение не существует, это может означать отсутствие зависимости и, соответственно, смысла продолжать проверку последующих файлов. Для данного сценария используйте `allowNull=”false”` в операторе вставки. Пример:

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

Документы конфигурации PROPERTIES

Документы конфигурации **PROPERTIES** хранят конфигурацию в формате `Key=value`. Например, документ конфигурации **PROPERTIES** может иметь следующий вид:

```
# Мой документ конфигурации
Hostname=MyNode
```

где `Hostname` — это ключ, а `MyNode` — связанное с ним значение.

"#" обозначает строку комментария. Строки комментариев игнорируются при тестировании выражения поиска.

Для указания того, что документ конфигурации является документом **PROPERTIES**, используйте элемент `<PropertiesConfigurationDocument>`. Пример:

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

Определение условий

Существует несколько способов тестирования на наличие переменных строк

подключения в файлах свойств. Для всех типов условий значение key, указанное в атрибуте **key**, сопоставляется с каким-то другим значением. Значение key всегда является константным (переменные не допускаются):

- Тестирование на наличие значения константы в качестве значения key.

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

Условие возвращает значение true, только если значение key равно (с учетом регистра) одному из значений константы.

- Тестирование на наличие значения переменной в качестве значения key.

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

Условие возвращает значение true, только если значение key равно (с учетом регистра) одному из значений переменной. Если значением переменной является список значений, тестируются все значения, и для возврата значения true достаточно совпадения значения key с одним из значений в списке.

Примечание. И значения переменных, и значения константы можно указать в одном и том же условии.

- Тестирование на соответствие значения key определенному регулярному выражению.

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

Регулярное выражение также может содержать одну или несколько переменных как часть условия. В ходе выполнения при создании конкретного выражения поиска переменные заменяются своими фактическими значениями.

Если переменная содержит список значений, создается выражение, подобное следующему:

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

В данном случае любое из значений может вернуть для условия значение true.

Пример условий в документе конфигурации PROPERTIES

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

Это означает, что документ конфигурации **MyConfig.properties** должен содержать все из перечисленного:

- Оператор key с именем TYPE, в котором должно быть значение HTTP или HTTPS.
- Оператор key с именем IPADDRESS, в котором должен быть IP-адрес поставщика (или один из его IP-адресов), либо имя его хоста.
- Оператор key с именем SITE, который должен начинаться в символа "/", далее содержать SITE_NAME поставщика, еще один символ "/" и любую строку.

Вставка значений переменных

Существует два способа извлечения значения или части значения key в переменную:

- Вставка значения key.
  - В отдельном разделе `<Variables>` необходимо использовать следующий синтаксис:

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

- Используйте следующий синтаксис как часть условия поиска:

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

Ключ переменной VARIABLE_NAME определяется в элементе <KeyCondition>.

При использовании <RegExp> вместо <Values> синтаксис будет тем же. В обоих случаях переменная будет вставляться с полным значением key.

- Вставка части значения key.
  - В отдельном разделе <Variables> необходимо использовать следующий синтаксис:

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

В данном случае GROUP_NUMBER — это индекс (начинающийся с 0) группы в регулярном выражении, заданном REGULAR_EXPRESSION. Например, в документе PROPERTIES, содержащем строку

```
myKey = 123Value123,
```

оператор

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

вставляет значение "Value" в переменную "Var".

Можно использовать переменные, заданные в том же файле, или в других файлах. Пример:

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

Исходя из того, что PREV_VAR содержит значение "123", переменная VAR получит значение "Value" точно так же, как и при вставке всего значения key. В любом случае переменная всегда будет содержать единственное значение.

- Используйте следующий синтаксис как часть условия поиска:

```
<KeyCondition key=”KEY_NAME”>
```

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

Ключ переменной **VARIABLE_NAME** определяется в элементе `<KeyCondition>`.

Это поддерживается только при использовании `<RegExp>`, и регулярное выражение, на которое ссылается атрибут переменной "**group**", являются тем же, что и в `<RegExp>`.

Документы конфигурации XML

С документом конфигурации **XML** можно использовать запросы **XPath**, обеспечивающие удобство создания выражений поиска для документов конфигурации на основе стандарта **XML**.

Для указания того, что документ конфигурации является документом **XML**, используйте элемент `<XmlConfigurationDocument>`. Пример:

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

Определение условий

Единственными допустимыми условиями поиска для документов конфигурации **XML** являются действительные запросы **XPath 2.0**. Если узел был выбран из запроса **XPath**, условие возвращает значение **true**, в противном случае — **false**. Запрос может содержать переменные из следующих расположений:

- В именах элементов

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

  Если переменная содержит несколько значений, платформа запускает несколько операторов **XPath**. Например, если `SITE_NAME` имеет значения **SITE1** и **SITE2**, оператор генерирует следующие два конкретных поиска:

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

- В тестах равенства

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

Равенство, как показано на примерах выше, работает только для переменных, содержащих единичные значения. Если в переменной может содержаться более одного значения, необходимо использовать следующий синтаксис:

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

Примечание.

- Используйте `${equals()}` для тестирования равенства с учетом регистра, либо `${equals-ignore-case()}` для равенства без учета регистра.
- Первый параметр должен быть функцией XPath или именем атрибута.
- Второй параметр всегда должен быть именем переменной. В имени переменной не должны присутствовать символы `${}`.
- Этот синтаксис работает для переменных с одним или несколькими значениями, и рекомендуется к использованию во всех случаях.
- Если имеется несколько значений, и значение первого параметра равно только одному из значений переменных, функция возвращает значение true.

- В регулярных выражениях

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

Если переменная содержит список значений, создается выражение, подобное следующему:

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

Каждое из значений возвращает для условия значение true.

Пример условий в документе конфигурации XML

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

```
<XPath>\URL\Site[matches(text(), '//${SITE_NAME}//.*')]</XPath>
</XPathCondition>
</Operator>
</Condition>
</XmlConfigurationDocument>
```

В примере показано, что документ конфигурации XML **MyConfig.xml** должен содержать все из перечисленного:

- В `\URL\Protocol\@name` должно быть значение HTTP или HTTPS.
- В `\URL\Host\@name` должен быть IP-адрес поставщика (или один из его IP-адресов), либо имя его хоста.
- Оператор `\URL\Site\text()` должен начинаться в символа "/", далее содержать SITE_NAME поставщика, еще один символ "/" и любую строку.

Вставка значений переменных

Запросы XPath можно использовать для извлечения текстовых значений и их вставки в переменные. Вставлять целые элементы в переменные невозможно. Для выполнения этого в отдельном разделе `<Variables>` необходимо использовать следующий синтаксис:

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

Вот несколько примеров:

- Выбор значения атрибута

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

  выбирает значения из атрибута **Att** из всех элементов \Root\Element в документе XML.
- Выбор текста элемента

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

  выбирает текст всех элементов, соответствующих \Root\Element.
- Выбор значения атрибута с условиями (с использованием переменных из одного или разных документов конфигурации)

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

  выбирает значение @AnotherAtt из всех элементов \Root\Element с `@Att=${VAR}`.

Переменные XPath можно использовать как часть условий XPath. В этом режиме XPath также может содержать относительный путь из узла результата условия, если такой результат существует (условие выполнено со значением true). Используйте следующий синтаксис:

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

Пример:

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

Если условие /XPathCondition выполнено со значением true, и какой-то узел XML (в данном случае — какой-то элемент) был выбран `<XPath>`, переменная будет содержать значение атрибута "type" этого узла.

XPath можно определить для переменной, которая не зависит от такой переменной, указав XPath из корневого элемента документа с символом "/" в начале.

После вставки XPath переменная может содержать список значений.

Текстовые документы конфигурации

Текстовый документ конфигурации — это документ с известным разработчику форматом, но не являющийся документом конфигурации формата PROPERTIES или XML. Для записи выражений поиска в текстовых документах конфигурации можно использовать регулярные выражения.

Для указания того, что документ конфигурации является текстовым документом, используйте элемент `<TextConfigurationDocument>`. Пример:

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

Определение условий

Единственными допустимыми условиями поиска для текстовых документов конфигурации являются регулярные выражения. Если образец совпадает, условие выполняется со значением true.

Регулярные выражения могут содержать одну или несколько переменных как часть условия. В ходе выполнения при создании конкретных выражений поиска переменные заменяются своими фактическими значениями.

Если переменная содержит список значений, создается выражение, подобное следующему:

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

Каждое из значений возвращает для условия значение true.

Пример условий в текстовом документе конфигурации

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

В примере показано, что текстовый документ конфигурации **MyConfig.txt** должен содержать все из перечисленного:

- В `\URL\Protocol\@name` должно быть значение HTTP или HTTPS.
- В `\URL\Host\@name` должен быть IP-адрес поставщика (или один из его IP-адресов), либо имя его хоста.
- Оператор `\URL\Site\text()` должен начинаться в символа "/", далее содержать SITE_NAME поставщика, еще один символ "/" и любую строку.

Вставка значений переменных

Для вставки значения в переменную в текстовых документах можно использовать регулярные выражения с группами. Группы выделяют текст, который должен быть вставлен. В отдельном разделе `<Variables>` необходимо использовать следующий синтаксис:

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

В данном случае GROUP_NUMBER — это индекс (начинающийся с 0) группы в регулярном выражении, заданном REGULAR_EXPRESSION.

Например, в документе, содержащем строку

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

оператор

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

добавляет значение "document" в переменную VAR. Можно использовать переменные, заданные в разных документах, вместе внутри одного документа. Пример:

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

Если PREV_VAR имеет значение "configuration", VAR вставляется вместе со строкой "document".

Можно использовать `<RegExpVariable>` как часть тега `<RegExpCondition>` при помощи следующего синтаксиса:

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

Регулярные выражения, на которые ссылается атрибут переменной "group", являются теми же, что и в `<RegExp>`.

Переменная всегда вставляется с единственным значением.

Указание области поиска

Область поиска служит для идентификации всех потребителей, которые потенциально могут быть частью зависимости от конкретного типа поставщика. Область может использоваться для двух целей:

- Удаление ненужных развертываемых компонентов из зоны поиска в UCMDB для сокращения количества результатов. (Обязательно)

  Для этого выполняется попытка найти подмножество строк подключения поставщика во всех документах конфигурации и ограничить поиск только развертываемыми компонентами, которые связаны с этими документами конфигурации.

  Это выражение поиска отличается от поиска зависимостей типа поставщик-потребитель, поскольку оно должно находить всех потенциальных потребителей максимально быстро, и при этом не возвращать точные данные о том, существует ли зависимость поставщик-потребитель.

  Синтаксис выражения поиска формируется так же, как и для выражений поиска документов конфигурации; однако, в нем отсутствуют какие-либо специальные условия для типов файлов, и нет необходимости указывать имена файлов. Подробнее см. в разделе "Составление выражения поиска" на странице 79.

  В этом примере приведено следующее выражение: `(x | ((y & z & w) & (h | g)))`.

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

x, y, z, w, h и g могут быть значениями констант, переменными или переменными понятий.

Значения (или значения переменных) будут разыскиваться при помощи данного выражения во всех связанных документах конфигурации потребителя в UCMDB.

Только те потребители, у которых имеются такие файлы, будут участвовать в поиске поставщик-потребитель и будут оцениваться с использованием точных выражений поиска.

- Ограничение развертываемых компонентов до подмножества подключенных компонентов поставщика; например, поиск только тех зависимостей, которые являются частью домена J2EE поставщика. (Необязательно)

  В данном случае TQL-запрос используется для поиска всех развертываемых компонентов, которые каким-либо образом подключены к типу поставщика. Подробнее см. в разделе "Определение TQL-запросов" на странице 102.

  Результатом TQL-запроса (с определенным ЭК поставщика) определяются все возможные развертываемые компоненты, которые могут иметь зависимость от поставщика. TQL-запрос должен соответствовать следующим правилам:

  - Узел запроса для развертываемого компонента должен быть назван "Deployable" (с учетом регистра). Результатами данного узла запроса являются возможные развертываемые компоненты для области.

  - Компонент "Deployable" представляет как компоненты потребителя, так и компоненты поставщика, которые могут находиться внутри одной области. Поэтому тип ЭК узла запроса должен быть родительским по отношению к развертываемым типам ЭК потребителя и поставщика.

  - В TQL-запросе может быть только один иной узел запроса (отличный от "Deployable"). Имя другого узла запроса может быть любым. Этот узел запроса имеет тип Scope.

  - Узлы запросов Deployable и Scope соединены связью типа Compound, содержащей все возможные пути между этими узлами.

- Все области явным образом отфильтровывают развертываемые компоненты потребителя, которые не входят в тот же домен маршрутизации, что и поставщик (при условии, что поставщик был обнаружен определенным доменом маршрутизации). Фильтрация доменом маршрутизации требует, чтобы развертываемый дескриптор имел путь к узлу. Подробнее см. в разделе "Определение дескриптора потребителя" на странице 77.

  Например, если развертываемым компонентом поставщика является какой-то тип запущенного программного обеспечения, домен маршрутизации этого программного обеспечения будет тем же, что и содержащийся в нем узел. Домен маршрутизации узла определяется доменом маршрутизации его IP-адресов. Узел может быть связан с несколькими доменами маршрутизации, если его IP-адреса относятся к разным доменам. С другой стороны, если развертываемым компонентом поставщика являет ЭК, который не связан ни с одним узлом или IP-адресом (например, бизнес-служба), он может быть связан с любым развертываемым компонентом потребителя, независимо от его домена маршрутизации.

Например, чтобы определить область домена J2EE, необходимо выполнить следующие действия:

1. Создать узел запроса Deployable с типом "J2EE Deployed Object". Это самый низкий класс общих знаменателей среди всех ЭК, представляющих развертываемые компоненты в домене J2EE. К примеру, развертываемым компонентом в данной области будет веб-модуль или приложение J2EE.
2. Создать узел запроса Scope с типом "J2EE Domain".
3. Создать составную связь между узлами запросов Dependency и Scope со всеми путями между узлами J2EE Deployed Object и J2EE Domain. Составная связь может иметь следующий вид:
   - Приложение J2EE — Composition — Домен J2EE
   - Веб-модуль — Composition — Приложение J2EE

Определение TQL-запроса должно быть встроено в XML области.

Значения переменных по умолчанию при поиске в области

Рассмотрим следующий пример:

- Используется переменная PORT, у которой значение по умолчанию — 80. Это значение может не отображаться в некоторых файлах конфигурации.
- В качестве выражения поиска в области используется `IP_ADDRESS`, и для некоторых поставщиков переменная PORT имеет значение 80. Развертываемые компоненты, связанные с документами конфигурации, которые не содержат номер порта по умолчанию (80), не будут возвращаться в результатах такого поиска (даже если должны), поскольку в выражении поиска указывается на то, что порт должен существовать в файле конфигурации.

Таким образом, если переменная PORT имеет значение по умолчанию, она будет убираться из выражения поиска в области точно так же, как она игнорируется в выражениях поиска зависимостей. Подробнее см. в разделе "Использование значений переменных по умолчанию" на странице 82.

Если переменная содержит значение, отличное от значения по умолчанию, такое значение будет вставлено в выражение поиска стандартным способом.

Устранение неполадок

Для тестирования возможности возврата ожидаемых потребителей из области следует использовать метод JMX **searchConfigurationDocuments** в службе "Диспетчер обнаружения" на сервере UCMDB. Параметр "searchString" должен быть XML-файлом выражения поиска, как описывается в разделе "Указание области поиска" на странице 99. XML должен содержать только значения констант и никаких переменных.

Этот XML также может быть извлечен из журнала связи поставщика в одном из адаптеров поиска. Подробнее см. в разделе "Адаптеры поиска зависимостей" на странице 105.

Определение TQL-запросов

1. Добавьте определение TQL-запроса в сигнатуру зависимости в разделе `<Queries>` следующим образом:

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

2. a. Воспользуйтесь текстовым редактором, чтобы создать пустой документ.
   b. Добавьте следующее содержимое:

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

</tql:query>
```

   c. Создайте TQL-запрос в Студии моделирования. Подробнее см. в разделе "Студия моделирования" в документе *Руководство по моделированию в HP Universal CMDB*.
   d. Экспортируйте TQL-запрос в XML-файл.
   e. Откройте экспортированный XML-документ и скопируйте следующее содержимое в элемент `<resource>`.
   f. Вставьте этот текст в элемент `<tql:query>` в документе, созданном в шаге 1.

g. Скопируйте все содержимое текстового документа и вставьте его в элемент <Query> файла сигнатуры зависимости.

Примечание. TQL-запросы встраиваются в XML-файл сигнатуры зависимости и не подвергаются проверке. Однако если XML-файл запроса сам по себе является недопустимым, во время развертывания будет создано исключение. Подробнее см. в разделе "Ошибки компиляции" на следующей странице.

Упаковка и развертывание нескольких файлов сигнатуры зависимости

В UCMDB может быть развернуто нескольких файлов сигнатуры зависимости. При поиске связей типа "потребитель-поставщик" задействуются все развернутые файлы.

Файлы сигнатуры зависимости — это файлы конфигурации обнаружения, имеющие префикс **dependencies/** и суффикс **.xml**, например, **dependencies/JEE.xml**.

Каждый файл сигнатуры зависимости полностью независим. Следует иметь в виду:

- Определения глобальных переменных могут использоваться только в файле, в котором они заданы. Настоятельно рекомендуется для схожих переменных максимально использовать одни и те же имена. Например, если переменная содержит IP-адрес, и существуют два файла зависимостей, в обоих файлах должна быть задана переменная с именем IP_ADDRESS. В этом случае различные адаптеры поиска смогут использовать одни и те же целевые данные с именем IP_ADDRESS. Подробнее см. в разделе "Указание значений для переменных" на странице 109.
- Определения понятий могут использоваться только в файле, в котором они заданы. Настоятельно рекомендуется для схожих понятий максимально использовать одни и те же имена и переменные. Подробнее см. в разделе "Указание значений для переменных понятий" на странице 109.
- Развертываемые компоненты могут иметь одно и то же имя, если они заданы в разных файлах. Они будут рассматриваться как различные развертываемые компоненты.
- TQL-запросы могут иметь одно и то же имя, если они заданы в разных файлах, но они будут рассматриваться как разные TQL-запросы. При обращении к TQL-запросу по имени в файле сигнатуры зависимости TQL-запрос с этим именем должен существовать в том же файле.
- Области могут иметь одно и то же имя, если они заданы в разных файлах. Они будут рассматриваться как различные области. При обращении к области по имени в файле сигнатуры зависимости область с этим именем должна существовать в том же файле.
- Функции условий могут иметь одно и то же имя, если они заданы в разных файлах. При обращении к функции по имени в файле сигнатуры зависимости функция с этим именем должна существовать в том же файле.
- Значения по умолчанию для переменных и переменных понятий зависят от файлов, в которых они заданы. Две переменных или переменных понятий с одним и тем же именем могут иметь разные значения по умолчанию в различных файлах.
- Не рекомендуется настраивать разные ключевые свойства для одного и того же понятия в различных файлах сигнатуры зависимости. Это усложнит обслуживание и корректное присвоение значений из разных адаптеров их переменным целевых

данных. Подробнее см. в разделе "Указание значений для переменных понятий" на странице 109.

Ошибки компиляции

Проверки компиляции

- Наличие двух и более развертываемых компонентов с одинаковыми именами в одном и том же файле.
- Наличие двух и более областей с одинаковыми именами в одном и том же файле.
- Наличие двух и более TQL-запросов с одинаковыми именами в одном и том же файле.
- Наличие двух и более функций условий с одинаковыми именами в одном и том же файле.
- Наличие двух зависимостей с одинаковыми именами в одном и том же развертываемом компоненте.
- Использование имени переменной в условии файла конфигурации, которая не заявлена как глобальная переменная в том же файле или как локальная переменная в зависимости, к которой принадлежит файл конфигурации.
- Использование имени переменной в функции условия, которая не заявлена как глобальная переменная или как параметр функции.
- Наличие циклической зависимости между переменными (например, если оператор вставки для переменной VAR_A использует значение переменной VAR_B, а оператор вставки для переменной VAR_B использует значение переменной VAR_A .
- Ссылка на TQL-запрос, который не существует в том же файле.
- Ссылка на область, которая не существует в том же файле.
- Ссылка на функцию условия, которая не существует в том же файле.
- Использование переменной, у которой нет значения по умолчанию в операторе игнорирования или альтернативном выражении, приведет к возникновению ошибки компиляции. Подробнее см. в разделе "Использование значений переменных по умолчанию" на странице 82.
- Если выражение поиска содержит переменные со значениями по умолчанию, необходимо учесть все случаи, в которых любая комбинация таких переменных получит значения по умолчанию. Это может быть сделано путем полного игнорирования выражения для некоторых переменных, либо за счет использования альтернативных выражений. Следует иметь альтернативные выражения для всех комбинаций переменных, которые отсутствуют в операторе игнорирования. Их отсутствие может привести к возникновению ошибки компиляции. Подробнее см. в разделе "Использование значений переменных по умолчанию" на странице 82.
- Если TQL-запрос используется в теге <DocumentCILocation>, но не соответствует следующим правилам:
  - TQL-запрос должен определять путь между развертываемым компонентом и документом конфигурации. Путь должен быть простым (без циклов).
  - TQL-запрос должен содержать следующие два конечных узла с особыми именами (с

учетом регистра):

- Deployable — ЭК развертываемого компонента в пути
- Configuration_document — ЭК в пути, указывающий на документ конфигурации

- Условия для узлов Deployable и Configuration_document не допускаются.
- Размерности между всеми узлами должны быть вида 1..1.
- Поддерживается только обычный тип связи. Составные связи, связи объединения и подграфы не допускаются.

- Если TQL-запрос используется в области, но не соответствует следующим правилам:
  - Узел запроса для развертываемого компонента должен быть назван "Deployable" (с учетом регистра). Результатами данного узла запроса являются возможные развертываемые компоненты для области.
  - В TQL-запросе может быть только один иной узел запроса (отличный от "Deployable"). Имя другого узла запроса может быть любым. Этот узел запроса имеет тип Scope.
  - Узлы запросов Deployable и Scope соединены связью типа Compound, содержащей все возможные пути между этими узлами.
  - Компонент "Deployable" представляет как компоненты потребителя, так и компоненты поставщика, которые могут находиться внутри одной области. Поэтому тип ЭК узла запроса должен быть родительским по отношению к развертываемым типам ЭК потребителя и поставщика.

- Если текстовый документ конфигурации содержит несколько тегов `<ReferenceLocation>` с разными приоритетами. Подробнее см. в разделе "Переопределение документа конфигурации" на странице 85.

Адаптеры поиска зависимостей

Платформа поиска зависимостей должна запускаться отдельным адаптером. Обязанности адаптера:

- Сбор всех необходимых строк подключения для поставщика.
- Запуск платформы поиска зависимостей.
- Отправка результатов поиска (зависимостей) в UCMDB.

Каждый адаптер обслуживает один тип поставщика. Например, адаптер может обслуживать поставщиков JAR.

Один и тот же тип поставщика также может обслуживаться несколькими адаптерами. Однако в этом случае каждый поставщик должен запускаться каждым из адаптеров только по одному разу для получения более однородных результатов.

Как и в случае с другими адаптерами, когда адаптер поиска зависимостей готов, необходимо создать задание обнаружения для запуска логики адаптера.

Этот раздел включает следующие подразделы:

- Создание адаптера поиска зависимостей 106
- Ограничения адаптера 114

Создание адаптера поиска зависимостей

1. Выберите **Управление потоком данных > Управление адаптерами**.
2. Щелкните **Создать**  и выберите **Создать адаптер**.
3. Введите данные адаптера и нажмите **ОК**.
4. На панели "Ресурсы" щелкните правой кнопкой созданный адаптер, а затем выберите **Изменить источник адаптера**.
5. Замените строку:

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

на

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

6. Замените строку:

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

на

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

```
<initialTimeout>180000</initialTimeout>
<retriesThreshold>12</retriesThreshold>
<multipleBy>2</multipleBy>
<maxRetry>10</maxRetry>
<timeoutThreshold>10800000</timeoutThreshold>
</timeoutParking>
</step>
</steps>
<finalStep />
<libraryScripts />
</workflow>
</params>
```

7. Подготовьте сценарий, который будет обрабатывать результаты поиска сигнатур зависимостей. Подробнее см. в разделе "Запись сценария Jython" на странице 112.
8. Добавьте в адаптер рабочего процесса следующий шаг:

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

9. На панели "Ввод" щелкните **Выбрать тип ЭК** и выберите тип ЭК "Provider" для данного адаптера.
10. Щелкните **Изменить входной запрос** и подготовьте входной TQL-запрос. Подробнее см. в разделе "Определение входного TQL-запроса и целевых данных" на следующей странице.
11. На панели "Обнаруженные типы ЭК" нажмите кнопку и добавьте тип поставщика, все возможные типы потребителей, а также тип связи "поставщик-потребитель".
12. Завершив настройку, нажмите **Сохранить**.

Определение адаптера потребитель-поставщик

Функция поиска выполняется при помощи адаптеров обнаружения. Каждый адаптер отвечает за поиск определенного набора строк подключения типа поставщика. Входной TQL-запрос адаптера отвечает за передачу всех ЭК, в которых могут быть найдены такие строки, и вставляет значения в переменные и понятия строк подключения при помощи целевых данных триггера. Подробнее см. в разделе "Разработка адаптеров Jython" на странице 37.

Эти адаптеры относятся к типу "Workflow Adapter", их работа по обнаружению связей потребитель-поставщик включает в себя несколько этапов:

1. Адаптер запускает логику составления конкретного выражения поиска для фильтра документа конфигурации области — вставляет значения строк подключения в глобальные переменные сигнатуры зависимости и создает экземпляры понятий.

2. Адаптер отправляет конкретное выражение поиска для области в механизм UCMDB Solr, запущенный на сервере UCMDB.
3. Адаптер отправляет в UCMDB TQL-запрос для получения всей необходимой информации (документы конфигурации, развертываемые дескрипторы и т.д.) с тем, чтобы можно было выполнить поиск сигнатуры зависимости.
4. Адаптер загружает и сохраняет содержимое требуемых документов конфигурации в зонд потока данных.
5. Затем адаптер выполняет поиски зависимостей, направленные на поставщика и потребителей, найденных в области на этапах 2 и 3.
6. После этого сценарий Jython сообщает результаты поиска зависимостей.

Определение входного TQL-запроса и целевых данных

Для составления выражения поиска из области и условий сигнатуры зависимости каждая переменная и понятие, указанные в выражении поиска, должны быть заменены конкретной строкой подключения (возвращаемой TQL-запросом). Такая замена выполняется на основе заданных для определенных адаптеров сопоставлений.

Входной TQL-запрос для адаптера сопоставления зависимостей должен определять:

- ЭК-триггеры для поиска потребителей для поставщика (узел запроса SOURCE).
- все строки подключения, необходимые для конкретного поставщика

Эти TQL-запросы должны возвращать все строки подключения, разбросанные по топологии ЭК, и реализовывать поставщика, предоставляющего службу. Например, если поставщиком является Oracle RAC, TQL-запрос должен возвращать все ЭК, включающие в себя любую строку подключения, необходимую потребителю для подключения к RAC и использования ее служб. Поэтому в определении структуры TQL-запроса должен упоминаться любой атрибут, хранящий такую строку подключения. Например, TQL-запрос для RAC должен возвращать IP-адрес и порт доступа к каждому экземпляру RAC, а также имя-службы-RAC.

Переменные и понятия должны сопоставляться с атрибутами узлов запроса во входном TQL-запросе при помощи определения целевых данных адаптера. Каждый элемент-образец должен иметь уникальное имя для обеспечения корректного сопоставления.

Как и для любого адаптера, входной TQL-запрос для адаптера поиска будет выполняться по любому триггеру, который соответствует TQL-запросу триггера для задания, связанного с этим адаптером.

При использовании адаптера для обнаружения служб может потребоваться ограничить триггер не только до уровня триггеров, обнаруженных сервером (что происходит автоматически), но также до уровня триггеров, которые связаны с ЭК бизнес-службы (по какому-то пути). Для этого задайте входной TQL-запрос таким образом, чтобы он содержал путь (обычно какую-то сложную связь) между триггером и ЭК бизнес-службы, и присвойте узлу запроса ЭК бизнес-службы имя "SERVICE" (с учетом регистра). Затем, когда платформа увидит узел запроса "SERVICE", она автоматически ограничит результаты только до службы (служб), к которой принадлежит этот ЭК-триггер.

Указание значений для переменных

Каждому адаптеру поиска требуется установка значений для всех глобальных переменных и понятий, которые имеются в условиях поиска в сигнатуре зависимости для зависимостей, которые должен обнаружить этот адаптер. Эти значения представляют строку подключения поставщика (подробнее см. в разделе "Переменные и понятия" на странице 73), и они будут использоваться для поиска потребителей для поставщика (триггера).

Примечание. Если один из атрибутов сопоставления пуст, оставьте его в следующем виде:

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

В этом примере CONTEXT_ROOT является атрибутом, который получит пустое значение в сигнатуре зависимости и будет игнорироваться.

Чтобы присвоить значение переменной:

1. Добавьте целевое значение данных, имеющее точно такое же имя, что и переменная (с учетом регистра).
2. Установите для целевых данных значение с жестким кодированием или переменную.

Подробнее см. в разделе "Разработка адаптеров Jython" на странице 37.

Указание значений для переменных понятий

Экземпляр понятия должен представлять собой набор неразрывных, тесно связанных строк подключения. Входной TQL-запрос может возвращать несколько неразрывных наборов строк подключения. Например, экземпляр running-software может прослушивать несколько пар IP:Port. Для каждого такого набора должен создаваться отдельный экземпляр понятия. Атрибут понятия key используется для разделения экземпляров понятий. key относится к элементу-образцу TQL-запроса. Для каждого экземпляра ЭК, возвращаемого для элемента-образца key, создается новый экземпляр понятия. Каждый такой экземпляр ЭК именуется **экземпляром ключевого ЭК (key CI instance)**.

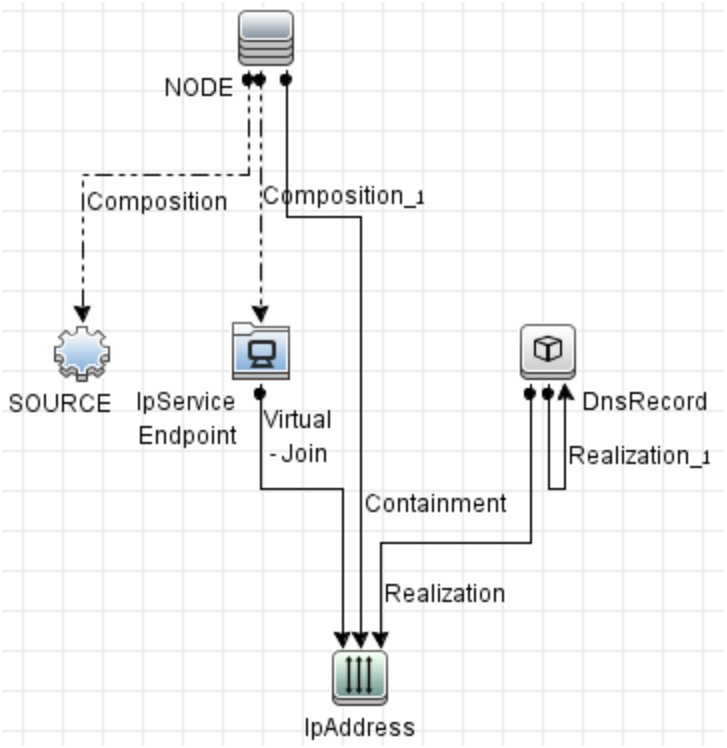
Примечание. TQL-запросы не могут содержать несколько определений сопоставления для одного и того же понятия.

Строки подключения для каждого экземпляра понятия должны извлекаться из соответствующего экземпляра ключевого ЭК, а также из ЭК, подключенных к этому ключевому ЭК. Ниже приведен пример файла сигнатуры зависимости, который содержит это определение понятия:

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

```
<Property name="DNS"/>
</Properties>
</Concept>
```

Входной TQL-запрос адаптера:



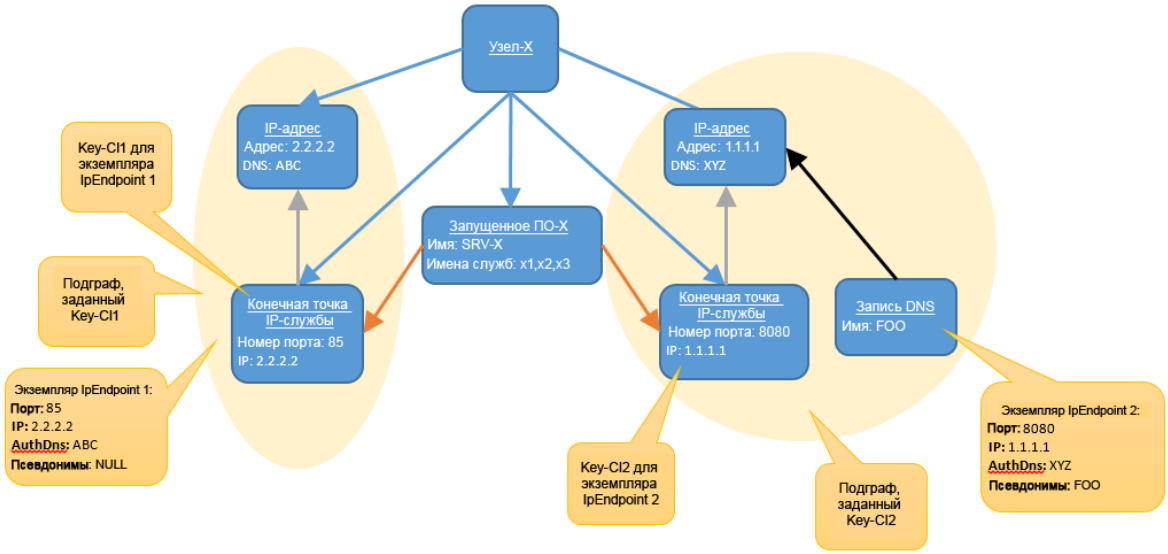
Для создания экземпляров понятий в адаптере, также как и для переменных, каждая переменная понятия должна быть сопоставлена с переменной целевых данных. Подробнее см. в разделе "Указание значений для переменных" на предыдущей странице.

Целевые данные адаптера:

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

Таким образом, элемент запроса **IpServiceEndpoint** является ключевым ЭК для понятия **IpEndpoint** в данном адаптере.

В графическом примере TQL-запроса (в котором представлен результат TQL-запроса для запущенного ПО X) можно распознать два ключевых ЭК. Строки подключения для заполнения атрибутов каждого экземпляра понятия **IpEndpoint** берутся из подграфа каждого ключевого ЭК. Таким образом, каждый экземпляр понятия представляет отдельный набор неразрывных строк подключения.



В отличие от других переменных при сопоставлении переменных понятия во входном TQL-запросе следует указывать путь от узла запроса триггера (всегда именуемого SOURCE) к узлу запроса, к которому будет привязана переменная. Кроме того, после настройки переменной key в префиксе ко всем остальным переменным понятия следует указать путь к переменной key.

В этом примере наглядно представлено следующее:

- Именем переменной целевых данных является имя понятия, за которым следует имя переменной понятия.
- Можно сопоставлять несколько понятий в одном и том же адаптере, указывая имя каждого из них в переменных целевых данных.
- Путь всегда начинается в элемента SOURCE.
- Пути переменных понятий, не являющихся переменными key, в префиксе будут иметь путь к переменной key.
- Путь состоит только из имен узлов запросов, без связей. Однако между двумя именами узлов запросов, указанными в пути, должна существовать минимум одна связь.

Если путь содержит один или несколько следующих элементов, триггер не сработает на этапе его диспетчеризации:

- Имя узла запроса, которое не существует во входном TQL-запросе.
- Два смежных имени узлов запросов, которые не связаны во входном TQL-запросе.
- Имя атрибута, не являющееся частью полученного типа ЭК.

Если для узла запроса, который не является ключевым узлом запроса, получено несколько ЭК, переменной целевых данных будет список всех значений свойств из этих

ЭК. Список может создаваться только для тех переменных целевых данных, которые содержат путь, не входящий в любые другие переменные целевых данных для того же понятия. В противном случае триггер не сработает на этапе его диспетчеризации. В приведенном выше примере переменная целевых данных `IpEndpoint.IPADDRESS` не может содержать список, поскольку ее путь содержится в переменной целевых данных `IpEndpoint.DNS`. `IpEndpoint.DNS` может содержать список, поскольку ее путь не содержится в какой-любой другой переменной. Это означает, что из узла запроса **IpAddress** может быть получен только один ЭК, связанный с результатом, полученным из узла запроса **IpServiceEndpoint**. Это следует учитывать при планировании TQL-запросов и создании путей.

Наличие ссылок на себя в TQL-запросе создаст список точно так же, как и наличие нескольких полученных ЭК в узле запроса, не являющемся ключевым узлом. Поэтому для узлов запросов со ссылками на себя существуют точно такие же ограничения.

Подробнее см. в разделе "Переменные и понятия" на странице 73.

Запись сценария Jython

Финальным этапом в создании адаптера сопоставления зависимостей является запись сценария Jython для отправки результатов.

Для доступа к результатам поиска зависимостей их следует извлечь из состояния рабочего процесса при помощи следующих строк кода:

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

Если этап поиска успешен, переменная searchResult гарантированно имеет ненулевое значение. Рекомендуется сделать этап поиска в рабочем процессе обязательным с тем, чтобы этап сценария Jython не выполнялся в случае, если этап поиска заканчивается сбоем. Эта переменная будет содержать объект типа **com.hp.ucmdb.discovery.probe.agents.probemgr.accuratedependencies.search.ConsumerDeployableSearchResult**.

Этот объект позволяет выполнять следующее:

1. Просмотр всех развертываемых потребителем компонентов, найденных во время поиска зависимостей. Помните: поиск получает строки подключения поставщика как входные данные и возвращает все развертываемые потребителем компоненты, которые зависят от поставщика.
2. Для каждого потребителя может быть более одной сигнатуры зависимости. Все они должны быть подвержены итерации. Каждая сигнатура зависимости может иметь различные выходные значения переменных. На самом деле значения всех переменных, использованных в сигнатуре зависимости (глобальных и локальных переменных, а также переменных понятий), доступны для сценария Jython из результатов поиска.
3. Создайте указатель Object State Holder (OSH), который содержит связь между потребителем и поставщиком. Сведения о поставщике также можно получить из

объекта результатов поиска.

4. Добавьте OSH в вектор результата (OSHV) и отправьте результаты в UCMDB.

Далее представлен код сценария Jython, который выполняет процесс, указанный выше:

```
# Получить объект результатов поиска из состояния рабочего процесса
workflowState = Framework.getWorkflowState()
searchResult = workflowState.getProperty
(DependenciesDiscoveryConsts.DEPENDENCIES_DISCOVERY_RESULT)

# Подготовить OSHV, который будет содержать зависимости
oshv = ObjectStateHolderVector()
dependencyCount = 0

# Извлечь OSH поставщика
providerServiceOsh = searchResult.getProviderDeployable().getDeployable()

# Прогнать все развертываемые потребителем компоненты
for index in range(0, searchResult.size()):
deployable = searchResult.get(index)

# Получить OSH развертываемого потребителем компонента
deployableOsh = deployable.getDeployable()

# Создать OSH для связи зависимости между потребителем и поставщиком
consumerProviderLink = modeling.createLinkOSH('consumer_provider',
deployableOsh, providerServiceOsh)

references = []

# Подвергнуть итерации все зависимости, найденные между потребителем и
поставщиком в deployable.getDependencies():

# Извлечь имя зависимости в том виде, в котором оно отображается в файле
сигнатуры зависимости
dependencyName = dependency.getDependencyName()

# Получить значения всех переменных, которые были использованы в зависимости
variables = dependency.getExportVariables()

# Агрегировать значения переменной с именем REFERENCE
# Примечание: Значением переменной может быть список, если она содержит
несколько значений
dependencyNames.append(dependencyName)
for var in variables:
varName = var.getName()
values = var.getValues()
if varName.lower() == REFERENCES:
references += list(values)
```

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

# Добавить связь в OSHV результатов
oshv.add(consumerProviderLink)

# Отправить результат в UCMDB
Framework.sendObjects(oshv)
```

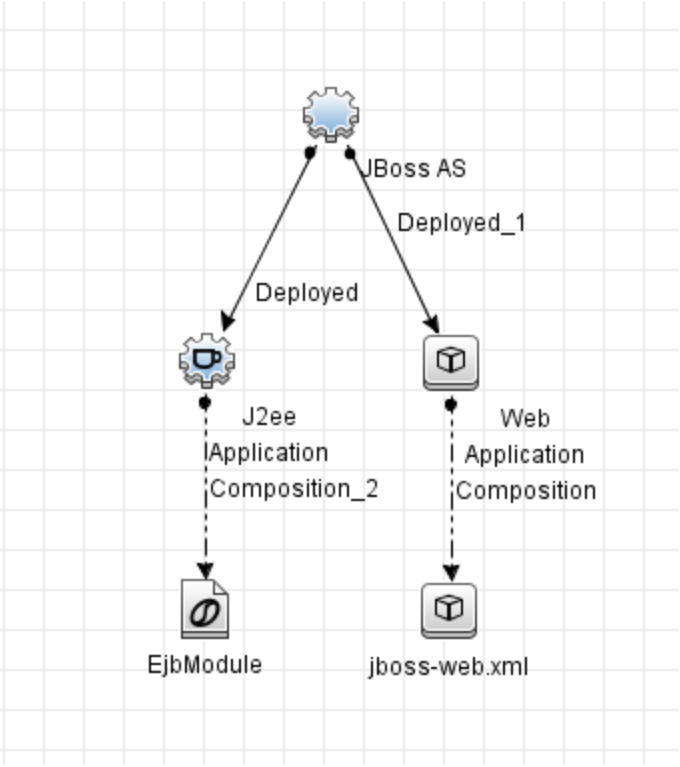
Примечание. Правила границы служб (набор правил для механизма правил обнаружения) выполняются на основе информации, отправляемой сценарием. По этой причине важно отправлять как можно больше информации о связи, потребителе и поставщике. Рекомендуется создавать связь с OSH потребителя и поставщика, которые были получены из объекта результатов поиска, как показано выше. Эти OSH будут содержать всю необходимую для правил границы служб (и других правил механизма) информацию, которая обеспечит корректную работу.

Ограничения адаптера

- Адаптеры сопоставления зависимостей не поддерживаются для зондов потока данных, которые установлены в отдельном режиме.
- При перезапуске сервера UCMDB триггеры для заданий адаптера сопоставления зависимостей могут не сработать из-за превышения времени ожидания. Эти триггеры должны сработать во время следующего запланированного запуска.

Полный пример

В этом разделе представлен полный пример разработки сигнатуры зависимости и адаптера, которые будут находить зависимости между приложением J2ee и веб-приложением, развернутыми внутри одной и той же системы JBoss AS.



Этот раздел включает следующие подразделы:

Рабочий процесс разработки

Общий процесс разработки сигнатуры зависимости и адаптера имеет следующий вид:

1. Определитесь, какой тип ЭК поставщика для сигнатуры вы собираетесь разрабатывать. В данном примере типом поставщика является приложение J2ee.
2. Решите, какие развертываемые компоненты потребителя, имеющие отношение к поставщику, требуется охватить. Поскольку сигнатура относится к документам конфигурации потребителя, каждый потребитель будет иметь собственные сигнатуры зависимостей.

   В этом примере мы выбираем один развертываемый компонент потребителя — веб-приложение в JBoss.
3. Решите, следует ли добавить эти сигнатуры к существующему файлу сигнатуры зависимости или создать новый файл.

   Оба метода функционально ничем не отличаются. Выберите тот вариант, который облегчит обслуживание сигнатур зависимостей. Например, если развертываемый компонент потребителя уже существует в одном из файлов сигнатур зависимостей, обслуживание может быть проще, если добавить эту новую зависимость в существующий развертываемый компонент. В данном примере мы создадим новый файл сигнатуры зависимости, который будет содержать новую сигнатуру зависимости.
4. Проверьте, требуется ли новый адаптер, или поставщик уже охвачен существующим

адаптером. Помните, что типом ЭК-триггера адаптеров поиска является тип ЭК "Provider". Необходимо проверить следующее:

a. Существует ли адаптер поиска, для которого ЭК-триггером уже является данный тип ЭК?

   Если нет, необходимо создать такой адаптер. Вот это мы будем делать в данном примере.

b. Если адаптер поиска существует — все ли требуемые переменные и понятия строк подключения сопоставлены с целевыми данными?

   Если нет, необходимо обновить существующий адаптер и добавить эти переменные в целевые данные. Помните, что целевые данные извлекаются из входного TQL-запроса. Это означает, что входной TQL-запрос может потребовать изменения для предоставления новой информации.

5. Подготовьте задание для адаптера, если оно еще не существует.
6. Убедитесь в том, что TQL-запрос триггера также возвращает поставщиков в качестве триггеров.

Разработка сигнатур зависимостей

Разработайте сигнатуры зависимостей между приложением JBoss J2EE и веб-приложением.

1. Выясните, какие строки подключения требуются для создания зависимости между потребителем и поставщиком. В данном примере требуемой строкой подключения является имя JNDI для EJB из модулей EJB, которые содержатся в приложении-поставщике J2EE.
2. Все строки подключения должны быть заданы как глобальные переменные или понятия. Значения этих переменных будут вставляться адаптером. При этом перед обозначением этих переменных в файле сигнатуры зависимости (новом или существующем) следует проверить все файлы на наличие уже существующих схожих переменных или понятий. Если такие существуют, дайте своим переменным, понятиям и переменным понятий те же имена, что и у существующих элементов. Это облегчит разработку адаптера, поскольку число глобальных переменных, требующих вставки, расти не будет. В данном примере мы добавим эти определения в новый файл, и этот файл будет выглядеть следующим образом:

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

3. Проанализируйте файлы конфигурации и определите следующее:
   - Расположение каждой строки подключения в каждом файле конфигурации.
   - Тип каждого файла: PROPERTIES, XML или другой текстовый файл.
   - Связи между всеми файлами (если файлов несколько).

В этом примере веб-приложение JBoss определяет ссылку на EJB в файле конфигурации **jboss-web.xml** следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-web>
<!-- Ссылка на EJB в том же сервере с настраиваемой привязкой JNDI -->
<ejb-ref>
<ejb-ref-name>ejb/BHome</ejb-ref-name>
<jndi-name>someapp/ejbs/beanB</jndi-name>
</ejb-ref>
<!-- Ссылка на EJB во внешнем сервере -->
<ejb-ref>
<ejb-ref-name>ejb/RemoteBHome</ejb-ref-name>
<jndi-name>jnp://otherserver/application/beanB</jndi-name>
</ejb-ref>
</jboss-web>
```

Значение в разделе `<ejb-ref-name>` является ссылкой строки подключения. Поскольку это XML-файл, можно использовать следующее выражение XPath для сопоставления имени JNDI с EJB:

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

4. Необходимо определить область, содержащую выражение поиска по умолчанию, которое поможет найти файл конфигурации, требуемый для сопоставления зависимостей. В этом примере ключевым словом является само имя JNDI. Далее приведено определение области:

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

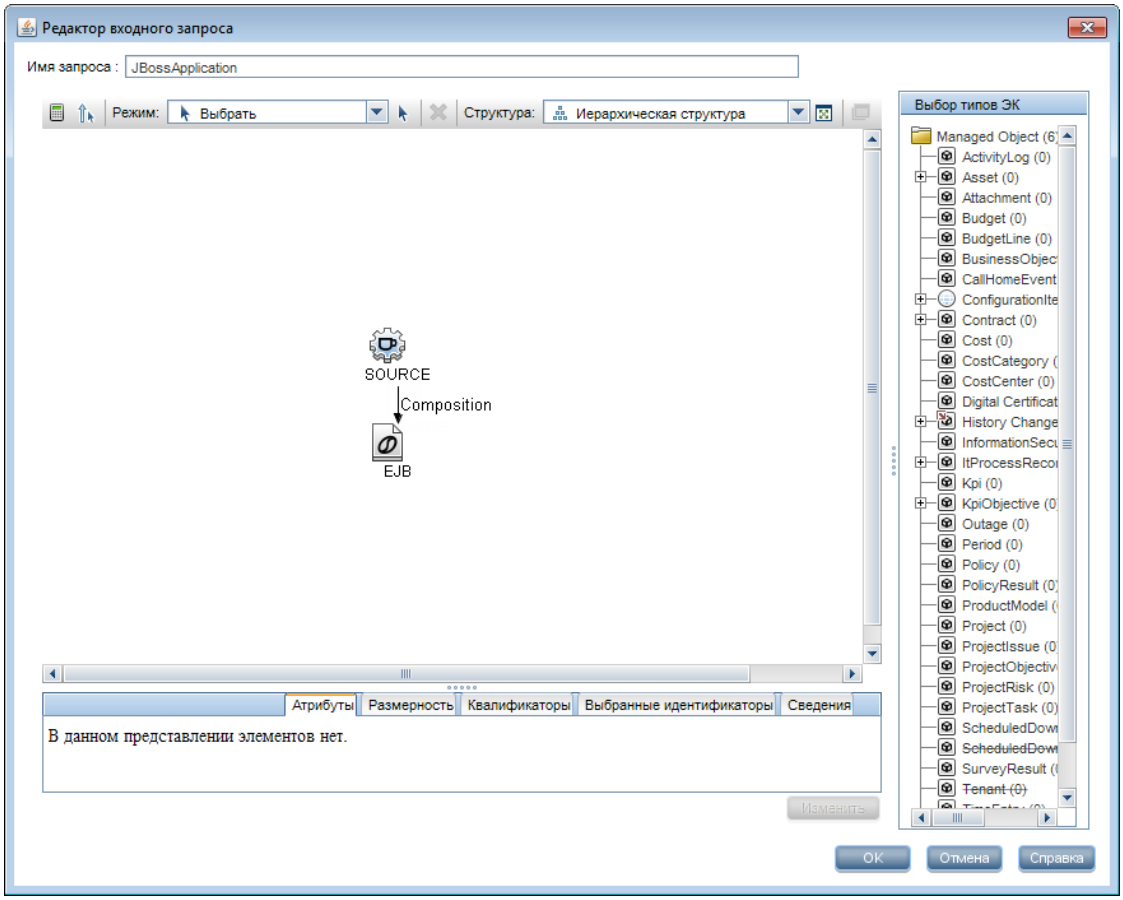
Полный пример сигнатуры зависимости выглядит так:

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

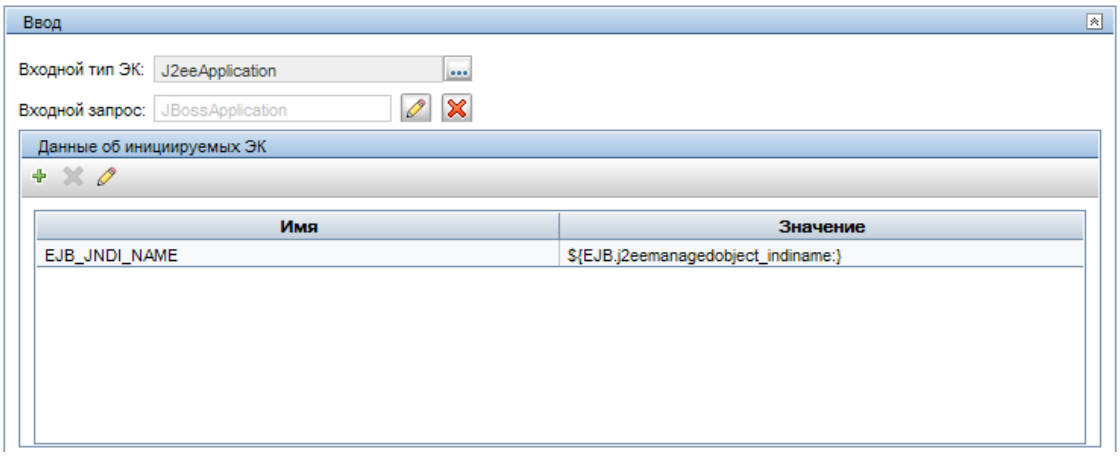
```
providerCiType="j2eeapplication" scope="JBoss_EJB_Same">
<XmlConfigurationDocument name="jboss-web.xml">
<Condition>
<Operator type="or">
<XPathCondition>
<XPath>//ejb-ref/jndi-name[matches(., '^${EJB_JNDI_NAME}$', 'i')]</XPath>
</XPathCondition>
</Operator>
</Condition>
</XmlConfigurationDocument>
</Dependency>
</Deployable>
<ScopeDefinitions>
<Scope name="JBoss_EJB_Same_Cell">
<ConfigurationDocumentContentFilter>
<Operator type="and">
<Operand value="${EJB_JNDI_NAME}"/>
</Operator>
</ConfigurationDocumentContentFilter>
</Scope>
</ScopeDefinitions>
</DependencySignatures>
```

Разработка адаптера

1. Создайте новый адаптер рабочего процесса, как описано в разделе "Создание адаптера поиска зависимостей" на странице 106.
2. Установите типом ЭК-триггера тип "Provider". В этом примере поставщиком является приложение J2EE.
3. Настройте выходной TQL-запрос для адаптера на получение требуемых ЭК и их атрибутов. В данном примере нам необходим ЭК триггер J2EEApplication вместе с ЭК EJBModule, принадлежащими приложению-поставщику J2EE.



4. Сопоставьте глобальные переменные, заданные в сигнатурах для требуемых переменных строк подключения и понятий, с данными ЭК-триггера. В этом примере мы сопоставляем атрибут j2eemanagedobject_jndiname из ЭК EJBModule с целевыми данными EJB_JNDI_NAME.



5. В разделе "Шаги рабочего процесса" вставьте следующее определение рабочего процесса:

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

   Длительность времени ожидания можно изменить при помощи параметра **timeoutParking**.

6. Создайте TQL-запрос триггера для нового адаптера.
7. Добавьте новое определение задания в тип операции Service Discovery следующим образом:

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

Глава 5: Разработка общих адаптеров БД

Данная глава включает:

- Обзор общего адаптера БД 122
- TQL-запросы для общего адаптера БД 123
- Выверка 124
- Hibernate как поставщик JPA 124
- Подготовка к созданию адаптера 127
- Подготовка пакета адаптера 131
- Настройка адаптера — Минимальный метод 134
- Настройка адаптера — Расширенный метод 138
- Реализация подключаемого модуля 142
- Развертывание адаптера 145
- Изменение адаптера 145
- Создание точки интеграции 145
- Создание представления 146
- Вычисление результатов 146
- Просмотр результатов 146
- Просмотр отчетов 146
- Активация файлов журнала 146
- Использование Eclipse для сопоставления атрибутов ЭК и таблиц базы данных 147
- Файлы конфигурации адаптеров 154
- Встроенные конвертеры 179
- Подключаемые модули 184
- Примеры конфигурации 185
- Файлы журнала адаптера 193
- Внешние ссылки 195
- Устранение неполадок и ограничения — Разработка общих адаптеров БД 195

Обзор общего адаптера БД

Цель платформы общего адаптера БД заключается в создании адаптеров, обеспечивающих интеграцию с реляционными СУБД, а также выполнение TQL-запросов и заданий заполнения БД. Реляционные СУБД, поддерживаемые общим адаптером БД: Oracle, Microsoft SQL Server и MySQL.

Эта версия адаптера БД основывается на стандарте JPA (Java Persistence API) с библиотекой Hibernate ORM в качестве поставщика постоянных данных.

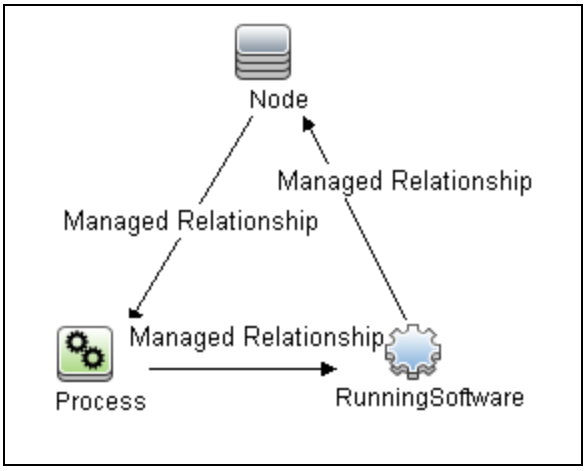
TQL-запросы для общего адаптера БД

Для заданий заполнения необходимо проверить каждую схему ЭК в диалоговом окне Настройки структуры в Студии моделирования. Подробнее см. в разделе Query Node/Relationship Properties Dialog Box в документе *Руководство по моделированию в HP Universal CMDB*. Важно отметить, что для ЭК может потребоваться определить какой-либо атрибут, без которого этот ЭК невозможно будет добавить в UCMDB.

Следующие ограничения действуют для TQL-запросов, вычисляемых только общим адаптером БД:

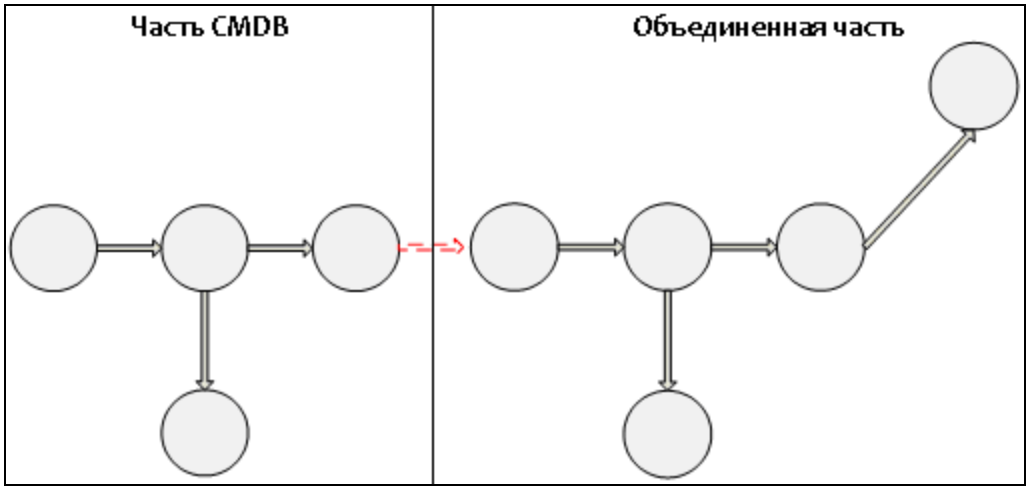
- подграфы не поддерживаются.
- составные связи не поддерживаются.
- циклы и части циклов не поддерживаются

  В следующем TQL-запросе представлен пример цикла:



- Макет функций не поддерживается.
- Размерность 0..0 не поддерживается.
- Связь типа `Join` не поддерживается.
- Условия квалификатора не поддерживаются.
- Для соединения двух ЭК во внешней базе данных должна существовать связь в виде

таблицы или внешнего ключа.



Выверка

Выверка выполняется в рамках вычисления TQL-запроса на стороне адаптера. Для выверки сторона CMDB сопоставляется с объединенным объектом, который называется типом ЭК выверки.

Сопоставление. Каждый атрибут в CMDB сопоставляется со столбцом источника данных.

Хотя сопоставление выполняется напрямую, также поддерживаются функции преобразования для данных сопоставления. Новые функции добавляются с помощью кода Java (например, lowercase, uppercase). Цель этих функций — обеспечить преобразование значений, которые хранятся в CMDB в одном формате, а в объединенной базе данных — в другом).

Примечание.

- Для соединения CMDB и внешнего источника базы данных необходимо создать соответствующую связь в базе данных. Дополнительные сведения см. в разделе "Необходимые условия" на странице 127.
- Также поддерживается выверка по ID CMDB.
- Также поддерживается выверка по глобальным ID.

Hibernate как поставщик JPA

Hibernate — это объектно-ориентированное средство сопоставления, которое обеспечивает сопоставление классов Java с реляционными БД нескольких типов, например Oracle и Microsoft SQL Server. Дополнительные сведения см. в разделе "Функциональные ограничения" на странице 195.

В простом сопоставлении каждый класс Java сопоставляется с одной таблицей. При более сложном сопоставлении используется наследование (как в базе данных CMDB).

Другие поддерживаемые функции включают сопоставление класса с несколькими таблицами, поддержку коллекций и связей типов один к одному, один ко многим и многие к одному. Дополнительные сведения см. в разделе "Связи" на следующей странице ниже.

В нашем случае создание классов Java не требуется. Сопоставление задается между типами ЭК модели классов CMDB и таблицами БД.

Данный раздел также включает следующие подразделы.

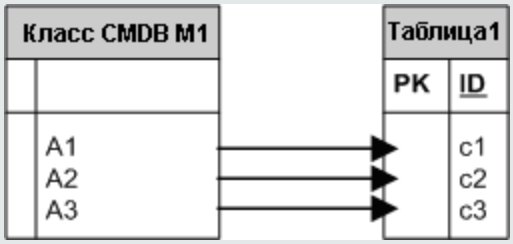
- "Примеры сопоставления между объектно-ориентированным приложением и реляционной базой данных" ниже
- "Связи" на следующей странице
- "Удобство использования" на следующей странице

Примеры сопоставления между объектно-ориентированным приложением и реляционной базой данных

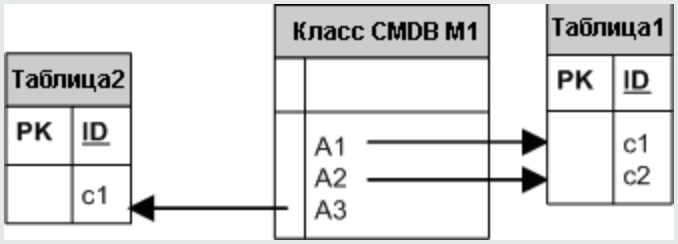
В следующих примерах приводится описание сопоставления между объектно-ориентированным приложением и реляционной базой данных:

Пример сопоставления одного класса CMDB с одной таблицей в БД:

Класс `M1` с атрибутами `A1`, `A2` и `A3` сопоставляется со столбцами таблицы `1` `c1`, `c2` и `c3`. Это значит, что для любого экземпляра `M1` существует соответствующая строка в таблице `1`.

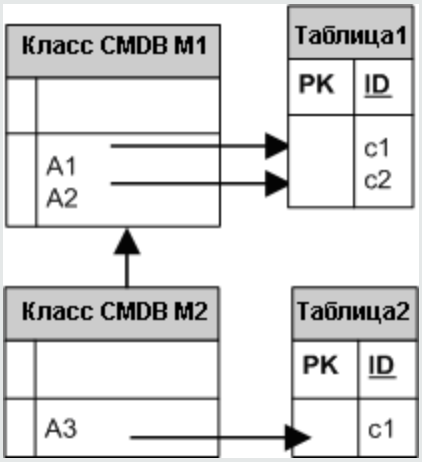


Пример сопоставления одного класса CMDB с двумя таблицами в БД:



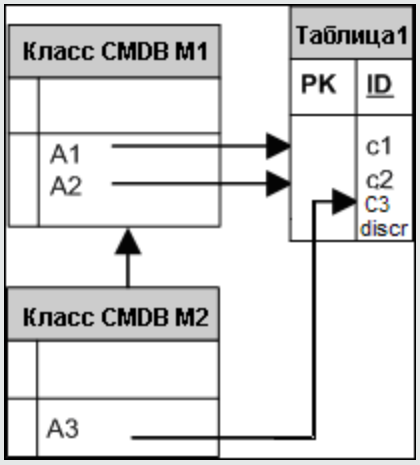
Пример наследования:

В данном случае используется база CMDB, в которой каждый класс имеет собственную таблицу БД.



Пример наследования одной таблицы с дискриминатором:

Вся иерархия классов сопоставляется с одной таблицей базы данных, столбцы которой включают сверхмножество всех атрибутов сопоставленных классов. Кроме того, таблица содержит дополнительный столбец (`Discriminator`), значение которого указывает, какой класс должен быть сопоставлен с этой записью.



Связи

Существует три типа связей: один ко многим, многие к одному и многие ко многим. Для соединения различных объектов базы данных одна из этих связей задается с помощью столбца внешнего ключа (один ко многим) или таблицы сопоставления (многие ко многим).

Удобство использования

Поскольку схема JPA крайне сложна, для упрощения определения связей доступен XML-файл.

Сценарий использования XML-файла приводится ниже: объединенные данные моделируются в один объединенный класс. Этот класс включает связь многие к одному с необъединенным классом CMDB. Кроме того, между объединенным и необъединенным классами может существовать только одна связь.

Подготовка к созданию адаптера

В этой задаче описывается подготовка к созданию адаптера.

Примечание. Примеры общего адаптера БД доступны в UCMDB API. В частности, адаптер DDMi содержит сложный файл **orm.xml**, а также реализации некоторых интерфейсов подключаемых модулей.

Эта задача включает следующие шаги:

- "Необходимые условия" ниже
- "Создание типа ЭК" на странице 129
- "Создание связи" на странице 129

1. Необходимые условия

   Чтобы проверить возможность использования адаптера БД с БД, убедитесь в следующем:

   - В базе данных хранятся классы выверки и их атрибуты (также известные как multinode). Например, если выверка выполняется по имени узла, убедитесь в наличии таблицы с именами узлов. Если выверка выполняются по узлу `cmdb_id`, убедитесь, что столбец с идентификаторами CMDB соответствует идентификаторам CMDB узлов в CMDB. См. дополнительные сведения о выверке в разделе "Выверка" на странице 124.

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

   - Для корреляции двух типов с ЭК со связью необходимы данные о корреляции между таблицами типов ЭК. Для корреляции может использоваться столбец внешних ключей или таблица сопоставления. Например, для корреляции узла и

заявки необходима таблица заявок со столбцом, содержащим идентификатор узла, таблица узла со столбцом с идентификатором заявки, которая с ним соединена, или таблица сопоставления, в которой значение `end1` соответствует идентификатору узла, а `end2` — идентификатору заявки. См. дополнительные сведения о данных корреляции в разделе "Hibernate как поставщик JPA" на странице 124.

В следующей таблице представлен столбец внешнего ключа `NODE_ID`:

NODE_ID	CARD_ID	CARD_TYPE	CARD_NAME
2015	1	Serial Bus Controller	Intel ® 82801EB USB Universal Host Controller
3581	2	System	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3581	3	Display	ATI ES1000
3581	4	Base System Peripheral	HP ProLiant iLO 2 Legacy Support Function

- Каждый тип ЭК может быть сопоставлен с одной или несколькими таблицами. Чтобы сопоставить один ЭК с несколькими таблицами, убедитесь в наличии основной таблицы, основные ключи которой существуют в других таблицах, а также уникального столбца значений.

  Например, заявка сопоставлена с двумя таблицами: `ticket1` и `ticket2`. Первая таблица включает столбцы `c1` и `c2`, вторая — столбцы `c3` и `c4`. Чтобы считаться одной таблицей, эти таблицы должны иметь одинаковые основные ключи. Или первичный ключ первой таблицы может быть столбцом во второй.

  В следующем примере таблицы используют общий первичный ключ, который называется `CARD_ID`:

CARD_ID	CARD_TYPE	CARD_NAME
1	Serial Bus Controller	Intel ® 82801EB USB Universal Host Controller
2	System	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3	Display	ATI ES1000
4	Base System Peripheral	HP ProLiant iLO 2 Legacy Support Function

CARD_ID	CARD_VENDOR
1	Hewlett-Packard Company
2	(Standard USB Host Controller)
3	Hewlett-Packard Company
4	(Standard system devices)
5	Hewlett-Packard Company

2. Создание типа ЭК

   Во время этого шага вы создадите тип ЭК, который представляет данные в реляционной СУБД (внешнем источнике данных).

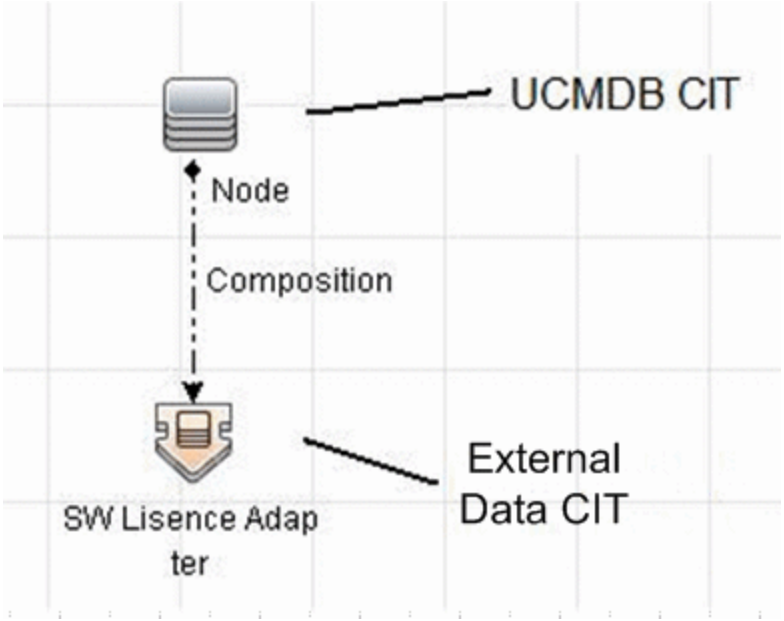
   a. В UCMDB откройте диспетчер типов ЭК и создайте новый тип ЭК. Подробнее см. в разделе "Создание типа ЭК" в документе *Руководство по моделированию в HP Universal CMDB*.

   b. Добавьте в тип ЭК необходимые атрибуты, такие как время доступа, поставщик и др. Это атрибуты, которые адаптер получит из внешнего источника данных и добавит в представления CMDB.

3. Создание связи

   Во время этого шага вы добавите связь между типом ЭК UCMDB и новым типом ЭК, представляющим данные из внешнего источника данных.

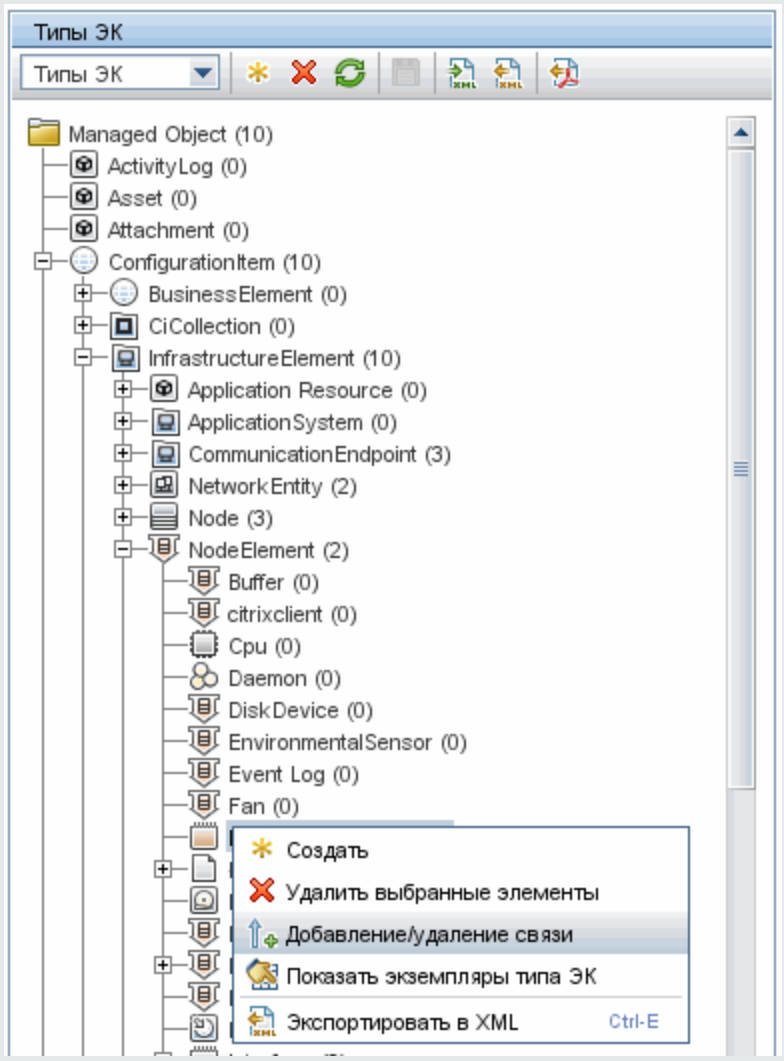
   Добавьте соответствующие связи к новому типу ЭК. Подробнее см. в разделе "Добавление/удаление связи" в документе *Руководство по моделированию в HP Universal CMDB*.



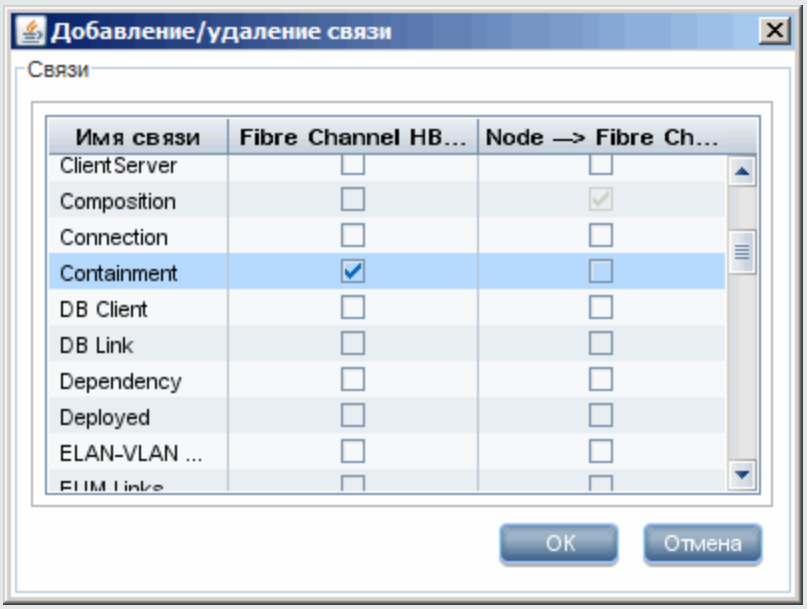
Примечание. На этом этапе еще невозможно просматривать объединенные данные или использовать внешние данные для заполнения, поскольку метод объединения данных еще не определен.

Пример создания связи «Containment»:

a. В диспетчере типов ЭК выберите два типа ЭК:



b. Создайте связь типа **Containment** между двумя типами ЭК:



Подготовка пакета адаптера

Во время этого шага вы найдете и настроите пакет общего адаптера БД.

1. Найдите пакет **db-adapter.zip** в папке **C:\hp\UCMDB\UCMDBServer\content\adapters**.
2. Извлеките пакет в локальный временный каталог.
3. Измените XML-файл адаптера:
   - Откройте файл **discoveryPatterns\db_adapter.xml** в текстовом редакторе.
   - Найдите атрибут **adapter id** и замените имя:

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

Если адаптер поддерживает заполнение, в элемент **<adapter-capabilities>** необходимо добавить следующую возможность:

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

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

Отображаемая метка или идентификатор появятся в списке адаптеров на панели «Точка интеграции» в HP Universal CMDB.

При создании общего адаптера БД нет необходимости изменять тег **changes-source** в теге **support-replicatioin-data**. Если реализован подключаемый модуль **FcmdbPluginForSyncGetChangesTopology**, будет возвращена измененная топология из последнего выполнения. Если подключаемый модуль не реализован, будет возвращена полная топология, а автоматическое удаление будет выполнено согласно возвращенным ЭК.

Подробные сведения о наполнении CMDB данными см. в разделе "Страница "Студия интеграции"" (*Руководство по управлению потоками данных в HP Universal CMDB*).

- Если в адаптере используется система сопоставления из версии 8.x (т.е. новая система сопоставления выверки не используется), замените следующий элемент:

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

  на:

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

  Чтобы вернуться к новой системе сопоставления, верните элемент к предыдущему значению:

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

- Найдите определение **category**:

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

  Измените имя категории **Generic** на нужную категорию.

  Примечание. Адаптеры, категории которых указаны как **Generic**, не отображаются в студии интеграции при создании новой точки интеграции.

- Соединение с базой данных можно описать именем пользователя (схемой), паролем, типом БД, именем хоста и именем или SID базы данных.

  Для этого типа подключения в разделе **parameters** XML-файла адаптера указываются следующие элементы:

  ```
  <parameters>
  <!--The description attribute may be written in simple text or HTML.-->
  <!--The host attribute is treated as a special case by UCMDB-->
  <!--and will automatically select the probe name (if possible)-->
  ```

```
<!--according to this attribute's value.-->
<!--Display name and description may be overwritten by I18N values-->
        <parameter name="host" display-name="Hostname/IP" type="string"
description="The host name or IP address of the remote machine"
mandatory="false" order-index="10" />
        <parameter name="port" display-name="Port" type="integer"
description="The remote machine's connection port" mandatory="false"
order-index="11" />
        <parameter name="dbtype" display-name="DB Type" type="string"
description="The type of database" valid-
values="Oracle;SQLServer;MySQL;BO" mandatory="false" order-
index="13">Oracle</parameter>
        <parameter name="dbname" display-name="DB Name/SID" type="string"
description="The name of the database or its SID (in case of Oracle)"
mandatory="false" order-index="13" />
        <parameter name="credentialsId" display-name="Credentials ID"
type="integer" description="The credentials to be used" mandatory="true"
order-index="12" />
</parameters>
```

Примечание. Это конфигурация по умолчанию. Поэтому данное определение уже содержится в файле **db_adapter.xml**.

В некоторых случаях настроить соединение с базой данных таким образом невозможно. Примеры таких случаев — подключение к Oracle RAC или подключение с помощью драйвера базы данных, отличного от входящего в комплект поставки CMDB.

В этих случаях соединение описывается именем пользователя (схемой), паролем и URL-адресом для подключения.

Чтобы изменить параметры подключения, отредактируйте XML-файл адаптера следующим образом:

```
<parameters>
<!--The description attribute may be written in simple text or HTML.-->
<!--The host attribute is treated as a special case by CMDBRTSM-->
<!--and will automatically select the probe name (if possible)-->
<!--according to this attribute's value.-->
<!--Display name and description may be overwritten by I18N values-->
        <parameter name="url" display-name="Connection String" type="string"
description="The connection string to connect to the database"
mandatory="true" order-index="10" />
        <parameter name="credentialsId" display-name="Credentials ID"
type="integer" description="The credentials to be used" mandatory="true"
order-index="12" />
</parameters>
```

Пример URL-адреса для подключения к Oracle RAC с помощью стандартного драйвера Data Direct:

jdbc:mercury:oracle://labm3amdb17:1521;ServiceName=RACQA;AlternateServers=(labm3amdb18:1521);LoadBalancing=true.

4. Во временном каталоге откройте папку **adapterCode** и переименуйте **GenericDBAdapter** в соответствии со значением **adapter id**, использованным в предыдущем шаге.

   Эта папка содержит конфигурацию адаптера, например имя адаптера, запросы и классы в CMDB и поля в реляционной СУБД, поддерживаемой адаптером.

5. Настройте адаптер. Подробнее см. в разделе "Настройка адаптера — Минимальный метод" ниже.

6. Создайте ZIP-файл с именем, которое было назначено атрибуту **adapter id** в шаге "Измените XML-файл адаптера:" на странице 131.

   Примечание. Файл **descriptor.xml** — это файл по умолчанию, который присутствует в каждом пакете.

7. Сохраните новый пакет, созданный во время предыдущего шага. Каталог адаптеров по умолчанию: **C:\hp\UCMDB\UCMDBServer\content\adapters**.

Настройка адаптера — Минимальный метод

Самым простым является метод создания файла сопоставления **simplifiedConfiguration.xml**, который используется адаптером. Этот метод включает базовое заполнение или объединение в рамках одного типа ЭК.

В следующем разделе описывается метод сопоставления модели классов определенных типов ЭК в CMDB с реляционной СУБД.

Все приведенные здесь файлы конфигурации находятся в пакете **db-adapter.zip** в директории **C:\hp\UCMDB\UCMDBServer\content\adapters**, которая была извлечена в разделе "Подготовка пакета адаптера" на странице 131.

Примечание. Файл **orm.xml**, который создается автоматически при использовании данного метода, — это хороший пример, который можно использовать для расширенного метода.

Этот минимальный метод можно использовать для решения следующих задач:

- Объединение/заполнение одного узла, например атрибута узла.
- Демонстрация возможностей общего адаптера БД.

Этот метод:

- Поддерживает только объединение/заполнение одного узла
- Поддерживает только связь многие к одному.

Настройка файла adapter.conf.

Во время этого шага вы измените параметры в файле `adapter.conf` так, чтобы адаптер использовал упрощенный метод настройки.

1. Откройте файл **adapter.conf** в текстовом редакторе.
2. Найдите следующую строку: **use.simplified.xml.config=<true/false>**.
3. Измените ее на **use.simplified.xml.config=true**.

Пример. Заполнение данных узла и IP-адреса упрощенным методом

В данном примере описывается заполнение ЭК типа **Node**, связанного с ЭК типа **IP Address** связью типа containment, в UCMDB. В RDBMS есть таблица **simpleNode**, содержащая имя компьютера, его узел и IP-адрес.

Содержимое таблицы **simpleNode** показано ниже:

	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

Заполнение выполняется в три этапа, как показано ниже:

1. "Создание файла simplifiedConfiguration.xml" ниже
2. "Создайте TQL-запрос" на следующей странице
3. "Создание точки интеграции" на странице 137

Создание файла simplifiedConfiguration.xml

Создайте файл **simplifiedConfiguration.xml** следующим образом:

1. Создайте объект **cmdb-class** следующим образом:

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

   Тип ЭК — **node**, а имя таблицы RDBMS — **simpleNode**.
2. Задайте первичный ключ таблицы следующим образом:

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

   Первичный ключ аналогичен идентификатору объекта в файле **orm.xml**.
3. Задайте правило **reconciliation-by-two-nodes** следующим образом:

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

   Этот тег определяет связь между типами ЭК **Node** и **IpAddress**. Используется тип связи Containment. Выверка выполняется по двум связанным типам ЭК. Сопоставление атрибутов связанного узла (в данном случае IpAddress) задается атрибутом **connected-node**.
4. Добавьте условие **or** между атрибутами выверки следующим образом:

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

Данный тег определяет связь OR между атрибутами выверки, т.е. если значение хотя бы одного атрибута равно **true**, все правило выверки получает значение **true**.

5. Добавьте следующие атрибуты:

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

Данный тег задает сопоставление между **node.name** в UCMDB и столбцом **host_name** в таблице **simpleNode**.

Выполните то же самое с атрибутом **data_note**:

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

Добавьте атрибут связанного узла:

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

Данный тег задает сопоставление между **ip_address.name** и столбцом **ip_address** в таблице **simpleNode**.

6. Закройте открытый тег путем:

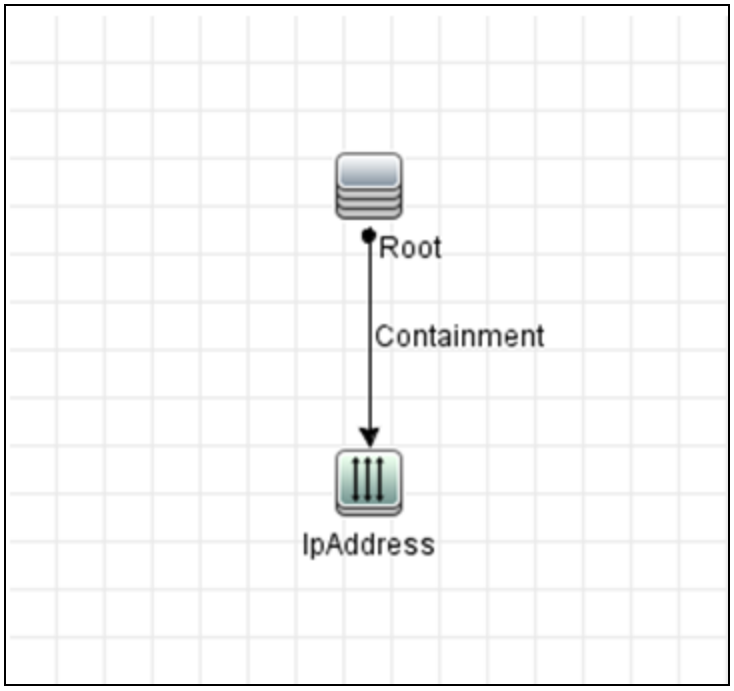
```
</or>

</reconciliation-by-two-nodes>

</cmdb-class>
```

Далее приведено содержимое файла simplifiedConfiguration.xml:

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

Создайте TQL-запрос

Результатом TQL-запроса будет ЭК **node**, соединенный связью containment с ЭК **ip_address**. Узел необходимо отметить как **root**, как показано ниже.



Для создания TQL-запроса:

1. Откройте раздел **Моделирование > Студия моделирования**.
2. Нажмите кнопку **Создать** и создайте новый запрос.
3. Откройте вкладку «Типы ЭК» и перетащите типы ЭК Node и IpAddress на страницу TQL.
4. Соедините между собой **Node** и **IpAddress** связью Containment.
5. Щелкните правой кнопкой мыши на элементе **Node** и выберите команду "Свойства узла запроса".
6. Измените значение **Имя элемента** на **Root**.
7. Откройте вкладку **Структура элемента**. В разделе "Условие атрибута" выберите **Особые атрибуты**. Выберите атрибуты **Name** и **Note** на панели «Доступные атрибуты» и переместите их на панель «Выбранные атрибуты».
8. Щелкните правой кнопкой мыши на элементе **IpAddress** и выберите команду "Свойства узла запросов".
9. Откройте вкладку **Структура элемента**. В разделе "Условие атрибута" выберите **Особые атрибуты**. Выберите атрибут **Name** на панели «Доступные атрибуты» и переместите его на панель «Выбранные атрибуты».
10. Сохраните TQL-запрос.

Создание точки интеграции

Создайте точку интеграции следующим образом:

1. Выберите **Управление потоком данных > Студия интеграции** и нажмите **Создать точку интеграции**.
2. Введите сведения о точке интеграции и нажмите **OK**:

3. Во вкладке «Заполнение» нажмите кнопку **Создать задание интеграции** и добавьте созданный ранее TQL-запрос.
4. Сохраните точку интеграции и нажмите кнопку **Выполнить полную синхронизацию**.

Настройка адаптера — Расширенный метод

Эти файлы конфигурации находятся в пакете **db-adapter.zip** в директории **C:\hp\UCMDB\UCMDBServer\content\adapters**, которая была извлечена при подготовке пакета адаптера. Дополнительные сведения см. в разделе "Подготовка пакета адаптера" на странице 131.

Эта задача включает следующие шаги:

- "Настройка файла orm.xml" ниже
- "Настройка файла reconciliation_rules.txt " на странице 141

Настройка файла orm.xml

Во время этого шага вы сопоставите типы ЭК и связи в CMDB с таблицами в реляционной СУБД.

1. Откройте файл **orm.xml** в текстовом редакторе.

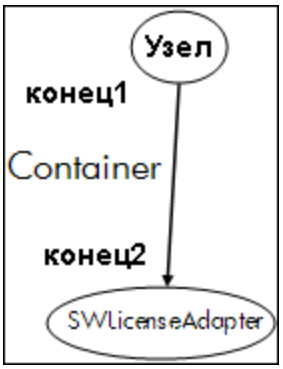
   По умолчанию этот файл содержит шаблон, который можно использовать для сопоставления любого числа типов ЭК и связей.

   > **Примечание.** Не изменяйте файл **orm.xml** в любой версии блокнота от корпорации Microsoft. Используйте Notepad++, UltraEdit или любой другой сторонний редактор.

2. Внесите изменения в файл в соответствии с сопоставляемыми объектами данных. См. дополнительные сведения в следующих примерах.

   Следующие типы связей могут быть сопоставлены в файле **orm.xml**:

   - Один к одному:

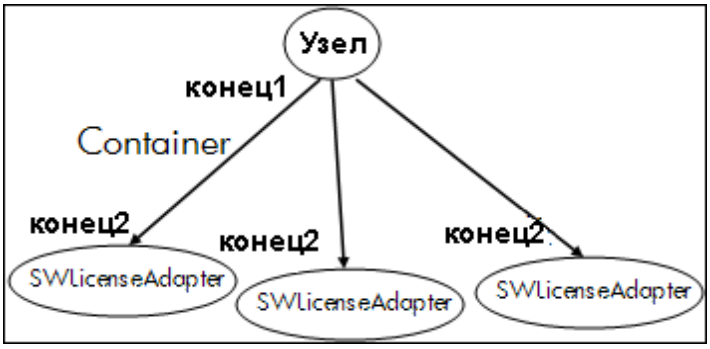
   

   Код связи этого типа:

```
<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>
```

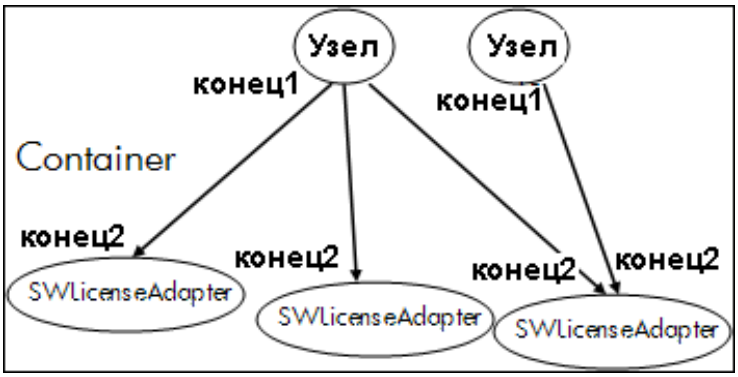
- Многие к одному:



Код связи этого типа:

```
<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>
```

- Многие ко многим:



Код связи этого типа:

```
<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>
```

См. дополнительные сведения о правилах именования в разделе "Правила именования" на странице 162.

Пример сопоставления объектов между моделью данных и реляционной СУБД:

Примечание. Атрибуты, которые не нужно настраивать, пропущены в следующих примерах.

- Класс типа ЭК CMDB:

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

- Имя таблицы в реляционной СУБД:

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

- Имя столбца уникального идентификатора в таблице реляционной СУБД:

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

- Имя атрибута типа ЭК в CMDB:

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

- Имя таблицы в во внешнем источнике данных:

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

- Имя атрибута нового типа ЭК, созданного в разделе "Создание типа ЭК" на странице 129:

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

- Имя соответствующей таблицы в реляционной СУБД:

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

- Уникальный идентификатор в реляционной СУБД:

- Имя атрибута в типе ЭК CMDB и имя соответствующего атрибута в реляционной СУБД:

Пример сопоставления связей между моделью данных и реляционной СУБД:

- Класс типа связи CMDB:

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

- Имя таблицы в реляционной СУБД, для которой действует связь:

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

- Уникальный идентификатор в реляционной СУБД:

```
<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>
```

- Тип связи и тип ЭК CMDB:

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

- Основной ключ и внешний ключ в реляционной СУБД:

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

Настройка файла reconciliation_rules.txt

Во время этого шага вы укажете правила, по которым адаптер выверяет CMDB и реляционную СУБД (только если используется система сопоставления, для обратной совместимости с версией 8.x):

1. Откройте файл **META-INF\reconciliation_rules.txt** в текстовом редакторе.
2. Внесите изменения в файл в соответствии с сопоставляемым типом ЭК. Например, для сопоставления типа ЭК узла используйте следующее выражение:

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

Примечание.

- Если данные в базе данных учитывают регистр, не удаляйте контрольный символ (**^**).
- Убедитесь, что для каждой открывающей квадратной скобки есть закрывающая скобка.

Дополнительные сведения см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 172.

Реализация подключаемого модуля

В этой задаче описывается порядок реализации и развертывания общего адаптера БД с подключаемыми модулями.

Примечание. Перед созданием подключаемого модуля для адаптера необходимо выполнить все шаги, описанные в разделе "Подготовка пакета адаптера" на странице 131.

1. Вариант 1 — Создать подключаемый модуль на Java
   a. Скопируйте следующие JAR-файлы из установочного каталога сервера UCMDB в каталог classpath средства разработки:
      - Скопируйте файлы **db-interfaces.jar** и **db-interfaces-javadoc.jar** из директории **tools\adapter-dev-kit\db-adapter-framework**.
      - Скопируйте файлы **federation-api.jar** и **federation-api-javadoc.jar** из каталога **\tools\adapter-dev-kit\SampleAdapters\production-lib**.

        Примечание. Дополнительные сведения о разработке подключаемых модулей содержатся в файлах **db-interfaces-javadoc.jar** и **federation-api-javadoc.jar**, а также в интерактивной документации по адресу:
        - **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. Создайте класс Java для реализации Java-интерфейса подключаемого модуля. Интерфейсы настраиваются в файле **db-interfaces.jar**. В таблице ниже представлен интерфейс, который должен быть реализован для каждого подключаемого модуля:

Тип подключаемого модуля	Имя интерфейса	Метод
Синхронизация всей топологии	FcmdbPluginForSyncGetFullTopology	getFullTopology

Тип подключаемого модуля	Имя интерфейса	Метод
Синхронизация изменений	FcmdbPluginForSyncGetChangesTopology	getChangesTopology
Синхронизация макета	FcmdbPluginForSyncGetLayout	getLayout
Получение поддерживаемых запросов	FcmdbPluginForSyncGetSupportedQueries	getSupportedQueries
Изменение определения и результатов TQL-запроса	FcmdbPluginGetTopologyCmdbFormat	getTopologyCmdbFormat
Изменение запроса макета для ЭК	FcmdbPluginGetCIsLayout	getCisLayout
Изменение запроса макета для связей	FcmdbPluginGetRelationsLayout	getRelationsLayout
Принудительная обратная отправка идентификаторов	FcmdbPluginPushBackIds	getPushBackIdsSQL

Класс подключаемого модуля должен иметь открытый конструктор по умолчанию. Кроме того, все интерфейсы предоставляют доступ к методу initPlugin. Этот метод гарантированно вызывается перед любым другим методом и используется для инициализации адаптера с объектом окружения, который включает этот адаптер.

Если реализован метод **FcmdbPluginForSyncGetChangesTopology**, существует два способа сообщить об изменениях:

- **Всегда сообщать всю корневую топологию.** Согласно этой топологии функция автоматического удаления находит ЭК, которые можно удалить. В этом случае необходимо включить функцию автоматического удаления следующим образом:

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

- **Сообщать о каждом экземпляре ЭК, который был удален или обновлен.** В

этом случае необходимо отключить функцию автоматического удаления следующим образом:

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

c. Убедитесь, что JAR-файлы SDK объединения и общего адаптера БД находятся в каталоге classpath перед компиляцией кода Java. SDK объединения — это файл **federation_api.jar**, который можно найти в каталоге **C:\hp\UCMDB\UCMDBServer\lib**.

d. Упакуйте класс в JAR-файл и поместите его в каталог adapterCode\<Your Adapter Name> пакета адаптера перед его развертыванием.

2. Вариант 2 — Создать подключаемый модуль на Groovy

a. Создайте файл Groovy (MyPlugin.groovy) (в разделе "Управление адаптерами", в папке файлов конфигурации соответствующего адаптера).

b. В классе Groovy следует применять соответствующие интерфейсы. Интерфейсы настраиваются в файле db-interfaces.jar (см. таблицу выше).

3. Подключаемые модули настраиваются с помощью файла **plugins.txt**, который находится в каталоге **\META-INF** адаптера.

Ниже представлен пример файла из адаптера DDMi:

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

Условные обозначения:

- строка комментария.

[<Adapter Type>] — начало раздела определения для указанного типа адаптера.

Под каждой строкой [<Adapter Type>] может находиться пустая строка. Это означает, что связанный класс подключаемого модуля или полное имя класса подключаемого модуля не могут быть отображены.

4. Упакуйте адаптер с новым JAR-файлом и обновленным файлом **plugins.xml**. Оставшиеся файлы в пакете должны быть одинаковы для всех адаптеров на основе общего адаптера БД.

Развертывание адаптера

1. В UCMDB откройте диспетчер пакетов. Подробнее см. в разделе "Страница "Диспетчер пакетов" в документе *Руководство по администрированию HP Universal CMDB*.
2. Щелкните **Развернуть пакеты на сервере (с локального диска)** и перейдите к пакету адаптера. Выберите пакет и нажмите **Открыть**, затем нажмите **Развернуть**, чтобы отобразить пакет в диспетчере пакетов.
3. Выберите пакет в списке в списке и щелкните значок **Просмотр ресурсов пакета** , чтобы проверить правильность распознания содержимого пакета в диспетчере пакетов.

Изменение адаптера

После создания и развертывания адаптера его можно изменить из UCMDB. Дополнительные сведения см. в разделе "Управление адаптерами" (*Руководство по управлению потоками данных в HP Universal CMDB*).

Создание точки интеграции

Во время этого шага вы проверите объединение, т. е. работоспособность подключения и правильность XML-файла. Однако эта проверка не включает проверку сопоставления XML-файла с правильными полями в реляционной СУБД.

1. В UCMDB откройте Студию интеграции (**Управление потоком данных > Студия интеграции**).
2. Создайте точку интеграции. Подробнее см. в разделе "Диалоговое окно "Создать точку интеграции/Изменить точку интеграции" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

   Во вкладке «Объединение» отображаются все типы ЭК, которые могут быть объединены с помощью этой точки интеграции. Подробнее см. в разделе "Вкладка "Объединение" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

Создание представления

Во время этого шага вы создадите представление для просмотра экземпляров типов ЭК.

1. В UCMDB откройте студию моделирования (**Моделирование > Студия моделирования**).
2. Создайте представление. Дополнительные сведения см. в разделе "Создание представления образца" в документе *Руководство по моделированию в HP Universal CMDB*.

Вычисление результатов

Во время этого шага вы проверите результаты.

1. В UCMDB откройте студию моделирования (**Моделирование > Студия моделирования**).
2. Откройте представление.
3. Рассчитайте результаты, нажав кнопку **Рассчитать число результатов запроса**  .
4. Нажмите кнопку **Предв. просмотр**, чтобы просмотреть ЭК в представлении.

Просмотр результатов

Во время этого шага выполняется просмотр результатов и устраняются проблемы в процедуре. Например, если в представлении ничего не отображается, проверьте определения в файле **orm.xml**, удалите атрибуты связи и перезагрузите адаптер.

1. В UCMDB откройте Диспетчер IT Universe (**Моделирование > IT Universe Manager**).
2. Выберите ЭК. Откроется вкладка «Свойства» с результатами объединения.

Просмотр отчетов

Во время этого шага вы ознакомитесь с отчетами по топологии. Подробнее см. в разделе Topology Reports Overview в документе *Руководство по моделированию в HP Universal CMDB*.

Активация файлов журнала

Чтобы понять потоки вычисления, жизненные циклы адаптера и просмотреть сведения об отладке, ознакомьтесь с файлами журнала. Дополнительные сведения см. в разделе "Файлы журнала адаптера" на странице 193.

Использование Eclipse для сопоставления атрибутов ЭК и таблиц базы данных

Внимание! Эта процедура предназначена для пользователей, хорошо владеющих разработкой содержимого. По любым вопросам обращайтесь в службу Поддержка ПО HP.

В этой задаче описывается установка и использование подключаемого модуля JPA, который входит в выпуск J2EE среды Eclipse, для решения следующих задач:

- Графическое сопоставление атрибутов классов CMDB и столбцов таблицы базы данных.
- Редактирование файла сопоставления (**orm.xml**) вручную с проверкой правильности изменений. Проверка правильности включает проверку синтаксиса, а также проверку правильности указанных атрибутов классов и сопоставленных столбцов таблицы.
- Развертывание файла сопоставления на сервере CMDB и просмотр сведений об ошибках в качестве дополнительной проверки.
- Формирование примера запроса на сервере CMDB и его выполнение непосредственно в Eclipse для тестирования файла сопоставления.

Версия 1.1. подключаемого модуля совместима с UCMDB 9.01 или более поздней версии и Eclipse IDE for Java EE Developers 1.2.2.20100217-2310 или более поздней версии.

Эта задача включает следующие шаги:

- "Необходимые условия" на следующей странице
- "Установка" на следующей странице
- "Подготовка рабочей среды" на следующей странице
- "Создание адаптера" на странице 149
- " Настройка подключаемого модуля CMDB " на странице 149
- "Импорт модели классов UCMDB" на странице 149
- "Построение ORM-файла — Сопоставление классов UCMDB с таблицами базы данных" на странице 149
- "Сопоставление идентификаторов" на странице 150
- "Сопоставление атрибутов" на странице 150
- "Сопоставление допустимой связи" на странице 150
- "Построение ORM-файла — Использование вторичных таблиц" на странице 151
- "Определение вторичной таблицы" на странице 151
- "Сопоставление атрибута с вторичной таблицей" на странице 152
- "Использование существующего ORM-файла в качестве основы" на странице 152
- "Импорт существующего ORM-файла из адаптера" на странице 152
- "Проверка файла orm.xml — Встроенная проверка правильности" на странице 153
- "Создание точки интеграции" на странице 153

- "Развертывание файла ORM в CMDB" на странице 153
- "Выполнение примера TQL-запроса" на странице 153

1. Необходимые условия

   Установите **Java Runtime Environment (JRE) 6** на компьютере Eclipse со следующего сайта:
   http://java.sun.com/javase/downloads/index.jsp.

2. Установка

   a. Загрузите и извлеките пакет **Eclipse IDE for Java EE Developers** с сайта **http://www.eclipse.org/downloads** в локальную папку, например, **C:\Program Files\eclipse**.

   b. Скопируйте файл **com.hp.plugin.import_cmdb_model_1.0.jar** из **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\bin** в **C:\Program Files\Eclipse\plugins**.

   c. Запустите **C:\Program Files\Eclipse\eclipse.exe**. При появлении сообщения, что виртуальная машина Java не найдена, запустите **eclipse.exe** со следующей командной строкой:

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

3. Подготовка рабочей среды

   Во время этого этапа вы настроите рабочую область, базу данных, подключения и свойства драйвера.

   a. Извлеките файл **workspaces_gdb.zip** из **C:\hp\UCMDB\UCMDBServer\tools\db-adapter-eclipse-plugin\workspace** into **C:\Documents and Settings\All Users.**

   > **Примечание.** Используйте точный путь к папке. Если распаковать файл по неверному пути или оставить его в архиве, процедура закончится неудачей.

   b. В Eclipse выберите **File > Switch Workspace > Other**:

   При использовании:

   - SQL Server выберите следующую папку: **C:\Documents and Settings\All Users\workspace_gdb_sqlserver**.
   - MySQL выберите следующую папку: **C:\Documents and Settings\All Users\workspace_gdb_mysql**.
   - Oracle выберите следующую папку: **C:\Documents and Settings\All Users\workspace_gdb_oracle**.

   c. Нажмите **OK**.

   d. В Eclipse откройте представление Project Explorer и выберите **<Active project> > JPA Content > persistence.xml > <имя активного проекта> > orm.xml**.

   e. В представлении Data Source Explorer (нижняя панель слева) щелкните подключение к базе данных правой кнопкой мыши и выберите меню **Properties**.

   f. В диалоговом окне **Properties for <Connection name>** выберите **Common** и установите

флажок **Connect every time the workbench is started**. Выберите **Driver Properties** и задайте свойства подключения. Нажмите **Test Connection** и проверьте работоспособность подключения. Нажмите **OK**.

g. В представлении Data Source Explorer щелкните подключение к базе данных правой кнопкой мыши и выберите **Connect**. Под значком подключения к базе данных откроется дерево, содержащее схемы базы данных и таблицы.

4. Создание адаптера

Создайте адаптер в соответствии с рекомендациями в разделе "Шаг 1: Создание адаптера" на странице 28.

5. Настройка подключаемого модуля CMDB

a. В Eclipse выберите **UCMDB > Settings**, чтобы открыть диалоговое окно **CMDB Settings**.

b. Если это еще не сделано, выберите недавно созданный проект JPA в качестве активного.

c. Введите имя хоста CMDB, например **localhost** или **labm1.itdep1**. Нет необходимости добавлять в адрес номер порта или префикс `http://`.

d. Укажите имя пользователя и пароль для доступа к CMDB API, обычно **admin/admin**.

e. Убедитесь, что папка **C:\hp** на сервере CMDB подключена как сетевой диск.

f. Выберите базовую папку соответствующего адаптера в **C:\hp**. Базовая папка — это папка, которая содержит файл **dbAdapter.jar** и вложенную папку **META-INF**. Путь должен иметь следующий формат: **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<имя адаптера>**. Убедитесь, что в конце пути отсутствует обратная косая черта (****).

6. Импорт модели классов UCMDB

Во время этого шага вы выберете типы ЭК для сопоставления с объектами JPA.

a. Выберите **UCMDB > Импорт модели классов CMDB**, чтобы открыть диалоговое окно **Выбор типа ЭК**.

b. Выберите типы ЭК, которые необходимо сопоставить с объектами JPA. Нажмите **OK**. Типы ЭК будут импортированы как классы Java. Убедитесь, что они отображаются в папке **src** активного проекта:

7. Построение ORM-файла — Сопоставление классов UCMDB с таблицами базы данных

Во время этого шага вы сопоставите классы Java (импортированные во время предыдущего шага) с таблицами базы данных.

a. Убедитесь, что подключение к БД активно. Щелкните активный проект правой кнопкой мыши в Project Explorer (по умолчанию называется myProject) в Project Explorer. Выберите представление JPA, установите флажок **Override default schema from connection** и выберите соответствующую схему базы данных. Нажмите **OK**.

b. Сопоставьте тип ЭК: в представлении структуры JPA щелкните правой кнопкой ветвь **Entity Mappings** и выберите **Add Class**. Откроется диалоговое окно **Add Persistent Class**. Не изменяйте поле **Map as** (**Entity**).

c. Нажмите **Browse** и выберите класс UCMDB для сопоставления (все классы UCMDB принадлежат пакету **generic_db_adapter**).

d. Нажмите кнопку **OK** в обоих диалоговых окнах. Выбранный класс будет отображаться в ветви **Entity Mappings** представления структуры JPA.

> **Примечание.** Если объект отображается без дерева атрибутов, щелкните активный проект в представлении Project Explorer правой кнопкой мыши. Нажмите **Close**, а затем **Open**.

e. В представлении JPA Details выберите основную таблицу базы данных, с которой должен быть сопоставлен класс UCMDB. Оставьте остальные поля без изменений.

8. Сопоставление идентификаторов

В соответствии со стандартами JPA каждый постоянный класс должен иметь хотя бы один атрибут идентификатора. Для классов UCMDB можно сопоставить до трех атрибутов в качестве идентификаторов. Потенциальные атрибуты идентификаторов называются **id1**, **id2** и **id3**.

Для сопоставления атрибута идентификатора выполните следующие действия.

a. Разверните соответствующий класс в ветви **Entity Mappings** представления структуры JPA, щелкните нужный атрибут правой кнопкой мыши (например, **id1**) и выберите **Add Attribute to XML and Map...**:

b. Откроется диалоговое окно **Add Persistent Attribute**. Выберите **Id** в поле **Map as** и нажмите кнопку **OK**.

c. В представлении JPA Details выберите столбец таблицы базы данных, с которой должно быть сопоставлено поле ID.

9. Сопоставление атрибутов

Во время этого шага вы сопоставите атрибуты со столбцами базы данных.

a. Разверните соответствующий класс в ветви **Entity Mappings** представления структуры JPA, щелкните нужный атрибут правой кнопкой мыши (например, **host_hostname**) и выберите **Add Attribute to XML and Map...**:

b. Откроется диалоговое окно **Add Persistent Attribute**. Выберите **Basic** в поле **Map as** и нажмите кнопку **OK**.

c. В представлении JPA Details выберите столбец таблицы базы данных, с которой должно быть сопоставлено поле атрибута.

10. Сопоставление допустимой связи

Выполните действия, описанные выше в шаге "Построение ORM-файла — Сопоставление классов UCMDB с таблицами базы данных" на предыдущей странице, чтобы сопоставить класс UCMDB, обозначающий допустимую связь. Имя каждого из таких классов будет иметь следующую структуру: **<end1 entity name>_<link name>_<end 2 entity name>**. Например, связь **Contains** между хостом и расположением будет обозначена классом Java с именем **generic_db_adapter.host_contains_location**. Подробнее см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 172.

a. Сопоставьте атрибуты идентификатора класса связи в соответствии с разделом "Сопоставление идентификаторов" на предыдущей странице. Для каждого атрибута идентификатора разверните группу флажков **Details** в представлении JPA Details и снимите флажки **Insertable** и **Updateable**.

b. Сопоставьте атрибуты **end1** и **end2** с классом следующим образом: для каждого атрибута **end1** и **end2** класса связи:

- Разверните соответствующий класс в ветви **Entity Mappings** представления структуры JPA, щелкните нужный атрибут правой кнопкой мыши (например, **end1**) и выберите **Add Attribute to XML and Map...**:
- В диалоговом окне **Add Persistent Attribute** выберите **Many to One** или **One to One** в поле **Map as**.
- Выберите **Many to One**, если указанный ЭК **end1** или **end2** может иметь несколько связей такого типа. В противном случае выберите **One to One**. Например, для связи **host_contains_ip** сторона **host** должна быть сопоставлена как **Many to One**, поскольку один хост может иметь несколько IP-адресов, а сторона **ip** должна быть сопоставлена как **One to One**, поскольку IP-адрес может быть назначен только одному хосту.
- В представлении JPA Details выберите **Target entity**, например **generic_db_adapter.host**.
- В разделе **Join Columns** представления JPA Details установите флажок **Override Default**. Щелкните **Edit**. В диалоговом окне **Edit Join Column** выберите столбец внешнего ключа таблицы базы данных связи, указывающий на запись в таблице целевого объекта **end1/end2**. Если имя столбца в таблице целевого объекта **end1/end2** сопоставлено с атрибутом идентификатора, оставьте значение **Referenced Column Name** без изменений. В противном случае выберите имя столбца, на который указывает столбец внешних ключей. Снимите флажки **Insertable** и **Updatable** и нажмите кнопку **OK**.
- Если целевой объект **end1/end2** имеет несколько идентификаторов, нажмите кнопку **Add**, чтобы добавить дополнительные столбцы и сопоставить их в соответствии с предыдущим шагом.

11. Построение ORM-файла — Использование вторичных таблиц

JPA обеспечивает сопоставление класса Java с несколькими таблицами базы данных. Например, **Host** можно сопоставить с таблицей **Device** для сохранения всех атрибутов и таблицей **NetworkNames** для сохранения **host_hostName**. В этом случае **Device** будет основной таблицей, а **NetworkNames** — вторичной таблицей. Можно задать любое количество вторичных таблиц. Единственное условие — между записями в основной и вторичной таблицах должна существовать связь один к одному.

12. Определение вторичной таблицы

Выберите соответствующий класс в представлении структуры JPA. В представлении **JPA Details** откройте раздел **Secondary Tables** и нажмите кнопку **Add**. В диалоговом окне **Add Secondary Table** выберите нужную вторичную таблицу. Оставьте остальные поля без изменений.

Если основная и вторичная таблица не включают одинаковых основных ключей, настройте дополнительные столбцы в разделе **Primary Key Join Columns** представления **JPA Details**.

13. Сопоставление атрибута с вторичной таблицей

Выполните следующие действия для сопоставления атрибута класса с полем вторичной таблицы.

a. Сопоставьте атрибут, как описано выше в разделе "Сопоставление атрибутов" на странице 150.

b. В разделе **Column** представления JPA Details выберите имя вторичной таблицы в поле **Table**, чтобы изменить значение по умолчанию.

14. Использование существующего ORM-файла в качестве основы

Для использования существующего файла **orm.xml** в качестве основы для разрабатываемого файла выполните следующие действия.

a. Убедитесь, что все типы ЭК, сопоставленные в существующем файле **orm.xml**, импортированы в существующий проект Eclipse.

b. Выберите и скопируйте все сопоставления объектов или некоторые из них из существующего файла.

c. Выберите вкладку **Source** файла **orm.xml** в перспективе Eclipse JPA.

d. Вставьте все скопированные сопоставления объектов под тегом **<entity-mappings>** в измененном файле **orm.xml** под тегом **<schema>**. Убедитесь, что тег schema настроен в соответствии с шагом "Построение ORM-файла — Сопоставление классов UCMDB с таблицами базы данных" на странице 149. Все вставленные объекты появятся в представлении структуры JPA. С этого момента сопоставления могут быть изменены как в графическом, так и в ручном режиме с помощью XML-кода в файле **orm.xml**.

e. Нажмите кнопку **Save**.

15. Импорт существующего ORM-файла из адаптера

Если адаптер уже существует, подключаемый модуль Eclipse можно использовать для редактирования ORM-файла в графическом режиме. Импортируйте файл **orm.xml** в Eclipse, измените его с помощью подключаемого модуля, а затем снова разверните на компьютере UCMDB. Чтобы экспортировать ORM-файл, нажмите кнопку на панели инструментов Eclipse. Появится диалоговое окно подтверждения. Нажмите **OK**. Файл ORM будет скопирован с компьютера UCMDB в активный проект Eclipse, и все соответствующие классы будут импортированы из модели классов UCMDB.

Если соответствующие классы не отображаются в представлении структуры JPA, щелкните активный проект правой кнопкой мыши в представлении Project Explorer, выберите **Close**, а затем **Open**.

С этого момента ORM-файл может быть изменен в графическом режиме с помощью Eclipse, а затем повторно развернут на компьютере UCMDB, как описано ниже в разделе "Развертывание файла ORM в CMDB" на следующей странице.

16. Проверка файла orm.xml — Встроенная проверка правильности

    Подключаемый модуль Eclipse JPA проверяет конфигурацию на наличие ошибок и отмечает их в файле **orm.xml**. Отслеживаются ошибки синтаксиса (например, неверные имена тегов, незакрытые теги и отсутствие идентификатора) и ошибки сопоставления (например, неверное имя атрибута или поля таблицы базы данных). Если ошибки существуют, их описания появятся в представлении **Problems**:

17. Создание точки интеграции

    Если для адаптера не существует точки интеграции в CMDB, ее можно создать с помощью студии интеграции. Подробнее см. в разделе "Студия интеграции" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

    Введите имя точки интеграции в открывшемся диалоговом окне. Файл **orm.xml** будет скопирован в папку адаптера. Точка интеграции будет создана со всеми импортированными типами ЭК в качестве поддерживаемых классов, кроме типов ЭК с несколькими узлами, если они настроены в файле **reconciliation_rules.txt**. Подробнее см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 172.

18. Развертывание файла ORM в CMDB

    Сохраните файл **orm.xml** и разверните его на сервере UCMDB. Для этого нажмите **UCMDB > Deploy ORM**. Файл **orm.xml** будет скопирован в папку адаптера, и адаптер будет перезагружен. Результат операции будет показан в диалоговом окне **Operation Result**. Если во время перезагрузки возникает ошибка, в диалоговом окне отображается трассировка стека исключений Java. Если точка интеграции еще не определена с помощью адаптера, ошибки сопоставления не будут обнаружены при развертывании.

19. Выполнение примера TQL-запроса

    a. Создайте запрос (не представление) в Студии моделирования. Подробнее см. в разделе "Студия моделирования" в документе *Руководство по моделированию в HP Universal CMDB*.

    b. Создайте точку интеграции с помощью адаптера, который использовался в шаге "Создание точки интеграции" выши. Подробнее см. в разделе "Диалоговое окно "Создать точку интеграции/Изменить точку интеграции" в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

    c. При создании адаптера убедитесь, что типы ЭК, которые должны участвовать в запросе, поддерживаются точкой интеграции.

    d. При настройке подключаемого модуля CMDB воспользуйтесь примером имени запроса в диалоговом окне настроек. Дополнительные сведения см. в описанном выше шаге " Настройка подключаемого модуля CMDB " на странице 149.

    e. Нажмите кнопку **Run TWL**, чтобы выполнить пример TQL-запроса и проверить, возвращает ли он необходимые результаты с использованием недавно созданного файла **orm.xml**.

Файлы конфигурации адаптеров

Файлы, описанные в этом разделе, находятся в пакете **db-adapter.zip** в каталоге **C:\hp\UCMDB\UCMDBServer\content\adapters**

В данном разделе описываются следующие файлы конфигурации:

- "Файл adapter.conf" на следующей странице
- "Файл simplifiedConfiguration.xml" на странице 156
- "Файл orm.xml" на странице 158
- "Файл reconciliation_types.txt" на странице 172
- "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 172
- "Файл transformations.txt" на странице 174
- "Файл discriminator.properties" на странице 175
- "Файл replication_config.txt" на странице 176
- "Файл fixed_values.txt" на странице 176
- "Файл persistence.xml" на странице 176

Общая конфигурация

- **adapter.conf.** Файл конфигурации адаптера. Дополнительные сведения см. в разделе "Файл adapter.conf" на следующей странице.

Простая конфигурация

- **simplifiedConfiguration.xml**. Файл конфигурации, который заменяет файлы **orm.xml**, **transformations.txt** и **reconciliation_rules.txt**, но поддерживает меньше возможностей. Дополнительные сведения см. в разделе "Файл simplifiedConfiguration.xml" на странице 156.

Расширенная настройка

- **orm.xml**. Файл сопоставлений объектов, в котором задается сопоставление типов ЭК CMDB и таблиц базы данных. Дополнительные сведения см. в разделе "Файл orm.xml" на странице 158.
- **reconciliation_rules.txt**. Содержит правила выверки. Дополнительные сведения см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 172.
- **transformations.txt.** Файл преобразований, в котором указываются конвертеры для преобразования значений CMDB в значения БД и наоборот. Дополнительные сведения см. в разделе "Файл transformations.txt" на странице 174.
- **Discriminator.properties.** В этом файле выполняется сопоставление всех поддерживаемых типов ЭК со списком возможных соответствующих значений, разделенных запятыми. Дополнительные сведения см. в разделе "Файл discriminator.properties" на странице 175.
- **Replication_config.txt.** Этот файл содержит список типов ЭК и связей, условия свойств

которых поддерживаются этим подключаемым модулем репликации. Дополнительные сведения см. в разделе "Файл replication_config.txt" на странице 176.

- **Fixed_values.txt.**Этот файл обеспечивает настройку фиксированных значений определенных атрибутов определенных типов ЭК. Дополнительные сведения см. в разделе "Файл fixed_values.txt" на странице 176.

Конфигурация Hibernate

- **persistence.xml.** Используется для переопределения встроенных конфигураций Hibernate. Дополнительные сведения см. в разделе "Файл persistence.xml" на странице 176.

Включение поддержки временных таблиц

Временные таблицы позволяют адаптеру более эффективно работать с удаленными базами данных. Таким образом снижается нагрузка на базу данных и на сеть, что увеличивает производительность системы.

Для включения поддержки временных таблиц на общем адаптере базы данных необходимо выполнить следующие условия:

- Учетная запись, используемая для подключения к базе данных, должна иметь права создания, изменения и удаления временных таблиц.
- Укажите следующие параметры в файле конфигурации адаптера adapter.conf:

  temp.tables.enabled=true

  performance.enable.single.sql=true

Примечание. Временные таблицы поддерживаются только при работе с Microsoft SQL и Oracle.

Файл adapter.conf

Этот файл содержит следующие параметры.

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

  Примечание. При использовании этого файла `orm.xml`, `transformations.txt` и `reconciliation_rules.txt` заменяются одним файлом с меньшим набором возможностей.

- **dal.ids.chunk.size=300.** Не изменяйте это значение.
- **dal.use.persistence.xml=false**. **true**: адаптер читает конфигурацию Hibernate из файла `persistence.xml`.

  Примечание. Рекомендуется переопределить конфигурацию Hibernate.

- **performance.memory.id.filtering=true**. Когда GDBA выполняет TQL-запросы, в некоторых случаях извлекается большое число идентификаторов, которые отправляются в базу данных с помощью SQL. Чтобы уменьшить число операций и повысить производительность, GDBA пытается прочитать представление/таблицу целиком и

отфильтровать результаты в памяти.

- **id.reconciliation.cmdb.id.type=string/bytes**. При сопоставлении общего адаптера БД с помощью выверки идентификаторов можно сопоставить **cmdb_id** со столбцом типа **string** или **bytes/raw** путем изменения свойства **META-INF/ adapter.conf**.
- **performance.enable.single.sql=true**. Это необязательный параметр. При его отсутствии в файле используется значение по умолчанию — **true**. Если параметр имеет значение **true**, общий адаптер БД пытается создать одно выражение SQL для каждого выполняемого запроса (запроса заполнения или объединенного запроса). Использование единых выражений SQL повышает производительность общего адаптера БД и снижает его требования к памяти. Если параметр имеет значение **false**, общий адаптер БД создает множество выражений SQL, на выполнение которых требуется больше времени и памяти. Даже если параметр имеет значение **true**, общий адаптер БД не создает одно выражение SQL в следующих ситуациях:
  - База данных, к которой подключается адаптер, не является ни Oracle, ни SQL Server.
  - В выполняемом TQL-запросе указано условие размерности, отличное от 0..* и 1..* (например, задано условие размерности 2..* или 0..2).
- **in.expression.size.limit=950** (по умолчанию). Этот параметр разделяет выражение 'IN' в выполняемом SQL-запросе при достижении предельного размера списка аргументов.
- **stringlist.delimiter.of.<CIT Name>.<Attribute Name>=<delimiter>**. Чтобы сопоставить атрибут из списка строковых переменных со столбцом базы данных, необходимо сопоставить атрибут со столбцом строковых переменных, в котором содержатся соединенные значения. Пример. Необходимо сопоставить атрибут **policy_category** с типом ЭК **policy**. При этом столбец строковых переменных содержит следующие значения: value1##value2##value3 (т.е., имеется список из 3 значений: value1, value2, value3). В этом случае следует использовать параметр: **stringlist.delimiter.of.policy.policy_category=##.**
- **temp.tables.enabled=true**. Включение поддержки временных таблиц для повышения производительности. Доступно только при включенном параметре **performance.enable.single.sql** (только для Microsoft SQL и Oracle). Могут потребоваться определенные права доступа к серверу базы данных.
- **temp.tables.min.value=50**. Число значений условий (или идентификаторов), необходимых для использования временных таблиц.

Файл simplifiedConfiguration.xml

Этот файл используется для простого сопоставления классов UCMDB с таблицами базы данных. Чтобы получить доступ к шаблону для редактирования файла, откройте в меню раздел **Управление адаптерами > db-adapter > Файлы конфигурации**.

Этот раздел охватывает следующие темы:

- "Шаблон файла simplifiedConfiguration.xml" ниже
- "Ограничения" на странице 158

Шаблон файла simplifiedConfiguration.xml

- Свойство **CMDB-class-name** — это тип multinode (узел, к которому объединенные типы ЭК

подключаются в TQL):

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

- **reconciliation-by-two-nodes**. Выверка может выполняться по одному или по двум узлам. В этом примере для выверки используется два узла.
- **connected-node-CMDB-class-name.** Второй тип класса, необходимый для TQL-запроса выверки.
- **CMDB-link-type.** Тип класса, необходимый для TQL-запроса выверки.
- **link-direction.** Направление связи в TQL-запросе выверки (от `node` к `ip_address` или от `ip_address` к `node`):

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

  Выражение выверки имеет форму OR, и каждое OR включает AND.
- **is-ordered.** Определяет способ выверки — в определенном порядке или с помощью обычного сравнения `OR`.

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

  Если свойство выверки получается из основного класса (multinode), используйте тег **attribute**. В противном случае используйте тег **connected-node-attribute**.
- **ignore-case.true**: если данные в модели классов UCMDB сравниваются с данными в реляционной БД , регистр не учитывается:

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

  Имя столбца — это имя столбца внешнего ключа (столбца со значениями, указывающими на столбец основных ключей multinode).

  Если основной столбец ключей multinode состоит из нескольких столбцов, необходимо несколько столбцов внешних ключей, по одному для каждого столбца основных ключей.

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

  Если существует не колько столбцов основных ключей, продублируйте этот столбец.

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

- Свойства **from-CMDB-converter** и **to-CMDB-converter** — это классы Java, которые реализуют следующие интерфейсы:
  - `com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB`

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

Используйте эти конвертеры, если значение в CMDB и значение в базе данных различаются.

В этом примере GenericEnumTransformer используется для преобразования счетчика в соответствии с XML-файлом, указанным в скобках (**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>
```

Ограничения

- Может использоваться для сопоставления только TQL-запросов с одним узлом (в источнике данных). Например, можно выполнить node > ticket и TQL-запрос ticket. Чтобы вызвать иерархию узлов из базы данных, используйте расширенный файл **orm.xml**.
- Поддерживаются только связи один ко многим. Например, можно вызвать одну или несколько заявок для каждого узла. Вызывать заявки, которые принадлежат нескольким узлам, нельзя.
- Подключение одного класса с разными типами ЭК CMDB. Например, если пользователь укажет, что элемент ticket подключен к узлу node, его нельзя также подключить к элементу application.

Файл orm.xml

Этот файл используется для сопоставления типов ЭК CMDB с таблицами базы данных.

Шаблон, используемый для создания нового файла, находится в директории **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\GenericDBAdapter\META-INF**.

Чтобы изменить XML-файл для развернутого адаптера, откройте в меню раздел **Управление адаптерами > db-adapter > Файлы конфигурации**.

Этот раздел охватывает следующие темы:

- "Шаблон файла orm.xml" на следующей странице
- "Несколько ORM-файлов" на странице 162

- "Правила именования" на странице 162
- "Использование встроенных инструкций SQL вместо имен таблиц" на странице 163
- "Схема orm.xml" на странице 163
- "Пример создания файла orm.xml" на странице 168
- " Настройка файлов orm.xml для работы с различными версиями продуктов" на странице 171

Шаблон файла orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.0"
xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd">
    <description>Generic DB adapter orm</description>
```

Не изменяйте имя пакета.

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

entity. Имя типа ЭК в CMDB . Это объект multinode.

Убедитесь, что **class** включает префикс **generic_db_adapter.**.

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

Используйте вторичную таблицу, если объект сопоставлен с несколькими таблицами.

```
        <secondary-table name="" />
        <attributes>
```

Для наследования одной таблицы с дискриминатором используйте следующий код:

```
        <inheritance strategy="SINGLE_TABLE" />
        <discriminator-value>node</discriminator-value>
        <discriminator-column name="[column_name]" />
```

Атрибуты с тегом **id** являются столбцами основных ключей. Убедитесь, что для столбцов основных ключей выполняются следующие правила именования: **idX** (id1, id2 и др.), где **X** — это индекс столбца первичного ключа.

```
            <id name="id1">
```

```
```

Изменение только имени столбца первичного ключа.

```
                <column updatable="false" insertable="false" name="[column_name]
" />
                <generated-value strategy="TABLE"/>
            </id>
```

basic. Используется для объявления атрибутов CMDB. Изменяйте только свойства **name** и **column_name**.

```
            <basic name="name">
                <column updatable="false" insertable="false" name="[column_name]
" />
            </basic>
```

Для наследования одной таблицы с дискриминатором сопоставьте дополнительные классы следующим образом:

```
    <entity name="[cmdb_class_name]" class="generic_db_adapter.nt" name="nt">

        <discriminator-value>nt        </discriminator-value>
        <attributes>
    </entity>
    <entity class="generic_db_adapter.unix" name="unix">
        <discriminator-value>unix</discriminator-value>
        <attributes>
    </entity>
    <entity name="[CMDB_class_name]" class="generic_db_adapter.[CMDB[cmdb_class_
name]">
        <table name="[default_table_name]" />
        <secondary-table name="" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_name]
" />
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column updatable="false" insertable="false" name="[column_name]
" />
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column updatable="false" insertable="false" name="[column_name]
" />
                <generated-value strategy="TABLE"/>
```

```
            </id>
```

В следующих примерах представлено имя атрибута CMDB без префикса:

```
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_name]
" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_name]
" />
            </basic>
            <basic name="[CMDB_attribute_name]">
                <column updatable="false" insertable="false" name="[column_name]
" />
            </basic>
        </attributes>
    </entity>
```

Это объект связи. Правило именования: **end1Type_linkType_end2Type**. В этом примере **end1Type** — это **node**, а **linkType** — **composition**.

```
    <entity name="node_composition_[CMDB_class_name]" class="generic_db_
adapter.node_composition_[CMDB_class_name]">
        <table name="[default_table_name]" />
        <attributes>
            <id name="id1">
                <column updatable="false" insertable="false" name="[column_name]
" />
                <generated-value strategy="TABLE"/>
            </id>
```

Целевой объект — это объект, на который указывает это свойство. В этом примере **end1** сопоставляется с объектом **node**.

many-to-one. С одним узлом можно соединить несколько связей.

join-column. Столбец, содержащий идентификаторы **end1** (идентификаторы целевого объекта).

referenced-column-name. Имя столбца целевого объекта (**node**), который содержит идентификаторы, используемые в добавляемом столбце.

```
            <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. Одна связь может быть соединена с одним элементом **[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>
```

атрибут node Ниже приведен пример добавления атрибута node.

```
<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>
```

Несколько ORM-файлов

Система поддерживает использование нескольких файлов сопоставления. Имя каждого файла сопоставления должно заканчиваться на **orm.xml**. Все файлы сопоставления должны находиться в папке META-INF адаптера.

Правила именования

- В каждом объекте свойство класса должно соответствовать свойству имени с префиксом generic_db_adapter.
- Столбцы основных ключей должны иметь имя **idX**, где **X** = **1**, **2**, ..., в соответствии с числом основных ключей в таблице.
- Имена атрибутов должны соответствовать именам атрибутов классов, регистр учитывается.
- Имя связи будет иметь следующий вид: end1Type_linkType_end2Type.
- Перед типами ЭК CMDB, которые также являются зарезервированными словами Java, должен стоять префикс **gdba_**. Например, для типа ЭК CMDB **goto** объект ORM должен

иметь имя **gdba_goto**.

Использование встроенных инструкций SQL вместо имен таблиц

Вы можете сопоставить объекты с внутренними выражениями `select` вместо таблиц баз данных. Это соответствует настройке представления в базе данных и сопоставлению объекта с этим представлением. Пример:

```
    <entity class="generic_db_adapter.node">
        <table name="(select d.id as id1, d.name as name , d.os as host_os from
Device d)" />
```

В этом примере атрибуты узла должны быть сопоставлены со столбцами id1, name и host_os, а не id, name и os.

Действуют следующие ограничения:

- Встроенные инструкции SQL доступны только при использовании Hibernate в качестве поставщика JPA.
- Использование круглых скобок вокруг внутренней инструкции SQL select обязательно.
- Элемент **<schema>** не должен присутствовать в файле **orm.xml**. При использовании Microsoft SQL Server 2005 это означает, что все имена таблиц должны иметь префикс `dbo.`, а не глобальное определение `<schema>dbo</schema>`.

Схема orm.xml

В следующей таблице приводится описание стандартных элементов файла **orm.xml**. Полная схема доступна по адресу http://java.sun.com/xml/ns/persistence/orm_1_0.xsd. Этот список не полон, он приводится, чтобы объяснить определенные действия Java Persistence API для общего адаптера БД.

Имя элемента и путь	Описание	Атрибуты
entity-mappings	Корневой элемент документа по сопоставлению объектов. Этот элемент должен совпадать с элементами в файлах образцов GDBA.	
description (entity-mappings)	Текстовое описание документа по сопоставлению объектов. (необязательно)	
package (entity-mappings)	Имя пакета Java, который содержит классы сопоставления. Значение должно содержать текст `generic_db_adapter`.	1. **Имя:** name **Описание:** Имя типа ЭК UCMDB, с которым сопоставляется объект. Если

Имя элемента и путь	Описание	Атрибуты
		объект сопоставляется со связью в CMDB, его имя должно иметь следующий формат: `<end_1>_<link_name>_<end_2>`. Например, `node_composition_cpu` определяет объект, который будет сопоставлен с составной связью между узлом и ЦП. Если имя типа ЭК совпадает с именем класса Java без префикса, это поле можно пропустить. **Обязательно?** Необязательно **Тип:** Строка 2. **Имя:** class **Описание:** Полное имя класса Java, который будет создан для этого объекта БД. Имя пакета класса Java должно совпадать с именем в элементе `package`. В качестве имени класса нельзя использовать зарезервированные слова Java, такие как interface или switch. Вместо этого к имени следует добавить префикс `gdba_`. Таким образом, интерфейс получит имя `generic_db_adapter.gdba_interface`. **Обязательно?** Обязательно **Тип:** Строка
table (entity-mappings>entity)	Этот элемент определяет основную таблицу объекта базы данных. Можно существовать только в одном экземпляре. Обязательно.	**Имя:** name **Описание:** Имя основной таблицы. Если имя таблицы не содержит схему, к которой принадлежит, поиск таблицы будет выполнен только в схеме пользователя, от имени которого была создана точка интеграции.

Имя элемента и путь	Описание	Атрибуты
		Кроме того, это может быть любая допустимая инструкция SELECT. Если это инструкция SELECT, ее необходимо взять в скобки. **Обязательно?** Обязательно **Тип:** Строка
secondary-table (entity-mappings>entity)	Этот элемент может использоваться для указания вспомогательной таблицы для объекта БД. Эта таблица должна быть подключена к основной таблице со связью один к одному. Можно настроить несколько вторичных таблиц. Необязательно.	**Имя:** name **Описание:** Имя вторичной таблицы. Если имя таблицы не содержит схему, к которой принадлежит, поиск таблицы будет выполнен только в схеме пользователя, от имени которого была создана точка интеграции. Кроме того, это может быть любая допустимая инструкция SELECT. Если это инструкция SELECT, ее необходимо взять в скобки. **Обязательно?** Обязательно **Тип:** Строка
primary-key-join-column (entity-mappings > entity > secondary-table)	Если основная и вторичная таблица не соединены с помощью полей с одинаковыми элементами, в этом элементе указывается имя поля первичного ключа во вторичной таблице, которое должно быть соединено с полем первичного ключа основной таблицы.	**Имя:** name **Описание:** Имя поля первичного ключа во вторичной таблице. Если элемент не существует, предполагается что поле первичного ключа имеет то же имя, что поле первичного ключа в основной таблице. **Обязательно?** Необязательно **Тип:** Строка
inheritance (entity-mappings>entity)	Если текущий объект является родительским объектом для семейства объектов БД, этот элемент отмечает его как родитель. Необязательно.	**Имя:** strategy **Описание:** Определяет способ наследования, реализованный в БД. **Обязательно?** Обязательно **Тип:** Одно из следующих значений: • SINGLE_TABLE: Этот объект и все родительские объекты существуют в одной таблице. • JOINED: Дочерние объекты

Имя элемента и путь	Описание	Атрибуты
		находятся в присоединенных таблицах. • TABLE_PER_CLASS: Каждый объект полностью определяется отдельной таблицей.
discriminator-column (entity-mappings>entity)	Если используется тип наследования SINGLE_TABLE, этот элемент используется для указания имени поля, используемого для определения типа объекта для каждой строки.	**Имя:** name **Описание:** Имя столбца дискриминатора. **Обязательно?** Обязательно **Тип:** Строка
discriminator-value (entity-mappings>entity)	Этот элемент определяет тип объекта в дереве наследования. Это имя должно совпадать с именем в файле **discriminator.properties** для группы значений этого типа объектов.	
attributes (entity-mappings>entity)	Корневой элемент всех сопоставлений атрибутов для объекта.	
id (entity-mappings>entity attributes)	Этот элемент определяет поле ключа объекта. Должно быть настроено хотя бы одно поле id. Если существует несколько элементов id, соответствующие поля образуют составной ключ объекта. Следует избегать использования составных ключей для объектов ЭК (но не для связей).	**Имя:** name **Описание:** Строка типа idX, где X — это число от 1 до 9. Первое значение id должно быть отмечено как id1, второе — как id2 и так далее. Это НЕ имена ключевых атрибутов в UCMDB. **Обязательно?** Обязательно **Тип:** Строка
basic (entity-mappings>entity attributes)	Этот элемент определяет сопоставление между полем в таблице, которое не является частью первичного ключа таблицы, и атрибутом UCMDB.	**Имя:** name **Описание:** Имя атрибута UCMDB, с которым сопоставляется поле. Атрибут должен существовать в типе ЭК UCMDB, с которым сопоставлен текущий объект.

Имя элемента и путь	Описание	Атрибуты
		Обязательно? Обязательно **Тип:** Строка
column (entity-mappings>entity> attributes> id -OR- (entity-mappings>entity> attributes> basic)	Определяет имя столбца в таблице для базового сопоставления или поля id.	1. **Имя:** name **Описание:** Имя поля. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** table **Описание:** Имя таблицы, к которой принадлежит поле. Это должна быть основная таблица или одна из вторичных таблиц объекта. Если этот атрибут пропущен, предполагается, что поле принадлежит к основной таблице. **Обязательно?** Необязательно **Тип:** Строка
one-to-one (entity-mappings>entity> attributes)	Определяет столбец, значение которого находится в другой таблице и две таблицы соединены с помощью связи один к одному. Этот элемент поддерживается для сопоставления объектов связей, но не поддерживается для других типов ЭК. Это единственный способ настроить сопоставление между таблицей и связью UCMDB.	1. **Имя:** name **Описание:** Какую из двух сторон представляет поле. **Обязательно?** Обязательно **Тип:** `end1` или `end2` 2. **Имя:** target-entity **Описание:** Имя объекта, на который ссылается сторона. **Обязательно?** Обязательно **Тип:** Одно из имен объектов, указанных в документе по сопоставлению объектов
join-column (entity-mappings>entity attributes>one-to-one)	Определяет способ присоединения целевого объекта, заданного в родительском элементе с сопоставлением «один к одному», к текущему объекту.	1. **Имя:** name **Описание:** Имя поля в текущей таблице, которое будет использоваться для присоединения «один к одному». **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** name

Имя элемента и путь	Описание	Атрибуты
		Описание: Имя поля в совместном объекте, по которому выполняется присоединение. Если атрибут пропущен, предполагается, что совместная таблица включает столбец с тем же именем, что поле, указанное в атрибуте имени. **Обязательно?** Необязательно **Тип:** Строка

Пример создания файла orm.xml

Ниже приведен пример создания файла **orm.xml**. В этом примере таблицы SQL из удаленной базы данных сопоставляются с типами ЭК в UCMDB.

Используя таблицы в следующем формате в удаленной базе данных, заполняет таблицу **Hosts** сведениями об узлах, таблицу **IP_Addresses** — IP-адресами, а также создает связи между ними следующим образом:

Таблица Hosts

host_name	host_id
Test1	1
Test2	2
Test3	3

Таблица 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

Таблица Host_IP_Link (связи между Node и IP Address)

host_id	ip_id
1	1
2	2
2	3
3	4

Первичным ключом для таблицы **Hosts** является поле **host_id**, а для таблицы **IP_Addresses Table** — поле **ip_id**. В таблице **Host_IP_Link** поля **host_id** и **ip_id** являются внешними ключами из таблиц **Hosts** и **IP_Addresses**.

Согласно описанным выше таблицам, файл **orm.xml** создается следующим образом: В данном примере используются объекты **node**, **ip_address** и **node_containment_ip_address**.

1. Создайте объект **node** путем сопоставления **host_id** из таблицы **Hosts** следующим образом:

```
<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>
```

В качестве объекта **class** необходимо использовать тип ЭК, который уже существует в UCMDB. Необходимо указать таблицу в базе данных, в которой уже содержатся сведения об идентификаторах и хостах. Атрибут ID необходим для определения конкретных хостов и будет использоваться в процессе сопоставления позже. В этом примере будет выполнено заполнение атрибута **name** данного объекта значениями столбца **host_name** в таблице Hosts.

2. Для следующего объекта выполните сопоставление IP-адресов из таблицы 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. Далее необходимо создать связь между Node и IP Address с помощью таблицы сопоставления со ссылкой на поле **ip_id** (хотя при желании можно создать ссылку и на **host_id**, и на **ip_id**).

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

```
       <one-to-one target-entity="ip_address" name="end2">
        <join-column name="ip_id"/>
       </one-to-one>
    </attributes>
</entity>
```

Имя объекта для контейнера имеет следующий формат: [end1 CIT]_[link CIT]_[end2 CIT]. В данном примере, поскольку используется тип связи **Containment**, имя объекта для контейнера имеет вид: **node_containment_ip_address**, а классом объекта является **generic_db_adapter.node_containment_ip_address**. Значение ID в данном блоке кода обязательно, и хотя данный пример работает с одним ID интерфейса, в обоих столбцах могут использоваться ссылки на id1 и id2. В этом случае код будет следующим:

```
<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>
```

Два конца этой связи имеют значения ‘много к одному’ и ‘одно к одному’, т.е. IP-адрес можно связать только с одним узлом, но узел можно связать с несколькими IP-адресами. Указываются поля из таблицы Links, которые ссылаются на таблицы Hosts и Interfaces.

Настройка файлов orm.xml для работы с различными версиями продуктов

В настройках файла **orm.xml** можно задать использование адаптером определенного файла **orm.xml** при работе с определенной версией ПО. Например, на удаленном хранилище данных установлены две версии продукта: x и y. Для каждой из них возможна отдельная процедура сопоставления.

Настройка файлов orm.xml для работы с различными версиями продуктов:

1. Добавьте в файл **adapter.xml** параметр **version** и укажите возможные версии как **valid-values**.
2. В пакете адаптера в папке META-INF создайте папку **VersionOrm**.

3. В папке **VersionOrm** создайте отдельный файл **orm.xml** для каждой версии ПО. В префиксе имени файла необходимо указать номер версии. Например, для версии **x** следует задать имя файла как **x_orm.xml**.

Примечание. Загрузка файла **orm.xml** из папки META-INF происходит при работе с любой версией ПО на удаленной машине, даже если отдельные версии **orm.xml** не создавались. Для всех версий может использоваться один принцип сопоставления.

Файл reconciliation_types.txt

В UCMDB 10.00 файл **reconciliation_types.txt** больше не используется. Типы ЭК можно использовать в целях выверки. Механизм объединения выполняет сопоставление автоматически.

Файл reconciliation_rules.txt (для обратной совместимости)

Этот файл используется для настройки правил выверки, если пользователю необходима выверка и служба DBMappingEngine настроена для адаптера. Если DBMappingEngine не используется, применяется общий механизм выверки UCMDB, и настройка этого файла не требуется.

Каждая строка файла представляет правило. Пример:

```
multinode[node] expression[^node.name OR ip_address.name] end1_type[node]
end2_type[ip_address] link_type[containment]
```

Для элемента multinode указано имя multinode (тип ЭК UCMDB, соединенный с типом ЭК объединенной базы данных в TQL-запросе).

Это выражение включает логику, которая определяет равенство элементов multinode (элемента multinode в UCMDB и элемента multinode в источнике базы данных).

Выражение состоит из операторов `OR` и `AND`.

Правило именования атрибутов в части выражения: `[className].[attributeName]`. Например, `attributeName` в `ip_address` записывается как `ip_address.name`.

Для упорядоченного сопоставления (если первое выражение `OR` возвращает ответ, что элементы multinode не равны, второе подвыражение OR не проверяется) используйте `ordered expression` вместо `expression`.

Чтобы игнорировать регистр при сравнении, используйте контрольный знак (**^**).

Параметры `end1_type`, `end2_type` и `link_type` используются, только если TQL-запрос выверки содержит два узла, а не просто элемент multinode. В этом случае TQL-запрос выверки будет иметь следующий вид: `end1_type > (link_type) > end2_type`.

Добавление соответствующего макета не требуется, так как он берется из выражения.

Типы правил выверки

Правила выверки принимают форму условий OR и AND. Эти правила можно определить для нескольких узлов (например, узел идентифицируется по `name from nodeAND/ORname from ip_address`).

Существуют следующие варианты сопоставления.

- **Сопоставление по порядку.** Выражение выверки читается слева направо. Два подвыражения `OR` считаются равными, если включают значения и являются равными. Два подвыражения `OR` считаются неравными, если включают значения и не являются равными. Во всех остальных случаях решение не принимается, и выполняется проверка следующего подвыражения `OR`.

  name from node OR from ip_address. Если UCMDB и источник данных включают значение `name` и они равны, узлы считаются равными. Если оба узла включают значение `name`, но не являются равными, узлы считаются неравными без проверки значения `name ip_address`. Если UCMDB или источник данных отсутствуют, проверяются `name of node` и `name of ip_address`.

- **Обычное сопоставление.** Если равенство имеет место в одном или нескольких подвыражениях `OR`, UCMDB и источник данных считаются верными.

  name from node OR from ip_address. Если соответствие по `name of node` отсутствует, `name of ip_address` проверяется на равенство.

Для сложных выверок, в которых объект выверки моделируется в модели классов как несколько типов ЭК со связями (такими как `node`), сопоставление узла сверхмножества включает все соответствующие атрибуты смоделированных типов ЭК.

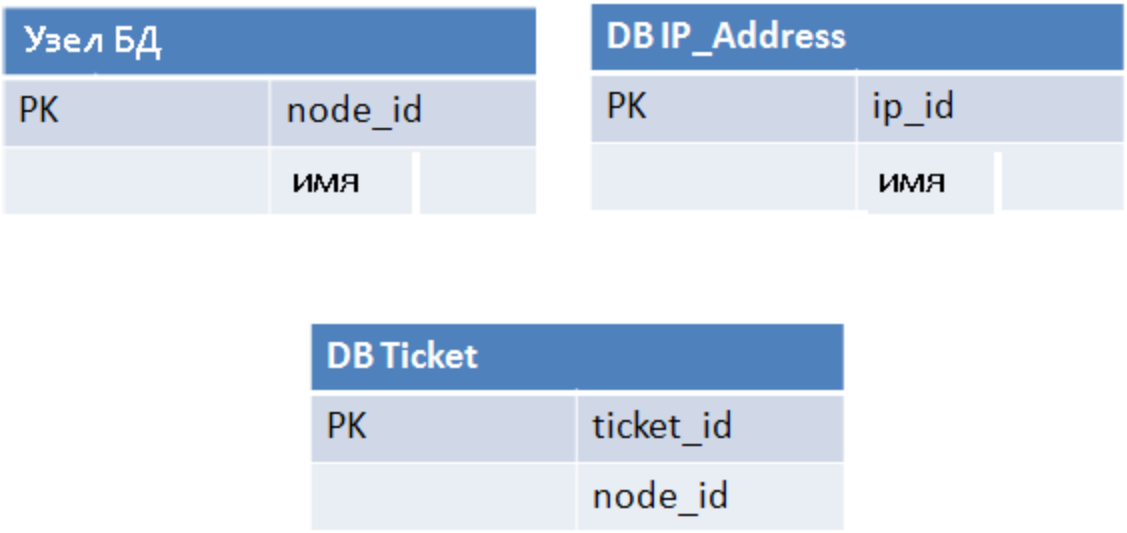
> **Примечание.** В результате будет действовать ограничение — все атрибуты выверки в источнике данных должны находиться в таблицах, которые используют общий первичный ключ.

Другое ограничение: TQL-запрос выверки должен включать не более двух узлов. Например, TQL-запрос `node > ticket` может включать узел UCMDB и заявку в источнике данных.

Для выверки результатов значение `name` должно быть получено из узла или элемента `ip_address`.

Если `name` в UCMDB имеет формат `*.m.com`, конвертер может использоваться для преобразования значений из UCMDB в объединенную базу данных и наоборот.

Столбец `node_id` в заявке базы данных используется для соединения объектов (определенная связь также может быть задана в таблице узла):



Примечание. Три таблицы должны быть часть объединенного источника в реляционной СУБД, а не базы UCMDB.

Файл transformations.txt

Этот файл содержит все определения конвертеров.

Каждая строка содержит новое определение.

Шаблон файла transformations.txt

```
entity[[CMDB_class_name]] attribute[[CMDB_attribute_name]] to_DB_class
[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.xml)]
from_DB_class[com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.
GenericEnumTransformer(generic-enum-transformer-example.xml)]
```

entity. Имя объекта в соответствии с файлом `orm.xml`.

attribute. Имя атрибута в соответствии с файлом `orm.xml`.

to_DB_class. Полное имя класса, который реализует интерфейс **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerToExternalDB**. Элементы в скобках назначаются в этом конструкторе классов. Используйте этот конвертер для преобразования значений CMDB в значения базы данных, например для добавления суффикса **.com** к каждому имени узла.

from_DB_class. Полное имя класса, который реализует интерфейс **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.FcmdbDalTransformerFromExternalDB**. Элементы в скобках назначаются в этом конструкторе классов. Используйте этот конвертер для преобразования значений базы данных в значения CMDB, например для добавления суффикса **.com** к каждому имени узла.

Дополнительные сведения см. в разделе "Встроенные конвертеры" на странице 179.

Файл discriminator.properties

В этом файле выполняется сопоставление всех поддерживаемых типов ЭК (которые также используются как значения дискриминатора в orm.xml) со списком возможных соответствующих значений, разделенных запятыми, либо условием, которому должны соответствовать значения в столбце дискриминатора.

При указании условий используется следующий синтаксис: `like(condition)`, где `condition` — строка, которая может содержать следующие групповые символы:

- **%** (процент) - обозначает любую последовательность символов любой длины (включая нулевую)
- **_** (подчеркивание) - обозначает один символ

Например, условию `like(%unix%)` отвечают слова unix, linux, unix-aix и т. д. Условия like применяются только для столбцов со строковыми данными.

Кроме того, можно сопоставить одно значение дискриминатора с любыми значениями, не относящимися к другим дискриминаторам, указав `'all-other'`.

Если создаваемый адаптер использует дискриминатор, необходимо указать все значения дискриминатора в файле **discriminator.properties**.

Пример сопоставления дискриминатора:

Например, адаптер поддерживает типы ЭК `node`, `nt` и `unix`, а в базе данных содержится одна таблица `t_nodes` со столбцом **type**. Если в столбце type указано значение 10001, строка соответствует ЭК типа node; если type имеет значение 10004, строка соответствует ЭК типа unix и т.д. Файл **discriminator.properties** в этом случае может выглядеть следующим образом:

```
node=10001, 10005
nt=10002,10003
unix=2%
mainframe=all-other
```

Файл **orm.xml** включает следующий код:

```
    <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>
```

Атрибут discriminator_column рассчитывается следующим образом:

- Если в столбце **type** у какой-либо записи содержится значение `10002` или `10003`, эта запись сопоставляется с ЭК типа **nt**.
- Если в столбце **type** у какой-либо записи содержится значение `10001` или `10005`, эта запись сопоставляется с ЭК типа **node**.
- Если значение в столбце **type** у какой-либо записи начинается на `2`, эта запись сопоставляется с ЭК типа **unix**.
- Записи с любыми другими значениями в столбце **type** сопоставляются с типом ЭК **mainframe**.

Примечание. Тип ЭК **node** также является родителем **nt** и **unix**.

Файл replication_config.txt

Этот файл содержит список типов ЭК и связей, условия свойств которых поддерживаются этим подключаемым модулем репликации. Дополнительные сведения см. в разделе "Подключаемые модули" на странице 184.

Файл fixed_values.txt

Этот файл обеспечивает настройку фиксированных значений определенных атрибутов определенных типов ЭК. Таким образом, каждому из этих атрибутов можно назначить фиксированное значение, не сохраненное в базе данных.

Файл должен содержать ноль или более записей в следующем формате:

```
entity[<entityName>] attribute[<attributeName>] value[<value>]
```

Пример:

```
entity[ip_address] attribute[ip_domain] value[DefaultDomain]
```

Файл также поддерживает список констант. Для указания списка констант используется следующий синтаксис:

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

Файл persistence.xml

Этот файл используется для переопределения параметров Hibernate по умолчанию и добавления поддержки типов базы данных, которые не работают в стандартной конфигурации (встроенные типы БД: Oracle Server, Microsoft SQL Server и MySQL).

Для поддержки нового типа базы данных укажите поставщика пула подключений (значение по умолчанию: `c3p0`) и драйвер JDBC для своих файлов (поместите `JAR`-файлы в папку адаптера).

Чтобы увидеть все доступные значения Hibernate, которые могут быть изменены, проверьте класс **org.hibernate.cfg.Environment** (подробнее см. на сайте http://www.hibernate.org.)

Пример файла persistence.xml:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
    <!-- Don't change this value -->
    <persistence-unit name="GenericDBAdapter">
        <свойства>
            <!--  Don't change this value  -->
            <property name="hibernate.archive.autodetection" value="class,
                hbm" />
            <!--The driver class name"/-->
            <property name="hibernate.connection.driver_class" value="com.
mercury.jdbc.MercOracleDriver" />
            <!--The connection url"/-->
            <property name="hibernate.connection.url" value="jdbc:mercury:
oracle://artist:1521;sid=cmdb2" />
            <!--DB login credentials"/-->
            <property name="hibernate.connection.username" value="CMDB" />
            <property name="hibernate.connection.password" value="CMDB" />
            <!--connection pool properties"/-->
            <property name="hibernate.c3p0.min_size" value="5" />
            <property name="hibernate.c3p0.max_size" value="20" />
            <property name="hibernate.c3p0.timeout" value="300" />
            <property name="hibernate.c3p0.max_statements" value="50" />
            <property name="hibernate.c3p0.idle_test_period" value="3000" />
            <!--The dialect to use-->
            <property name="hibernate.dialect" value="org.hibernate.dialect.
                OracleDialect" />
        </properties>
    </persistence-unit>
</persistence>
```

Подключение к базе данных при помощи проверки подлинности NT

Существует возможность подключения к серверу MS SQL, который требует проверки подлинности NT. Для этого необходим драйвер анализа доменов (например, драйвер jTDS JDBC).

При проверке подлинности используется не учетная запись текущего процесса NT, а заданные параметры (домен, имя пользователя, пароль).

1. Укажите в файле **persistence.xml** следующие параметры:

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

2. Сохраните файл драйвера JDBC в папке: **<директория установки зонда>\lib**.
3. Перезапустите зонд.

Настройка в файле persistence.xml использования в интеграции SCCM аутентификации NTLM

Примечание: Этот раздел относится только к интеграции SCCM.

Для того, чтобы в интеграции SCCM использовалась аутентификация NTLM, настройте файл **persistence.xml** следующим образом:

1. Сохраните файл драйвера JDBC в папке: **<каталог установки зонда>\lib**.

   Например, можно поместить файл **jtds-1.3.1.jar** из http://sourceforge.net/projects/jtds/files/ в папку **DataFlowProbe\lib**.
2. Запустите сервер и зонд.
3. В UCMDB перейдите в **Управление потоком данных > Управление адаптерами > SCCMAdapter >** .
4. На панели "Ресурсы" выберите файл конфигурации адаптера SCCM в папке **Пакеты > SCCMAdapter > Файлы конфигурации**.
5. В файле **adapter.conf** задайте значение **dal.use.persistence.xml=true**.
6. В файле **persistence.xml** добавьте следующее:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
    <persistence-unit name="GenericDBAdapter">
        <properties>
            <!-- added to fix: org.hibernate.HibernateException:
'hibernate.dialect' must be set when no Connection avalable -->
            <property name="hibernate.dialect"
```

```
value="org.hibernate.dialect.HSQLDialect"/>
            <property name="hibernate.hbm2ddl.auto" value="create-drop"/>

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

Примечание: Замените выделенную часть URL-адресом для подключения.

7. Имя пользователя или пароль для файла **persistence.xml** не требуются.
8. Перейдите в **Управление потоком данных > Студия интеграции** и нажмите кнопку **Создать точку интеграции**  .
9. Введите значения в обязательных полях.

   Когда потребуется ввести учетный идентификатор, выполните следующие действия:

   a. В диалоговом окне "Выбрать учетные данные" выберите **Generid DB Protocol (SQL)** из панели "Протокол" слева.
   b. На панели "Учетные данные" справа нажмите кнопку **Добавить сведения о новом подключении для указанного типа протокола**  .
   c. В новом диалоговом окне выберите **MicrosoftSQLServerNTLM** в качестве типа базы данных.
   d. Введите номер порта.
   e. Укажите имя пользователя в следующем формате: **домен\имя пользователя**.
   f. Укажите пароль.

Встроенные конвертеры

Следующие конвертеры можно использовать для преобразования объединенных запросов и заданий репликации в данные БД и обратно.

Этот раздел охватывает следующие темы:

- "Встроенные конвертеры" выши
- "Конвертер SuffixTransformer" на странице 182
- "Конвертер PrefixTransformer" на странице 183
- "Конвертер BytesToStringTransformer" на странице 183
- "Конвертер StringDelimitedListTransformer" на странице 183
- "Настраиваемый конвертер" на странице 183

Конвертер enum-transformer

Этот конвертер использует XML-файл, указанный как входной параметр.

XML-файл сопоставляет значения CMDB с жестким кодированием и значения БД (enum). Если одно из значений не существует, вы можете выбрать возврат того же значения, возврат значения null или создание исключения.

Конвертер сравнивает две строки с учетом или без учета регистра. По умолчанию регистр учитывается. Чтобы включить режим без учета регистра, укажите: `case-sensitive="false"` в элементе enum-transformer.

Используйте файл сопоставления XML для каждого атрибута объекта.

Примечание. Этот конвертер можно использовать для полей `to_DB_class` и `from_DB_class` в файле **transformations.txt**.

Входной XSD-файл:

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

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

```
  </xs:element>
  <xs:element name="value">
    <xs:complexType>
        <xs:attribute name="cmdb-value" type="xs:string" use="required"/>
        <xs:attribute name="external-db-value" type="xs:string" use="required"/>
        <xs:attribute name="is-cmdb-value-null" type="xs:boolean"
use="optional"/>
        <xs:attribute name="is-db-value-null" type="xs:boolean" use="optional"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Пример преобразования 'sys' в значение 'System':

В этом примере значение `sys` в CMDB преобразуется в значение `System` в объединенной базе данных, а значение `System` в объединенной базе данных преобразуется в значение `sys` в CMDB.

Если значение не существует в XML-файле (например, строка `demo`), конвертер возвращает полученное входное значение.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-value-
action="return-original" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/generic-enum-transformer.xsd">
    <value CMDB-value="sys" external-DB-value="System" />
</enum-transformer>
```

Пример преобразования внешнего значения или значения CMDB в значение null:

В данном примере значение **NNN** в удаленной базе данных преобразуется в значение null в базе данных CMDB.

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

В данном примере значение **OOO** в базе данных в базе данных CMDB преобразуется в значение null в удаленной базе данных.

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

Конвертер SuffixTransformer

Этот конвертер используется для добавления и удаления суффиксов в значениях CMDB и источника объединенной базы данных.

Существует две реализации:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddSuffixTransformer**. Добавление суффикса (в качестве входного значения) при преобразовании значения в объединенной базе данных в значение CMDB и удаление суффикса при преобразовании значения CMDB в значение объединенной базы данных.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemoveSuffixTransformer**. Удаление суффикса (в качестве входного значения) при преобразовании значения в объединенной базе данных в значение CMDB и добавление суффикса при преобразовании значения CMDB в значение объединенной базы данных.

Конвертер PrefixTransformer

Этот конвертер используется для добавления и удаления префиксов в значениях CMDB и объединенной базы данных.

Существует две реализации:

- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb AddPrefixTransformer**. Добавление префикса (в качестве входного значения) при преобразовании значения в объединенной базе данных в значение CMDB и удаление префикса при преобразовании значения CMDB в значение объединенной базы данных.
- **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.AdapterToCmdb RemovePrefixTransformer**. Удаление префикса (в качестве входного значения) при преобразовании значения в объединенной базе данных в значение CMDB и добавление префикса при преобразовании значения CMDB в значение объединенной базы данных.

Конвертер BytesToStringTransformer

Этот конвертер используется для преобразования массивов байтов в CMDB в их представление строки в объединенной базе данных.

Конвертер:
com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl.CmdbToAdapterBytes ToStringTransformer.

Конвертер StringDelimitedListTransformer

Этот конвертер используется для преобразования списка строк в список целых чисел/строк в CMDB.

Конвертер: **com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.transform.impl. StringDelimitedListTransformer.**

Настраиваемый конвертер

Допускается создание нового пользовательского конвертера (преобразователя). Это позволяет создавать конвертеры с любыми необходимыми функциями.

Существует два метода создания пользовательских конвертеров:

1. Конвертер Java
   a. Создайте Java Project в интегрированной среде разработки Java (Eclipse, Intellij или Netbeans).
   b. Добавьте файлы federation_api.jar и db-interfaces.jar в classpath.
   c. Создайте класс Java, который реализует следующие интерфейсы (из **db-interfaces.jar**):
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   d. Скомпилируйте проект и создайте файл jar.
   e. Поместите файл jar в пакет адаптера (adapterCode\<ID адаптера>\)
   f. Разверните пакет.
   g. Добавьте имя класса конвертера в файл **transformations.txt**.
2. Конвертер Groovy

   Пример такого конвертера находится в исходном пакете GDBA (**GroovyExampleTransformer.groovy**).
   a. Поместите созданный файл Groovy в пакет адаптера (adapterCode\<ID адаптера>\) Это можно сделать напрямую через раздел меню "Управление адаптерами".
   b. Создайте класс Groovy, который реализует следующие интерфейсы (из **db-interfaces.jar**):
      - FcmdbDalTransformerFromExternalDB
      - FcmdbDalTransformerValuesToExternalDB
      - FcmdbDalTransformerInit
   c. Добавьте имя класса конвертера в файл **transformations.txt**.

Примечание. Groovy — это скриптовый язык на основе Java. Стандартный код Java также применим и в Groovy.

Подключаемые модули

Общий адаптер БД поддерживает следующие подключаемые модули:

- Дополнительный модуль для полной синхронизации топологии.
- Дополнительный модуль для синхронизации изменений топологии. Если подключаемый модуль для синхронизации изменений не реализован, можно выполнить разностную синхронизацию, но она фактически будет полной.
- Дополнительный модуль для синхронизации макета.
- Дополнительный модуль для получения поддерживаемых запросов для синхронизации. Если подключаемый модуль не указан, возвращаются все имена TQL-запросов.
- Внутренний дополнительный модуль для изменения определения и результата TQL-запроса.

- Внутренний дополнительный модуль для изменения запроса макета и ЭК в результате.
- Внутренний дополнительный модуль для изменения запроса макета и связей в результате.
- Внутренний дополнительный модуль для изменения действия принудительной обратной отправки идентификаторов.

Подробные сведения о реализации и развертывании подключаемых модулей см. в разделе "Реализация подключаемого модуля" на странице 142.

Примеры конфигурации

В этом разделе приводятся примеры конфигурации.

Этот раздел охватывает следующие темы:

- "Сценарий использования" ниже
- "Выверка одного узла" ниже
- "Выверка двух узлов" на странице 188
- "Использование первичного ключа, содержащего несколько столбцов" на странице 190
- "Использование преобразований" на странице 192

Сценарий использования

TQL-запрос:

node > (composition) > card

где:

- **node** — это объект CMDB
- **card** — это объект источника объединенной базы данных
- **composition** — это связь между ними

Этот пример запроса выполняется для базы данных ED. Узлы ED хранятся в таблице Device, а элемент card находится в таблице hwCards. В следующих примерах элемент card всегда сопоставляется одинаковым способом.

Выверка одного узла

В этом примере выверка выполняется для свойства name.

Упрощенное определение

Выверка выполняется по элементу node и выделяется специальным тегом **CMDB-class**.

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

```
        <reconciliation-by-single-node>
            <или>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"
/>
            </or>
        </reconciliation-by-single-node>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMDB-
class-name="node" link-class-name="composition">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"
/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

Расширенное определение

Файл orm.xml

Обратите внимание на добавление сопоставления связей. См. подраздел определений в документе "Файл orm.xml" на странице 158.

Пример файла orm.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/
persistence/orm http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
version="1.0">
        <description>Generic DB adapter orm</description>
        <package>generic_db_adapter</package>
        <entity class="generic_db_adapter.node" >
                <table name="Device"/>
                <attributes>
                        <id name="id1">
                                <column name="Device_ID"
                                   insertable="false"
                                   updatable="false"/>
                                <generated-value strategy="TABLE"/>
                        </id>
                        <basic name="name">
                                <column name="Device_Name"/>
                        </basic>
```

```
                </attributes>
        </entity>
        <entity class="generic_db_adapter.card" >
                <table name="hwCards"/>
                <attributes>
                        <id name="id1">
                                <column name="hwCards_Seq" insertable="false"
                                    updatable="false"/>
                                <generated-value strategy="TABLE"/>
                        </id>
                        <basic name="card_class">
                                <column name="hwCardClass" insertable="false"
                                    updatable="false"/>
                        </basic>
                        <basic name="card_vendor">
                                <column name="hwCardVendor" insertable="false"
                                   updatable="false"/>
                        </basic>
                        <basic name="card_name">
                                <column name="hwCardName" insertable="false"
                                   updatable="false"/>
                        </basic>
                </attributes>
        </entity>
        <entity class="generic_db_adapter.node_composition_card" >
                <table name="hwCards"/>
                <attributes>
                        <id name="id1">
                                <column name="hwCards_Seq" insertable="false"
                                   updatable="false"/>
                                <generated-value strategy="TABLE"/>
                        </id>
                        <many-to-one name="end1" target-entity="node">
                                <join-column name="Device_ID" insertable="false"
                                   updatable="false"/>
                        </many-to-one>
                        <one-to-one name="end2" target-entity="card"
>                               <join-column name="hwCards_Seq"
                                referenced-column-name="hwCards_Seq" insertable=
                                "false" updatable="false"/>
                        </one-to-one>
                </attributes>
        </entity>
</entity-mappings>
```

Файл reconciliation_types.txt

Дополнительные сведения см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 172.

```
multinode[node] expression[node.name]
```

Файл transformations.txt

Этот файл остается пустым, поскольку в данном примере преобразование значений не требуется.

Выверка двух узлов

В этом примере выверка рассчитывается в соответствии со свойством `name` элемента `node` и элемента `ip_address` с различными вариациями.

TQL-запрос сверки: **node > (containment) > ip_address**.

Упрощенное определение

Выверка выполняется по элементу `name` элемента `node` ИЛИ элемента `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment">
            <или>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"
/>
                <connected-node-attribute CMDB-attribute-name="name" column-
name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMDB-
class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"
/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

Выверка выполняется по элементу `name` элемента `node` И элемента `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
```

```
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment">
            <and>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"
/>
                <connected-node-attribute CMDB-attribute-name="name" column-
name="Device_PreferredIPAddress" />
            </and>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMDB-
class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"
/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

Выверка выполняется по элементу `name` элемента `ip_address`:

```
<?xml version="1.0" encoding="UTF-8"?>
<generic-DB-adapter-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/simplifiedConfiguration.xsd">
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment">
            <или>
                <connected-node-attribute CMDB-attribute-name="name" column-
name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMDB-
class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"
/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName" />
```

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

Расширенное определение

Файл orm.xml

Поскольку выражение выверки на задано в этом файле, эта версия будет использоваться для всех выражений выверки.

Файл reconciliation_types.txt

Дополнительные сведения см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 172.

```
multinode[node] expression[ip_address.name OR node.name] end1_type[node] end2_
type[ip_address] link_type[containment]

multinode[node] expression[ip_address.name AND node.name] end1_type[node] end2_
type[ip_address] link_type[containment]

multinode[node] expression[ip_address.name] end1_type[node] end2_type[ip_
address] link_type[containment]
```

Файл transformations.txt

Этот файл остается пустым, поскольку в данном примере преобразование значений не требуется.

Использование первичного ключа, содержащего несколько столбцов

Если первичный ключ состоит из нескольких столбцов, следующий код добавляется в определения XML:

Упрощенное определение

Существует несколько тегов одного ключа, тег указан для каждого столбца.

```
    <class CMDB-class-name="card" default-table-name="hwCards" connected-CMDB-
class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-primary-key-
column="Device_ID" />
        <primary-key column-name="Device_ID"/>
        <primary-key column-name="hwBusesSupported_Seq" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-name="hwCardVendor"
/>
        <attribute CMDB-attribute-name="card_name" column-name="hwCardName" />
    </class>
```

Расширенное определение

Файл orm.xml

Добавляется новый объект `id`, связывающий столбцы первичного ключа. К объектам, использующим объект `id`, необходимо добавить специальный тег.

При использовании внешнего ключа (тег join-column) для такого первичного ключа необходимо сопоставить каждый столбец внешнего ключа со столбцом первичного ключа.

Дополнительные сведения см. в разделе "Файл orm.xml" на странице 158.

Пример файла orm.xml:

```
<entity class="generic_db_adapter.card" >
        <table name="hwCards"/>
        <attributes>
            <id name="id1">
                <column name="Device_ID" insertable="false" updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false"
updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column name="hwCards_Seq" insertable="false"
updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>

<entity class="generic_db_adapter.node_containment_card" >
        <table name="hwCards"/>
        <attributes>
            <id name="id1">
                <column name="Device_ID" insertable="false" updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id2">
                <column name="hwBusesSupported_Seq" insertable="false"
updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <id name="id3">
                <column name="hwCards_Seq" insertable="false"
updatable="false"/>
                <generated-value strategy="TABLE"/>
            </id>
            <many-to-one name="end1" target-entity="node">
                <join-column name="Device_ID" insertable="false"
```

```
updatable="false"/>
            </many-to-one>
            <one-to-one name="end2" target-entity="card">
                <join-column name="hwCards_Seq" referenced-column-name="hwCards_
Seq" insertable="false" updatable="false"/>
                <join-column name="hwBusesSupported_Seq" referenced-column-
name="hwBusesSupported_Seq" insertable="false" updatable="false"/>
                <join-column name="hwCards_Seq" referenced-column-name="hwCards_
Seq" insertable="false" updatable="false"/>
            </one-to-one>
        </attributes>
    </entity>
</entity-mappings>
```

Использование преобразований

В следующем примере общий конвертер **enum** преобразован из значений 1, 2, 3 в значения a, b, c соответственно в столбце name.

Файл сопоставления: generic-enum-transformer-example.xml.

```
<enum-transformer CMDB-type="string" DB-type="string" non-existing-value-
action="return-original" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../META-CONF/generic-enum-transformer.xsd">
    <value CMDB-value="1" external-DB-value="a" />
    <value CMDB-value="2" external-DB-value="b" />
    <value CMDB-value="3" external-DB-value="c" />
</enum-transformer>
```

Упрощенное определение

```
    <CMDB-class CMDB-class-name="node" default-table-name="Device">
        <primary-key column-name="Device_ID"/>
        <reconciliation-by-two-nodes connected-node-CMDB-class-name="ip_address"
CMDB-link-type="containment">
            <or>
                <attribute CMDB-attribute-name="name" column-name="Device_Name"
from-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.
xml)" to-CMDB-converter="com.mercury.topaz.fcmdb.adapters.dbAdapter.dal.
transform.impl.GenericEnumTransformer(generic-enum-transformer-example.
xml)" />
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
```

Расширенное определение

Изменяется только файл **transformation.txt**.

Файл transformations.txt

Убедитесь, что имена атрибутов и объектов соответствуют файлу `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)]
```

Файлы журнала адаптера

Чтобы понять потоки вычисления, жизненные циклы адаптера и просмотреть сведения об отладке, ознакомьтесь с файлами журнала.

Этот раздел охватывает следующие темы:

- "Уровни журнала" ниже
- "Расположение журналов" ниже

Уровни журнала

Уровень журнала можно настроить для каждого из журналов в отдельности.

В текстовом редакторе откройте файл **C:\hp\UCMDB\UCMDBServer\conf\log\fcmdb.gdba.properties**
.

Уровень журнала по умолчанию: **ERROR**:

```
#loglevel can be any of DEBUG INFO WARN ERROR FATAL
loglevel=ERROR
```

- Чтобы повысить уровень всех файлов журналов, измените значение **loglevel=ERROR** на **loglevel=DEBUG** или **loglevel=INFO**.
- Чтобы изменить уровень журнала для определенного файла, измените строку категории **log4j** соответствующим образом. Например, чтобы изменить уровень журнала для файла fcmdb.gdba.dal.sql.log на **INFO**, измените

  ```
  log4j.category.fcmdb.gdba.dal.SQL=${loglevel},fcmdb.gdba.dal.SQL.appender
  ```

  на:

  ```
  log4j.category.fcmdb.gdba.dal.SQL=INFO,fcmdb.gdba.dal.SQL.appender
  ```

Расположение журналов

Файлы журналов находятся в каталоге **C:\hp\UCMDB\UCMDBServer\runtime\log**

- **Fcmdb.gdba.log**

  Журнал жизненного цикла адаптера. Предоставляет сведения о том, когда адаптер был запущен и остановлен, и какие типы ЭК поддерживаются адаптером.

  Здесь можно просмотреть ошибки запуска (загрузки и выгрузки адаптеров).

- **fcmdb.log**

  Здесь можно просмотреть исключения.

- **cmdb.log**

  Здесь можно просмотреть исключения.

- **Fcmdb.gdba.mapping.engine.log**

  Журнал системы сопоставления. Предоставляет сведения о TQL-запросе выверки, который используется системой сопоставления, и топологиях выверки, которые сравниваются на этапе подключения

  С этим журналом следует ознакомиться, если TQL-запрос не возвращает результаты, несмотря на то, что соответствующие ЭК присутствуют в базе данных, или возвращает непредвиденные результаты (проверьте выверку).

- **Fcmdb.gdba.TQL.log**

  Журнал TQL. Содержит сведения о TQL-запросах и их результатах.

  Ознакомьтесь с этим журналом, если TQL-запрос не возвращает результаты и журнал системы сопоставления показывает отсутствие результатов в объединенном источнике данных.

- **Fcmdb.gdba.dal.log**

  Журнал жизненного цикла DAL. Содержит сведения о создании типов ЭК и подключении к базе данных.

  Ознакомьтесь с этим журналом, если вам не удается подключиться к базе данных или если типы ЭК или атрибуты не поддерживаются запросом.

- **Fcmdb.gdba.dal.command.log**

  Журнал операций DAL. Содержит сведения о вызванных внутренних операциях DAL. (Этот журнал аналогичен `cmdb.dal.command.log`).

- **Fcmdb.gdba.dal.SQL.log**

  Журнал SQL-запросов DAL. Содержит сведения о вызванных JPAQL (объектно-ориентированных SQL-запросов) и их результатах.

  Ознакомьтесь с этим журналом, если вам не удается подключиться к базе данных или если типы ЭК или атрибуты не поддерживаются запросом.

- **Fcmdb.gdba.hibrnate.log**

  Журнал Hibernate. Содержит сведения о выполненных SQL-запросах, обработке каждого JPAQL-запроса в SQL-запрос, результаты запросов, данные о кэшировании Hibernate и др. См. дополнительные сведения о Hibernate в разделе "Hibernate как поставщик JPA" на странице 124.

Внешние ссылки

Подробнее о спецификации JavaBeans 3.0 см. по адресу http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

Устранение неполадок и ограничения — Разработка общих адаптеров БД

В данном разделе описываются процедуры поиска и устранения неполадок, а также ограничения общего адаптера базы данных.

Общие ограничения

При обновлении пакета адаптера используйте Notepad++, UltraEdit или другой сторонний текстовый редактор, а не блокнот (любой версии) от корпорации Microsoft для редактирования файлов шаблонов. Это позволит предотвратить применение специальных символов, которые приведут к сбою развертывания подготовленного пакета.

Ограничения JPA

- Все таблицы должны включать столбец первичного ключа.
- Имена атрибутов классов CMDB должны соответствовать правилам именования JavaBeans (например, имена должны начинаться со строчных букв).
- Два ЭК, соединенные одной связью в модели классов, должны иметь прямую связь в базе данных (например, если элемент `node` соединен с элементом `ticket`, должна существовать таблица внешних ключей или связей, которая соединяет их).
- Несколько таблиц, сопоставленных с одним типом ЭК, должны использовать одну таблицу первичного ключа.

Функциональные ограничения

- Создание связей между CMDB и объединенными типами ЭК вручную не поддерживается. Для настройки виртуальных связей необходимо задать специальную логику связей (она может основываться на свойствах объединенного класса).
- Объединенные типы ЭК не могут вызывать типы ЭК в правиле влияния, но могут входить в TQL-запрос анализа влияния.
- Объединенный тип ЭК может быть частью TQL-запроса, но не может использоваться как узел, для которого выполняется расширение (нельзя добавлять, обновлять и удалять объединенный тип ЭК).
- Использование квалификатора класса в условии не поддерживается.
- Подграфы не поддерживаются.
- Составные связи не поддерживаются.

- Внешний CMDB`id` ЭК включает первичный ключ, но не включает ключевые атрибуты.
- Столбец `bytes` не может использоваться как столбец первичного ключа в Microsoft SQL Server.
- Вычисление TQL-запроса закончится неудачей, если имена условий атрибутов, указанные в объединенном узле, не сопоставлены в файле **orm.xml**.

Глава 6: Разработка адаптеров Java

Данная глава включает:

- Обзор Federation Framework 197
- Взаимодействие адаптера и сопоставления в Federation Framework 202
- Поток Federation Framework для объединенных TQL-запросов 203
- Взаимодействие между Federation Framework, сервером, адаптером и системой сопоставления 204
- Поток Federation Framework для заполнения 213
- Интерфейсы адаптера 214
- Устранение неполадок в ресурсах адаптеров 215
- Добавление адаптера для нового внешнего источника данных 216
- Создание примера адаптера 223
- Теги конфигурации и свойства XML 224
- Интерфейс DataAdapterEnvironment 226

Обзор Federation Framework

Примечание.

- Термин **relationship** является эквивалентом термина **link** (связь).
- Термин **ЭК** является эквивалентом термина **объект**.
- Граф является набором узлов и связей.

Функция Federation Framework использует API-интерфейс для получения данных из объединенных источников. Federation Framework предоставляет три основные возможности:

- **Объединение** в оперативном режиме. Все запросы выполняются для исходных репозиториев данных, а результаты формируются в CMDB в оперативном режиме.
- **Заполнение.** Заполнение данных (топологических данных и свойств ЭК) в CMDB из внешнего источника данных.
- **Принудительная отправка данных.** Отправка данных (топологических данных и свойств ЭК) в CMDB в удаленный источник данных.

Все типы действий требуют адаптера для каждого репозитория данных, который предоставляет специальные возможности репозитория, и извлекает и обновляет необходимые данные. Каждый запрос для репозитория данных проходит через адаптер.

Данный раздел также включает следующие подразделы.

- "Объединение в оперативном режиме" ниже
- "Принудительная отправка данных" на следующей странице
- "Заполнение" на странице 200

Объединение в оперативном режиме

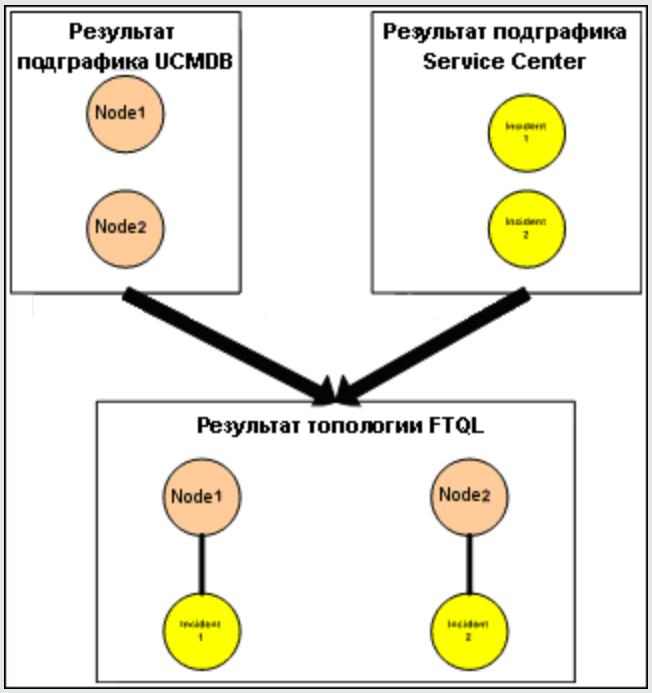
Объединенные TQL-запросы обеспечивают извлечение данных из любого внешнего репозитория без извлечения его данных.

Объединенный TQL-запрос использует адаптеры, представляющие внешние репозитории данных, для создания соответствующих внешних связей между ЭК из других внешних репозиториев и ЭК UCMDB.

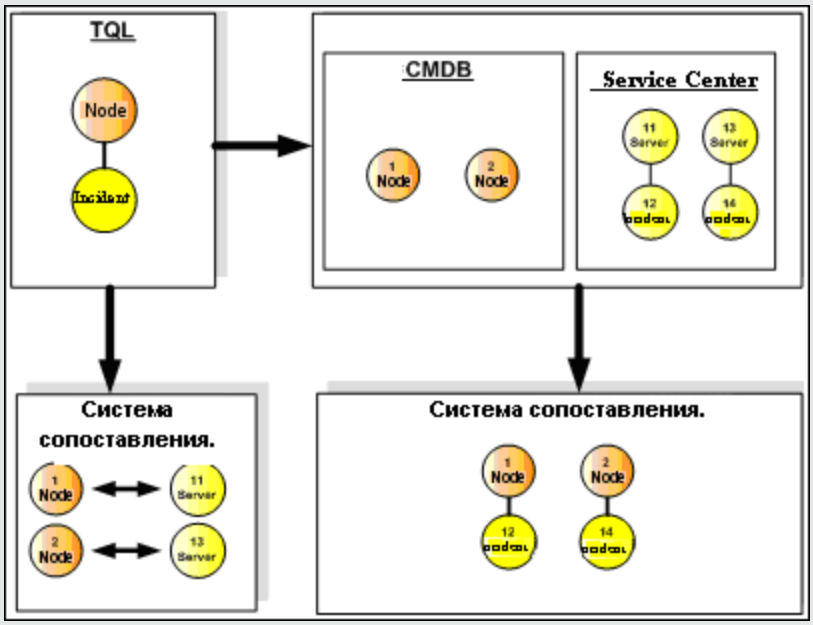
Пример оперативного объединения:

1. Federation Framework разделяет объединенный TQL-запрос на несколько подграфов, причем все узлы в подграфе относятся к одному репозиторию данных. Каждый подграф соединен с другими подграфами посредством виртуальной связи (но сам по себе не содержит виртуальных связей).

2. После разделения объединенного TQL-запроса на подграфы Federation Framework рассчитывает топологию каждого подграфа и соединяет два соответствующих подграфа путем создания виртуальной связи между соответствующими узлами.



3. После вычисления объединенной топологии TQL-запроса Federation Framework получает макет результата топологии.



Принудительная отправка данных

Принудительная отправка данных используется для синхронизации данных между текущей локальной базой CMDB и удаленной службой или целевым репозиторием данных.

При отправке данных репозитории разделяются на две категории: исходный (локальная база CMDB) и целевой. Данные извлекаются из исходного репозитория данных и обновляются в целевом репозитории. Процесс принудительной отправки данных основывается на именах запросов. Это значит, что данные синхронизируются между исходным (локальная база CMDB) и целевым репозиториями данных и возвращаются по имени TQL-запроса из локальной базы CMDB.

Процесс принудительной отправки данных включает следующие шаги:

1. Получение результата топологии с сигнатурами из исходного репозитория данных.
2. Сравнение новых результатов с предыдущими результатами.
3. Получение полного макета ЭК (т.е. всех свойств ЭК) и связей только для измененных результатов.
4. Обновление целевого репозитория данных с использованием полученного полного макета ЭК и связей. Если любые из ЭК и связей удалены в исходном репозитории данных и запрос является эксклюзивным, процесс репликации также удаляет ЭК или связи из целевого репозитория данных.

База CMDB включает 2 скрытых источника данных: (**hiddenRMIDataSource** и **hiddenChangesDataSource**), которые всегда являются первичным источником данных в потоках принудительной отправки. Чтобы реализовать новый адаптер для потоков принудительной отправки данных, необходимо реализовать целевой адаптер.

Заполнение

Поток наполнения используется для наполнения базы CMDB данными из внешних источников.

Поток всегда использует один исходный источник для получения данных и передает эти данных зонду, используя тот же процесс, что при обнаружении.

Чтобы реализовать новый адаптер для потоков наполнения, необходимо всего лишь реализовать исходный адаптер, поскольку зонд потока данных выступает в качестве целевого объекта.

Адаптер в потоке наполнения выполняется в зонде. Отладка и ведение журналов выполняется в зонде, а не в CMDB.

Поток наполнения основывается на именах запросов. Это значит, что данные синхронизируются между исходным репозиторием данных зондом потока данных и возвращаются по имени запроса в исходном репозитории. Например, в UCMDB имя запроса — это имя TQL-запроса. Однако в другом репозитории имя запроса может быть кодовым именем, которое возвращает данные. Адаптер разработан для правильной обработки по имени запроса.

Каждое задание может быть определено как эксклюзивное. Это значит, что ЭК и связи в результатах задания уникальны в локальной базе CMDB, и другие запросы не могут перенести их в целевой объект. Адаптер исходного репозитория данных поддерживает определенные запросы и может получать данные из этого репозитория. Адаптер целевого репозитория данных обеспечивает обновление полученных данных в этом репозитории.

Поток SourceDataAdapter

- Получение результата топологии с сигнатурами из исходного репозитория данных.
- Сравнение новых результатов с предыдущими результатами.
- Получение полного макета ЭК (т.е. всех свойств ЭК) и связей только для измененных результатов.
- Обновление целевого репозитория данных с использованием полученного полного макета ЭК и связей. Если любые из ЭК и связей удалены в исходном репозитории данных и запрос является эксклюзивным, процесс репликации также удаляет ЭК или связи из целевого репозитория данных.

Поток SourceChangesDataAdapter

- Получение результата топологии, возвращенного после последней даты.
- Получение полного макета ЭК (т.е. всех свойств ЭК) и связей только для измененных результатов.
- Обновление целевого репозитория данных с использованием полученного полного макета ЭК и связей. Если любые из ЭК и связей удалены в исходном репозитории данных и запрос является эксклюзивным, процесс репликации также удаляет ЭК или связи из целевого репозитория данных.

Поток PopulateDataAdapter

- Получение полной топологии с запрошенным результатом макета.
- Использование механизма разделения топологии для получения данных в виде блоков.
- Фильтр зонда исключает все данные, которые уже возвращались в предыдущих выполнениях.
- Обновление целевого репозитория данных с использованием полученного макета ЭК и связей. Если любые из ЭК и связей удалены в исходном репозитории данных и запрос является эксклюзивным, процесс репликации также удаляет ЭК или связи из целевого репозитория данных.

PopulateChangesDataAdapter

- Получение топологии с запрошенным результатом макета, измененным после последнего выполнения.
- Использование механизма разделения топологии для получения данных в виде блоков.
- Фильтр зонда исключает все данные, которые уже возвращались в предыдущих выполнениях (включая этот поток).
- Обновление целевого репозитория данных с использованием полученного макета ЭК и связей. Если любые из ЭК и связей удалены в исходном репозитории данных и запрос является эксклюзивным, процесс репликации также удаляет ЭК или связи из целевого репозитория данных.

Поток заполнения на основе экземпляров

Если адаптер поддерживает потоки заполнения на основе экземпляров (тег **<instance-based-data>**, см. раздел "Теги конфигурации и свойства XML" на странице 224), механизм заполнения осуществляет поиск ЭК внутри экземпляра и удаляет их из UCMDB (при условии, что для задания заполнения разрешено удаление). В каждом экземпляре содержится корневой ЭК, указанный в определении TQ-запроса под именем **Root**. При

каждой передаче корневого ЭК сведения об экземпляре (обо всех ЭК, подключенных к нему) сопоставляются с данными, полученными при предыдущей отправке в UCMDB, при этом из UCMDB удаляются все ЭК, которые больше не связаны с корневым ЭК. Для корректной поддержки потока заполнения на основе экземпляров необходимо, чтобы любые изменения ЭК или его атрибута инициировали повторную отправку всего экземпляра в UCMDB.

Взаимодействие адаптера и сопоставления в Federation Framework

Адаптер — это объект в UCMDB, который представляет внешние данные (данные, не сохраненные в UCMDB). В объединенных потоках все взаимодействие с внешними источниками данных производится через адаптеры. Поток взаимодействия Federation Framework и интерфейсы адаптеров отличаются для репликации и объединенных TQL-запросов.

Данный раздел также включает следующие подразделы.

- "Жизненный цикл адаптера" ниже
- "Методы assist адаптера" ниже

Жизненный цикл адаптера

Экземпляр адаптера создается для каждого внешнего источника данных. Адаптер начинает свой жизненный цикл с первого действия, которое к нему применено (например, `calculate TQL` или `retrieve/update data`). При вызове метода **start** адаптер получает сведения об окружении, такие как конфигурация репозитория, средство ведения журнала и др. Жизненный цикл адаптера завершается удалением репозитория из конфигурации и вызовом метода **shutdown**. Это значит, что адаптер учитывает состояние и может содержать соединение с внешним источником данных (при необходимости).

Методы assist адаптера

Адаптер включает несколько методов `assist`, которые могут добавлять конфигурации внешних репозиториев. Эти методы не входят в жизненный цикл адаптера и создают новый адаптер при каждом вызове.

- Первый метод проверяет подключение к внешнему репозиторию данных. testConnection можно выполнить на сервере UCMDB или зонде потока данных — в зависимости от типа адаптера.
- Второй метод действителен только для источника данных и возвращает запросы, репликация которых поддерживается. (Этот метод выполняется только на зонде.)
- Третий метод действует только для потоков объединения и наполнения и возвращает поддерживаемые внешние классы по внешнему источнику данных. (Этот метод выполняется на сервере UCMDB.)

Все эти методы используются при создании и просмотре конфигураций интеграции.

Поток Federation Framework для объединенных TQL-запросов

Этот раздел охватывает следующие темы:

- "Определения и термины" ниже
- "Система сопоставления." ниже
- "Объединенный адаптер" на следующей странице

В разделе "Взаимодействие между Federation Framework, сервером, адаптером и системой сопоставления" на следующей странице представлены диаграммы, демонстрирующие взаимодействие между Federation Framework, сервером UCMDB, адаптером и системой сопоставления.

Определения и термины

Данные выверки. Правило сопоставления ЭК указанного типа, полученных из CMDB и внешнего репозитория данных. Существует три типа правил выверки:

- **Выверка идентификатора.** Может использоваться, только если внешний репозиторий данных содержит идентификатор CMDB объектов выверки.
- **Выверка свойств.** Используется, если сопоставление может выполняться только по свойствам типа ЭК выверки.
- **Выверка топологии.** Используется, если для отбора ЭК выверки необходимы свойства дополнительных типов ЭК (не только ЭК выверки). Например, можно выполнить выверку узла по свойству `name`, которое принадлежит типу ЭК `ip_address`.

Объект выверки. Объект создается адаптером в соответствии с полученными данными выверки. Этот объект должен ссылаться на внешний ЭК и использоваться системой сопоставления для соединения внешних ЭК и ЭК в CMDB.

Тип ЭК выверки. Тип ЭК, представляющий объекты выверки. Эти ЭК должны храниться в CMDB и внешних репозиториях данных.

Система сопоставления. Компонент, который идентифицирует связи между ЭК из различных репозиториев, между которыми установлены виртуальные связи. Идентификация выполняется путем выверки объектов CMDB и внешних объектов выверки ЭК.

Система сопоставления.

Federation Framework использует систему сопоставления для вычисления объединенного TQL-запроса. Система сопоставления связывает ЭК, полученные из различных репозиториев и соединенные через виртуальные связи. Кроме того, система сопоставления предоставляет данные выверки для виртуальной связи. Одна сторона виртуальной связи должна ссылаться на CMDB. Эта сторона имеет тип `reconciliation`. Для вычисления двух подграфов виртуальная связь может начать с любого конечного узла.

Объединенный адаптер

Объединенный адаптер вызывает данные двух типов из внешних репозиториев: данные внешних ЭК и объекты выверки, принадлежащие внешним ЭК.

- **Данные внешних ЭК.** Внешние данные, отсутствующие в CMDB. Это целевые данные внешнего репозитория.
- **Данные объекта выверки.** Вспомогательные данные, используемые Federation Framework для соединения ЭК CMDB и внешних данных. Каждый объект выверки должен ссылаться на внешний ЭК. Тип объекта выверки — это тип (или подтип) одной из сторон виртуальной связи, из которых получаются данные. Объекты выверки должны сопоставить полученный адаптер с данными выверки. Существует три типа объектов выверки: `IdReconciliationObject`, `PropertyReconciliationObject` и `TopologyReconciliationObject`.

В интерфейсах на основе DataAdapter (DataAdapter, PopulateDataAdapter и PopulateChangesDataAdapter) выверка запрашивается как часть определения запроса.

Взаимодействие между Federation Framework, сервером, адаптером и системой сопоставления

В следующих диаграммах демонстрируется взаимодействие между Federation Framework, сервером UCMDB, адаптером и системой сопоставления. Объединенный TQL-запрос на диаграммах включает только одну виртуальную связь, т.е. только одна система UCMDB и один внешний репозиторий участвуют в объединенном TQL-запросе.

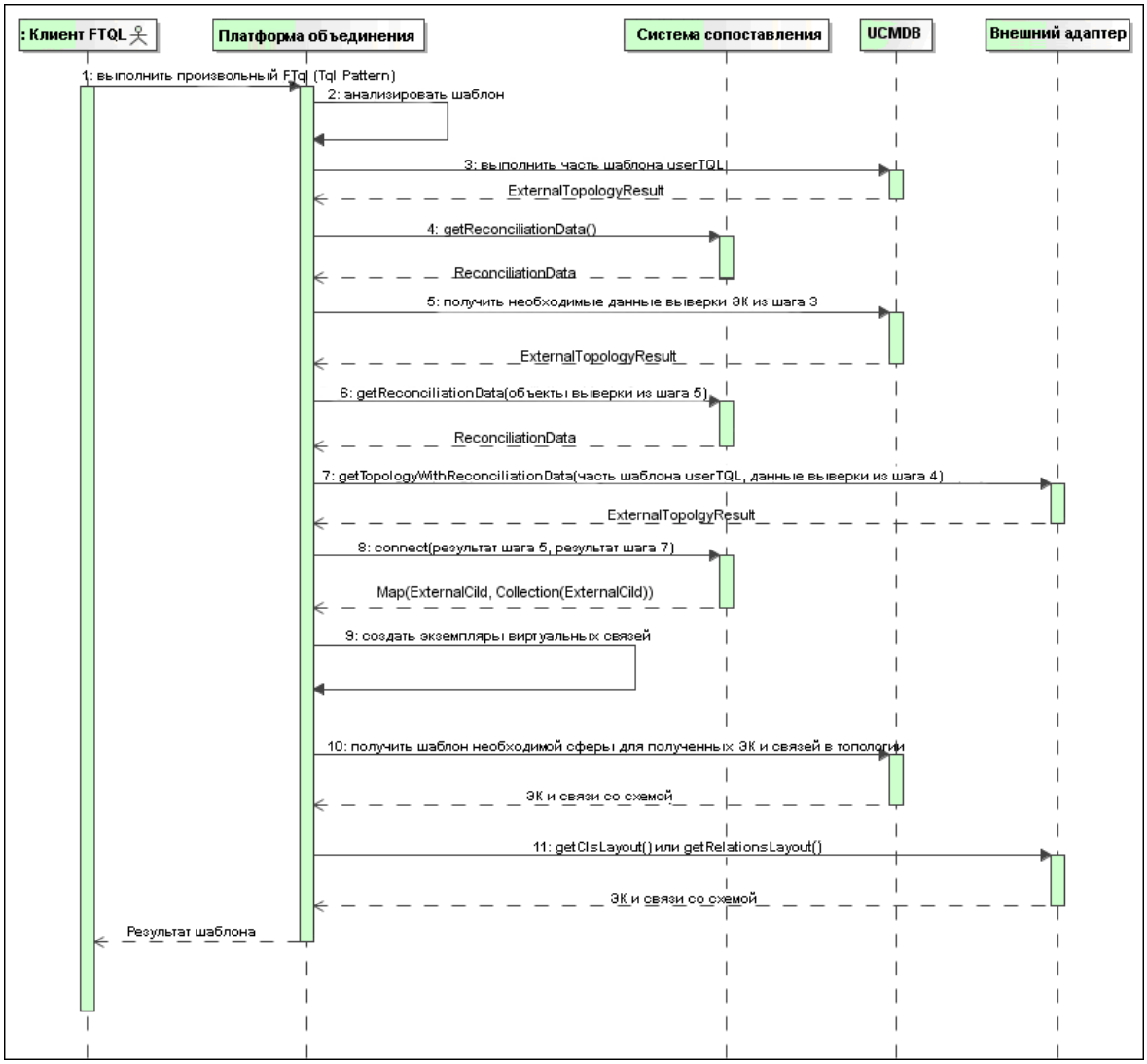
Этот раздел охватывает следующие темы:

- " Вычисление начинается на стороне сервера" ниже
- "Вычисление начинается на стороне внешнего адаптера" на странице 207
- "Пример потока Federation Framework для объединенных TQL-запросов" на странице 209

В первой диаграмме расчет начинается в UCMDB, а во второй диаграмме — во внешнем адаптере. Каждый этап диаграммы включает ссылки на соответствующий вызов метода адаптера или интерфейс системы сопоставления.

Вычисление начинается на стороне сервера

На следующей диаграмме демонстрируется взаимодействие между Federation Framework, UCMDB, адаптером и системой сопоставления. Объединенный TQL-запрос на диаграмме включает только одну виртуальную связь, т.е. только одна система UCMDB и один внешний репозиторий участвуют в объединенном TQL-запросе.

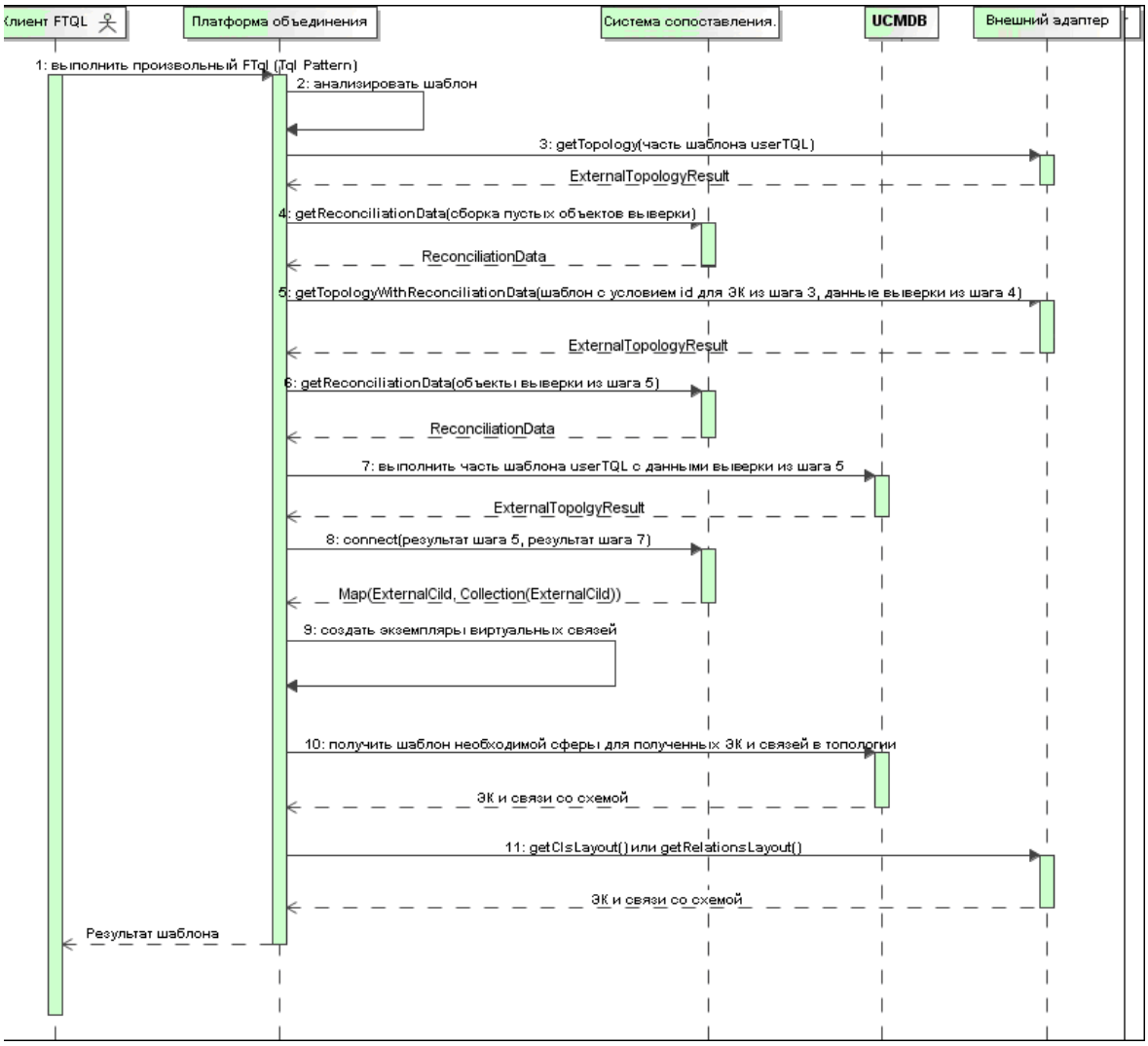


Числа на изображении объясняются ниже:

Число	Пояснение
1	Federation Framework получает вызов от объединенного вычисления TQL.
2	Federation Framework анализирует адаптер, находит виртуальное отношение и разделяет исходный TQL-запрос на два подадаптера — один для UCMDB и второй для внешнего репозитория данных.
3	Federation Framework запрашивает топологию подзапроса TQL у UCMDB.
4	После получения результатов топологии Federation Framework вызывает соответствующую систему сопоставления для текущего виртуального отношения и запрашивает данные выверки. Параметр `reconciliationObject` является пустым на этом этапе, то есть условия не добавлены к данным выверки в рамках этого вызова. Возвращенные данные выверки определяют,

Число	Пояснение
	какие данные необходимы для сопоставления ЭК выверки в UCMDB с внешним репозиторием. Существует три типа данных выверки: • **IdReconciliationData.** ЭК выверяются по идентификатору. • **PropertyReconciliationData.** ЭК выверяются по свойствам одного из ЭК. • **TopologyReconciliationData.** ЭК выверяются по топологии (например, для выверки ЭК узлов также необходим IP-адрес **IP**).
5	Federation Framework запрашивает данные выверки для ЭК сторон виртуального отношения, полученные во время шага "3" на предыдущей странице от UCMDB.
6	Federation Framework вызывает систему сопоставления для получения данных выверки. В этом состоянии (в отличие от шага "3" на предыдущей странице) система получает объекты выверки из шага "5" выши в качестве параметров. Система сопоставления преобразует полученный объект выверки в условие для данных выверки.
7	Federation Framework запрашивает топологию подзапроса TQL у внешнего репозитория. Внешний адаптер получает данные выверки из шага "6" выши в качестве параметра.
8	Federation Framework вызывает систему сопоставления для соединения полученных результатов. Параметр `firstResult` — это результат внешней топологии, полученный от UCMDB во время шага "5" выши, параметр `secondResult` — это внешний результат топологии, полученный от внешнего адаптера во время шага "7" выши. Система сопоставления возвращает сопоставление, в котором внешний идентификатор ЭК из первого репозитория (в нашем случае UCMDB) сопоставляется с внешними идентификаторами ЭК из второго (внешнего) репозитория.
9	Для каждого сопоставления Federation Framework создает виртуальное отношение.
10	После вычисления результата объединенного TQL-запроса (только на этапе топологии) Federation Framework получает исходный макет TQL для конечных ЭК и связей из соответствующих репозиториев.

Вычисление начинается на стороне внешнего адаптера



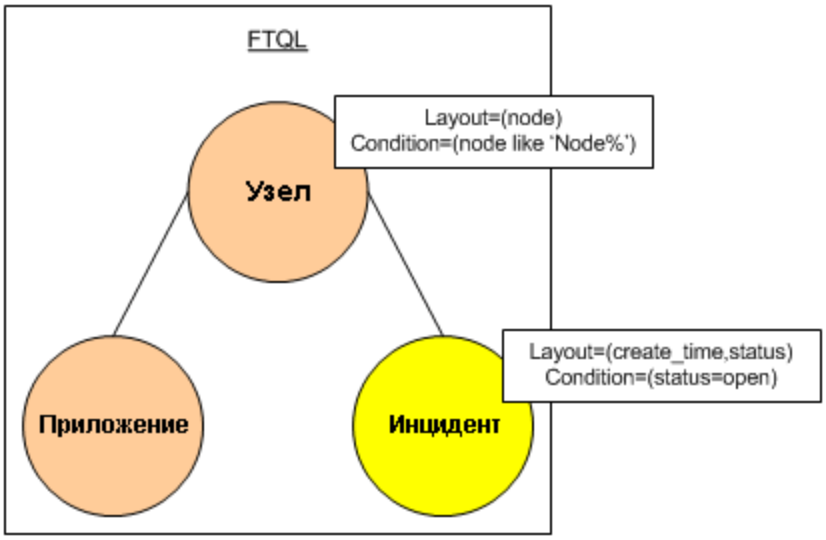
Числа на изображении объясняются ниже:

Число	Пояснение
1	Federation Framework получает вызов от объединенного вычисления TQL.
2	Federation Framework анализирует адаптер, находит виртуальное отношение и разделяет исходный TQL-запрос на два подадаптера — один для UCMD и второй для внешнего репозитория данных.
3	Federation Framework запрашивает топологию подзапроса TQL у внешнего адаптера. Возвращенный элемент `ExternalTopologyResult` не должен содержать объекты выверки, поскольку данные выверки не являются частью запроса.

Число	Пояснение
4	После получения результатов топологии Federation Framework вызывает соответствующую систему сопоставления для текущего виртуального отношения и запрашивает данные выверки. Параметр `reconciliationObjects` является пустым в этом состоянии, то есть условия не добавлены к данным выверки в рамках этого вызова. Возвращенные данные выверки определяют, какие данные необходимы для сопоставления ЭК выверки в UCMDB с внешним репозиторием. Существует три типа данных выверки: • **IdReconciliationData.** ЭК выверяются по идентификатору. • **PropertyReconciliationData.** ЭК выверяются по свойствам одного из ЭК. • **TopologyReconciliationData.** ЭК выверяются по топологии (например, для выверки ЭК узлов также необходим IP-адрес **IP**).
5	Federation Framework запрашивает данные выверки для ЭК сторон виртуального отношения, полученные во время шага 3 от внешнего репозитория. Federation Framework вызывает метод **getTopologyWithReconciliationData()** во внешнем адаптере, в котором запрошенная топология является одноузловой топологией с ЭК, полученными во время шага 3 в качестве условия идентификатора, и данными выверки из шага 4.
6	Federation Framework вызывает систему сопоставления для получения данных выверки. В этом состоянии (в отличие от шага 3) система получает объекты выверки из шага 5 в качестве параметров. Система сопоставления преобразует полученный объект выверки в условие для данных выверки.
7	Federation Framework запрашивает топологию подзапроса TQL с данными выверки, полученными во время шага 6 у UCMDB.
8	Federation Framework вызывает систему сопоставления для соединения полученных результатов. Параметр `firstResult` — это результат внешней топологии, полученный от внешнего адаптера во время шага 5, параметр `secondResult` — это внешний результат топологии, полученный от UCMDB во время шага 7. Система сопоставления возвращает сопоставление, в котором внешний идентификатор ЭК из первого репозитория (в нашем случае внешний репозиторий) сопоставляется с внешними идентификаторами ЭК из второго репозитория (UCMDB).
9	Для каждого сопоставления Federation Framework создает виртуальное отношение.
10	После вычисления результата объединенного TQL-запроса (только на этапе топологии) Federation Framework получает исходный макет TQL для конечных ЭК и связей из соответствующих репозиториев.

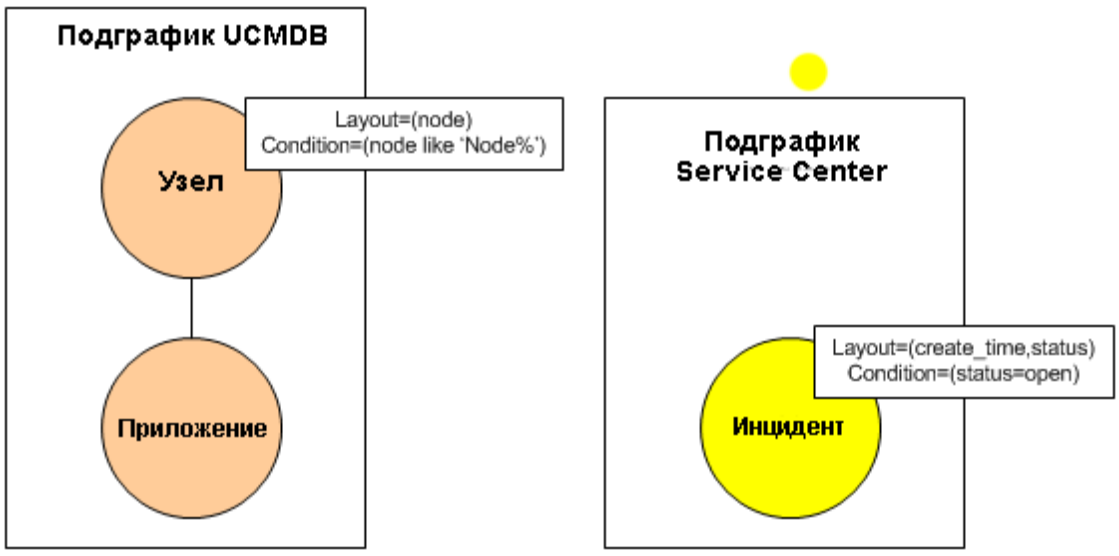
Пример потока Federation Framework для объединенных TQL-запросов

В этом примере описывается способ просмотра всех открытых инцидентов для определенных узлов. Репозиторий данных ServiceCenter — это внешний репозиторий. Экземпляры узлов хранятся в UCMDB, а экземпляры инцидентов хранятся в ServiceCenter. Предполагается, что для соединения экземпляров с соответствующим узлом необходимы свойства `node` и `ip_address` элементов host и IP. Это свойства выверки, которые идентифицируют узлы из ServiceCenter в UCMDB.

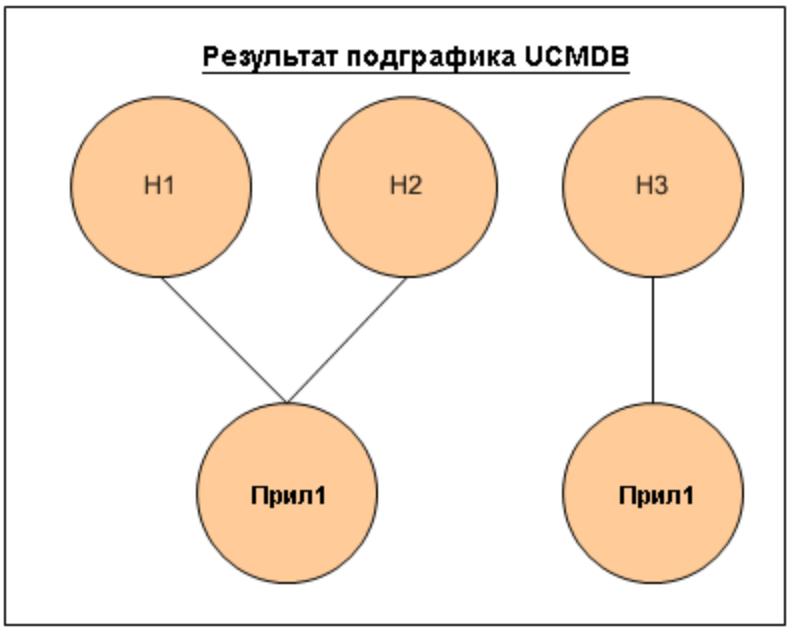


Примечание. Для объединения атрибутов вызывается метод адаптера **getTopology**. Данные выверки адаптируются в TQL-запросе пользователя (в нашем случае элементе ЭК).

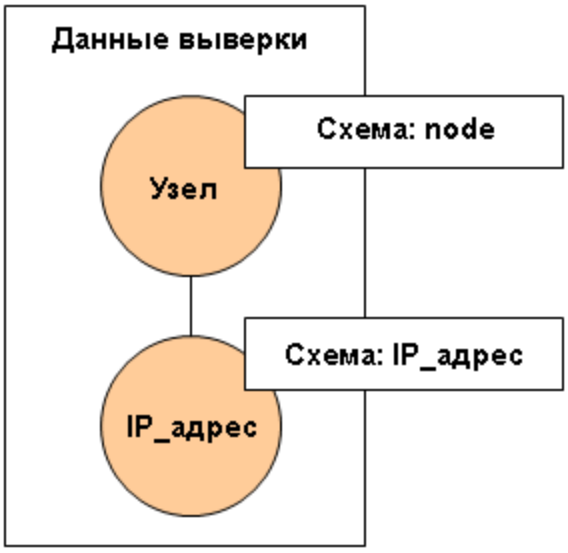
1. После анализа адаптера Federation Framework распознает виртуальное отношение между элементами `Node` и `Incident` и разделяет объединенный TQL-запрос на два подграфа:



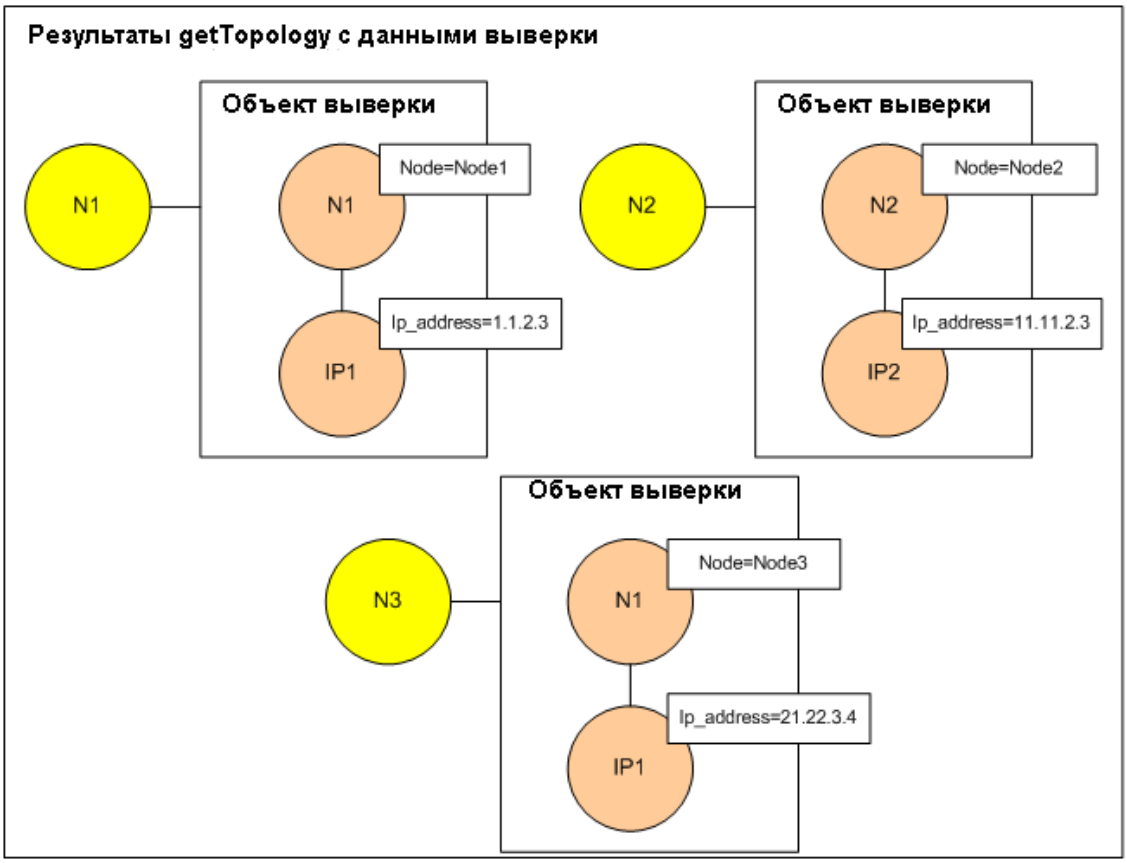
2. Federation Framework выполняет подграф UCMDB для запроса топологии и получает следующие результаты:



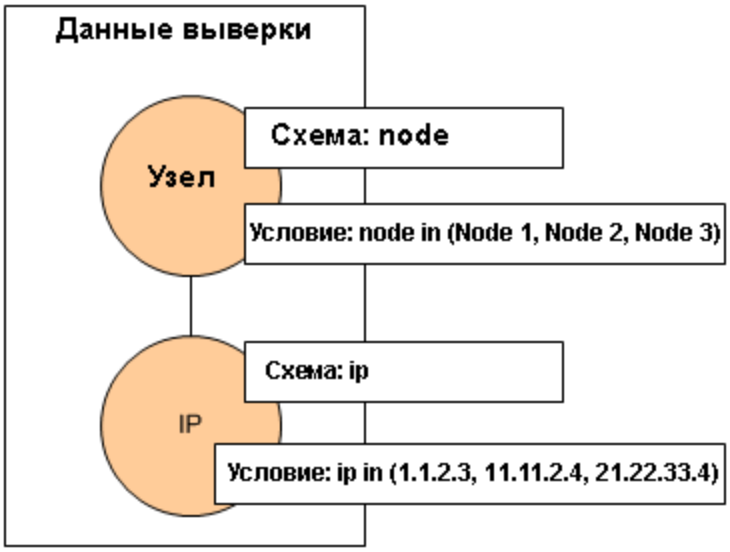
3. Federation Framework запрашивает данные выверки для первого репозитория данных (UCMDB) у соответствующей системы сопоставления, содержащие сведения для соединения полученных данных из двух репозиториев. В этом случае данные выверки примут следующий вид:



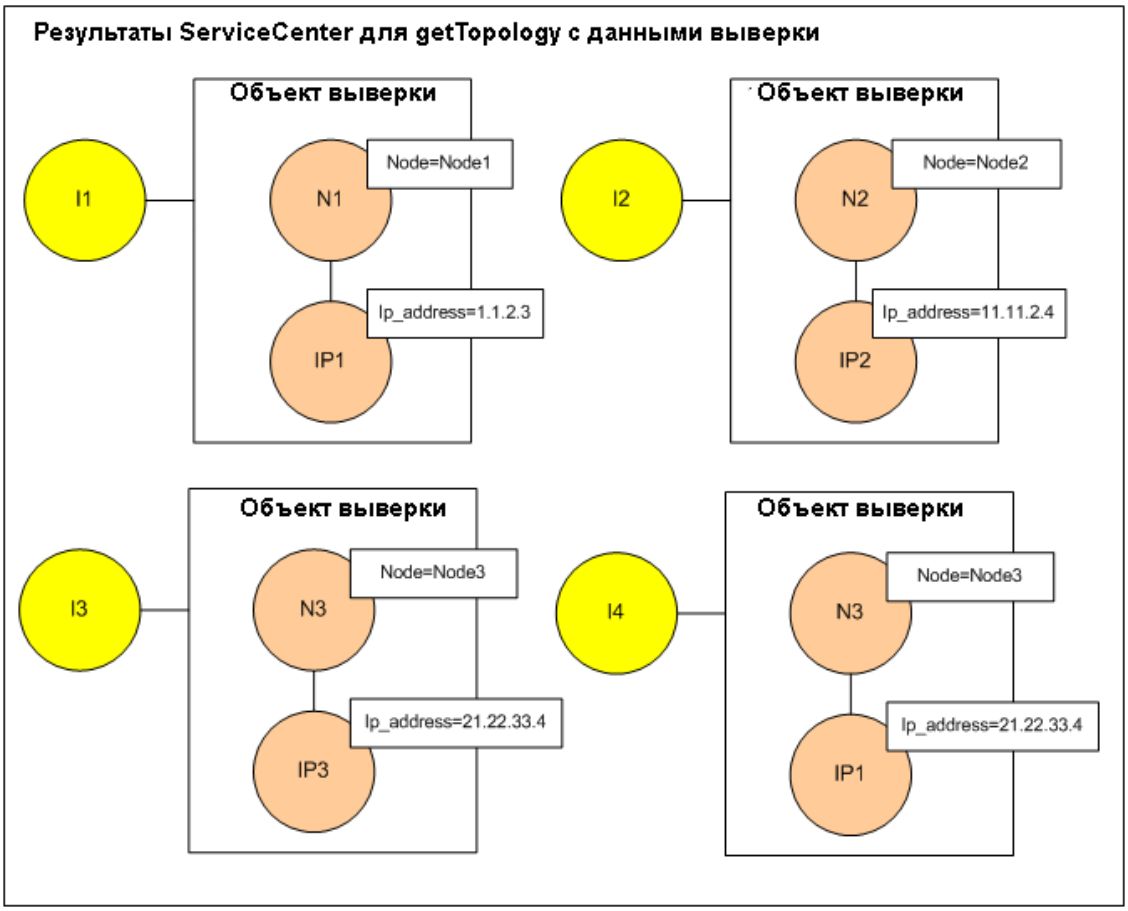
4. Federation Framework создает одноузловой запрос топологии с условиями Node и ID из предыдущего результата (node в H1, H2, H3) и выполняет этот запрос с необходимыми данными выверки в UCMDB. Результат включает ЭК узла, связанные с условием идентификатора и соответствующим объектом выверки для каждого ЭК:



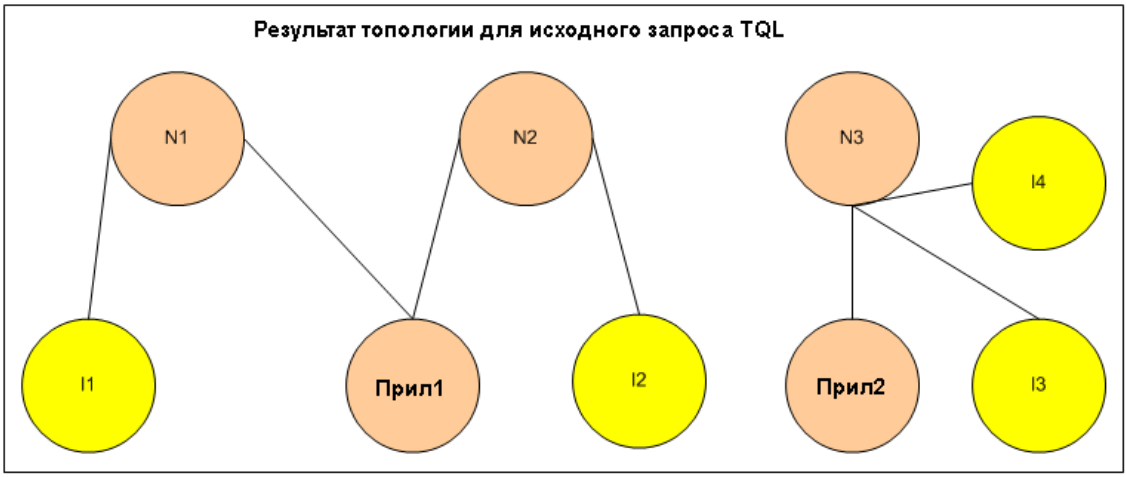
5. Данные выверки для ServiceCenter должны содержать условие для `node` и `ip`, производное от объектов выверки, полученных от UCMDB:



6. Federation Framework выполняет подграф ServiceCenter с данными выверки для запроса топологии и соответствующих объектов выверки и получает следующие результаты:



7. Результат после соединения в системе сопоставления и создания виртуального отношения:



8. Federation Framework запрашивает исходный макет TQL для экземпляров, полученных от UCMDB и ServiceCenter.

Поток Federation Framework для заполнения

Этот раздел охватывает следующие темы:

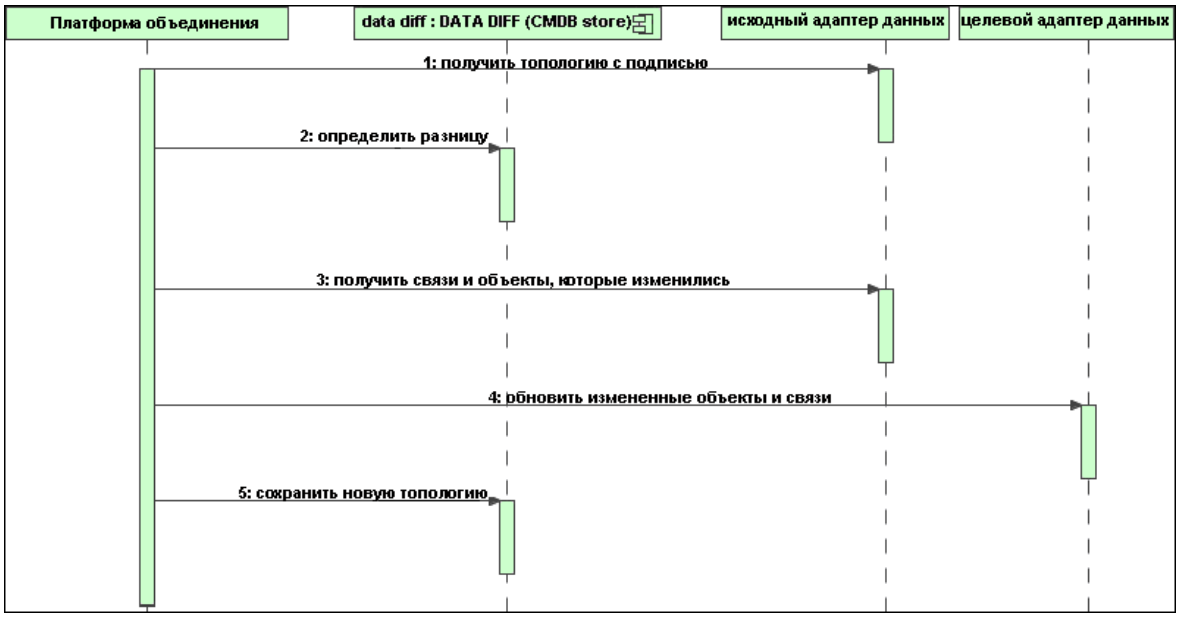
- "Определения и термины" ниже
- "Диаграмма потока" ниже

Определения и термины

Сигнатура. Обозначает состояние свойств ЭК. Если в значения свойств ЭК вносятся изменения, сигнатура ЭК также должна быть изменена. Сигнатура ЭК помогает обнаружить изменения ЭК без получения и сравнения свойств ЭК. ЭК и сигнатура ЭК предоставляются соответствующим адаптером. Адаптер изменяет сигнатуру ЭК при изменении свойств ЭК.

Диаграмма потока

В следующей последовательной диаграмме представлено взаимодействие между Federation Framework и исходными/целевыми адаптерами в потоке наполнения.



1. Federation Framework получает топологию результата запроса от исходного адаптера. Адаптер распознает запрос по имени и выполняет его во внешнем репозитории данных. Результат топологии содержит идентификатор и сигнатуру каждого ЭК и связи в результате. Идентификатор — это логический идентификатор, который определяет ЭК, уникальный во внешнем репозитории данных. Сигнатура должна быть изменена в случае изменения ЭК или связи.
2. Federation Framework использует сигнатуры для сравнения недавно полученных результатов запроса топологии с сохраненными результатами и выявления измененных ЭК.

3. После того как компонент Federation Framework находит измененные ЭК и связи, он вызывает исходный адаптер, используя идентификаторы измененных ЭК и связей в качестве параметра, чтобы извлечь их полный макет.
4. Federation Framework отправляет обновление целевому адаптеру. Целевой адаптер обновляет внешний источник данных с использованием полученных данных.
5. После обновления Federation Framework сохраняет последний результат запроса.

Интерфейсы адаптера

Этот раздел охватывает следующие темы:

- "Определения и термины" ниже
- "Интерфейсы адаптера для объединенных TQL-запросов" ниже

Определения и термины

Внешнее отношение. Отношение между двумя внешними типами ЭК, поддерживаемыми одним адаптером.

Интерфейсы адаптера для объединенных TQL-запросов

Используйте соответствующие интерфейсы для каждого адаптера (см. ниже).

- **Интерфейс топологии Single Node** используется, если адаптер не поддерживает внешние связи, т.е. если он не предназначен для получения запросов более чем с одним внешним ЭК. Данные выверки, необходимые для завершения операции, можно указать в сложном запросе (см. SingleNodeFederationTopologyReconciliationAdapter ниже).

  Все интерфейсы SingleNode создаются для упрощения рабочего процесса. В случаях, когда необходимы более сложные запросы, используйте интерфейс **FederationTopologyAdapter**.
- **Интерфейс FederationTopologyAdapter** используется для настройки адаптеров, поддерживающих сложные объединенные запросы. Запрос выверки в этих адаптерах является частью параметра **QueryDefinition**.

  Механизм объединения использует данные выверки для подключения объединенных данных к соответствующим локальным ЭК. Данные выверки могут извлекаться при помощи нескольких запросов (выполняемых рекурсивно, в соответствии с результатами). При этом адаптер получает запрос только с данными выверки.

Интерфейсы SingleNode

В следующих интерфейсах применяются разные типы данных выверки:

- **SingleNodeFederationIdReconciliationAdapter.** Используется, если адаптер поддерживает **single-node TQL** и выверка между репозиториями вычисляется по идентификатору.
- **SingleNodeFederationPropertyReconciliationAdapter.** Используется, если адаптер поддерживает **single-node TQL** и выверка между репозиториями вычисляется по свойствам ЭК.
- **SingleNodeFederationTopologyReconciliationAdapter.** Используется, если адаптер поддерживает **single-node TQL** и выверка между репозиториями вычисляется по

топологии. Необходимо, чтобы адаптер корректно обрабатывал пустые элементы запросов, в которых запрашиваются только сведения о топологии выверки.

Интерфейсы Data Adapter

- **FederationTopologyAdapter.** Используется для сложных объединенных TQL-запросов. Обеспечивает максимальное разнообразие. Необходимо, чтобы адаптер корректно обрабатывал запросы, в которых содержатся только данные выверки.
- **PopulateDataAdapter.** Используется для сложных объединенных TQL-запросов и потоков заполнения. В потоке заполнения этот адаптер получает весь набор данных и позволяет зонду отфильтровать различия, возникшие с момента последнего выполнения задания.
- **PopulateChangesDataAdapter.** Используется для сложных объединенных TQL-запросов и потоков заполнения. В потоке заполнения этот адаптер получает только изменения, возникшие с момента последнего выполнения задания.

Примечание. При разработке адаптера, который может возвращать большие объемы данных, важно предусмотреть возможность разбиения данных на фрагменты, реализовав интерфейс ChunkGetter. Дополнительные сведения см. в документации Java для конкретного адаптера.

Интерфейсы обнаружения ресурсов

Ниже приведены интерфейсы обнаружения ресурсов, при помощи которых можно влиять на поведение адаптера. Вносить изменения в ресурсы можно напрямую из Студии интеграции. Эти интерфейсы следует использовать в дополнение к основным интерфейсам адаптеров.

- **PopulationQueriesResourcesLocator.** Определяет изменяемые ресурсы для каждого запроса заполнения.
- **PushQueriesResourceLocator.** Определяет изменяемые ресурсы для каждого запроса принудительной отправки данных.
- **GeneralResourcesLocator.** Определяет общие изменяемые ресурсы адаптера.

Дополнительные интерфейсы

- **SortResultDataAdapter.** Используется для сортировки полученных ЭК во внешнем репозитории.
- **FunctionalLayoutDataAdapter.** Используется для вычисления функционального макета во внешнем репозитории.

Интерфейсы адаптера для синхронизации

- **SourceDataAdapter.** Используется для исходных адаптеров в потоках заполнения.
- **TargetDataAdapter.** Используется для целевых адаптеров в потоках принудительной отправки.

Устранение неполадок в ресурсах адаптеров

В данной задаче описаны приемы использования консоли JMX для создания, просмотра и удаления ресурсов состояния адаптеров (любых ресурсов, созданных методами,

реализованными в интерфейсе DataAdapterEnvironment, и сохраненными в базе данных CMDB или зонда) в процессе устранения неполадок или разработки.

1. Запустите веб-браузер и введите адрес сервера:
   - Для сервера CMDB: `http://localhost:8080/jmx-console`
   - Для зонда: `http://localhost:1977`

   Возможно, потребуется ввести имя пользователя и пароль для входа в систему.
2. Чтобы открыть страницу JMX MBEAN View, выполните одно из следующих действий:
   - На сервере CMDB : нажмите **UCMDB:service=FCMDB Adapter State Resource Services**
   - На зонде: нажмите **type=AdapterStateResources**
3. Введите значения для необходимых операций и нажмите **Invoke**.

Добавление адаптера для нового внешнего источника данных

В этой задаче описывается создание адаптера для поддержки нового внешнего источника данных.

Эта задача включает следующие шаги:

- "Необходимые условия" ниже
- "Указание действующих связей для виртуальных связей" на следующей странице
- "Определение конфигурации адаптера" на следующей странице
- "Указание поддерживаемых классов" на странице 220
- "Реализация адаптера" на странице 221
- "Укажите правила выверки или реализуйте систему сопоставления" на странице 222
- "Добавить JAR-файлы, необходимые для реализации Classpath" на странице 222
- "Развертывание адаптера" на странице 222
- "Обновление адаптера" на странице 223

1. Необходимые условия

Классы адаптера, поддерживаемые моделью, для ЭК и связей в модели данных UCMDB. Разработчик адаптеров должен:

- знать иерархию типов ЭК UCMDB и понимать связь внешних типов ЭК с типами ЭК UCMDB;
- моделировать внешние типы ЭК в модели классов UCMDB;
- добавлять определения новых типов ЭК и их связей;
- формировать допустимые связи в модели классов UCMDB для допустимых связей

между внутренними классами адаптера. (Типы ЭК могут быть помещены на любом уровне дерева модели классов UCMDB.)

Моделирование должно быть одинаковым независимо от типа объединения (в оперативном режиме или репликация). Подробные сведения о добавлении новых типов ЭК в модель классов UCMDB см. в разделе "Работа с селектором ЭК" в документе *Руководство по моделированию в HP Universal CMDB*.

Чтобы адаптер поддерживал объединенные атрибуты в типах ЭК, этот тип ЭК необходимо добавить в поддерживаемые классы с поддерживаемыми атрибутами и правилом выверки для этого типа ЭК.

2. Указание действующих связей для виртуальных связей

Примечание. Этот раздел относится только к объединению.

Для получения объединенных типов ЭК, соединенных с локальными типами ЭК CMDB, должно существовать определение допустимой связи между двумя ЭК в CMDB.

a. Создайте XML-файл с этими связями (если они не существуют).
b. Добавьте XML-файл связей в пакет адаптера в каталог **\validlinks**. Подробнее см. в разделе в документе *Руководство по администрированию HP Universal CMDB*.

Пример определения действующего отношения:

В следующем примере связь типа `containment` между экземплярами типа `node` и экземплярами типа `myclass1` является допустимым определением связи.

```
<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. Определение конфигурации адаптера

a. Откройте раздел **Управление адаптерами**.
b. Нажмите кнопку **Создать новый ресурс**  и выберите **Создать адаптер**.
c. В диалоговом окне создания адаптера выберите **Интеграция** и **Адаптер Java**.
d. Щелкните созданный адаптер правой кнопкой мыши и выберите **Изменить источник адаптера** в меню ярлыков.
e. Измените следующие теги XML:

```
<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="newAdapterIdName"
xsi:noNamespaceSchemaLocation="../../Patterns.xsd"
```

```
description="Adapter Description" schemaVersion="9.0"
displayName="New Adapter Display Name">
<deletable>true</deletable>
<discoveredClasses>
<discoveredClass>link</discoveredClass>
<discoveredClass>object</discoveredClass>
</discoveredClasses>
<taskInfo
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
 AdapterService">
<params
className="com.hp.ucmdb.discovery.probe.services.dynamic.core.
 AdapterServiceParams" enableAging="true"
enableDebugging="false" enableRecording=
"false" autoDeleteOnErrors="success" recordResult="false"
maxThreads="1" patternType="java_adapter"
maxThreadRuntime="25200000">
<className>com.yourCompany.adapter.MyAdapter.MyAdapterClass
</className>
</params>
<destinationInfo
className="com.hp.ucmdb.discovery.probe.tasks.BaseDestinationDa
ta">
<!-- check -->
<destinationData name="adapterId"
description="">${ADAPTER.adapter_id}</destinationData>
<destinationData name="attributeValues"
description="">${SOURCE.attribute_values}</destinationData>
<destinationData name="credentialsId"
description="">${SOURCE.credentials_id}</destinationData>
<destinationData name="destinationId"
description="">${SOURCE.destination_id}</destinationData>
</destinationInfo>
<resultMechanism isEnabled="true">
<autoDeleteCITs isEnabled="true">
<CIT>link</CIT>
<CIT>object</CIT>
</autoDeleteCITs>
</resultMechanism>
</taskInfo>
```

```
<adapterInfo>
<adapter-capabilities>
<support-federated-query>
<!--<supported-classes/> <!—see the section about supported
classes-->
<topology>
<pattern-topology /> <!—or <one-node-topology> -->
</topology>
</support-federated-query>
<!--<support-replicatioin-data>
<source>
<changes-source/>
</source>
<target/>
</adapter-capabilities>
<default-mapping-engine />
<queries />
<removedAttributes />
<full-population-days-interval>-1</full-population-days-
interval>
</adapterInfo>
<inputClass>destination_config</inputClass>
<protocols />
<parameters>
<!--The description attribute may be written in simple text or
HTML.-->
<!--The host attribute is treated as a special case by UCMDB-->
<!--and will automatically select the probe name (if possible)-
->
<!--according to this attribute’s value.-->
<parameter name="credentialsId" description="Special type of
property, handled by UCMDB for credentials menu" type="integer"
display-name="Credentials ID" mandatory="true" order-index="12"
/>
<parameter name="host" description="The host name or IP address
of the remote machine" type="string" display-name="Hostname/IP"
mandatory="false" order-index="10" />
```

```
<parameter name="port" description="The remote machine's
connection port" type="integer" display-name="Port"
mandatory="false" order-index="11" />
</parameters>
<parameter name="myatt" description="is my att true?"
type="string" display-name="My Att" mandatory="false" order-
index="15" valid-values="True;False"/>True</parameters>
<collectDiscoveredByInfo>true</collectDiscoveredByInfo>
<integration isEnabled="true">
<category >My Category</category>
</integration>
<overrideDomain>${SOURCE.probe_name}</overrideDomain>
<inputTQL>
<resource:XmlResourceWrapper
xmlns:resource="http://www.hp.com/ucmdb/1-0-
0/ResourceDefinition" xmlns:ns4="http://www.hp.com/ucmdb/1-0-
0/ViewDefinition" xmlns:tql="http://www.hp.com/ucmdb/1-0-
0/TopologyQueryLanguage">
<resource xsi:type="tql:Query" group-id="2" priority="low" is-
live="true" owner="Input TQL" name="Input TQL">
<tql:node class="adapter_config" id="-11" name="ADAPTER" />
<tql:node class="destination_config" id="-10" name="SOURCE" />
<tql:link to="ADAPTER" from="SOURCE" class="fcmdb_conf_
aggregation" id="-12" name="fcmdb_conf_aggregation" />
</resource>
</resource:XmlResourceWrapper>
</inputTQL>
<permissions />
</pattern>
```

См. дополнительные сведения о тегах XML в документе "Теги конфигурации и свойства XML" на странице 224.

4. Указание поддерживаемых классов

Укажите поддерживаемые классы в коде адаптера путем реализации метода *getSupportedClasses()* или с помощью XML-файла.

```
<supported-classes>
    <supported-class name="HistoryChange" is-derived="false" is-
reconciliation-supported="false" federation-not-supported="false" is-id-
reconciliation-supported="false">
    <supported-conditions>
```

```
        <attribute-operators attribute-name="change_create_time">
            <operator>GREATER</operator>
            <operator>LESS</operator>
            <operator>GREATER_OR_EQUAL</operator>
            <operator>LESS_OR_EQUAL</operator>
            <operator>CHANGED_DURING</operator>
        </attribute-operators>
    </supported-conditions>
</supported-class>
```

name	Имя типа ЭК
is-derived	Указывает, включает ли определение всех наследующих потомков
is-reconciliation-supported	Указывает, используется ли класс для выверки
is-id-reconciliation-supported	Указывает, используется ли класс для выверки идентификатора
federation-not-supported	Указывает, что объединение этого типа ЭК должно быть запрещено (при этом некоторые типы ЭК, например указанные только для объединения, будут недоступны)
<supported-conditions>	Указывает подер иваемые условия для каждого атрибута

5. Реализация адаптера

Выберите правильный класс реализации адаптера в соответствии с указанными возможностями. Класс реализации адаптера реализует соответствующие интерфейсы согласно указанным возможностям.

Если адаптер использует метод **getTopologyWithReconciliationData**, и его можно использовать в качестве начальной точки, необходимо, чтобы он также поддерживал запросы топологии с данными выборки без указания условий (только тип). При этом адаптер должен возвращать полные данные выверки полученных результатов.

Поддержка выверки адаптеров может определяться в соответствии с **global_id**. При этом необходимо определить **global_id** в составе атрибутов выверки в поддерживаемых адаптерами классах. Если выверка адаптеров осуществляется по **global_id**, **getTopologyWithReconciliationData()** будет возвращать **global_id** в составе свойств объекта выверки. В UCMDB **global_id** используется для выверки объединенных результатов для типа ЭК, а не правила идентификации.

Интерфейс DataAdapterEnvironment входит в API объединения. Этот интерфейс представляет собой среду адаптера данных. Он содержит API, необходимый для работы адаптера. Подробнее об интерфейсе DataAdapterEnvironment см. в разделе "Интерфейс DataAdapterEnvironment" на странице 226.

6. Укажите правила выверки или реализуйте систему сопоставления

   Если адаптер поддерживает объединенные TQL-запросы, существует два варианта настройки системы сопоставления:

   - Использовать систему сопоставления по умолчанию, использующую внутренние правила CMDB для сопоставления. Для этого оставьте XML-тег **<default-mapping-engine>** пустым.
   - Создать собственную систему сопоставления путем реализации интерфейса системы сопоставления и добавления JAR-файла в код адаптера. Для этого добавьте следующий XML-тег: **<default-mapping-engine>com.yourcompany.map.MyMappingEngine</default-mapping-engine>**

7. Добавить JAR-файлы, необходимые для реализации Classpath

   Для реализации классов добавьте файл **federation_api.jar** в classpath редактора кода.

8. Развертывание адаптера

   Разверните пакет адаптера. Общие сведения о развертывании пакета см. в разделе "Диспетчер пакетов" в документе *Руководство по администрированию HP Universal CMDB*.

   Пакет должен содержать следующие объекты:

   - Определение нового типа ЭК (необязательно):
   - Используется, только если адаптер поддерживает новые типы ЭК, которые еще не существуют в UCMDB.
   - Определения новых типов ЭК находятся в папке `class` пакета.
   - Определение нового типа данных (необязательно):
   - Используется, только если новые типы ЭК требуют новых типов данных.
   - Определения новых типов данных находятся в папке `typedef` пакета.
   - Определение действующих связей (необязательно):
   - Используется, только если адаптер поддерживает объединенные TQL-запросы.
   - Определения новых допустимых связей находятся в папке `validlinks` пакета.
   - XML-файл конфигурации образца должен находиться в папке `validlinks` пакета.
   - **Descriptor.** Определяет связи пакета.
   - Поместите скомпилированные классы (обычно JAR-файл) в папку **adapterCode\<adapter id>** пакета.

     Примечание. Имя папки `adapter id` имеет то же значение, что в конфигурации адаптера.

- Если вы создаете собственный файл конфигурации, поместите файл в папку **adapterCode\<adapter id>** пакета.

9. Обновление адаптера

   Изменения любых недвоичных файлов адаптера можно внести с помощью модуля управления адаптерами. После внесения изменений в файлы конфигурации в модуле управления адаптерами выполняется перезагрузка адаптера с новыми конфигурациями.

   Кроме того, обновления можно внести путем редактирования файлов в пакете (двоичных и недвоичных) с последующим повторным развертыванием пакета с помощью диспетчера пакетов. Подробнее см. в разделе "Развертывание пакета" в документе *Руководство по администрированию HP Universal CMDB*.

Создание примера адаптера

В этом разделе иллюстрируется создание примера адаптера. Эта задача включает следующие шаги:

- "Выбор логики адаптера" ниже
- "Загрузка проекта" ниже

1. Выбор логики адаптера

   При реализации адаптера необходимо выбрать способ обработки логики условий в реализации (условия свойств, условия идентификаторов, условия выверки и условия связи).

   a. Извлечение всего набора данных в память адаптера, выбор или фильтрация необходимых экземпляров.
   b. Преобразование всех условий в язык источника данных, фильтрация и выбор данных с его помощью. Пример:
      - Преобразование условия в SQL-запрос.
      - Преобразование условия в объект фильтра Java API.
   c. Фильтрация части данных в удаленной службе и выбор/фильтрация оставшихся данных с помощью адаптера.

   В примере MyAdapter используется логика шага *a*.

2. Загрузка проекта

   Скопируйте файлы из папки **C:\hp\UCMDB\UCMDBServer\tools\adapter-dev-kit\SampleAdapters** и следуйте инструкциям в файлах сведений.

   Примечание. При использовании адаптера с большими наборами данных используйте кэширование и индексацию для улучшения производительности объединения.

   Интерактивная документация javadocs доступна по адресу:

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

Теги конфигурации и свойства XML

id="newAdapterIdName"		Определяет реальное имя адаптера. Используется для поиска журналов и папок.
displayName="New Adapter Display Name"		Определяет отображаемое имя адаптера в интерфейсе.
<className>...</className>		Определяет интерфейс адаптера, который реализует класс Java.
<category >My Category</category>		Определяет категорию адаптера.
<parameters>		Определяет свойства конфигурации, доступные в интерфейсе при установке новой точки интеграции.
	name	Имя свойства (в основном используется кодом).
	description	Подсказка свойства.
	type	Строка или число (используйте допустимые значения строки для типа «логический»).
	display-name	Имя свойства в интерфейсе.
	mandatory	Определяет обязательность свойства конфигурации для пользователя.
	order-index	Порядок размещения свойства (чем меньше размер, тем выше будет место свойства в списке).
	valid-values	Список допустимых значений, разделенных символами ';' (например, alid-values="Oracle;SQLServer;MySQL" or valid-values="True;False").
<adapterInfo>		Определение статических параметров и возможностей адаптера.
	<support-federated-query>	Определяет способность адаптера к объединению.
	<start-point-adapter>	Указывает адаптер в качестве начальной точки расчета TQL-запроса.
	<one-node-topology>	Возможность объединения запросов с

		одним объединенным узлом запроса.
	<pattern-topology>	Возможность объединять сложные запросы.
	<support-replicatioin-data>	Определяет возможность выполнения потоков отправки данных и наполнения.
	<source>	Адаптер может использоваться для потоков наполнения.
	<push-back-ids>	Возврат глобального ID ЭК в столбец global_id таблицы (необходимо задать в файле orm.xml). Для переопределения этого поведения можно подключить модуль **FcmdbPluginPushBackIds**.
	<changes-source>	Адаптер может использоваться для потоков наполнения изменений.
	<instance-based-data>	Определяет поддержку адаптером потока заполнения на основе экземпляра.
	<target>	Адаптер может использоваться для потоков принудительной отправки данных.
	<default-mapping-engine>	Обеспечивает определение системы сопоставления для адаптера (по умолчанию адаптер использует систему сопоставления по умолчанию). Для любой другой системы сопоставления введите имя реализующего класса системы сопоставления
	<removedAttributes>	Принудительное удаление определенных атрибутов из результата.
	<full-population-days-interval>	Определяет частоту выполнения полного задания наполнения вместо разностного (каждые 'x' дней). Использует механизм старения вместе с потоком изменений.
	<adapter-settings>	Список настроек адаптера.
	<list.attributes.for.set>	Указывает, какие атрибуты переопределяют предыдущее значение (если оно существует).

Интерфейс DataAdapterEnvironment

OutputStream openResourceForWriting(String resourceName) throws FileNotFoundException;

Данный метод открывает ресурс с указанным именем для записи. Он используется для сохранения постоянных данных. Следует использовать его вместо загрузки файлов средствами Java. После завершения записи необходимо закрыть поток. За сохранение ресурса отвечает команда close()/flush(). Этот метод создает ресурс среды выполнения (без записи поверх файлов пакета адаптера).

Параметр

- **resourceName:** Имя извлекаемого ресурса. Это имя должно быть уникальным в рамках всех интеграций одного адаптера.

Возвращаемое значение

Поток для записи.

Исключения

- Этот метод выдает исключение *FileNotFoundException*, если в качестве типа ресурса указан несуществующий файл, если вместо файла задана директория, либо по какой-то причине ресурс не удается открыть для чтения.
- Этот метод выдает исключение *SecurityException* при наличии Диспетчера безопасности, метод *checkRead* которого запрещает доступ к файлу.

InputStream openResourceForReading(String resourceName) throws FileNotFoundException;

Данный метод открывает ресурс с указанным именем для чтения. Он используется для чтения постоянных данных интеграции. Следует использовать его вместо загрузки файлов средствами Java. После завершения чтения необходимо закрыть поток. Сначала этот метод пытается загрузить файлы из пакета адаптера. Если таких файлов не обнаружено, метод пытается загрузить ресурс, созданный при выполнении *DataAdapterEnvironment.openResourceForWriting(String)*. Ресурсы среды выполнения можно просматривать при помощи JMX (зонда или сервера).

Параметр

- **resourceName:** Имя извлекаемого ресурса. Это имя должно быть уникальным в рамках всех интеграций одного адаптера.

Возвращаемое значение

Поток для чтения.

Исключения

- Этот метод выдает исключение *FileNotFoundException*, если в качестве типа ресурса указан несуществующий **файл**, если вместо файла задана директория, либо по какой-то причине ресурс не удается открыть для чтения.
- Этот метод выдает исключение *SecurityException* при наличии Диспетчера безопасности, метод *checkRead* которого запрещает доступ к файлу.

Properties openResourceAsProperties(String propertiesFile) throws IOException;

Данный метод открывает ресурс с указанным именем и загружает его в виде структуры *Свойств*. Он используется для чтения постоянных данных интеграции. Следует использовать его вместо загрузки файлов **.properties** средствами Java. Сначала этот метод пытается загрузить файлы из пакета адаптера. Если таких файлов не обнаружено, метод пытается загрузить ресурс, созданный при выполнении *DataAdapterEnvironment.openResourceForWriting(String)*. Ресурсы среды выполнения можно просматривать при помощи JMX (зонда или сервера).

Параметр

- **propertiesFile:** Имя извлекаемого ресурса. Это имя должно быть уникальным в рамках всех интеграций одного адаптера.

Возвращаемое значение

Содержимое файла в виде раздела "Свойства".

Исключения

- Этот метод выдает исключение *FileNotFoundException*, если в качестве типа ресурса указан несуществующий **файл**, если вместо файла задана директория, либо по какой-то причине ресурс не удается открыть для чтения.
- Этот метод выдает исключение *SecurityException* при наличии Диспетчера безопасности, метод *checkRead* которого запрещает доступ к файлу.
- Исключение *IOException* означает, что файл свойств не удалось преобразовать в объект *Свойства*.

String openResourceAsString(String resourceName) throws IOException;

Данный метод открывает ресурс с указанным именем и загружает его в виде строки. Он используется для чтения постоянных данных интеграции. Следует использовать его вместо загрузки файлов средствами Java.

Сначала этот метод пытается загрузить файлы из пакета адаптера. Если таких файлов не обнаружено, метод пытается загрузить ресурс, созданный при выполнении *DataAdapterEnvironment.openResourceForWriting(String)*. Ресурсы среды выполнения можно просматривать при помощи JMX (зонда или сервера).

Параметр

- **resourceName:** Имя извлекаемого ресурса. Это имя должно быть уникальным в рамках всех интеграций одного адаптера.

Возвращаемое значение

Содержимое файла в формате строки.

Исключения

- Этот метод выдает исключение *FileNotFoundException*, если в качестве типа ресурса указан несуществующий **файл**, если вместо файла задана директория, либо по какой-то причине ресурс не удается открыть для чтения.
- Этот метод выдает исключение *SecurityException* при наличии Диспетчера безопасности, метод *checkRead* которого запрещает доступ к файлу.
- Исключение *IOException* означает ошибку ввода/вывода.

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

Этот метод извлекает сроку и сохраняет ее в виде ресурса. Он используется для сохранения постоянных данных. Следует использовать его вместо сохранения файлов средствами Java. Метод преобразует строку в поток и сохраняет в виде ресурса. Этот метод создает ресурс среды выполнения (без записи поверх файлов пакета адаптера). Ресурсы среды выполнения можно просматривать при помощи JMX (зонда или сервера).

Параметр

- **relativeFileName:** Имя извлекаемого ресурса. Это имя должно быть уникальным в рамках всех интеграций одного адаптера.
- **значение:** Строка, сохраняемая в виде ресурса

Исключения

Исключение IOException означает ошибку ввода/вывода.

boolean resourceExists(String resourceName);

Этот метод проверяет наличие ресурса с указанным именем. Этот метод пытается искать файлы из пакета адаптера, а также среду выполнения, созданную методом *DataAdapterEnvironment.openResourceForWriting(String)*.

Параметр

- **resourceName:** Имя извлекаемого ресурса. Это имя должно быть уникальным в рамках всех интеграций одного адаптера.

Возвращаемое значение

Возвращает значение **True**, если *resourceName* существует.

boolean deleteResource(String resourceName);

Этот метод удаляет указанный ресурс из постоянных данных. Этот метод удаляет ресурс среды выполнения (без удаления файлов пакета адаптера). Ресурсы среды выполнения можно просматривать при помощи JMX (зонда или сервера).

Параметр

- **resourceName:** Имя извлекаемого ресурса. Это имя должно быть уникальным в рамках всех интеграций одного адаптера.

Возвращаемое значение

Возвращает значение **True** при успешном удалении ресурса.

Collection<String> listResourcesInPath(String path);

Этот метод извлекает список ресурсов, расположенных по указанному адресу. Этот метод пытается искать файлы из пакета адаптера, а также среду выполнения, созданную методом *DataAdapterEnvironment.openResourceForWriting(String)*. Ресурсы среды выполнения можно просматривать при помощи JMX (зонда или сервера).

Параметр

- **path:** Адрес ресурса. Например, "META-INF/myfiles/"

Возвращаемое значение

Список ресурсов, расположенных по указанному адресу.

DataAdapterLogger getLogger();

Извлекает модуль ведения журнала, который будет записывать события адаптера.

Возвращаемое значение

Модуль ведения журнала DataAdapter.

DestinationConfig getDestinationConfig();

Этот метод извлекает конфигурацию целевого устройства интеграции. В конфигурации содержатся сведения о подключении и работе интеграции.

Возвращаемое значение

Возвращает файл DestinationConfig.

int getChunkSize();

Этот метод извлекает размер блоков данных для заполнения в рамках интеграции.

Возвращаемое значение

Размер блоков данных для заполнения.

int getPushChunkSize();

Этот метод извлекает размер блоков принудительной отправки данных в рамках интеграции.

Возвращаемое значение

Размер блоков принудительной отправки данных.

ClassModel getLocalClassModel();

Этот метод извлекает модель классов для для запроса сведений о модели классов локальной базы UCMDB. Метод обновляет ClassModel. После получения объект ClassModel не будет обновляться в связи с какими-либо обновлениями модели классов. Для получения обновленной модели классов необходимо повторно вызвать этот метод.

Возвращаемое значение

Модель классов UCMDB.

CustomerInformation getLocalCustomerInformation();

Этот метод извлекает сведения о пользователе, от имени которого запускается адаптер.

Возвращаемое значение

Сведения о пользователе, от имени которого запускается адаптер.

Object getSettingValue(String name);

Этот метод извлекает определенный параметр адаптера.

Параметр

name: Имя параметра.

Возвращаемое значение

Значение параметра объекта.

Map<String, Object> getAllSettings();

Этот метод извлекает все параметры адаптера.

Возвращаемое значение

Параметры адаптера.

boolean isMTEnabled();

Этот метод проверяет наличие поддержки множественной аренды (MT) в среде сервера.

Возвращаемое значение

Возвращает значение **True**, если на сервере поддерживается функция множественной аренды. В противном случае возвращает значение **False**.

String getUcmdbServerHostName();

Этот метод возвращает имя хоста сервера локальной базы UCMDB.

Возвращаемое значение

Имя хоста сервера локальной базы UCMDB.

Глава 7: Разработка адаптеров принудительной отправки

Данная глава включает:

- Разработка и развертывание адаптеров принудительной отправки данных 232
- Создание пакета адаптера 233
- Создание сопоставлений 236
- Создание сценариев Jython 240
- Поддержка разностной синхронизации 244
- SQL-запросы общего адаптера принудительной отправки в формате XML 245
- Общий адаптер принудительной отправки данных веб-служб 246
- Справочные сведения о файлах сопоставления 265
- Схема файла сопоставления 267
- Схема результатов сопоставления 276
- Настройка 278

Разработка и развертывание адаптеров принудительной отправки данных

Общий адаптер принудительной отправки предоставляет платформу для быстрой разработки адаптеров интеграции, выполняющих принудительную отправку данных UCMDB во внешние репозитории (базы данных и сторонние приложения). В зависимости от протокола, используемого для отправки данных, адаптеры относят к той или иной категории. Подробные сведения о принудительной отправке данных через XML с помощью общего адаптера принудительной отправки данных в формате XML см. в разделе "SQL-запросы общего адаптера принудительной отправки в формате XML" на странице 245. Подробные сведения о принудительной отправке данных через веб-службу с помощью общего адаптера принудительной отправки данных веб-служб см. в разделе "Общий адаптер принудительной отправки данных веб-служб" на странице 246.

Для разработки настраиваемых адаптеров интеграции на основе общего адаптера принудительной отправки требуются следующие условия:

- Создание пакета адаптера, созданный на базе соответствующего шаблона общего адаптера. Дополнительные сведения см. в разделе "Создание пакета адаптера" на следующей странице.
- Сопоставление между типами связей ЭК UCMDB и внешними элементами данных. Сведения о сопоставлении хранятся в файле XML. Для каждого внешнего источника данных в этот файл требуется внести соответствующие изменения. Дополнительные сведения см. в разделе "Создание сопоставлений" на странице 236.

- Сценарий Jython для принудительной отправки элементов данных во внешний репозиторий. Дополнительные сведения см. в разделе "Создание сценариев Jython" на странице 240.
- Дополнительные шаги, необходимые при работе с определенными адаптерами. Например, указание пути к файлу записи для адаптера принудительной отправки данных в формате XML или создание устройства приема данных веб-служб.

Создание пакета адаптера

Чтобы создать новый адаптер принудительной отправки данных MDR, необходимо создать копию общего адаптера и внести в нее изменения в соответствии с предполагаемым целевым устройством.

Пакеты общих адаптеров находятся в одной из двух папок:

- Общий адаптер принудительной отправки данных в XML-формате: **hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip**
- Общий адаптер данных веб-служб: **hp\UCMDB\UCMDBServer\content\adapters\web-service-push-adapter.zip**

Для создания нового адаптера принудительной отправки данных на базе общего выполните следующие шаги.

1. Извлеките содержимое из архивного файла выбранного пакета в рабочую папку.
2. Проанализируйте содержимое следующих директорий перед тем, как начать переименование и замену файлов:
   - **adapterCode:** Содержит директорию с путем развертывания **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase**. Jar-файлы, которые записываются при развертывании, не отвечают за автоматический перезапуск адаптера и не переносятся автоматически в каталог зонда CLASSPATH.
   - **discoveryConfigFiles:** Содержит определения метода сопоставления адаптера и указание на сценарий Jython (**push.properties**)
   - **discoveryPatterns:** Содержит определение файла XML, развертываемого на сервере UCMDB
   - **discoveryScripts:** Содержит сценарии Jython, используемые для подключения адаптера к сторонним хранилищам данных
   - **discoveryResources:** Содержит файл **UCMDBDataReceiver.jar** с классами интеграции Java для работы с веб-службами.

   Примечание. При развертывании этого пакета происходит перезапуск зонда для того, чтобы включить файл **.jar** в каталог зонда CLASSPATH. Помимо развертывания пакета других действий не требуется.
3. Внесите следующие изменения в структуру распакованного каталога:

a. **discoveryConfigFiles\<Имя_пользовательского_адаптера>:** Переименуйте директорию "PushAdapter" или "XMLtoWebService" в соответствии с созданным адаптером (например, “myPushAdapter”).

b. **discoveryConfigFiles\<Имя_пользовательского_адаптера>\push.properties:** Внесите следующие изменения в файл **push.properties**:

   - Вместо **jythonScript.name** укажите имя сценария Jython для созданного адаптера (например, **pushToMyService.py**).
   - Обновите имя файла сопоставления для созданного адаптера (например, myPushAdapter_mappings). Указывать расширение **.xml** не следует — это делается автоматически.

c. **discoveryPatterns\<имя адаптера>.xml:** Переименуйте этот файл в соответствии с XML-файлом определения созданного адаптера (например, **my_push_adapter.xml**).

d. **discoveryPatterns\<имя_пользовательского_адаптера>.xml:** Внесите в файл следующие изменения:

   - В элементе XML **<pattern>:** укажите ID и описание атрибутов. Пример.

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

     Вариант замены:

     ```
     <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">
     ```

   - В элементе XML **<parameters>:** обновите дочерние элементы в соответствии с настройками адаптера. По умолчанию для определения адаптера принудительной отправки данных используются следующие дочерние элементы. Эти значения указываются при определении точки интеграции в Студии интеграции после настройки адаптера. Обновите список параметров в соответствии с необходимыми атрибутами подключения. Не удаляйте атрибут **probeName**.

     - **host:** имя сервера, на котором размещена веб-служба
     - **port:** порт прослушивания службы устройства приема данных UCMDB
     - **Адаптер принудительной отправки данных веб-служб: uri** — отрезок URL-адреса для указания адреса конечной точки службы устройства приема данных.
     - **probeName:** зонд Data Flow Probe, который отвечает за принудительную отправку данных.

   - В элементе XML **<integration>:** смените значение дочернего элемента **<category>** с "Generic" на любое другое. По умолчанию адаптеры интеграции, которые принадлежат к категории Generic, не отображаются в Студии интеграции. При интеграции со сторонним хранилищем данных здесь следует указать "Third Party". При интеграции с продуктом HP BTO здесь следует указать "HP BTO Products".

e. **adapterCode\PushAdapter:** переименуйте папку в соответствии с идентификатором адаптера, используемым в предыдущем шаге (например, **adapterCode\MyPushAdapter**).

f. **discoveryScripts\<сценарий_принудительной_отправки_Jython>.py:** Создайте файл с именем, указанным в свойстве **push.properties jythonScript.name**. Файл **discoveryScript** содержит сценарий, который помещает ЭК и связи во внешнюю базу данных Oracle. Замените **discoveryScripts\pushScript.py** на созданный сценарий (см. дополнительные сведения в разделе "Создание сценариев Jython" на странице 240). Если сценарий переименован, свойство **jythonScript.name** в **adapterCode\<ID адаптера>\push.properties** должно быть обновлено соответствующим образом.

   - Адаптер принудительной отправки данных в XML-формате: **pushScript.py**
   - Адаптер принудительной отправки данных веб-служб: **XMLtoWebService.py**

g. **tql\<TQL_запросы_интеграции>:** как и при работе с обычным пакетом, следует поместить определение TQL-запросов интеграции в формате XML в эту папку. Все TQL-запросы в ней будут развернуты в ходе развертывания пакета.

h. **discoveryConfigFiles\<Имя_пользовательского_адаптера>\mappings:** создайте XML-файл сопоставления для каждого TQL-запроса, который будет использоваться в интеграции. Следует помнить, что адаптер применяет метод преобразования, указанный в файле сопоставления, к результатам TQL-запроса, а затем передает полученные данные при помощи трех параметров (**addResult**, **updateResult** и **deleteResult**) в виде стандартной задачи зонда Data Flow Probe.

i. **adapterCode\<ID адаптера>\mappings:** замените файл **mappings.xml** на подготовленные файлы сопоставления (см. дополнительные сведения в разделе "Создание сопоставлений" на следующей странице).

   Адаптер принудительной отправки данных в XML-формате: Этот пример соответствует примеру таблиц в ORACLE в файле sql_queries.

   Чтобы использовать файл сопоставления для каждого метода TQL, назначьте имя соответствующего TQL-запроса каждому XML-файлу с расширением **.xml**. В данном случае, если для TQL-запроса не найден определенный файл сопоставления, по умолчанию будет использоваться файл **mappings.xml** . Имя файла сопоставления по умолчанию можно заменить на любое другое в параметре **mappingFile.default** в **adapterCode\<adapterID>\push.properties**.

4. После внесения перечисленных изменений, упакуйте папки и файлы, указанные в шаге 3, в файл **.zip** (например, **my_Push_Adapter.zip**).
5. Разверните этот файл **.zip** на сервере UCMDB при помощи Диспетчера пакетов (**Администрирование > Диспетчер пакетов**).
6. Создайте точку интеграции (**Управление потоком данных > Студия интеграции**) и задайте для нее TQL-запросы. Составьте расписание принудительной отправки данных.

Устранение неполадок

Процедура создания адаптера отправки данных требует полного и последовательного переименования и замены. Любая ошибка может привести к сбою работы адаптера.

Необходимо также корректно проводить упаковку и распаковку пакетов (чтобы обеспечить их совместимость с UCMDB). В качестве примеров следует использовать стандартные пакеты. Типичные ошибки:

- Лишний директорий в ZIP-файле пакета.
  Решение: Производите упаковку ZIP-файла в той же директории, где находятся директории пакетов (**discoveryResources**, **adapterCode**). Не включайте в пакет директории верхнего уровня.
- Непереименованная директория, файл или строка в файле.
  Решение: Четко следуйте инструкциям, приведенным в данном разделе.
- Неверное написание имени директории, файла или строки в файле.
  Решение: Придерживайтесь одних и тех же правил именования. При необходимости изменить какое-либо имя, возвратитесь к началу процедуры, в противном случае возможны ошибки. Также, вместо изменения строк вручную, следует воспользоваться функцией поиска и замены.
- Развертывание адаптеров с дублирующимися именами, в особенности в директориях **discoveryResources** и **adapterCode**.
  Решение: Возможно, используется версия UCMDB, которая запрещает использовать для файла сопоставления то же имя, что и у другого адаптера в окружении UCMDB. При попытке развертывания пакета с дублирующимися именами произойдет сбой. Ошибка возможна, даже если файлы находятся в различных директориях. Не следует дублировать файлы как в рамках одного пакета, так и в рамках всех ранее развернутых пакетов.

На данном этапе возможно создание в Студии интеграции задания адаптера принудительной отправки, которое будет использовать только что развернутый адаптер.

Оптимальные методы создания TQL-запросов для адаптеров принудительной отправки данных

1. Создайте структуру папок в соответствии с древовидными структурами TQL-запросов и представлений для сохранения в ней всех новых TQL-запросов и представлений. Используйте правила именования.
2. Копируйте сперва наиболее схожие TQL-запросы (это правило может не распространяться на небольшие запросы).
3. Вносите изменения постепенно. Сохраняйте, тестируйте и просматривайте результат каждого изменения. Повторяйте описанные шаги до тех пор, пока требуемый результат не будет достигнут.

Создание сопоставлений

Необработанные результаты TQL-запроса соответствуют схеме модели класса UCMDB. В разных системах могут использоваться различные модели данных. Адаптер принудительной отправки данных обеспечивает механизм сопоставления, который позволяет преобразовывать данные в подходящий формат. При сопоставлении происходит прямое и комплексное преобразование — от прямого преобразования имен и типов до обработки данных о родительских и дочерних элементах и их связях.

Требования к сопоставлению изложены в разделе "Справочные сведения о файлах сопоставления" на странице 265. Воспользуйтесь справочными материалами при создании файла сопоставления.

Примечание. В файле свойств адаптера указывается имя файла сопоставления. Файлы конфигурации адаптера расположены в папке с именем, соответствующим имени адаптера. При внедрении адаптера следует переименовать эту папку для сохранения уникальности, необходимой Диспетчеру пакетов.

Создание файла сопоставления

1. Начните с файла сопоставления по умолчанию.
2. Разверните адаптер и запустите его один раз.
3. Проанализируйте полученные результаты.
4. Определите параметры, которые требуется изменить.
5. Внесите соответствующие изменения. При внесении изменений следует придерживаться следующего порядка.
   a. Начинают с верхнего раздела, который не относится к преобразованию данных. Рекомендуется запускать адаптер после каждого изменения.
   b. Затем следует заменить исходные ЭК на имена UCMDB в результатах TQL-запроса.
   c. В первую очередь выполняется сопоставление ключей.
   d. Далее добавление прямых сопоставлений.
   e. Затем добавляют комплексные сопоставления
   f. и сопоставления связей.

Шаги 2-5 повторяют до получения искомого файла. Для создание нового адаптера выберите пакет общего адаптера в качестве основы.

Принцип работы с файлами сопоставления одинаков для всех типов адаптеров принудительной отправки. Общий адаптер принудительной отправки данных в формате XML записывает результаты сопоставления в файл. Общий адаптер принудительной отправки данных веб-служб отправляет результаты в формате XML устройству приема. Дополнительные сведения см. в разделе "Общий адаптер принудительной отправки данных веб-служб" на странице 246.

Подготовка файлов сопоставления

Примечание. Чтобы извлечь ЭК и связи из CMDB без сопоставления, не создавайте файл **mappings.xml**. В этом случае будут возвращены все ЭК и связи со всеми атрибутами.

Существует два способа подготовки файлов сопоставления:

- Можно подготовить один глобальный файл сопоставления.

  Все сопоставления будут помещены в один файл с именем **mappings.xml**.

- Можно подготовить отдельный файл для каждого запроса принудительной отправки. Каждый файл сопоставления получит имя **<имя запроса>.xml**.

Дополнительные сведения см. в разделе "Схема файла сопоставления" на странице 267.

Эта задача включает следующие шаги:

- "Создание файла mappings.xml" ниже
- "Сопоставление ЭК" ниже
- "Сопоставление связей" на следующей странице

1. Создание файла mappings.xml

   Структура файла сопоставления будет иметь следующий вид (в качестве шаблона используйте существующий файл):

```
<?xml version="1.0" encoding="UTF-8"?>
<integration>
    <info>
        <source name="UCMDB"  versions="9.x" vendor="HP" >
        <!-- for example: -->
        <target name="Oracle" versions="11g" vendor="Oracle" >
    </info>
    <targetcis>
        <!--- CI Mappings --->
    </targetcis>
    <targetrelations>
        <!--- Link Mappings --->
    </ targetrelations>
</integration>
```

2. Сопоставление ЭК

   Есть два способа сопоставления типов ЭК из CMDB:

   - Сопоставление типа ЭК таким образом, чтобы были одинаково сопоставлены ЭК данного типа и всех наследованных типов.

```
<source_ci_type_tree name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey>
            <pkey>name</pkey>
        </targetprimarykey
        <target_attribute name=" name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
        <!-- more target attributes --->
    </target_ci_type>
</source_ci_type_tree>
```

- Сопоставление типа ЭК таким образом, чтобы обрабатывались только ЭК данного типа. ЭК наследуемых типов не обрабатываются, если только данный тип не был сопоставлен отдельно (одним из описанных способов):

```
<source_ci_type name="node" mode="update_else_insert">
    <apioutputseq>1</apioutputseq>
    <target_ci_type  name="host">
        <targetprimarykey>
            <pkey>name</pkey>
        </targetprimarykey
        <target_attribute name=" name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
        <!-- more target attributes --->
    </target_ci_type>
</source_ci_type>
```

Тип ЭК, который сопоставляется косвенно (один из его предков сопоставлен с помощью **source_ci_type_tree**), также может переопределить сопоставление своего родительского типа, если тот будет указан в **source_ci_type_tree** или **source_ci_type**.

По возможности рекомендуется использовать **source_ci_type_tree**. В противном случае итоговые ЭК типов, не включенных в файлы сопоставления, не будут переданы в сценарий Jython.

3. Сопоставление связей

Существует два способа сопоставления связей:

- Сопоставление связи таким образом, чтобы были одинаково сопоставлены связи данного типа и всех наследуемых связей.

```
<source_link_type_tree name="dependency" target_link_type="dependency"
mode="update_else_insert" source_ci_type_end1="webservice" source_ci_
type_end2="sap_gateway">
    <target_ci_type_end1 name="webservice" >
    <target_ci_type_end2 name="sap_gateway" >
        <target_attribute name="name" datatype="STRING">
            <map type="direct" source_attribute="name" >
        </target_attribute>
</source_link_type_tree>
```

- Сопоставление связи таким образом, чтобы обрабатывались связи данного типа. Связи наследуемых типов не обрабатываются, если только данный тип не был сопоставлен отдельно (одним из описанных способов):

```
<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>
```

Создание сценариев Jython

Сценарий сопоставления — это обычный сценарий Jython, который должен соответствовать правилам сценариев Jython. Дополнительные сведения см. в разделе "Разработка адаптеров Jython" на странице 37.

Сценарий должен содержать функцию **DiscoveryMain**, которая может вернуть пустой экземпляр **OSHVResult** или экземпляр **DataPushResults** в случае успеха.

Для сообщения об ошибках сценарий должен создавать исключение, например:

```
raise Exception('Failed to insert to remote UCMDB using TopologyUpdateService. See log of the remote UCMDB')
```

В функции DiscoveryMain элементы данных, которые принудительно отправлены или удалены из внешнего приложения, можно получить следующим образом:

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

Объект клиента внешнего приложения можно получить следующим образом:

```
oracleClient = Framework.createClient()
```

Этот клиент автоматически применяет идентификатор учетных данных, имя хоста и номер порта, переданные адаптером с помощью Framework.

Чтобы применить параметры подключения, указанные для адаптера (подробнее см. процедуру изменения файла **discoveryPatterns\push_adapter.xml** в разделе "Создание пакета адаптера" на странице 233), используется следующий код:

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

Пример:

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

Этот раздел также содержит следующие подразделы.

- "Работа с результатами сопоставления" ниже
- "Обработка тестовых подключений в сценарии" на странице 243

Работа с результатами сопоставления

Адаптер принудительной отправки создает строки XML, которые описывают данные для

добавления, обновления или удаления из целевой системы. Сценарий Jython должен проанализировать эти строки XML, а затем выполнить операцию добавления, обновления или удаления в целевом объекте.

В строках XML операции добавления, полученных сценарием Jython, атрибут `mamId` объектов и связей всегда представляет собой идентификатор исходного объекта или связи в UCMDB до изменения их типа, атрибута или другой информации в схеме на удаленной системе.

В строках XML операций обновления или удаления атрибут `mamId` объектов и связей содержит представление строки `ExternalId`, возвращенной из сценария Jython при предыдущей синхронизации.

В строках XML атрибут ЭК `id` содержит `cmdbId` в качестве внешнего идентификатора или `ExternalId` данного ЭК, если у ЭК в момент его отправки сценарию есть `ExternalId`. Поля связи `end1Id` и `end2Id` содержат для каждой из сторон связи `cmdbId` как внешний идентификатор или `ExternalId` данной стороны связи, если у ЭК на стороне связи есть `ExternalId` в момент ее отправки сценарию.

При обработке ЭК сценарий Jython возвращает сопоставление между ID ЭК в CMDB и данным ID (идентификатором, присвоенным каждому из ЭК в сценарии). Если ЭК передается впервые, в XML-коде ЭК указывается его идентификатор в CMDB. Если же ЭК передается не первый раз, указывается идентификатор, присвоенный ему сценарием при первой передаче.

Идентификатор извлекается из XML-кода ЭК следующим образом:

1. извлекается значение атрибута id из элемента ЭК в XML-файле. Пример: `id = objectElement.getAttributeValue('id')`.
2. После извлечения ID из XML идентификатор восстанавливается из атрибута (строки). Пример: `objectId = CmdbObjectID.Factory.restoreObjectID(id)`.
3. Проверяется, является ли `objectId`, полученный в предыдущем шаге, идентификатором из CMDB. Для этого можно проверить, есть ли у `objectId` новый идентификатор, присвоенный сценарием. Если он есть, возвращенный ID не является идентификатором из CMDB. Например:
   `newId = objectId.getPropertyValue(<имя атрибута ID, присвоенного сценарием>)`.

   Если `newId` имеет значение null, идентификатор, возвращенный в XML, является идентификатором из CMDB.
4. Если ID является идентификатором из CMDB, (т.е., если `newId` имеет значение null), выполняются следующие действия (если ID не является идентификатором из CMDB, перейти к шагу 5):
   a. Для данного ЭК создается свойство, в котором хранится новый идентификатор. Например: `propArray =[TypesFactory.createProperty('<имя атрибута ID, присвоенного сценарием>', '<new id>')]`.
   b. Для ЭК создается `externalId`. Пример:
   ```
   cmdbId = extI.getPropertyValue('internal_id')
   className = extI.getType()
   externalId = ExternalIdFactory.createExternalCiId(className, propArray)
   ```
   c. Выполняется сопоставление ID из CMDB с новым `externalId` (в следующем шаге это сопоставление возвращается адаптеру). Пример: `objectMappings.put(cmdbId,`

```
externalId)
```

d. Когда сопоставление всех ЭК и связей выполнено:

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

5. Если идентификатор является новым (т.е. значение `newId` не равно null), `externalId` является `newId`.

Получение ссылки и отчета о принудительной отправке данных по каждому ЭК осуществляется при помощи следующей строки:

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

   где `updateStatus` — экземпляр класса `UpdateStatus`, который содержит статусы ЭК и ссылки.

2. Статус добавляется в `updateStatus` методом `reportCIStatus` или `reportRelationStatus`.

   Пример.

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

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

   где `ERROR_CODE_CI` — порядковый номер сообщения об ошибке в файле **properties.errors** (подробные сведения о файле **properties.errors** см. в разделе "Правила сообщений об ошибках" на странице 65), а `errorParams` содержит параметры для передачи. Подробнее см. javadoc **ReplicationActionDataFactory** .

3. Пример результата принудительной отправки с указанием статуса:

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

   return updateResult
   ```

Пример результата XML

```
<root>

  <data>

    <objects>

      <Object mode="update_else_insert" name="UCMDB_UNIX" operation="add"
mamId="0c82f591bc3a584121b0b85efd90b174"
id="HiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A">

        <field name="NAME" key="false" datatype="char" length="255">UNIX5</field>

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

      </Object>
```

```
    </objects>

    <links>

        <link targetRelationshipClass="TALK" targetParent="unix" targetChild="unix"
operation="add" mode="update_else_insert"
mamId="265e985c6ec51a8543f461b30fa58f81"
id="end1id%5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D41372a1cbcaba27b214b84a2ec9eb535%0A%5D%0Aend2id%
5BHiddenRmiDataSource%0Aunix%0A1%0Ainternal_
id%3DSTRING%3D0c82f591bc3a584121b0b85efd90b174%0A%5D%0AHiddenRmi
DataSource%0Atalk%0A1%0Ainternal_
id%3DSTRING%3D265e985c6ec51a8543f461b30fa58f81%0A">

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

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

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

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

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

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

        </link>

    </links>

  </data>

</root>
```

Примечание. Если `datatype="BYTE"`, возвращается результат типа **String**, который создается следующим образом: `new String([the byte array attribute])`. Объект `byte[]` можно восстановить следующим образом: `<the received String>.getBytes()`. Если региональные настройки сервера и зонда по умолчанию различаются, восстановление происходит согласно настройкам сервера по умолчанию.

Обработка тестовых подключений в сценарии

Сценарий Jython можно вызвать для тестирования подключения к внешнему приложению. В этом случае целевой атрибут `testConnection` будет иметь значение `true`. Атрибут можно получить у Framework следующим образом:

```
testConnection = Framework.getTriggerCIData('testConnection')
```

При выполнении в тестовом режиме сценарий должен создать исключение, если установка подключения невозможна. Если подключение устанавливается успешно, функция **DiscoveryMain** должна вернуть пустой экземпляр **OSHVResult**.

Поддержка разностной синхронизации

Чтобы адаптер принудительной отправки поддерживал разностную синхронизацию, функция **DiscoveryMain** должна вернуть объект, реализующий интерфейс **DataPushResults**, который содержит сопоставление идентификаторов, полученных сценарием Jython из XML, и идентификаторов, созданных сценарием Jython на удаленном компьютере. Идентификаторы на удаленном компьютере имеют тип **ExternalId**.

Команда **ExternalIdUtil.restoreExternal**, которая принимает идентификатор ЭК в CMDB в качестве параметра, восстанавливает значение внешнего ID по ID ЭК в CMDB. Данную команду можно использовать, например, при выполнении разностной синхронизации, если одна из сторон принятой связи не входит в пакет (уже была синхронизирована ранее).

Если метод **DiscoveryMain** в сценарии Jython, на котором основывается сценарий принудительной отправки, возвращает пустой экземпляр **ObjectStateHolderVector**, адаптер не поддерживает разностную синхронизацию. Это значит, что даже при выполнении задания разностной синхронизации, фактически выполняется полная синхронизация. Поэтому данные не могут быть обновлены или удалены в удаленной системе, поскольку при каждой синхронизации в CMDB добавляются все данные.

Внимание! При реализации разностной синхронизации для существующего адаптера, созданного в версиях 9.00 и 9.01, используйте файл push-adapter.zip из версии 9.02 или более поздней для восстановления пакета адаптера. Дополнительные сведения см. в разделе "Создание пакета адаптера" на странице 233.

Эта задача позволяет адаптеру принудительной отправки выполнять разностную синхронизацию.

Сценарий Jython возвращает объект **DataPushResults**, который содержит два сопоставления Java, — одно для сопоставлений идентификаторов объектов (ключи и значения — объекты типа ExternalCiId) и второе для идентификаторов связей (ключи и значения являются объектами типа ExternalRelationId).

- Добавьте следующие инструкции **from** в сценарий Jython:

```
from com.hp.ucmdb.federationspi.data.query.types import ExternalIdFactory
from com.hp.ucmdb.adapters.push import DataPushResults
from com.hp.ucmdb.adapters.push import DataPushResultsFactory
from com.mercury.topaz.cmdb.server.fcmdb.spi.data.query.types import
ExternalIdUtil
```

- Используйте класс фабрики **DataPushResultsFactory** для получения объекта **DataPushResults** из функции **DiscoveryMain**.

```
# Create the UpdateResult object
updateResult = DataPushResultsFactory.createDataPushResults(objectMappings,
linkMappings);
```

- Используйте следующие команды, чтобы создать сопоставления Java для объекта **DataPushResults**:

```
# Prepare the maps to store the mappings if IDs
objectMappings = HashMap()
linkMappings = HashMap()
```

- Используйте класс **ExternalIdFactory** для создания следующих внешних идентификаторов:
  - Значения ExternalId для объектов и связей, полученных из CMDB (например, все ЭК в операции добавления происходят из CMDB):

```
externaCIlId = ExternalIdFactory.createExternalCmdbCiId(ciType, ciIDAsString)
externalRelationId = ExternalIdFactory.createExternalCmdbRelationId(linkType,
end1ExternalCIId,
end2ExternalCIId, linkIDAsString)
```

  - Значения ExternalId для объектов из связей, полученных не из CMDB (как правило, все операции обновления и удаления содержат такие объекты):

```
myIDField = TypesFactory.createProperty("systemID", "1")
myExternalId = ExternalIdFactory.createExternalCiId(type, myIDField)
```

> **Примечание.** Если сценарий Jython обновил существующие сведения и идентификатор объекта (или связи) меняется, необходимо вернуть сопоставление между предыдущим внешним идентификатором и новым идентификатором.

- Используйте методы **restoreCmdbCiIDString** или **restoreCmdbRelationIDString** из класса **ExternalIdFactory** для получения строки идентификатора UCMDB из внешнего идентификатора объекта или связи, полученного из UCMDB.
- Используйте методы **restoreExternalCiId** и **restoreExternalRelationId** из класса **ExternalIdUtil** для восстановления объекта **ExternalId** из значения атрибута mamId строки XML операций обновления или удаления.

> **Примечание.** Объекты **ExternalId** фактически представляют собой массив свойств. Это значит, что объект **ExternalId** можно использовать для хранения любой информации, которая может потребоваться для идентификации данных в удаленной системе.

SQL-запросы общего адаптера принудительной отправки в формате XML

Файл **sql_queries** в пакете адаптера, расположенный в разделе **adapterCode > PushAdapter > sqlTablesCreation**, содержит запросы, необходимые при создании таблиц в новой схеме Oracle для тестирования адаптера. Таблицы соответствуют файлу adapterCode\<ID адаптера>\mappings\mappings.xml.

Примечание. Файл **sql_queries** адаптеру не требуется. Он служит только для примера.

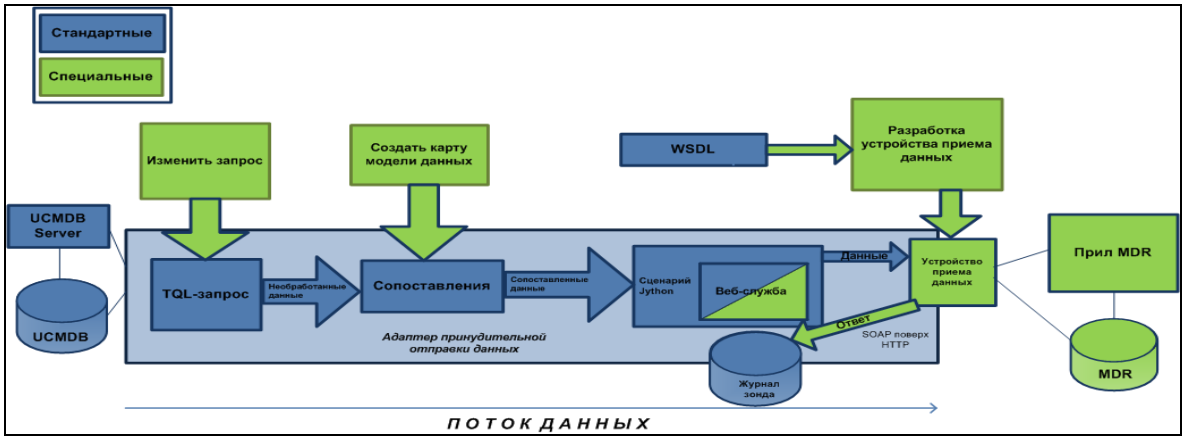
Общий адаптер принудительной отправки данных веб-служб

Общий адаптер принудительной отправки данных веб-служб обеспечивает (по триггеру UCMDB) отправку сообщений SOAP с данными запроса к устройству приема данных веб-службы. Сопоставленные результаты отправляются устройству приема данных в виде стандартного сообщения SOAP через протокол HTTP POST. Необходимо, чтобы устройство приема данных воспринимало сообщения SOAP, отправленные адаптером. Для простоты разработки таких устройств приема адаптер снабжен WSDL-файлом.

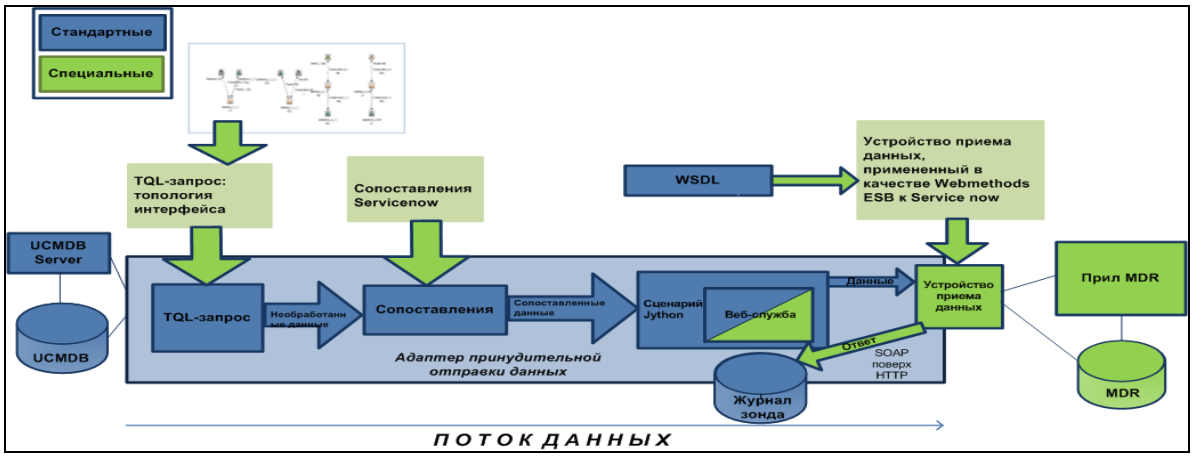
Обработку XML-формата сообщений SOAP можно настроить при помощи сценария Jython.

Для обеспечения соответствия формата входящих данных необходимо взаимодействие разработчика устройства приема и разработчика файлов сопоставления. В данной версии адаптера принудительной отправки данных веб-служб отсутствует файл **.xsd**, поэтому при обработке данных требуется учитывать формат получаемых данных, которые представляют собой комбинацию исходного TQL-запроса и соответствующих сопоставлений.

Функции адаптера принудительной отправки данных веб-служб клиенту приведены ниже. Зеленым отмечены настраиваемые компоненты или компоненты, которые входят в комплект клиента для работы с определенными устройствами приема. Синим отмечены стандартные компоненты.

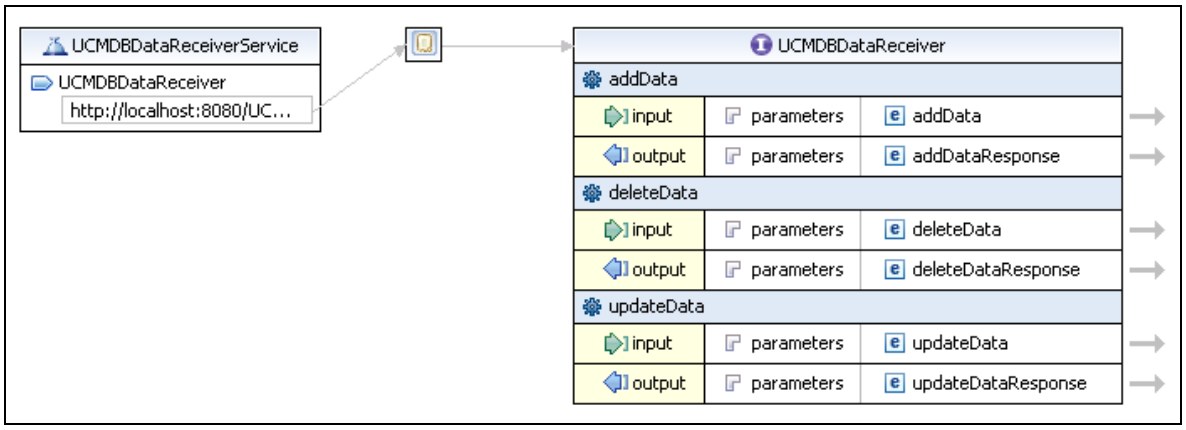


Ниже приведен пример использования общего адаптера принудительной отправки данных веб-служб в качестве основы для адаптера MDR, работающего на базе сервисной шины предприятия (ESB):



WSDL-файл

WSDL-файл необходим при разработке клиента для обеспечения связи между устройством приема данных и адаптера принудительной отправки данных UCMDB посредством веб-служб. **UCMDBDataReceiver.wsdl** содержит описание сообщений SOAP, которые будут использоваться для передачи данных UCDMB устройству приема. Ниже приведена схема WSDL:



Устройство приема данных (обычно это сервер или “конечная точка службы" в терминах SOAP) использует три метода: **addData**, **deleteData** и **updateData**, соответствующие наборам данных UCMDB. Заголовки HTTP содержат ключевое слово **SoapAction**, которое указывает на тип передаваемых данных. Устройство приема данных отвечает за применение бизнес-логики и обработку данных.

URL-адрес WSDL по умолчанию:

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

После обработки устройством приема данных URL-адрес будет иметь следующий вид:

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

URL-адрес веб-службы соответствует URL-адресу WSDL без добавления “?wsdl” в конце.

Ниже приведен исходный код WSDL:

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

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

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

    <wsdl:types>

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

            <element name="addData">

                <complexType>

                    <sequence>

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

                    </sequence>

                </complexType>

            </element>

            <element name="addDataResponse">

                <complexType/>

            </element>

            <element name="deleteData">

                <complexType>

                    <sequence>

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

                    </sequence>

                </complexType>

            </element>

            <element name="deleteDataResponse">

                <complexType/>

            </element>

            <element name="updateData">
```

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

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

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

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

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

    <wsdl:operation name="addData">

        <wsdl:input name="addDataRequest">

            <wsdlsoap:body use="literal" />

        </wsdl:input>

        <wsdl:output name="addDataResponse">

            <wsdlsoap:body use="literal" />

        </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="deleteData">

        <wsdl:input name="deleteDataRequest">

            <wsdlsoap:body use="literal" />

        </wsdl:input>

        <wsdl:output name="deleteDataResponse">

            <wsdlsoap:body use="literal" />

        </wsdl:output>

    </wsdl:operation>

    <wsdl:operation name="updateData">

        <wsdl:input name="updateDataRequest">

            <wsdlsoap:body use="literal" />

        </wsdl:input>

        <wsdl:output name="updateDataResponse">

            <wsdlsoap:body use="literal" />

        </wsdl:output>

    </wsdl:operation>

</wsdl:binding>

<wsdl:service name="UCMDBDataReceiverService">

    <wsdl:port binding="impl:UCMDBDataReceiverSoapBinding"
    name="UCMDBDataReceiver">
```

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

        </wsdl:port>

    </wsdl:service>

</wsdl:definitions>
```

Обработка ответа

Устройство приема данных возвращает строку в виде структуры **addDataResponse**, **deleteDataResponse** или **updateDataResponse**. Адаптер передает необработанные данные в журнал зонда **probeMgr-adaptersDebug.log**. Устройство приема возвращает строковые данные любого вида. Ответ имеет формат XML SOAP сообщений. Для анализа сообщений в рамках сценария Jython можно использовать **SOAPMessage** и соответствующие классы Java. Ниже приведен пример ответа, полученного от устройства приема данных:

```
<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>
```

Это сообщение является сообщением об ошибке **<Error publishing config item changes>**. Фактически содержание сообщения может быть любым — в зависимости от целей создания адаптера. В данном случае адаптер ожидает сообщений со сведениями об успешном выполнении или сбое. Ответ может включать идентификаторы выверки успешно добавленных ЭК или сообщения об ошибках при их добавлении. Процесс настройки GWSPA может включать анализ ответа и соответствующие действия (повторная отправка ЭК или запись в журнал).

Тестирование WSDL:

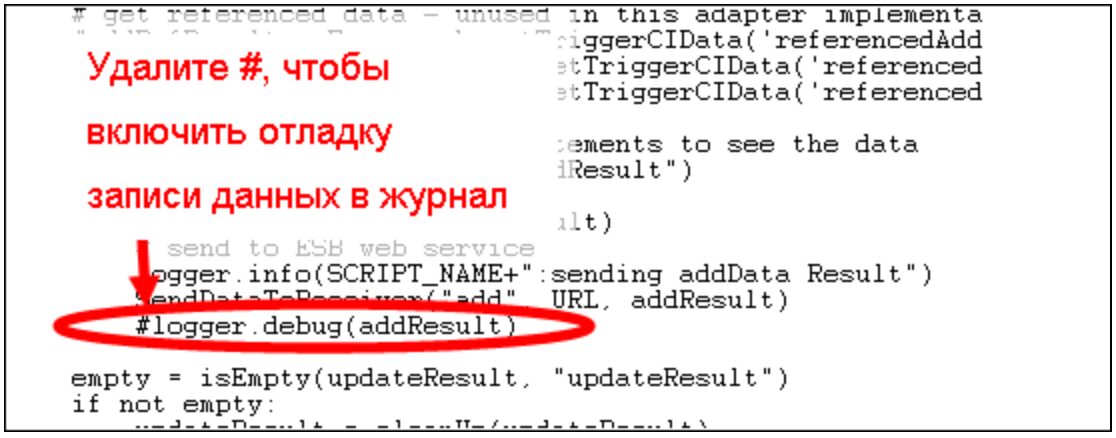
Для тестирования слоев веб-служб в ходе разработки используется подключаемый модуль SOAPUI Eclipse. SOAPUI позволяет настраивать работу веб-служб. SOAPUI использует встроенную среду разработки (IDE) для создания, отправки и получения сообщений SOAP. В рамках SOAPUI WSDL (исходный код которого приведен на стр. 248-252) создает сообщения следующего вида:

```
<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>
```

“**?**” в строке с элементом **xmlAdded** указывает местоположение данных, полученных в результате интеграции веб-службы и адаптера принудительной отправки данных.

Полученные результаты

При нормальной работе адаптера принудительной отправки данных (вне режима отладки) данные не записываются в файл до тех пор, пока не будет получен окончательный результат (промежуточные результаты TQL-запросов и сопоставления данных обычно не отражаются в файлах журналов). Для сохранения этих результатов необходимо удалить символы комментирования (“#”) в файле отладки зонда в строке **logger.debug** в разделе DiscoveryMain (см. ниже).



Необходимо, чтобы выражение "logger" имело отступ, соответствующий предшествующей и последующей строкам. Формат Jython требует соблюдения отступов, в противном случае произойдет сбой сценария.

Содержимое файла журнала отладки **probeMgr-adaptersDebug.log** выглядит следующим образом:

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

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

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

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

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

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

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

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

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

<root>

    <data>

        <objects>

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

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

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

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

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

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

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

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

            </Object>
```

Изменение сценария Jython

XMLtoWebService.py

Сценарий Jython, используемый адаптером принудительной отправки данных, похож на адаптер отправки данных в формате XML. Сценарий использует файл **UCMDBDataReceiver.jar**, который входит в комплект адаптера. В сценарии используется метод **SendDataToReceiver()**. **SendDataToReceiver()** содержит три параметра:

1. Действие (добавить, обновить, удалить)
2. URL-адрес устройства приема данных
3. Данные

Пример добавления: **SendDataToReceiver("add", URL, addResult)**

Слои веб-служб и SOAP подлежат преобразованию. URL-адрес — это адрес конечной точки службы устройства приема данных UCMDB. Он соответствует URL-адресу получения wsdl с добавлением "?wsdl".

Ниже приведен исходный код сценария Jython. Строки оболочки интеграции веб-служб выделены зеленым цветом.

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

# script: XMLtoWebService.py

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

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

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

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

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

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

# regular expressions

import re

# logging

import logger

# web service interface

from com.hp.ucmdb import SendDataToReceiver

from com.hp.ucmdb.SendDataToReceiver import locateService

from com.hp.ucmdb.SendDataToReceiver import SendData

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

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

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

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

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

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

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

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

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

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

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

    return str
def isEmpty(xml, type = ""):
```

```
    objectsEmpty = 0
    linksEmpty = 0

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

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

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

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

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

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

# get referenced data - unused in this adapter implementation
```

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


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

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

Настройка обработки ответных сообщений

Устройство приема данных возвращает строку с необходимым ответом или статусом. По умолчанию адаптер принудительной отправки данных веб-служб передает эти данные в журнал зонда. Ответное сообщение в формате XML/SOAP содержит строку (строки) ответа. Устройство получения данных может отправлять разнообразные сообщения (сведения об ошибках или об успешном выполнении). Если требуется дополнительная обработка, ответное сообщение может обрабатываться при помощи сценария Jython. Использование Java не требуется.

Пример ответа, полученного от устройства приема данных:

```
// 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;
}
```

приведен далее:

```
<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>
```

Изменение устройства приема данных

Как и клиент Jython, клиент Java может применять классы файла **UCMDBDataReceiver.jar** для вызова веб-служб. Кроме того, возможно использование методов без преобразования. Для классов **UCMDBDataReceiver.jar** используется Javadoc. Ниже приведен исходный код с примерами основных методов преобразования данных в сообщение SOAP и передачи по HTTP.

Процедура состоит в создании объекта **UCMDBDataReceiverServiceLocator**и назначении параметру **UCMDBDataReceiverEndPointAddress** URL-адреса устройства передачи данных.

Для отправки данных вызывается метод **getUCMDBDataReceiver**, который создает объект **UCMDBDataReceiver**. Объект **UCMDBDataReceiver** применяет методы для фактической отправки команды добавить/изменить /удалить данные. Для обработки каждого типа запроса используется один из трех фрагментов кода.

Ниже приведен исходный код для класса **SendDataToReceiver**. Цветом выделены основные используемые объекты и методы.

```
/**
* Test SendData for the UCMDB Data Receiver for the UCMDB Web Service Push
Adapter
*/
package com.hp.ucmdb;
import com.hp.ucmdb.SendDataToReceiver;
/**
* TestSendData can be used to verify the SOAP classes are working.
* TestSendData creates a SendDataToReceiver class and invokes its SendData
method.
* a response String is returned.
* The test URL is typically appended with "?wsdl" to get the WSDL of the
service.
*/
public class TestSendData {
    /**
    * @param args - test SOAP message.
    *  optional arguments [0] a test string [1] a service endpoint URL of a
    Data Receiver.
    * the default URL is sent the incoming argument as a test message.
    * the default URL is
    "http://localhost:8080/UCMDBDataReceiver/services/UCMDBDataReceiver".
    * If any errors are encountered, TestClient will attempt to throw
    exceptions.
    */
    public static void main(String[] args) {
        // use test message if supplied, otherwise supply a default test string
```

```
        String teststring = new String("Test SOAP message from
        UCMDBDataReceiver TestSendData.");
        if(args.length > 0) {
            teststring = args[0];
        }
        // use test URL if supplied, otherwise supply the default URL
        String URL = new String("");
        if(args.length > 1) {
            URL = args[1];
        }
        // return response
        String response = new String("");
        // perform the tests
        try {
            if(URL.equals("")) {
                UCMDBDataReceiverServiceLocator locator = new
                UCMDBDataReceiverServiceLocator();
                UCMDBDataReceiver receiver = locator.getUCMDBDataReceiver();
                URL = locator.getUCMDBDataReceiverAddress();
                System.out.println("TestClient: tested
                URL="+locator.getUCMDBDataReceiverAddress());
                System.out.println("TestClient: receiver="+receiver.toString
                ());
            }
            SendDataToReceiver sdtr = new SendDataToReceiver();
            // this sends a test push and gets a response message
            response = sdtr.SendData("add", URL, args[0]);
            System.out.println("Response received was:"+response);
        } catch (Exception e) {
            System.out.println("TestClient: Remote Error:");
            e.printStackTrace();
        }
    }
```

```
}
```

Исходный код для других классов также включен в файл **UCMDBDataReceiver.jar** :

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

Код был создан в Eclipse IDE, а затем преобразован. При изменении кода UCMDB следует соблюдать осторожность, поскольку основная его часть создается автоматически для обеспечения соответствия формату SOAP и формату устройства получения данных UCMDB.

Javadoc

Javadoc со всеми комментариями входит в комплект общего адаптера принудительной отправки данных веб-служб. Он находится в папке документов **javadoc**. Основным файлом является **index.html**. Обзорная страница предоставляет доступ к документам по всем класса и методам в SDK.

Все классы

- **SendDataToReceiver:** API для оболочки веб-службы
- **TestClient:** проверка подключения клиента к конечной точке службы
- **UCMDBDataReceiver:** оболочка веб-службы

Остальные классы создаются конструктором веб-служб автоматически:

- UCMDBDataReceiverProxy
- UCMDBDataReceiverService
- UCMDBDataReceiverServiceLocator
- UCMDBDataReceiverSoapBindingStub

Обзор

Методы работы с SDK, а также примеры исходного кода включены в документацию к пакету. Данный **javadoc** предназначен для адаптера принудительной отправки данных веб-служб UCMDB. Вызов API может осуществляться Jython или Java.

SDK содержит два примера исходного кода: **TestClient** и **SendDataToReceiver**. **TestClient** используется для ограниченного тестирования связи с локальным клиентом. **SendDataToReceiver** — основной класс отправки данных веб-службам.

Сначала SDK (в основном WSDL) используется для обеспечения связи между устройством приема данных UCMDB и веб-службой. Затем создается адаптер для принудительной отправки результатов TQL-запроса UCMDB устройству приема. Основные методы использования API приведены ниже (для Jython и Java).

Применение SendDataToReceiver()

SendDataToReceiver() преобразовывает все функции в единый метод:

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

Альтернативный вариант: создается объект **SendDataToReceiver** (например, для управления другими параметрами), а затем отдельно вызывается метод **SendData**:

- Jython:

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

- Java:

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

Вариант пошагового выполнения процедуры:

1. Создается объект x **UCMDBDataReceiverServiceLocator()**, а затем указывается адрес конечной точки объекта:
   - **Jython:**

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

   - **Java :**

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

2. Затем создается UCMDBDataReceiver
   - **Jython:** `y = x.getUCMDBDataReceiver()`
   - **Java:** `UCMDBDataReceiver y = x.getUCMDBDataReceiver();`
3. Далее данные отправляются веб-службой SOAP:
   - **Jython:**
     - `y.addData(yourData)`
     - or `y.updateData(yourData)`
     - or `y.deleteData(yourData)`
   - **Java:**
     - `y.addData(yourData);`
     - or `y.updateData(yourData);`
     - or `y.deleteData(yourData);`
4. Может потребоваться проверка подключения. При успешном подключении этот же объект используется для возврата **UCMDBDataReceiver** в целях передачи данных.

Классы не содержат деструкторов и не обеспечивают управление памятью.

Справочные сведения о файлах сопоставления

Использование сопоставлений

Для каждого целевого атрибута в преобразованном XML-файле выходных данных необходимо создать соответствующий метод сопоставления. Эти методы указывают место и способ получения данных. Если данные находятся в другом соответствующем атрибуте UCMDB, используется прямое сопоставление.

Для получения данных из нескольких атрибутов (или при наличии разноуровневых связей) могут потребоваться более сложные методы сопоставления. Ниже приведена схема со всеми возможными вариантами сопоставления.

Файл сопоставления - это XML-файл, содержащий определения типов ЭК/связей в UCMDB, которые сопоставляются с типами ЭК/связей в хранилище данных. Ниже приведены подробные требования к формату файла. Файл сопоставления определяет типы ЭК и связей, а также атрибуты, подлежащие принудительной отправке.

Запись сопоставления каждого атрибута направляется в регистр данных. Каждая запись может содержать один или несколько атрибутов необработанных данных UCMDB. Записи сопоставления обеспечивают полноценный контроль конечной структуры и именования данных перед их отправкой в регистр.

Прямое сопоставление

Преобразование одной модели данных в другую (в данном случае модели UCMDB в модель регистра данных). При одноуровневых связях между атрибутами UCMDB и атрибутами целевого объекта (которые различаются только именем и, возможно, типом) используется простое преобразование.

В большинстве случаев используется прямое сопоставление. Например, сервер “ServerX” представлен в базе UCMDB в виде ЭК с типом **unix** и атрибутом **primary_server_name** типа **string** длиной не более 50. Модель целевого регистра данных содержит такую же логическую единицу с ЭК типа **linux** и атрибутом **hostname** типа **char[]** с длиной не более 250. В этом случае достаточно прямого сопоставления.

Ниже приведен пример прямого сопоставления:

```
 <target_attribute name="dns_domain" datatype="char">

<map type="direct" source_attribute="domain_name" />

</target_attribute>
```

При прямом сопоставлении происходит создание связи между атрибутом UCMDB **dns_ domain** и атрибутом целевой модели данных **domain_name**.

Независимо от фактического типа данных следует использовать тип **char**, если иное прямо не указано.

Комплексное сопоставление

При комплексном сопоставлении происходит дополнительное преобразование.

- Сопоставление значений атрибутов нескольких ЭК с одним целевым ЭК.
- Сопоставление дочерних ЭК (которые содержат связь **container_f** или contained) с родительским ЭК в целевом хранилище данных. Например, указание значения под именем **Number of CPUs** для целевого ЭК Host. Либо передача значения **Total Memory** (получаемого сложением значений объемов памяти все ЭК Memory и ЭК Host в UCMDB) целевому ЭК Host.
- Сопоставление атрибутов родительских ЭК (которые содержат связь **container_f** или contained) с ЭК в целевом хранилище данных. Например, передача значения **Container Server** целевому атрибуту **Installed Software** через ЭК-контейнер Host, который содержит ЭК Software в UCMDB.

Ниже приведен пример комплексного сопоставления двух исходных атрибутов, разделенных запятой, с целевым атрибутом **os**":

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

Отслеживание направления связи.

В зависимости от источника, данные в UCMDB могут иметь различную структуру. Например, между ЭК IpAddress и ЭК Interface может существовать связь **parent** (в рамках HP Network Node Manager integration). Также между ними может существовать связь **containment** (в рамках Universal Discovery). При этом указанные связи могут иметь противоположные направления.

В настоящее время определить направление связи при помощи файла сопоставления невозможно. При перестановке переменных **_end1** и **_end2** происходит изменение порядка данных в преобразованном XML-файле или потеря связи в исходных данных.

Одним из методов решения этой проблемы является определение правила расширения:

1. TQL-запрос расширения представляет собой блок запросов, используемый адаптером принудительной отправки данных. Этот запрос собирает сведения о связях, направления которых не соответствуют указанным в преобразованном XML-файле.
2. При помощи расширения определяется новая связь с требуемым направлением и типом.
3. Создание связей происходит после активации расширения.

4. TQL-запросы задания интеграции при этом будут использовать расширенные связи вместо исходных.
5. При сопоставлении <связей> адаптер принудительной отправки данных использует расширенные связи и создает набор связей с соответствующими типами и направлением.

Схема файла сопоставления

Имя элемента и путь	Описание	Атрибуты
integration	Определяет содержимое файла сопоставления. Это должен быть первый блок файла, не считая начальной строки и комментариев.	
info (integration)	Информация об интегрируемых репозиториях.	
source (integration > info)	Информация об исходном репозитории.	1. **Имя:** type **Описание:** Имя исходного репозитория. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** versions **Описание:** Версии исходных репозиториев. **Обязательно?** Обязательно **Тип:** Строка 3. **Имя:** vendor **Описание:** Поставщик исходного репозитория. **Обязательно?** Обязательно **Тип:** Строка
target (integration > info)	Информация о целевом репозитории.	1. **Имя:** type **Описание:** Имя исходного репозитория. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** versions **Описание:** Версии исходного

Имя элемента и путь	Описание	Атрибуты
		репозитория. **Обязательно?** Обязательно **Тип:** Строка 3. **Имя:** vendor **Описание:** Поставщик исходного репозитория. **Обязательно?** Обязательно **Тип:** Строка
targetcis (integration)	Элемент-контейнер для всех сопоставлений типов ЭК.	
source_ci_type_tree (integration > targetcis)	Исходный тип ЭК и все наследующие от него типы.	1. **Имя:** name **Описание:** Имя исходного типа ЭК. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** mode **Описание:** Тип обновления для текущего типа ЭК. **Обязательно?** Обязательно **Тип:** Одна из следующих строк: a. **insert:** Используется, только если ЭК не существует. b. **update:** Используется, только если ЭК существует. c. **update_else_insert:** Если ЭК существует, он обновляется, в противном случае создается новый ЭК. d. **ignore:** Пропустить этот тип ЭК.
source_ci_type (integration > targetcis)	Исходный тип ЭК без типов-наследников.	1. **Имя:** name **Описание:** Имя исходного типа ЭК. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** mode **Описание:** Тип обновления для текущего типа ЭК. **Обязательно?** Обязательно **Тип:** Одна из следующих строк: a. **insert:** Используется, только если ЭК не существует. b. **update:** Используется, только если ЭК существует.

Имя элемента и путь	Описание	Атрибуты
		c. **update_else_insert:** Если ЭК существует, он обновляется, в противном случае создается новый ЭК. d. **ignore:** Пропустить этот тип ЭК.
target_ci_type (integration > targetcis > source_ci_type -OR- integration > targetcis > source_ci_type_tree)	Целевой тип ЭК.	1. **Имя:** name **Описание:** Имя целевого типа ЭК. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** schema **Описание:** Имя схемы, которая будет использоваться для хранения этого типа ЭК в целевой системе. **Обязательно?**Необязательно **Тип:** Строка 3. **Имя:** namespace **Описание:** Пространство имен типа ЭК на целевом объекте **Обязательно?**Необязательно **Тип:** Строка
targetprimarykey (integration > targetcis > source_ci_type) -OR- (integration > targetcis > source_ci_type_tree -OR- (integration > targetrelations > link) -OR- (integration > targetrelations > source_link_type_tree)	Основные ключевые атрибуты целевого типа ЭК.	
pkey (integration > targetcis> source_ci_type > targetprimarykey -OR-	Один первичный ключевой атрибут. Требуется только в режиме **update** или **insert_else_update**.	

Имя элемента и путь	Описание	Атрибуты
integration > targetcis > source_ci_type_tree > targetprimarykey -OR- (integration > targetrelations > link > targetprimarykey) -OR- integration > targetrelations > source_link_type_tree > targetprimarykey)		
target_attribute (integration > targetcis> source_ci_ type -OR- integration > targetcis > source_ci_type_tree -OR- integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)	Атрибут целевого типа ЭК.	1. **Имя:** name **Описание:** Имя атрибута целевого типа ЭК. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** datatype **Описание:** Тип данных атрибута целевого типа ЭК. **Обязательно?** Обязательно **Тип:** Строка 3. **Имя:** length **Описание:** Для типов данных string и char размер целевого атрибута (целое число). **Обязательно?**Необязательно **Тип.** Целое число 4. **Имя.** option **Описание.** Функция преобразования для применения к значению. **Обязательно?** Нет **Тип.** Одна из следующих строк: a. **uppercase** — преобразование в верхний регистр b. **lowercase** — преобразование в нижний регистр Если этот атрибут пуст, функция преобразования применяться не будет.

Имя элемента и путь	Описание	Атрибуты
map (integration > targetcis > source_ci_type > target_attribute -OR- integration > targetcis > source_ci_type_tree > target_attribute) -OR- (integration > targetrelations > link > target_attribute -OR- integration > targetrelations > source_link_type_tree > target_attribute)	Способ получения значения атрибута исходного типа ЭК.	1. **Имя.** type **Описание.** Тип сопоставления между исходным и целевым значениями. **Обязательно?** Обязательно **Тип.** Одна из следующих строк: a. **direct** — сопоставление исходного атрибута со значением целевого атрибута один к одному. b. **compoundstring** — подэлементы объединяются в одну строку, устанавливается значение целевого атрибута c. **childattr** — подэлементы являются одним или несколькими значениями дочерних атрибутов типов ЭК. Дочерние типы ЭК имеют связь **composition** или **containment**. d. **constant** — статическая строка. 2. **Имя.** value **Описание.** Строка постоянной для type=**constant** **Обязательно?** Обязательно, только при использовании type=**constant** **Тип.** Строка 3. **Имя.** attr **Описание.** Имя исходного атрибута для type=**direct** **Обязательно?** Обязательно, только при использовании type=**direct** **Тип.** Строка
aggregation (integration > targetcis > source_ci_type > target_attribute > map -OR- integration > targetcis > source_ci_type_tree > target_attribute > map -OR-	Указывает, как значения дочерних атрибутов исходного ЭК объединяются в одно значение для сопоставления с целевым атрибутом ЭК. Необязательно.	**Имя:** type **Описание.** Тип функции сведения. **Обязательно?** Обязательно **Тип.** Одна из следующих строк: • **csv** — объединяет все включенные значения в список с разделителем-запятой (числа или строки/символы). • **count** — возвращает число включенных значений. • **sum –** Возвращает сумму всех включенных числовых значений. • **average** — возвращает среднее всех

Имя элемента и путь	Описание	Атрибуты
(integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > target_attribute > map) Действительно только при использовании типа сопоставления **childattr**		включенных значений. • **min** — возвращает минимальное включенное значение (число или символ). • **max** — возвращает максимальное включенное значение (число или символ).
source_child_ci_type (integration > targetcis> source_ci_type > target_attribute > map -OR- integration > targetcis > source_ci_type_tree > target_attribute > map -OR- (integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > target_attribute > map) Действительно только при использовании типа сопоставления **childattr**.	Указывает, от какого связанного ЭК берется значение дочернего атрибута.	1. **Имя.** name **Описание.** Тип дочернего ЭК **Обязательно?** Обязательно **Тип.** Строка 2. **Имя.** source_attribute **Описание.** Сопоставляемый атрибут дочернего ЭК. **Обязательно?** Обязательно, только если тип агрегации **childAttr** (по тому же пути) не =count. **Тип.** Строка

Имя элемента и путь	Описание	Атрибуты
validation (integration > targetcis > source_ci_type > target_attribute > map -OR- integration > targetcis > source_ci_type_tree > target_attribute > map -OR- (integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > target_attribute > map) Действительно только при использовании типа сопоставления **childattr**	Обеспечивает фильтрацию дочерних ЭК исходных ЭК по значениям атрибутов. Использует подэлемент сведения для обеспечения детализации дочерних атрибутов, сопоставленных со значением целевого атрибута типа ЭК. Необязательно.	1. **Имя.** minlength **Описание.** Исключает строки короче указанного значения. **Обязательно?** Необязательно **Тип.** Целое число 2. **Имя.** maxlength **Описание.** Исключает строки длиннее указанного значения. **Обязательно?** Необязательно **Тип.** Целое число 3. **Имя.** minvalue **Описание.** Исключает строки меньше указанного значения. **Обязательно?** Необязательно **Тип.** Числовой 4. **Имя.** maxvalue **Описание.** Исключает числа больше указанного значения. **Обязательно?** Необязательно **Тип.** Числовой
targetrelations (integration)	Элемент-контейнер для всех сопоставлений связей. Необязательно.	
source_link_type_tree (integration > targetrelations)	Сопоставление исходного типа связи без наследующих типов с целевой связью. Обязательно, только если присутствует элемент **targetrelation**.	1. **Имя:** name **Описание.** Имя исходной связи. **Обязательно?** Обязательно **Тип.** Строка 2. **Имя:** target_link_type **Описание.** Имя целевой связи **Обязательно?** Обязательно **Тип.** Строка 3. **Имя:** nameSpace **Описание:** Пространство имен для связи, которая будет создана для

Имя элемента и путь	Описание	Атрибуты
		целевого объекта. **Обязательно?** Необязательно **Тип:** Строка 4. **Имя:** mode **Описание:** Тип обновления для текущей связи. **Обязательно?** Обязательно **Тип:** Одна из следующих строк: • **insert** — используется, только если ЭК не существует. • **update** — используется, только если ЭК существует. • **update_else_insert** — если ЭК существует, он обновляется, в противном случае создается новый ЭК. • **ignore** — пропустить этот тип ЭК. 5. **Имя:** source_ci_type_end1 **Описание:** Тип ЭК End1 исходной связи. **Обязательно?** Обязательно **Тип:** Строка 6. **Имя:** source_ci_type_end2 **Описание:** Тип ЭК End2 исходной связи **Обязательно?** Обязательно **Тип:** Строка
link (integration > targetrelations)	Сопоставление исходной связи с целевой. Обязательно, только если присутствует элемент **targetrelation**.	1. **Имя:** source_link_type **Описание:** Имя исходной связи. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** target_link_type **Описание:** Имя целевой связи. **Обязательно?** Обязательно **Тип:** Строка 3. **Имя:** nameSpace **Описание:** Пространство имен для связи, которая будет создана для целевого объекта. **Обязательно?** Необязательно

Имя элемента и путь	Описание	Атрибуты
		Тип: Строка 4. **Имя:** mode **Описание:** Тип обновления для текущей связи. **Обязательно?** Обязательно **Тип:** Одна из следующих строк: • **insert** — используется, только если ЭК не существует. • **update** — используется, только если ЭК существует. • **update_else_insert** — если ЭК существует, он обновляется, в противном случае создается новый ЭК. • **ignore** — пропустить этот тип ЭК. 5. **Имя:** source_ci_type_end1 **Описание:** Тип ЭК End1 исходной связи. **Обязательно?** Обязательно **Тип:** Строка 6. **Имя:** source_ci_type_end2 **Описание:** Тип ЭК End2 исходной связи **Обязательно?** Обязательно **Тип:** Строка
target_ci_type_end1 (integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)	Тип ЭК End1 целевой связи.	1. **Имя:** name **Описание:** Имя типа ЭК End1 целевой связи. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** superclass **Описание:** Имя суперкласса типа ЭК End1. **Обязательно?** Необязательно **Тип:** Строка
target_ci_type_end2 (integration > targetrelations > link -OR-	Тип ЭК End2 целевой связи.	1. **Имя:** name **Описание:** Имя типа ЭК End2 целевой связи. **Обязательно?** Обязательно **Тип:** Строка

Имя элемента и путь	Описание	Атрибуты
integration > targetrelations > source_link_type_tree)		2. **Имя:** superclass **Описание:** Имя суперкласса типа ЭК End2. **Обязательно?** Необязательно **Тип:** Строка

Схема результатов сопоставления

Имя элемента и путь	Описание	Атрибуты
root	Корень документа с результатами.	
data (root)	Корень данных	
objects (root > data)	Корневой элемент объектов для обновления.	
Object (root > data > objects)	Описание операции обновления для одного объекта и всех его атрибутов.	1. **Имя:** name **Описание:** Имя типа ЭК. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** mode **Описание:** Тип обновления для текущего типа ЭК. **Обязательно?** Обязательно **Тип:** Одна из следующих строк: a. **insert** — используется, только если ЭК не существует. b. **update** — используется, только если ЭК существует. c. **update_else_insert** — если ЭК существует, он обновляется, в противном случае создается новый ЭК. d. **ignore** — пропустить этот тип ЭК. 3. **Имя:** operation **Описание:**Операция, выполняемая с этим ЭК. **Обязательно?** Обязательно **Тип:** Одна из следующих строк:

Имя элемента и путь	Описание	Атрибуты
		a. **add** – добавить ЭК b. **update** – обновить ЭК c. **delete** – удалить ЭК Если значение не установлено, используется значение **add** по умолчанию. 4. **Имя:** mamId **Описание:** Идентификатор объекта в исходной базе CMDB. **Обязательно?** Обязательно **Тип:** Строка
field (root > data > objects> Object -OR- root > data > links > link)	Описание значения одного поля объекта. Текст поля — это новое значение в поле, и если поле содержит связь, значением будет идентификатор одной из сторон. Каждый конечный идентификатор отображается как объект (в разделе <objects>).	1. **Имя:** name **Описание:** Имя поля. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** key **Описание:** Указывает, является ли поле ключом для объекта. **Обязательно?** Обязательно **Тип:** Логическое 3. **Имя:** datatype **Описание:** Тип поля. **Обязательно?** Обязательно **Тип:** Строка 4. **Имя:** length **Описание:** Для типов данных string и character размер целевого атрибута (целое число). **Обязательно?**Необязательно **Тип:** Целое число
links (root > data)	Корневой элемент связей для обновления.	1. **Имя:** targetRelationshipClass **Описание:** Имя связи в целевой системе. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** targetParent **Описание:** Тип первой стороны связи (родитель). **Обязательно?** Обязательно **Тип:** Строка 3. **Имя:** targetChild **Описание:** Тип второй стороны связи (потомок).

Имя элемента и путь	Описание	Атрибуты
		Обязательно? Обязательно **Тип:** Строка 4. **Имя:** mode **Описание:** Тип обновления для текущего типа ЭК. **Обязательно?** Обязательно **Тип:** Одна из следующих строк: a. **insert** — используется, только если ЭК не существует. b. **update** — используется, только если ЭК существует. c. **update_else_insert** — если ЭК существует, он обновляется, в противном случае создается новый ЭК. d. **ignore** — пропустить этот тип ЭК. 5. **Имя:** operation **Описание:**Операция, выполняемая с этим ЭК. **Обязательно?** Обязательно **Тип:** Одна из следующих строк: a. add – добавить ЭК b. update – обновить ЭК c. delete – удалить ЭК Если значение не установлено, используется значение **add** по умолчанию. 6. **Имя:** mamId **Описание:** Идентификатор объекта в исходной базе CMDB. **Обязательно?** Обязательно **Тип:** Строка

Настройка

В этом разделе объясняются основные процедуры стандартных типов настройки адаптеров принудительной отправки данных.

Добавление атрибутов

1. Включите атрибут в результаты TQL-запроса.
2. Добавьте сопоставление атрибута в файл сопоставления в разделе соответствующего ЭК.
3. Подготовьте устройство приема к получению добавленного атрибута.

Удаление атрибутов

Чтобы удалить атрибут, удалите его из файла сопоставления. Если данный атрибут больше не используется в качестве результата или условия, его также следует удалить из TQL-запроса.

Добавление типа ЭК

1. Добавьте тип ЭК в TQL-запрос.
2. Убедитесь, что данные атрибута отображаются в результатах TQL-запроса (при помощи функций расчета и предварительного просмотра).
3. Добавьте сопоставление типа ЭК в файл сопоставления. Для быстрого создания нового типа ЭК можно воспользоваться копированием сопоставления существующего типа.
4. Измените имя XML-файла и сопоставление атрибутов в соответствии с требованиями этого типа ЭК. Сведения о доступных типах сопоставления см. в разделе "Справочные сведения о файлах сопоставления" на странице 265.

Удаление типа ЭК

1. Удалите тип ЭК из TQL-запроса.
2. Удалите соответствующий раздел в файле сопоставления.

Добавление связей

1. Проверьте наличие обоих связываемых ЭК в потоке данных.
2. Проверьте связь (в Диспетчере типов ЭК).
3. Добавьте элементы связи в раздел связей XML-файла сопоставления.

Удаление связей

1. Удалите соответствующий раздел из файла сопоставлений.
2. Если это возможно, удалите связь из TQL-запроса (если это не повлияет на его работу).

Глава 8: Разработка общих адаптеров

Данная глава включает:

- Синхронизация экземпляров 280
- Принудительная отправка данных с помощью общего адаптера 280
- Принудительная отправка данных с помощью общего адаптера 289
- Объединение данных с помощью общего адаптера 304
- Выверка 318
- API общего адаптера 318
- API-интерфейсы указателей ресурсов 319
- Создание пакета общего адаптера 319
- Различия в сопоставлении принудительной отправкой и заполнением 325
- Файлы журнала общего адаптера 325
- Адаптеры, использующие Generic Adapter Framework 326
- Справочные материалы по XML-схеме общего адаптера 326

Синхронизация экземпляров

Операции общего адаптера по принудительной отправке и заполнению направлены на работу с данными экземпляров. Подробнее о понятиях *экземпляр* и *корневой объект* см. в разделах "Поток заполнения на основе экземпляров" на странице 201 и "Принудительная отправка данных с помощью общего адаптера" ниже.

Принудительная отправка данных с помощью общего адаптера

Принудительная отправка данных использует существующую платформу расширенного адаптера принудительной отправки с небольшими изменениями XML-схемы.

Примечание. Общий адаптер работает в режиме экземпляра (то есть он работает не с отдельными типами ЭК, а с коллекциями ЭК, сгруппированными основным корневым ЭК). Подробнее см. в разделе "Поток заполнения на основе экземпляров" на странице 201.

Ниже приведены изменения XML-схемы, необходимые для размещения семантики двустороннего сопоставления:

- тег **<targetcis>** переименован в **<target_entities>**.
- тег **<source_instance_type>** переименован в **<source_instance>**.

- тег **<target_ci_type>** переименован в **<target_entity>**.
- тег **<for-each-source-ci>** переименован в **<for-each-source-entity>**.
- атрибут заголовка **versions** переименован в **version** и больше не требует десятичного значения.

В этом разделе представлены сведения о принудительной отправке данных при помощи Generic Adapter Framework:

- Принудительная отправка — Обзор 281
- Файл сопоставления 281
- The Groovy Traveler 284
- Создание сценариев Groovy 287
- Реализация интерфейса PushAdapterConnector 288

Принудительная отправка — Обзор

Общий адаптер работает со структурами данных, представляющими результат TQL-запроса. Все адаптеры, созданные на базе Generic Adapter Framework, обрабатывают эту структуру данных и отправляют ее в соответствующие целевые точки.

Структура данных называется **ResultTreeNode (RTN)**. RTN создается на основании файла сопоставления для адаптера и результатов TQL-запроса. Запросы, используемые для Generic Adapter Framework, должны быть основаны на корневом элементе, т.е. запрос должен содержать один узел с именем элемента **root**, либо один или несколько элементов связей, имена которых начинаются с префикса **root**. Этот ЭК или эта связь служит корневым элементом запроса. Подробнее см. в разделе Data Push в документе *Руководство по управлению потоками данных в HP Universal CMDB*.

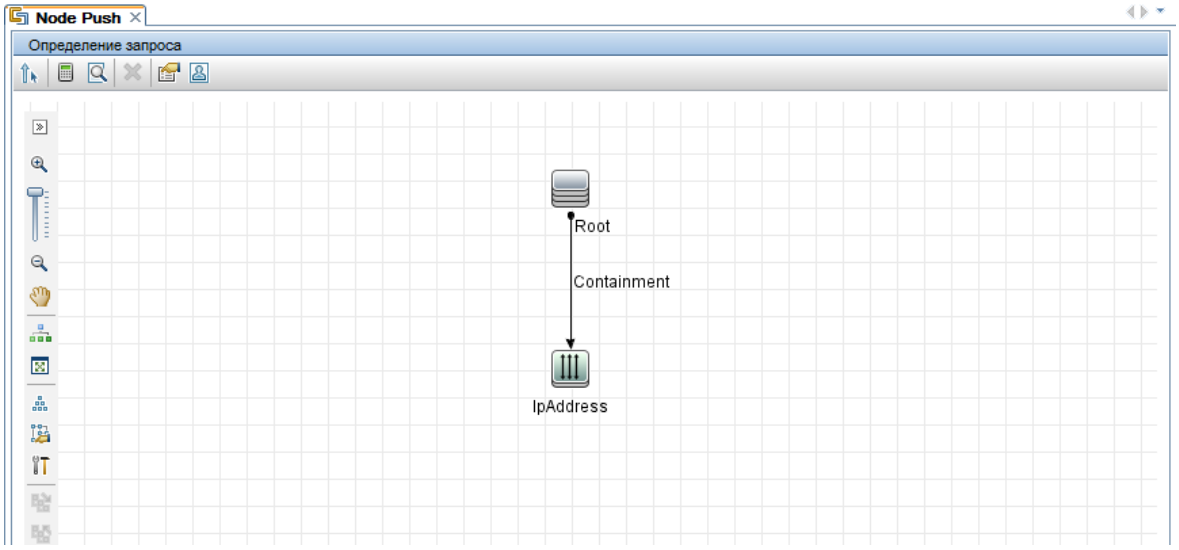
Разработка расширенного адаптера принудительной отправки данных состоит из двух основных шагов:

1. Реализация интерфейса PushAdapterConnector — этот интерфейс принимает данные, которые необходимо добавить, обновить или удалить, в виде списка RTN, а затем отправляет их в целевую точку.
2. Создание файла сопоставления — файл сопоставления определяет создание структуры RTN путем сопоставления ЭК и атрибутов из результатов TQL-запроса.

Файл сопоставления

В следующем примере показана процедура создания файла сопоставления.

В этом примере демонстрируется принудительная отправка данных об узле и IP-адресе. Мы создадим TQL-запрос с именем: **Принудительная отправка данных об узле** осуществляется следующим образом:

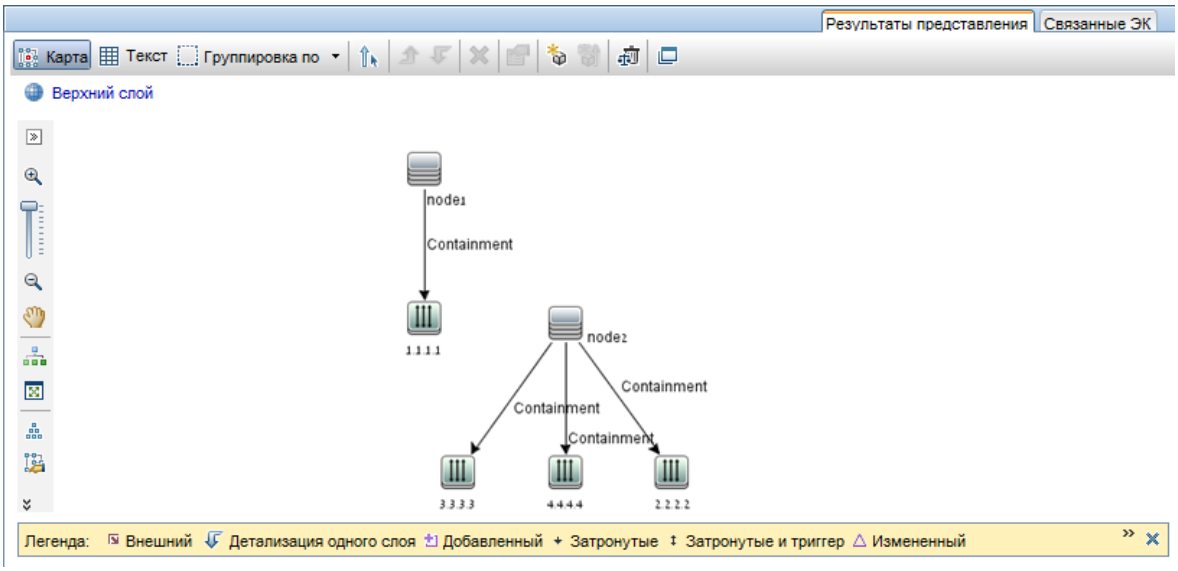


В файле сопоставления создается два целевых типа ЭК: **Computer** и **IP**. Computer имеет одну переменную и два атрибута. IP имеет один атрибут.

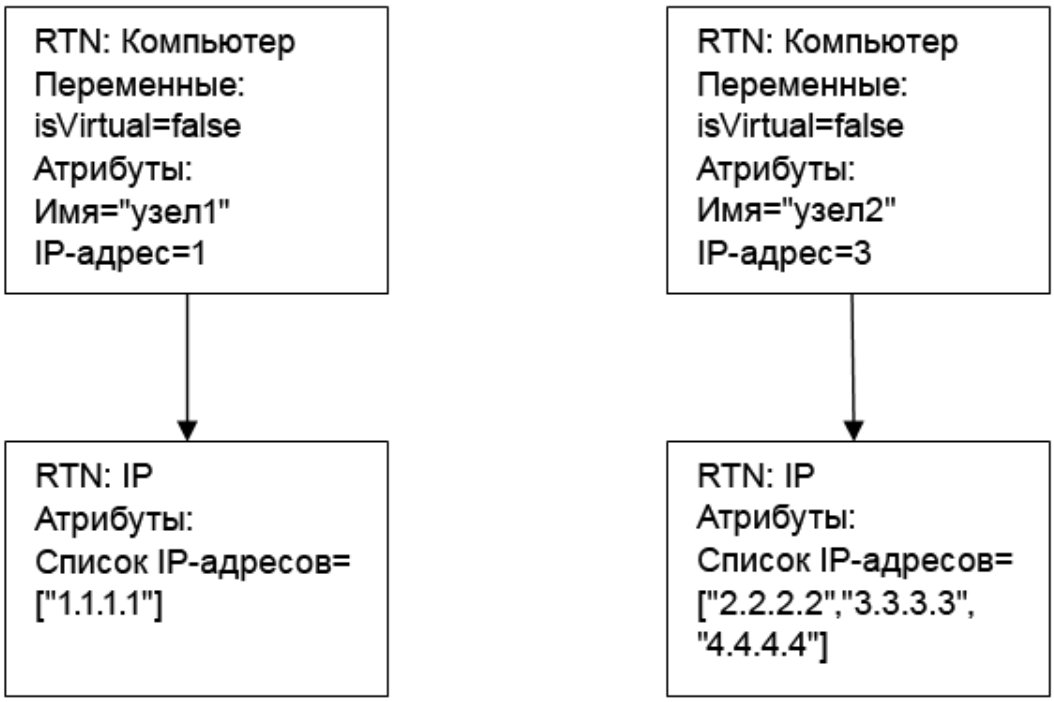
XML-файл сопоставления выглядит следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../generic-adapter.xsd">
    <info>
        <source name="UCMDB" version="10.20" vendor="HP"/>
        <target name="PushProduct" version="9.3" vendor="HP"/>
    </info>

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

    <target_entities>
        <source_instance query-name="Node Pusy" root-element-name="Root">
            <target_entity name="Computer" is-valid="(Root['root_iscandidatefordeletion'] == null) ? true : !Root['root_iscandidatefordeletion']">
                <variable name="isVirtual" datatype="BOOLEAN" value="PushFunctions.isVirtual(Root['root_class'])"/>
                <target_mapping name="name" datatype="STRING" value="Root['name']"/>
                <target_mapping name="ipNumber" datatype="INTEGER" value="Root.IpAddress.size()"/>
                <target_mapping name="description" datatype="STRING" value="PushFunctions.getDescription(isVirtual)"/>
                <target_entity name="IP">
                    <target_mapping name="IpAddressList" datatype="STRING_LIST" value="Root.IpAddress*.getAt('name')"/>
                </target_entity>
            </target_entity>
        </source_instance>
    </target_entities>

</integration>
```

Результат запроса имеет следующий вид:



Согласно данному файлу сопоставления создается следующий список RTN:



Каждый корневой экземпляр сопоставляется отдельно с помощью файла сопоставления. Поэтому в данном примере PushAdapterConnector получает список из двух корневых RTN.

Примечание. Предыдущий адаптер принудительной отправки данных мог создавать общее сопоставление для типа ЭК. Новый адаптер выполняет сопоставление для каждого TQL-запроса. При выполнении задания принудительной отправки данных на основании запроса **x** адаптер ищет соответствующий файл сопоставления (имеющий

атрибут): query-name=x).

Значения атрибутов в файле сопоставления можно рассчитать с помощью языка сценариев groovy. Дополнительные сведения см. в разделе "The Groovy Traveler" ниже.

The Groovy Traveler

Доступ к результатам TQL-запроса осуществляется следующим образом:

- **Root[attr]** возвращает атрибут **attr** элемента Root.
- **Root.Query_Element_Name** возвращает список экземпляров ЭК Query_Element_Name в TQL-запросе, связанных с текущим корневым ЭК.
- **Root.Query_Element_Name[2][attr]** возвращает атрибут **attr** третьего ЭК Query_Element_Name, связанного с текущим корневым ЭК.
- **Root.Query_Element_Name*.getAt(attr)** возвращает список атрибутов **attr** экземпляров ЭК Query_Element_Name в TQL-запросе, связанных с текущим корневым ЭК.

С помощью groovy traveler можно получить доступ к ряду других атрибутов:

- **cmdb_id** – возвращает идентификатор ЭК или связи в UCMDB в виде строки.
- **external_cmdb_id** – возвращает внешний идентификатор ЭК или связи в виде строки.
- **Element_type** – возвращает тип элемента ЭК или связи в виде строки.

Метка импорта:

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

Таким образом объявляется импорт для всех сценариев groovy в файле сопоставления. В данном примере **PushFunctions** — это сценарий groovy, содержащий несколько статических функций, доступ к которым возможен в процессе сопоставления (напр., value=" PushFunctions.foo()")

source_instance_type

Сопоставление выполняется для каждого TQL-запроса. Значение query-name указывает, какой запрос связан с текущим сопоставлением. Символ '*' означает, что файл сопоставления связан со всеми запросами, имена которых начинаются с префикса: **Принудительная отправка данных об узле**.

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

Метка source_instance_type обозначает корневой элемент в сопоставлении.

Значение root-element-name должно в точности совпадать с именем корневого элемента в TQL-запросе.

target_entity

Данная метка используется для создания RTN.

Атрибут name соответствует имени target_entity: name=Computer

Атрибут **is-valid** — это логическое значение, рассчитываемое в процессе сопоставления и указывающее, является ли допустимым текущий target_ci. Недопустимые значения target_entity не добавляются в RTN. В данном примере мы не создаем экземпляр target_entity, для которого значение атрибута **root_iscandidatefordeletion** в UCMDB равно true.

Атрибут target_entity может иметь переменные, значения которых рассчитываются в процессе сопоставления:

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

Переменная **vSerialNo** получает значение, равное **serial_number** текущего корневого элемента.

Атрибут RTN создается с помощью метки **target_mapping**. Результат выполнения сценария groovy в поле **value** присваивается атрибуту RTN.

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

SerialNo получает значение переменной **vSerialNo**.

Назначить target_entity дочерним элементом другого target_entity можно следующим образом:

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

RTN **Portfolio** будет иметь дочернюю RTN **Asset**.

for-each-source-entity

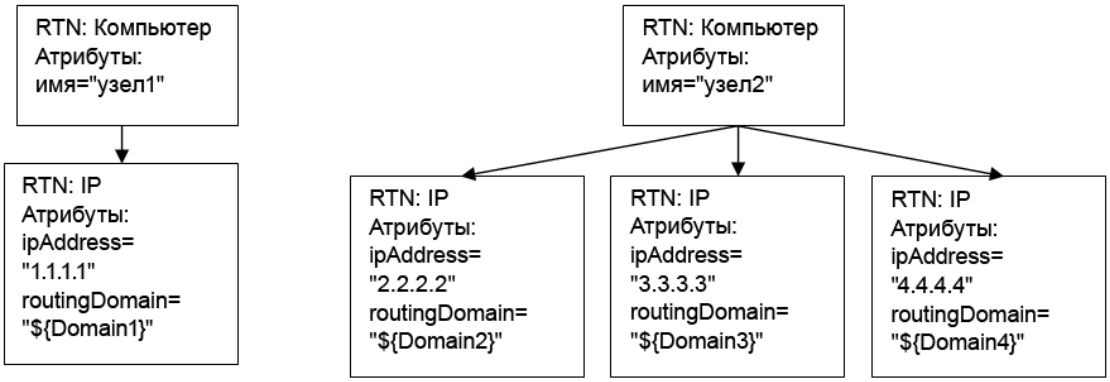
В данной метке перечисляются конкретные ЭК корневого экземпляра. У нее есть следующие поля:

- **source-entities=" "** — список ЭК, для которых создается целевой ЭК. Этот список определяется groovy traveler в поле **Root.IpAddress**.
- **count-index=" "** — переменная, хранящая индекс ЭК в текущей итерации цикла.
- **var-name=" "** — имя ЭК в текущей итерации цикла.

Внесем некоторые изменения в пример файла сопоставления:

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

Список RTN, создаваемый согласно данному файлу сопоставления, будет выглядеть следующим образом:



dynamic_mapping

Эта метка добавляет возможность сопоставления данных из целевого хранилища во время создания структуры RTN.

Пример: Предположим, что в качестве целевой точки используется база данных с таблицей **Computer**, имеющей столбцы **id** и **name**, причем последний связан с **Node.name** в UCMDB. Оба столбца являются уникальными. Кроме того, в базе данных есть таблица **IP** со ссылочным ключом на **parentID** в таблице Computer. Механизм 'динамического сопоставления' может создать схему, в которой атрибуты name и ID хранятся в виде <name,id>. На основании этой схемы адаптер сопоставляет идентификаторы с компьютерами и передает необходимое значение атрибуту **parentID** в таблице IP. С помощью этой схемы можно присвоить значение атрибуту parentID при создании RTN.

Сопоставление определяется свойством **map_property**. Выполнение dynamic_mapping осуществляется однократно для каждого фрагмента.

```
<dynamic_mapping name="IdByName " keys-unique="true">
```

Атрибут **name** представляет имя схемы. Атрибут **keys-unique** указывает, являются ли ключи уникальными (ключ сопоставлен с одним значением или набором значений).

В данном примере схема называется **IdByName** и имеет уникальные ключи. Для доступа к схеме сопоставления в сценарии выполните следующую команду:

```
DynamicMapHolder.getMap('IdByName')
```

Команда возвратит ссылку на схему.

Метка **map_property** создает свойство, на котором основано сопоставление.

Пример:

```
<map_property property-name="SQLQuery" datatype="STRING"
property-value="SELECT name, id FROM Computer"/>
```

В данном примере свойство называется **SQLQuery**, а его значением является SQL-оператор, создающий схему сопоставления. Реализация методов **retrieveUniqueMapping** и **retrieveNonUniqueMapping** для интерфейса PushConnector определяет фактическое содержание возвращенной схемы.

Глобальные переменные

Сценарий groovy может работать со следующими глобальными переменными в файле сопоставления.

- **Topology** – Тип: Топология. Экземпляр топологии текущего фрагмента.
- **QueryDefinition** - Тип: QueryDefinition. Экземпляр определения текущего TQL-запроса.
- **OutputCI** – Тип: ResultTreeNode. RTN корневого элемента в текущем сопоставлении дерева.
- **ClassModel** – Тип: ClassModel. Экземпляр модели классов.
- **CustomerInformation** – Тип: CustomerInformation. Сведения о заказчике, запускающем задание.
- **Logger** – Тип: DataAdapterLogger. Модуль ведения журнала в адаптере для передачи сведений в платформу журналов UCMDB.

Создание сценариев Groovy

В данном разделе описано создание файла **PushFunctions.groovy**. В этом файле будут содержаться статические функции, используемые при сопоставлении корневого элемента.

```
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.
        }
}
```

Реализация интерфейса PushAdapterConnector

Реализация должна поддерживать следующие основные шаги:

```
public class PushExampleAdapter implements PushAdapterConnector

{

public UpdateResult pushTreeNodes(PushConnectorInput input) throws
DataAccessException{

// 1. build an UpdateResult instance - the UpdateResult is used to return
mappings between the sent ids to the actual ids that entered the data store.
// Also has an update status which allows to pass the status of data that was
actually pushed, detailed status reports on failed IDs, and actions actually
performed on successful ids.
// 2. обработка данных:
// a. handle data to add. Can be retrieved by:
input.getResultTreeNodes.getDataToAdd();
//  b. handle data to update.
//  c. handle data to delete.
// 3. Возврат результата обновления.
        }

public void start(PushDataAdapterEnvironment env) throws DataAccessException{
        // this method is called when the integration point created,
or when the adapter is reloaded
        //(i.e after changing one of the mapping files
        // and pressing ‘save’).

}



public void testConnection(PushDataAdapterEnvironment env) throws
DataAccessException {
                // this method is called when pressing the 'test
connection' button in the
```

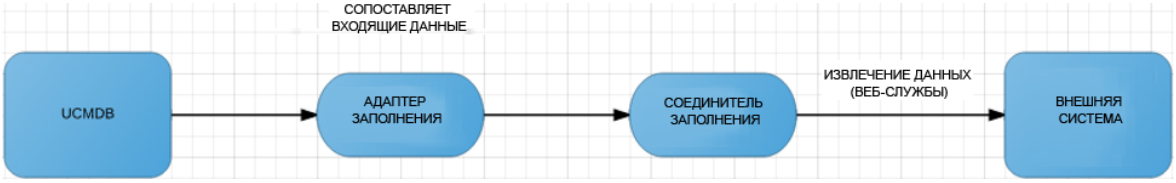
```
                //creation of the integration point.
                // For example if we push data to RDBMS this method
can create a connection
                //to the database and will run a dummy SQL statement.
                // If it fails it writes an error message to the log
and throws an exception.
        }



Map<Object, Object> retrieveUniqueMapping(MappingQuery mappingQuery){
//This method will create the map according to the given mappingQuery. It will
be called in the
// mapping stage of the adapter execution, before the 'UpdateResult
pushTreeNodes' method.
// This method is called when the 'keys-unique' attribute of the 'dynamic_
mapping' tag is true.
}

Map<Object, Set<Object>> retrieveNonUniqueMapping(MappingQuery mappingQuery){
// This method is called when the 'keys-unique' attribute of the 'dynamic_
mapping' tag is false.
//  In this case a key can be mapped to several values.
}
}
```

Принудительная отправка данных с помощью общего адаптера

Этот раздел включает следующие подразделы:

- Архитектура Population Framework 289
- Основные артефакты, включенные в заполнение 290
- Режимы адаптера заполнения 302
- Явное сопоставление внешних ID 303
- Принудительная обратная отправка глобальных идентификаторов 303

Архитектура Population Framework



Механизм работы схож с механизмом Push Adapter Framework, он подразумевает, что пользователь должен предоставить файл сопоставления и использовать соединитель, а также собрать их вместе в пакете адаптера UCMDB.

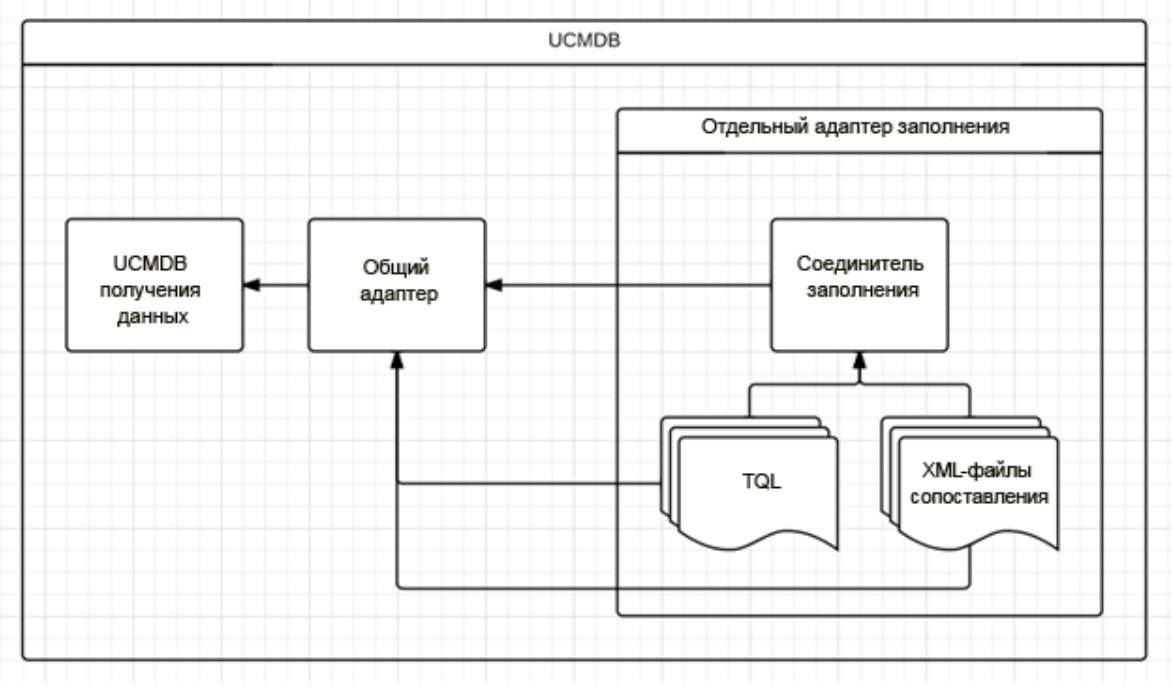
Поток операций состоит из следующих этапов

1. Пользователь UCMDB инициирует операцию заполнения из пользовательского интерфейса.
2. В адаптер заполнения отправляется команда.
3. Адаптер заполнения вызывает соединитель заполнения и извлекает данные блоками.
4. Адаптер заполнения применяет заданное сопоставление к данным из каждого блока и перенаправляет их в UCMDB Server.

Основные артефакты, включенные в заполнение

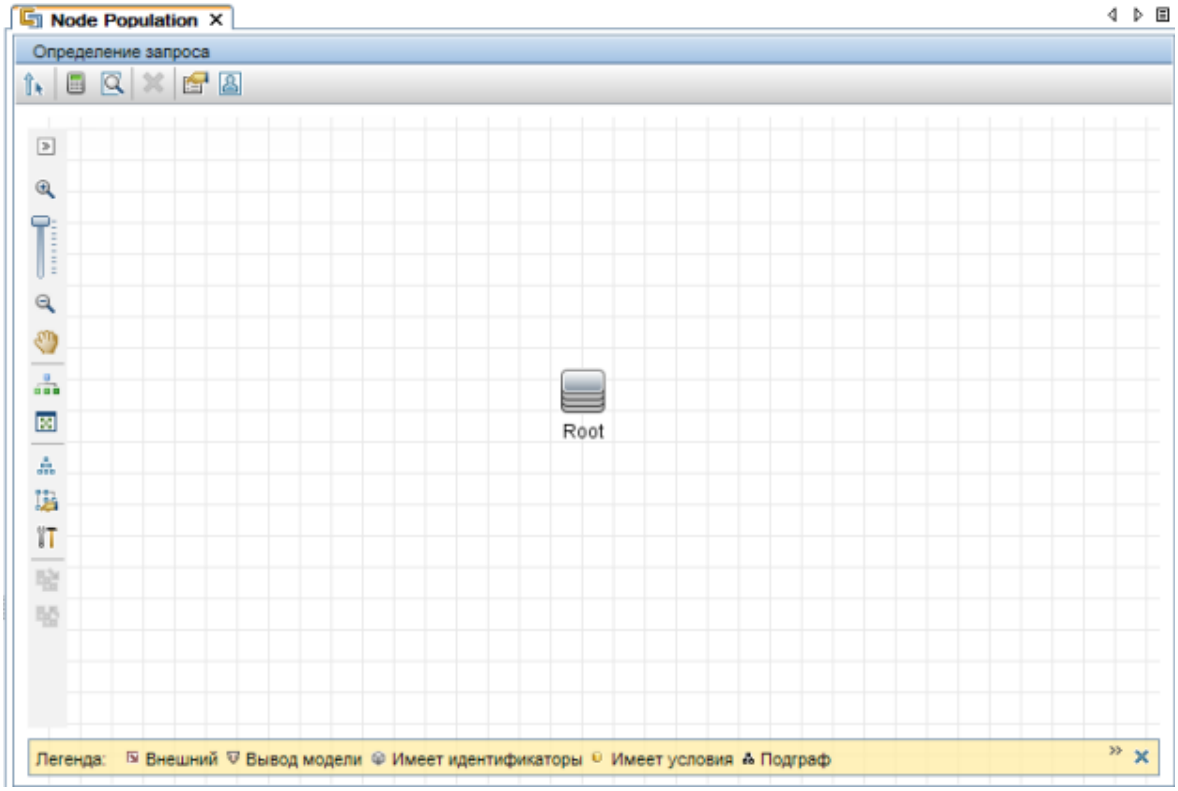
Основными артефактами, включенными в заполнение, являются:

- TQL-запросы, которые указывают данные для заполнения в UCMDB
- XML-файлы сопоставления, которые указывают, каким образом возвращенные соединителем данные будут сопоставлены в UCMDB
- обязательные данные
- соединитель заполнения, который отвечает за извлечение данных внешней системы и их возврат в общий адаптер UCMDB.



TQL-запросы заполнения

Роль TQL-запроса заполнения заключается в указании данных, которые будут заполняться в UCMDB. К примеру, на следующем рисунке TQL используется для передачи экземпляров узлов в UCMDB.



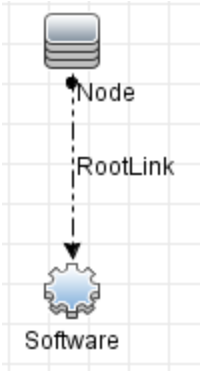
Соединитель заполнения отвечает за понимание TQL-запросов заполнения и предоставление необходимых данных из внешней системы.

Файлы сопоставления заполнением

Файлы сопоставления XML имеют ту же задачу, что и для операций принудительной отправки, но направление является обратным. Эти файлы сопоставления описывают то, каким образом возвращенные соединителем данные будут сопоставлены с данными UCMDB.

Приведенные здесь сведения относятся к сопоставлению заполнением и еще не рассматриваются для расширенного адаптера принудительной отправки.

Ниже приведен пример сопоставления для узлов и запущенного программного обеспечения в UCMDB. На первом изображении показан TQL-запрос узлов и запущенного ПО. На втором изображении показано сопоставление узлов и запущенного ПО заполнением.




```
<target_entities>
    <!--The query name must match the one selected in the UI-->
    <source_instance query-name="Nodes And Software Population" root-element-name="PC">
        <!-- need to match case in UCMDB TQL -->
        <target_entity name="RootLink">
            <target_mapping name="name" datatype="STRING" value="PC['name'] + 'has' + PC.Programs[0]['name']"/>
        </target_entity>
        <target_entity name="Node" type="Util.getNodeType(PC)">
            <target_mapping name="name" datatype="STRING" value="PC['name']"/>
            <target_mapping name="description" datatype="STRING" value="PC['description']"/>
        </target_entity>
        <target_entity name="Software">
            <target_mapping name="name" datatype="STRING" value="PC.Programs[0]['name']"/>
            <target_mapping name="root_container_name" datatype="STRING" value="PC['name']"/>
            <target_mapping name="product_name" datatype="STRING" value="'vmware_hypervisor'"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Это задание заполнения передает данные из внешней системы в форме ResultTreeNode (RTN) PC. ResultTreeNode API представлен расширенным адаптером принудительной отправки и находится в файле **push-interfaces.jar**, расположенном в папке **lib** в UCMDB Server. Подробнее см. в разделе "Принудительная отправка данных с помощью общего адаптера" на странице 280.

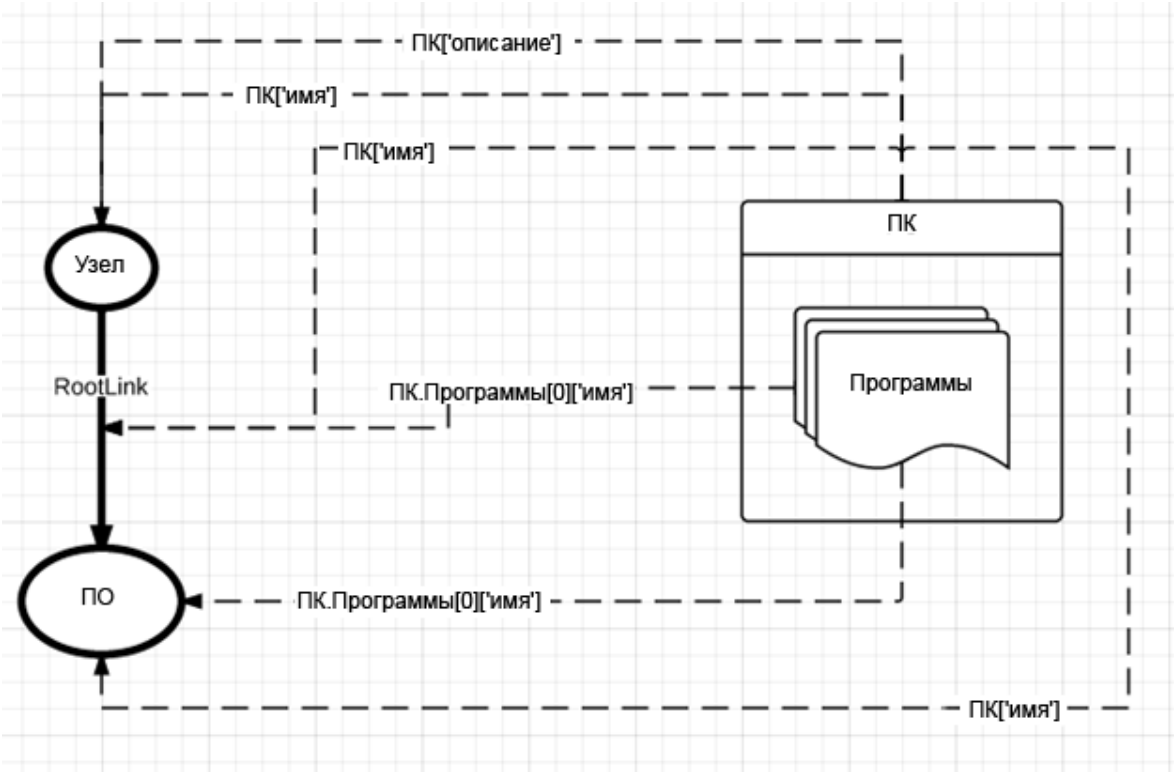
PC RTN содержит общие сведения об узле в форме атрибутов, а также встроенный объект Programs, содержащий объекты типа "программное обеспечение" с собственными атрибутами.

Один экземпляр PC будет сопоставлен с тремя экземплярами в UCMDB:

- ЭК типа Node
- ЭК типа Running Software
- ЭК со связью Composition

Подробнее о формате экземпляра PC см. в разделе "Выходные данные запроса заполнения" на странице 301.

Процедура сопоставления данных соединителя с данными UCMDB представлена на следующем рисунке:



Давайте проанализируем ключевые строки:

```
<!--The query name must match the one selected in the UI-->
<source_instance query-name="Nodes And Software Population" root-element-name="PC">
```

Определение **source_instance** сообщает о том, что объекты будут переданы в UCMDB, а топология UCMDB, группирующая эти объекты, определяется TQL-запросом заполнения узлов и ПО. Кроме того, структурой данных, которая возвращается соединителем и будет использоваться для создания данных UCMDB, является **ResultTreeNode** с именем **PC**.

```
<target_entity name="RootLink">
    <target_mapping name="name" datatype="STRING" value="PC['name'] + 'has' + PC.Programs[0]['name']"/>
</target_entity>
```

Тег **target_entity** сообщает о том, что новый объект UCMDB начинается здесь, и этот объект соответствует элементу **RootLink** внутри TQL-запроса заполнения узлов и ПО. Он также сообщает тип ЭК нового объекта в UCMDB.

Создаваемый объект **RootLink** будет иметь один атрибут, **name**, значением которого будет что-то вроде **Computer_22 has MySQL Server**.

В этом примере сопоставления используется заполнение связей вручную. Рекомендуется использовать автоматическое заполнение, описанное в разделе "Автоматическое заполнение связей" на следующей странице.

Атрибут заполнения type

```
<target_entity name="Node" type="Util.getNodeType(PC)">
```

Обратите внимание — объект Node имеет атрибут **type**. Атрибут **type** указывает конкретный тип ЭК, который этот объект будет иметь в UCMDB. Атрибут **type** не является обязательным, поскольку тип объекта, по умолчанию присваиваемый при создании, извлекается из элемента TQL, к которому этот объект относится (в данном случае это Node). Однако, если необходимо вернуть несколько экземпляров ЭК UCMDB с типом Node, и некоторые из них относятся к Windows, а некоторые — к Unix, можно воспользоваться атрибутом **type**, чтобы указать точный тип создания в UCMDB. Таким образом, в данном случае мы создаем функцию **getNodeType** внутри файла сценариев функции **Util**, которая в качестве входных данных получает дерево PC и возвращает действительный идентификатор типа ЭК UCMDB ("unix" для Unix и "nt" для Windows).

Примечание. Атрибут типа **target_entity** доступен только в контексте потока заполнения, и его значением должно быть допустимое выражение Groovy.

Точно так же можно описать создание объекта Software.

Автоматическое заполнение связей

На примере сопоставления, приведенном в разделе "Файлы сопоставления заполнением" мы узнали, что необходимо для явного сопоставления заполненной связи. Сопоставляемый целевой объект должен быть в наличии для каждого элемента связи TQL, такого, как показанный ниже:

```
<target_entity name="RootLink">
    <target_mapping name="name" datatype="STRING" value="PC['name'] + 'has' + PC.Programs[0]['name']"/>
</target_entity>
```

При использовании механизма автоматического заполнения связей общего адаптера больше нет необходимости сопоставлять элементы связи TQL с разделами сопоставления, такими, как представленный выше. Платформа создаст экземпляр ЭК связи с типом, указанным в TQL-запросе с пустыми свойствами. Эта операция будет выполняться для всех связей в TQL-запросе заполнения.

Представленное в примере сопоставление заканчивается созданием ЭК связи с типом Composition, который в качестве конечных точек связи (end1 и end2) имеет экземпляры ЭК Node и Running Software.

Следует использовать метод заполнения связей вручную, если заполняемая связь требует следующее:

- Динамический тип связи (с использованием атрибута **type**)
- Свойства связи

Заполнение связей вручную

Общий адаптер осуществляет заполнение связей путем определения (сопоставления) трех объектов, которые требуются для связи:

- объект Связь
- объект End 1 Связи
- объект End 2 Связи

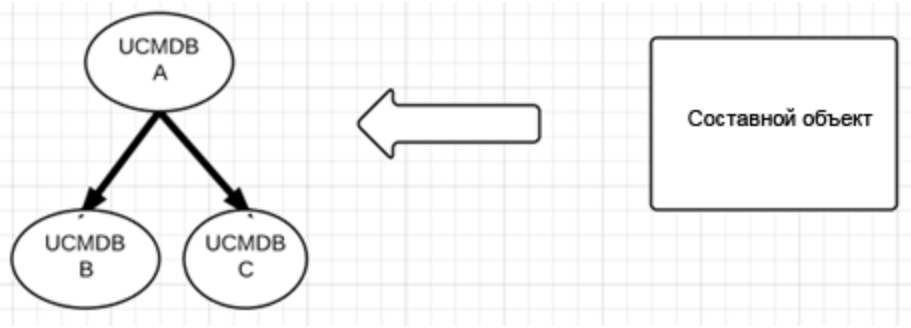
Давайте проанализируем пример сопоставления, представленный в разделе "Файлы сопоставления заполнением" на странице 291. В данном случае мы заполняем три типа объектов в UCMDB: Nodes, Running Software и связь типа Composition между ними. Поскольку мы хотим заполнить связь (связь с именем **RootLink** и типом Composition), нам также необходимо сопоставить две конечных точки связи. Таким образом, взглянув на TQL-запрос, мы видим, что двумя объектами для сопоставления являются Node (End 1) и Software (End 2). Generic Adapter Framework понимает структуру связей, изучая имя и определение элемента созданного объекта в TQL-запросе. Поскольку задание заполнения также должно приносить экземпляры типа Node и Running Software, требуемое сопоставление конечных точек уже присутствует.

Типы заполнения связей

Существует два типа заполнения связей:

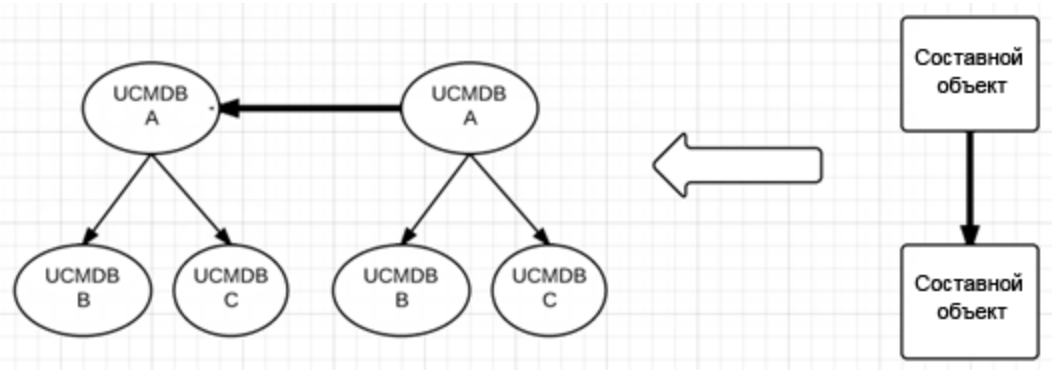
- Разделение составного внешнего объекта на несколько связанных объектов UCMDB

  В это случае составной внешний объект, такой как PC, преобразуется в типы объектов UCMDB **Node** и **Running Software**, которые должны быть объединены связью типа **Composition**. Такой тип связи существует только в контексте UCMDB.



- Связи между составными внешними объектами

  В этом случае требуется смоделировать связь между двумя составными внешними объектами, такими как PC.



Соединитель заполнения

Соединитель заполнения отвечает за извлечение данных внешней системы. Данные передаются в общий адаптер в установленном формате API (**ResultTreeNode**), а затем сопоставляются со структурами данных UCMDB и вставляются в UCMDB соответствующим процессом ввода.

Так же, как и соединитель принудительной отправки, соединитель заполнения может работать на базе Java и Groovy, а также должен иметь Java-интерфейс соединителя заполнения (см. рисунок ниже).

Чтобы настроить соединитель заполнения, добавьте следующую строку в XML-файл конфигурации адаптера:

```
<adapter-settings>
    <adapter-setting name="PopulationConnector.class.name">com.hp.ucmdb.connector.dummy.DummyPopulationConnector</adapter-setting>
```

```java
public interface PopulationAdapterConnector extends GenericConnector, DynamicMappingConnector {

    /**
     * Executes a population request and provides the entities that will be populated in the format of
     * {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode}
     * Used for both population and federation with the flow type flag which is present in the
     * {@link com.hp.ucmdb.adapters.population.connector.PopulationConnectorInput}
     *
     * @param input population request
     * @return a population response
     * @throws DataAccessException
     */
    PopulationConnectorOutput populate(PopulationConnectorInput input) throws DataAccessException;

    /**
     * Returns the collection of queries the current connector supports for population
     * Note: this method is independent to the adapter life cycle (i.e. start/shutdown methods) and must work at all times.
     * <p/>
     * The method also supports return of queries with their folders path: E.g.
     * "Folder/Secondary Folder/Query Name 1"
     * "Folder/Secondary Folder/Query Name 2"
     *
     * @param env the adapter environment
     * @return a collection of the supported queries
     */
    Collection<String> getPopulationQueries(DataAdapterEnvironment env);

    /**
     * Returns the collection of queries the current connector supports for federation
     * Note: this method is independent to the adapter life cycle (i.e. start/shutdown methods) and must work at all times.
     * <p/>
     * The method also supports return of queries with their folders path: E.g.
     * "Folder/Secondary Folder/Query Name 1"
     * "Folder/Secondary Folder/Query Name 2"
     *
     * @param env the adapter environment
     * @return a collection of the supported queries
     */
    Collection<String> getFederationQueries(DataAdapterEnvironment env);

    /**
     * Update target id (global id) for each source object.
     *
     * @param idMapping mapping between source id (external id) and target id (string)
     */
    void updateGlobalIDsFromTarget(FCmdbExtenalToTargetIdMappingSet idMapping);

    /**
     * This methods reports population queries resources used by the adapter.<br>
     * This allows editing these resources directly from the Integration Studio.
     *
     * @param input  the requested information
     * @param output the returned resources
     * @see com.hp.ucmdb.federationspi.adapter.resource.PushQueriesResourceLocator
     */
    void locatePopulationQueriesResources(DataAdapterEnvironment env, LocatePopulationQueriesResourcesInput input,
                                          LocatePopulationQueriesResourcesOutput output);

    /**
     * Returns the collection of classes the current adapter supports for query.<br>
     * Notes:<br>
     * 1. This method is independent of the adapter life cycle (i.e. start/shutdown methods).<br>
     * 2. If adapter configuration (xml) defines supported classes, this method doesn't need to be implemented (return null).<br>
     *
     * @param env the Adapter env
     * @return collection of the supported classes
     * @throws DataAccessException in case of an error
     */
    Collection<SupportedClassConfig> getSupportedClasses(DataAdapterEnvironment env);
}
```

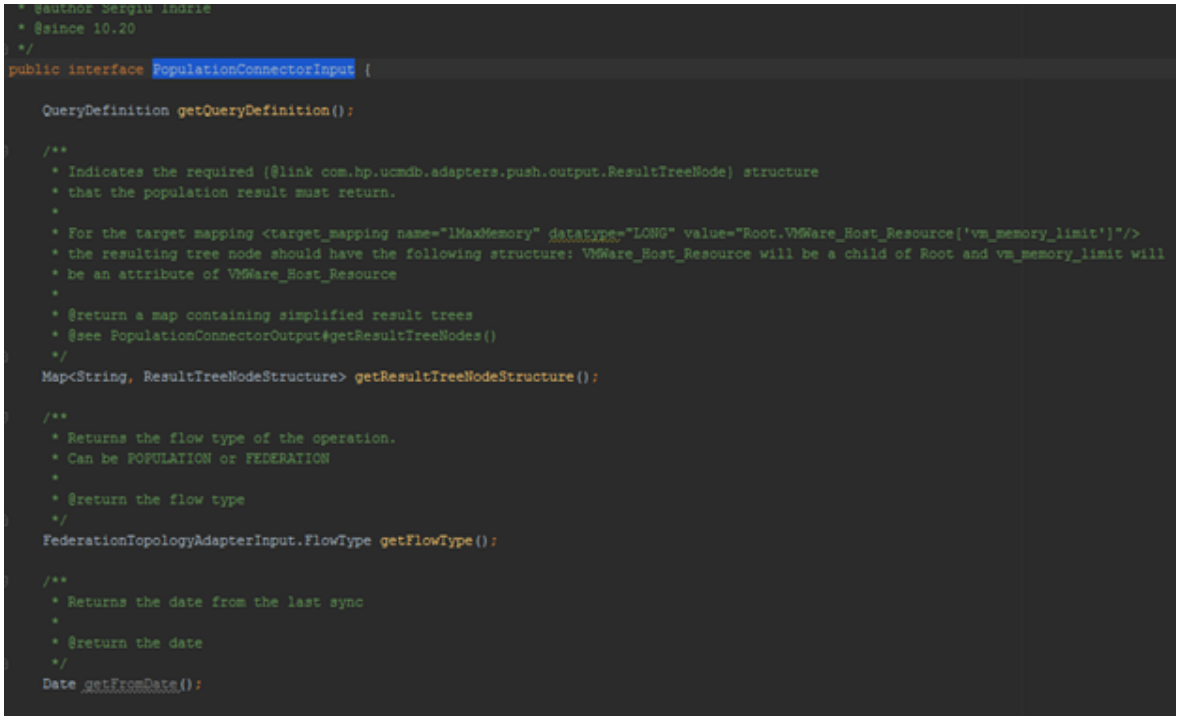
Первый метод, `populate`, является основным методом соединения, отвечающим за извлечение данных из внешней системы. Этот метод в качестве входных данных получает TQL-запрос и затем отправляет результаты в общем формате *ResultTreeNode*. Подробнее см. в разделе "Принудительная отправка данных с помощью общего адаптера" на странице 280. Вместе с основными бизнес-данными соединитель также возвращает сведения о статусе и разбивке на блоки.

Второй метод, `getSupportedQueries`, указывает на TQL-запросы, которые поддерживаются соединителем заполнения.

Третий и четвертый методы относятся к более сложным сценариям использования — они возвращают идентификаторы заполненных данных и находят релевантные ресурсы заполнения внутри адаптера для конкретного запроса. Подробнее об этих API-интерфейсах см. в файле **push-interfaces.jar**.

Входные данные запроса заполнения

Запрос заполнения определяется объектом **QueryDefinition**, описывающим такой запрос в UCMDB. Соединитель заполнения отвечает за чтение этого объекта запроса и его перевод на язык запросов внешней системы.



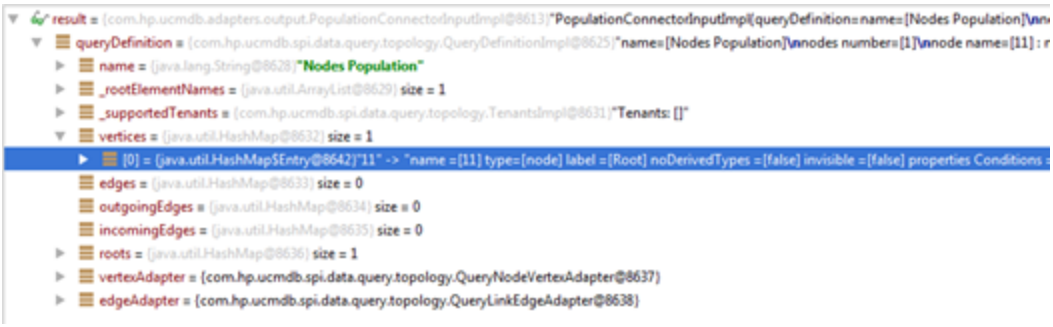
```
 * @author Sergiu Indrie
 * @since 10.20
 */
public interface PopulationConnectorInput {

    QueryDefinition getQueryDefinition();

    /**
     * Indicates the required {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode} structure
     * that the population result must return.
     *
     * For the target mapping <target_mapping name="lMaxMemory" datatype="LONG" value="Root.VMWare_Host_Resource['vm_memory_limit']"/>
     * the resulting tree node should have the following structure: VMWare_Host_Resource will be a child of Root and vm_memory_limit will
     * be an attribute of VMWare_Host_Resource
     *
     * @return a map containing simplified result trees
     * @see PopulationConnectorOutput#getResultTreeNodes()
     */
    Map<String, ResultTreeNodeStructure> getResultTreeNodeStructure();

    /**
     * Returns the flow type of the operation.
     * Can be POPULATION or FEDERATION
     *
     * @return the flow type
     */
    FederationTopologyAdapterInput.FlowType getFlowType();

    /**
     * Returns the date from the last sync
     *
     * @return the date
     */
    Date getFromDate();
```

Кроме объекта QueryDefinition существуют следующие объекты:

- **getResultTreeNodeStructure** — указывает требуемую структуру, которую должен вернуть результат заполнения.
- **getFlowType** — используется для определения типа соединителя (ЗАПОЛНЕНИЕ или ОБЪЕДИНЕНИЕ).

getFromDate — указывает дату последней синхронизации. Если значение даты нулевое, выполняется ПОЛНОЕ ЗАПОЛНЕНИЕ, в противном случае — ДИФФЕРЕНЦИАЛЬНОЕ ЗАПОЛНЕНИЕ (если типом потока является ОБЪЕДИНЕНИЕ, метод getFromDate всегда возвращает нулевое значение).

На следующем рисунке представлен пример запроса заполнения:



В этом примере представлен запрос **Nodes Population**. Как можно увидеть, запрос содержит только один TQL-элемент типа **Node**.

ResultTreeNodeStructure

Для использования **PopulationAdapterConnector** необходимо прочесть TQL заполнения UCMDB, понять, что запрашивает UCMDB, и предоставить результаты с помощью объектов внешней системы. Например, UCMDB может запрашивать все узлы, связанные с экземплярами бизнес-служб во внешней системе, и может так быть, что эквивалентом внешней системы для компьютера является **PC**, который связан в объектом **Service**. Таким образом соединитель заполнения должен возвращать экземпляры **PC**, соединенные с экземплярами **Service**. В этом случае сопоставление будет выглядеть где-то вот так:

```
<target_entities>
    <source_instance query-name="Nodes And BS Population" root-element-name="PC">
        <!-- Node -->
        <target_entity name="Root">
            <target_mapping name="name" datatype="STRING" value="PC['name']"/>
        </target_entity>

        <!-- Business Service -->
        <target_entity name="BusinessService">
            <target_mapping name="name" datatype="STRING" value="PC.Service['name']"/>
            <target_mapping name="description" datatype="STRING" value="PC.Service['description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

В этом случае для возврата экземпляров **PC**, связанных с экземплярами **Service**, мы возвращаем RTN **PC**, содержащий **Service** в качестве дочернего узла. Однако мы могли создать сопоставление в формате RTN **Service** с дочерним узлом **PC** следующего характера (что делает сопоставление недействительным):

```
<target_entities>
    <source_instance query-name="Nodes And BS Population" root-element-name="Service">
        <!-- Node -->
        <target_entity name="Root">
            <target_mapping name="name" datatype="STRING" value="Service.PC['name']"/>
        </target_entity>

        <!-- Business Service -->
        <target_entity name="BusinessService">
            <target_mapping name="name" datatype="STRING" value="Service['name']"/>
            <target_mapping name="description" datatype="STRING" value="Service['description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Таким образом, для облегчения разработки соединителя заполнения запрос заполнения, отправляемый общим адаптером, также содержит структуру RTN данных, используемых в файле сопоставления. Это указывает внедряемому соединителю требуемый формат возвращаемого RTN.

ResultTreeNodeStructure в первом случае:

PC
- name

  Service
  - name
  - description

ResultTreeNodeStructure во втором случае:

Service
- name
- description

  PC
  - name

Выходные данные запроса заполнения

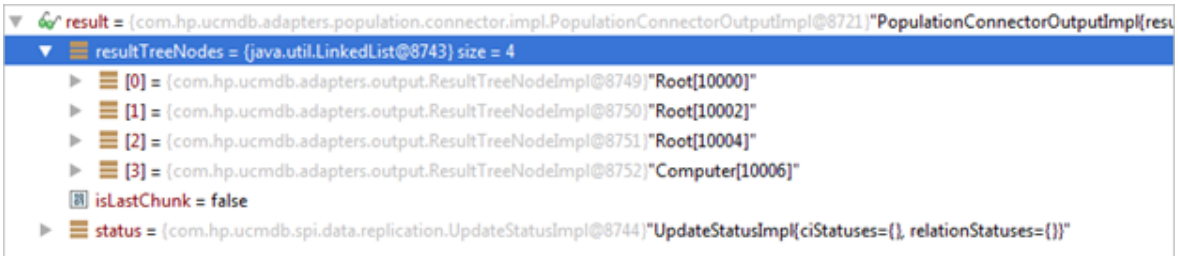
После обработки запроса заполнения соединитель заполнения должен вернуть ответ `PopulationConnectorOutput`.

```
public interface PopulationConnectorOutput {

    /**
     * The result trees representing the external entities in {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode} format.
     *
     * @return a list of result trees or an empty list
     */
    List<ResultTreeNode> getResultTreeNodes();

    /**
     * Adds a result tree to the population output object.
     *
     * @param resultTreeNode the result tree to add
     */
    void addResultTreeNode(ResultTreeNode resultTreeNode);

    /**
     * This method indicates if the result received is the last chunk of data.
     *
     * @return true if this is the last chunk, false otherwise.
     */
    boolean isLastChunk();

    /**
     * Set whether this result object is the last chunk or not.
     */
    void setLastChunk(boolean isLastChunk);

    /**
     * Returns the status information about the population result.
     */
    UpdateStatus getStatus();

    /**
     * Set the population result status.
     */
    void setStatus(UpdateStatus status);

}
```

Этот объект выходного значения содержит:

- данные в очереди (в формате `ResultTreeNode`)
- информарцию о статусе (необходимую в случае сбоя)
- информацию о блоках

На следующем рисунке представлен пример ответа заполнения:



В приведенном ответе соединитель вернул четыре экземпляра данных (соответствующих типу `Node` в **UCMDB**), пустой статус (сообщающий об успехе) и флаг, указывающий на то, что это не последний блок.

Режимы адаптера заполнения

UCMDB Adapter Framework предусматривает два типа адаптеров заполнения:

- Стандартный адаптер заполнения
  - Адаптер характеризуется отсутствием тега **<changes-source/>** в своем XML-файле.
  - Всегда представляет данные запроса из внешней системы в полном объеме. В этом случае UCMDB Probe Framework отвечает за определение различий между двумя последовательными выполнениями. Probe Framework делает это, сравнивая предыдущий результат конкретного запроса с текущим результатом и вычисляя различия. Полное заполнение достигается за счет того, что текущий результат запроса ни с чем не сравнивается, а используется как окончательный результат. Этот поток подразумевает, что заполняемые данные не фильтруются по дате начала, поскольку фильтрация по дате сделает сравнение данных бессмысленным.
- Адаптер заполнения changes-source
  - Адаптер характеризуется использованием тега **<changes-source/>** в своем XML-файле:

```
<adapterInfo>
    <adapter-capabilities>
        <support-federated-query/>
        <support-replicatioin-data>
            <source>
                <changes-source/>
                <push-back-ids/>
                <re-populate/>
            </source>
            <target/>
        </support-replicatioin-data>
```

  - Адаптер changes-source отвечает за вычисление различий между двумя последовательными выполнениями.

Удаление ЭК при использование адаптера changes-source

При использовании адаптера заполнения changes-source, последний отвечает за явное удаление ЭК. Это осуществляется путем использования XML-атрибута файла сопоставления **is-deleted**, который принимает действительное выражение Groovy.

Например, в файле сопоставления, представленном ниже, соединитель заполнения возвращает экземпляры **Service**. Несмотря на то, что экземпляры все еще действуют, некоторые ЭК, которые были частью тех экземпляров **Service**, были удалены. Чтобы сообщить об удалении этих ЭК, необходимо использовать атрибут **is-deleted** в сопоставлении **BusinessService**.

```
<target_entities>
    <source_instance query-name="Nodes And BS Population" root-element-name="Service">
        <!-- Node -->
        <target_entity name="Root">
            <target_mapping name="name" datatype="STRING" value="Service.PC['name']"/>
        </target_entity>

        <!-- Business Service -->
        <target_entity name="BusinessService" is-deleted="Functions.isOlderThanThreeMonths()">
            <target_mapping name="name" datatype="STRING" value="Service['name']"/>
            <target_mapping name="description" datatype="STRING" value="Service['description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Явное сопоставление внешних ID

Могут возникать ситуации, в которых заполненным данным (ЭК) потребуется управляемый соединителем/адаптером атрибут **ExternalId**. Для этого используйте следующую концепцию сопоставления:

```
<target_entities>
    <!--The query name must match the one selected in the UI-->
    <source_instance query-name="Node with ID" root-element-name="Computer">
        <!-- need to match case in UCMDB TQL -->
        <target_entity name="Root">
            <!--This is how the RTN External ID is set-->
            <variable name="external_id_obj" datatype="STRING" value="Computer['external_id_obj']"/>
            <!--RTN Attributes-->
            <target_mapping name="name" datatype="STRING" value="Computer.Asset[0]['name']"/>
            <target_mapping name="description" datatype="STRING" value="Computer['name']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

В данном случае корневой ЭК заполняется ExternalId, созданным на уровне соединителя и размещенным на компьютере ['external_id_obj']. Создание ExternalId также может осуществляться на уровне сопоставления при помощи сценария Groovy.

Примечание. Механизм явного создания внешних ID переопределяет атрибут `target_entity` **type**. Таким образом, при создании внешнего ID при помощи файла сценария сопоставления или внутри соединителя атрибут **type** игнорируется, и окончательный тип UCMDB заполненного ЭК будет типом, заданным в объекте **ExternalId**.

Принудительная обратная отправка глобальных идентификаторов

Существуют ситуации, когда заполненные ЭК в UCMDB также должны быть синхронизированы во внешней системе. Для такого сценария Generic Adapter Framework предусматривает принудительную обратную отправку идентификаторов. Для использования этой функции выполняется обратный вызов всех ЭК, которые были заполнены в UCMDB, и информирование адаптера заполнения о назначенных глобальных идентификаторах для каждого из ЭК.

Чтобы активировать эту функцию, добавьте выделенную в приведенном ниже примере строку в XML-файл конфигурации адаптера:

```
<adapterInfo>
    <adapter-capabilities>

        <support-federated-query>
        <supported-classes>
            <supported-class is-derived="true" all-attributes-supported="true" name="node" is-reconciliation-supported="true"/>
            <supported-class is-derived="true" all-attributes-supported="true" name="business_service" is-reconciliation-supported="true"/>
            <supported-class is-derived="true" all-attributes-supported="true" name="incident" is-reconciliation-supported="true"/>
        </supported-classes>
        </support-federated-query>
        <support-replicatioin-data>
            <source>
                <push-back-ids/>
                <instance-based-data/>
                <population-queries-resources-locator/>
            </source>
            <target>
                <instance-based-data>true</instance-based-data>
                <push-queries-resources-locator/>
            </target>
        </support-replicatioin-data>
        <general-resources-locator/>
    </adapter-capabilities>
```

Также необходимо использовать метод `PopulationAdapterConnector`, как указано далее:

```
/**
* <b>Description:</b><br>
 * This Interface is used for implementing
 * an adapter to allow the definition of Integration Points.<br>
 * An adapter that implements this Interface will expose
 * ths ability to run Population flows.<br>
 * The only Population flow exposed by this adapter is the Simple Flow using {@link #getDataResult(com.hp.ucmdb.federationspi.adapter.federation.
 * which will return the entire data set of data each time, and compare it to the last data set (handled by framework).<p>
 * It is highly recommended to allow better link validation by implementing {@link com.hp.ucmdb.federationspi.adapter.ReportsLinks}<br>
 * If possible, it's recommended to implement the {@link PopulationChangesAdapter} instead,
 * allowing the adapter to have better performance and control.<p>
 *
 * @see com.hp.ucmdb.federationspi.data.query.topology.TopologyFactory
 * @see com.hp.ucmdb.federationspi.adapter.ChunkTopologyResultGetter
 * @see PopulationAdapter
 * @since 10.10
 *
*/
public interface PopulationAdapter extends BasicSourceDataAdapter{

    /**
     * Retrieves the calculated result of the given {@code QueryDefinition}.<br>
     * <li>
     * This method is called by the population framework.<br>
     * </li>
     * <p>
     * This method implementation may use the {@code UpdateStatus} to report warnings for specific CIs or Relations.
     * <p/>
     * When implementing this method, it should return the result by being aware to all the conditions
     * that appear on the {@code QueryDefinition} like: topology, cardinality, id conditions and property conditions.
     * </p><br>
     * When implementing this method, one must be aware that different flows may actually be used:<br>
     * 1) A topology only flow (a query definition with conditions or id conditions but with out any layout requested).<br>
     * 2) A layout only flow (a query definition with only ids condition and layout).<br>
     * 3) A full topology flow (a query with property conditions, id conditions and layout)<br>
     *
     * @param input contains the logged in user and queryDefinition that contains the topology, conditions and layout requested by the framework
     * @return {@link com.hp.ucmdb.federationspi.adapter.federation.FederationTopologyAdapterOutput} containing the query's calculated result.
     * @throws com.hp.ucmdb.federationspi.exception.DataAccessException
     */
    public FederationTopologyAdapterOutput getDataResult(FederationTopologyAdapterInput input) throws DataAccessException;

}
```

Объединение данных с помощью общего адаптера

Объединение данных осуществляется при помощи следующих средств:

- Сопоставительный подход к объединению 305
- API объединения общего адаптера 305

• Настройка объединения 308

Сопоставительный подход к объединению

Сопоставление объединением осуществляется путем сопоставления подзапросов TQL, используемых Federation Framework в UCMDB для обработки запроса объединения. Основной смысл заключается в том, что когда запрос объединения получается общим адаптером, происходит следующее:

1. Проанализируйте динамический TQL-запрос объединения и сравните его со списком статических TQL-запросов объединения, предоставленным соединителем объединения.
2. Выполняется сравнение со статическим TQL-запросом. Этот TQL-запрос используется для идентификации требуемого сопоставления для конкретного запроса объединения и для создания входного аргумента структуры RTN (объекта Java, который будет иллюстрировать древовидную структуру узлов, необходимую соединителю), который будет отправляться в соединитель объединения. (Подробнее см. в файле **push-interfaces.jar**).
3. Отправьте запрос объединения с аргументом TQL в соединитель.
4. Сопоставьте входящие деревья RTN, отправляемые соединителем, тем же способом, что и для заполнения. См. "Принудительная отправка данных с помощью общего адаптера" на странице 289.

Сопоставление связей объединением

Сопоставление связей объединением выполняется автоматически, также как и сопоставление связей заполнением. См. "Автоматическое заполнение связей" на странице 294.

API объединения общего адаптера

API объединения общего адаптера имеет очень схожие характеристики с API заполнения общего адаптера. Это связано с тем, что Java-интерфейс общего адаптера объединения идентичен интерфейсу общего адаптера заполнения.

```
/**
* <b>Description:</b><br>
 * This Interface is used for implementing
 * an adapter to allow the definition of Integration Points.<br>
 * An adapter that implements this Interface will expose
 * ths ability to run Population flows.<br>
 * The only Population flow exposed by this adapter is the Simple Flow using {@link #getDataResult(com.hp.ucmdb.federationspi.adapter.federation.
 * which will return the entire data set of data each time, and compare it to the last data set (handled by framework).<p>
 * It is highly recommended to allow better link validation by implementing {@link com.hp.ucmdb.federationspi.adapter.ReportsLinks}<br>
 * If possible, it's recommended to implement the {@link PopulationChangesAdapter} instead,
 * allowing the adapter to have better performance and control.<p>
 *
 * @see com.hp.ucmdb.federationspi.data.query.topology.TopologyFactory
 * @see com.hp.ucmdb.federationspi.adapter.ChunkTopologyResultGetter
 * @see PopulationAdapter
 * @since 10.10
 *
 */
public interface PopulationAdapter extends BasicSourceDataAdapter{

    /**
     * Retrieves the calculated result of the given {@code QueryDefinition}.<br>
     * <li>
     * This method is called by the population framework.<br>
     * </li>
     * <p>
     * This method implementation may use the {@code UpdateStatus} to report warnings for specific CIs or Relations.
     * <p/>
     * When implementing this method, it should return the result by being aware to all the conditions
     * that appear on the {@code QueryDefinition} like: topology, cardinality, id conditions and property conditions.
     * </p><br>
     * When implementing this method, one must be aware that different flows may actually be used:<br>
     * 1) A topology only flow (a query definition with conditions or id conditions but with out any layout requested).<br>
     * 2) A layout only flow (a query definition with only ids condition and layout).<br>
     * 3) A full topology flow (a query with property conditions, id conditions and layout)<br>
     *
     * @param input contains the logged in user and queryDefinition that contains the topology, conditions and layout requested by the framework
     * @return {@link com.hp.ucmdb.federationspi.adapter.federation.FederationTopologyAdapterOutput} containing the query's calculated result.
     * @throws com.hp.ucmdb.federationspi.exception.DataAccessException
     */
    public FederationTopologyAdapterOutput getDataResult(FederationTopologyAdapterInput input) throws DataAccessException;

}
```

```
import ...

/**
  * <b>Description:</b><br>
  * This Interface is used for implementing
  * an adapter to allow the definition of Integration Points.<br>
  * An adapter that implements this Interface will expose to run Federation flows.<br>
  * The adapter can report status per CI during the flow with {@link com.hp.ucmdb.federationspi.data.replication.UpdateStatus}.<br>
  * It is highly recommended to allow better link validation by implementing {@link com.hp.ucmdb.federationspi.adapter.ReportsLinks}
  * </p>
  *
 */
public interface FederationTopologyDataAdapter extends FTqlDataAdapter {

    /**
     * Retrieves the calculated result of the given {@code QueryDefinition}.<br>
     * <li>
     * This method is called by the federation framework when the user query includes any
     * federated elements(node or link).<br>
     * </li>
     * <p>
     * This method implementation may use the {@code UpdateStatus} to report warnings for specific CIs or Relations.
     * <p/>
     * When implementing this method, it should return the result by being aware to all the conditions
     * that appear on the {@code QueryDefinition} like: topology, cardinality, id conditions and property conditions.
     * </p><br>
     * When implementing this method, one must be aware that different flows may actually be used:<br>
     * 1) A topology only flow (a query definition with conditions or id conditions but with out any layout requested).<br>
     * 2) A layout only flow (a query definition with only ids condition and layout).<br>
     * 3) A full topology flow (a query with property conditions, id conditions and layout)<br>
     *
     * @param input contains the logged in user and queryDefinition that contains the topology, conditions and layout requested by th
     * @return {@link FederationTopologyAdapterOutput} containing the query's calculated result.
     * @throws com.hp.ucmdb.federationspi.exception.DataAccessException
     */
    public FederationTopologyAdapterOutput getDataResult(FederationTopologyAdapterInput input) throws DataAccessException;

}
```

Интерфейс соединителя общего адаптера для объединения

Запросы объединения используют тот же метод, который используется запросами заполнения, поэтому можно применять тот же соединитель заполнения. Новый атрибут был добавлен в Java-класс PopulationConnectorInput с именем **FlowType**. Атрибут **FlowType** может иметь два значения, ОБЪЕДИНЕНИЕ или ЗАПОЛНЕНИЕ. Общий адаптер узнает тип запроса по этому атрибуту.

```java
/**
 * Holds data needed to process a population request.
 *
 * @author Sergiu Indrie
 * @since 10.20
 */
public interface PopulationConnectorInput {

    QueryDefinition getQueryDefinition();

    /**
     * Indicates the required {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode} structure
     * that the population result must return.
     *
     * For the target mapping <target_mapping name="lMaxMemory" datatype="LONG" value="Root.VMWare_Host_Resource['vm_memory_limit']"/>
     * the resulting tree node should have the following structure: VMWare_Host_Resource will be a child of Root and vm_memory_limit will
     * be an attribute of VMWare_Host_Resource
     *
     * @return a map containing simplified result trees
     * @see PopulationConnectorOutput#getResultTreeNodes()
     */
    Map<String, ResultTreeNodeStructure> getResultTreeNodeStructure();

    /**
     * Returns the flow type of the operation.
     * Can be POPULATION or FEDERATION
     *
     * @return the flow type
     */
    FederationTopologyAdapterInput.FlowType getFlowType();
```

```java
import ...

/**
 * Population Connector that will be used by the UCMDB's Generic Population Adapter to provide the external data. The
 * data provided by the connector will be mapped to the UCMDB entities by the adapter configured mapping files.
 * <p/>
 * The Population Connector must be able to process population requests by analyzing the input query and the providing
 * corresponding data from the external system.
 *
 * @author Sergiu Indrie
 * @since 10.20
 */
public interface PopulationAdapterConnector extends GenericConnector, DynamicMappingConnector {

    /**
     * Executes a population request and provides the entities that will be populated in the format of
     * {@link com.hp.ucmdb.adapters.push.output.ResultTreeNode}
     * Used for both population and federation with the flow type flag which is present in the
     * {@link com.hp.ucmdb.adapters.population.connector.PopulationConnectorInput}
     *
     * @param input population request
     * @return a population response
     * @throws DataAccessException
     */
    PopulationConnectorOutput populate(PopulationConnectorInput input) throws DataAccessException;
```

Поддерживаемые запросы объединения

Запросы объединения и заполнения располагаются в разных папках. Java-интерфейс **PopulationAdapterConnector** предлагает следующие два метода указания поддерживаемых запросов заполнения и объединения:

- **getPopulationQueries** — возвращает коллекцию запросов, которые поддерживаются текущим соединителем для заполнения.
- **getFederationQueries** — возвращает коллекцию запросов, которые поддерживаются текущим соединителем для объединения.

```
/**
 * Returns the collection of queries the current connector supports for population
 * Note: this method is independent to the adapter life cycle (i.e. start/shutdown methods) and must work at all times.
 * <p/>
 * The method also supports return of queries with their folders path: E.g.
 * "Folder/Secondary Folder/Query Name 1"
 * "Folder/Secondary Folder/Query Name 2"
 *
 * @param env the adapter environment
 * @return a collection of the supported queries
 */
Collection<String> getPopulationQueries(DataAdapterEnvironment env);

/**
 * Returns the collection of queries the current connector supports for federation
 * Note: this method is independent to the adapter life cycle (i.e. start/shutdown methods) and must work at all times.
 * <p/>
 * The method also supports return of queries with their folders path: E.g.
 * "Folder/Secondary Folder/Query Name 1"
 * "Folder/Secondary Folder/Query Name 2"
 *
 * @param env the adapter environment
 * @return a collection of the supported queries
 */
Collection<String> getFederationQueries(DataAdapterEnvironment env);
```

Настройка объединения

Этот раздел включает следующие подразделы:

- Настройка параметров адаптера 309
- Настройка запросов объединения из файлов журнала 309
- Пример настройки объединения 313

Настройка параметров адаптера

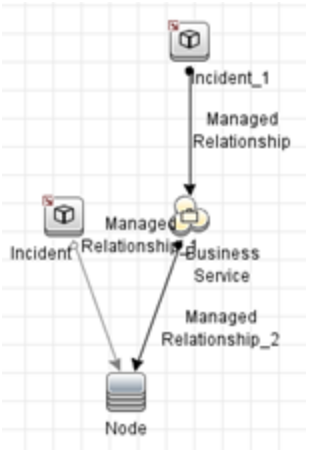
Для конкретного TQL-запроса общий адаптер должен объявлять все узлы из TQL-запроса в теге **<supported-classes>**. Например, если TQL-запрос имеет форму инцидента, связанного с узлом, следует объявить и узел, и инцидент как поддерживаемые классы в XML-файле параметров адаптера, расположенном в архивном файле адаптера в папке **discoveryPatterns**.


```
<supported-classes>
    <supported-class is-derived="true" all-attributes-supported="true" name="node" is-reconciliation-supported="true"/>
    <supported-class is-derived="true" all-attributes-supported="true" name="incident" is-reconciliation-supported="true"/>
</supported-classes>
```

Настройка запросов объединения из файлов журнала

Перед началом использования Federation Framework должны быть выполнены некоторые обязательные условия. Federation Framework делает несколько различных TQL-запросов в адаптер для извлечения необходимых данных. Для каждого TQL-запроса, отправляемого Federation Framework, в адаптере должен создаваться динамический TQL-запрос (тем же способом, что и в потоке заполнения).

Разница между объединением и заполнением заключается в том, что TQL-запросы объединения отправляются динамически, они не могут быть заранее известны. Возьмем в качестве примера следующий TQL-запрос:



Federation Framework отправляет в адаптер следующие запросы:

- Запрос только с одним инцидентом
- Запрос с инцидентом, связанным с узлом связью типа "connection"
- Запрос с инцидентом, связанным с узлом связью типа "membership"
- Запрос с инцидентом, связанным с бизнес-службой связью типа "connection"
- Запрос с инцидентом, связанным с бизнес-службой связью типа "membership"

Примечание. Все запросы, которые отправляются механизмом объединения в адаптер и которые должны быть сохранены, имеют имя **User mapping union FTQL**.

После обработки результатов запроса **User mapping union FTQL** выполняются другие вызовы для извлечения атрибутов объектов. Эти вызовы содержат запрос с именем **objects layout**. Механизм объединения попытается получить все атрибуты ЭК, но соединителю необязательно предоставлять их все; достаточно отправить лишь те, которые запрашиваются файлом сопоставления.

Причиной для отправки одного и того же запроса с различными связями является то, что в TQL-запросе имеется связь типа **managed_relationship** между узлом и инцидентом/бизнес-службой и инцидентом, но единственными допустимыми связями при попытке соединения этих типов ЭК являются "connection" и "membership".

```
<tql:link from="incident_12" to="node_10050" class="connection" name="connection_1" id="1"/>
<tql:link from="incident_12" to="node_1000050" class="connection" name="connection_2" id="2"/>
<tql:link from="datacenter_20050" to="node_10050" class="composition" name="composition_30050" id="30050"/>
```

```
<tql:link from="incident_12" to="node_10050" class="membership" name="membership_1" id="1"/>
<tql:link from="incident_12" to="node_1000050" class="membership" name="membership_2" id="2"/>
<tql:link from="datacenter_20050" to="node_10050" class="composition" name="composition_30050" id="30050"/>
```

При использовании этого подхода нам всего лишь необходимо задать один статический TQL с общей связью типа **managed_relationship** вместо того, чтобы задавать два практически идентичных TQL с различными типами связей.

```
2014-08-07 16:43:55,231 [AdHoc:AD_HOC_TASK_PATTERN_ID-179-1407419035224] TRACE -  >> Received federation call with the following query:
2014-08-07 16:43:55,231 [AdHoc:AD_HOC_TASK_PATTERN_ID-179-1407419035224] TRACE -  >>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tql:query name="User mapping union FTQL" is-live="true" priority="low" xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRuleT
    <tql:node class="incident" name="incident_16" id="16">
        <tql:where>
            <tql:links>
                <tql:or>
                    <tql:link-ref name="membership_1"/>
                    <tql:link-ref name="membership_2"/>
                </tql:or>
            </tql:links>
        </tql:where>
    </tql:node>
    <tql:node class="business_service" name="business_service_10050" id="10050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="name"/>
                    <tql:list type="string">
                        <tql:string>MyBusServ</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="name"/>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:node class="business_service" name="business_service_1000050" id="1000050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="global_id"/>
                    <tql:list type="string">
                        <tql:string>183ad8038405644e67aba201334714ea</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:link from="incident_16" to="business_service_10050" class="membership" name="membership_1" id="1"/>
    <tql:link from="incident_16" to="business_service_1000050" class="membership" name="membership_2" id="2"/>
</tql:query>
```

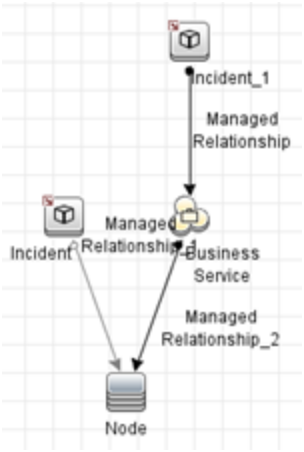
Эти статические TQL-запросы должны быть предоставлены адаптером. Для содействия в разработке адаптера TQL-запросы, отправляемые Federation Framework, записываются в файл **fcmdb.push.mapping.log** (с включенным уровнем журнала TRACE). Чтобы облегчить разработку, используйте параметр `<adapter-setting name="dev.mode">true</adapter-setting>`. Если для этого параметра после выполнения запроса объединения устанавливается значение true, Framework автоматически создаст пустой результат объединения для текущего несогласованного TQL-запроса.

Пример запроса объединения из журнала **fcmdb.push.mapping.log**:

```
2014-08-07 16:43:55,231 [AdHoc:AD_HOC_TASK_PATTERN_ID-179-1407419035224] TRACE -  >> Received federation call with the following query:
2014-08-07 16:43:55,231 [AdHoc:AD_HOC_TASK_PATTERN_ID-179-1407419035224] TRACE -  >>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tql:query name="User mapping union FTQL" is-live="true" priority="low" xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRuleD
    <tql:node class="incident" name="incident_16" id="16">
        <tql:where>
            <tql:links>
                <tql:or>
                    <tql:link-ref name="membership_1"/>
                    <tql:link-ref name="membership_2"/>
                </tql:or>
            </tql:links>
        </tql:where>
    </tql:node>
    <tql:node class="business_service" name="business_service_10050" id="10050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="name"/>
                    <tql:list type="string">
                        <tql:string>MyBusServ</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="name"/>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:node class="business_service" name="business_service_1000050" id="1000050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="global_id"/>
                    <tql:list type="string">
                        <tql:string>183ad8038405644e67aba201394714ea</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:link from="incident_16" to="business_service_10050" class="membership" name="membership_1" id="1"/>
    <tql:link from="incident_16" to="business_service_1000050" class="membership" name="membership_2" id="2"/>
</tql:query>
```

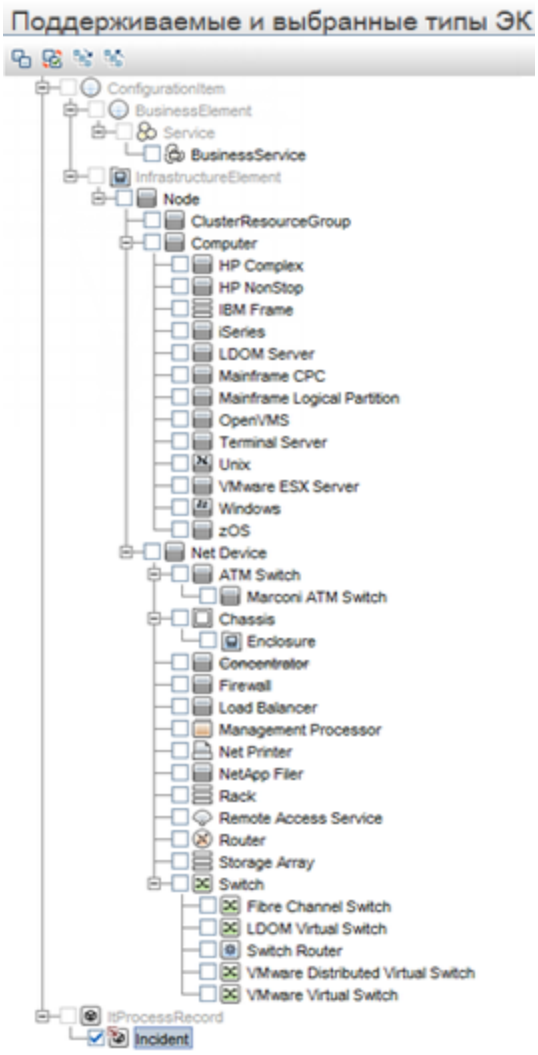
Пример настройки объединения

В примере будет использован следующий TQL-запрос объединения:



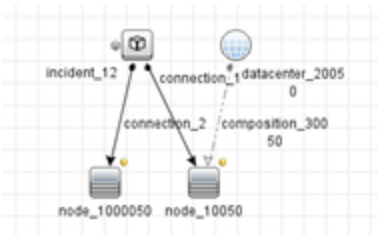
Для данного TQL-запроса адаптер должен объявить поддерживаемые классы в XML-файле параметров адаптера, расположенном в архивном файле адаптера в папке **discoveryPatterns**. Поддерживаемые классы — **node**, **incident** и **business_service**.

В Студии интеграции инцидент необходимо выбрать на вкладке **Объединение**, как показано ниже:

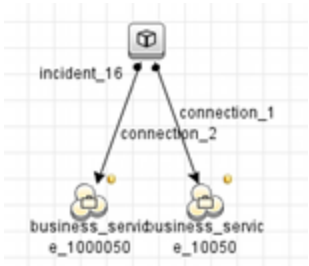


Для данного TQL-запроса необходимо иметь следующие три TQL-запроса в адаптере:

Incident



Incident linked to Node



Incident linked to business_service

Подробнее о процедуре получения статических TQL-запросов см. в разделе "Настройка запросов объединения из файлов журнала" на странице 309. Несмотря на то, что эти TQL-

запросы имеют условия, которые зависят от представленных в UCMDB данных, это не должно повлиять на структуру TQL-запросов, а также на ход выполнения сопоставления.

Для каждого из этих TQL-запросов требуется файл сопоставления в адаптере:

- Incident

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resource:XmlResourceWrapper xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRuleDefiniti
    <resource xsi:type="tql:Query" name="SM_Incident" is-active="false" priority="low" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<tql:node class="incident" name="incident_12" id="12"/>
    </resource>
    <resourceBundle>integration_tqls_bundle</resourceBundle>
</resource:XmlResourceWrapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../generic-adapter.xsd">
    <!-- add scheme reference -->
    <info>
        <source name="Dummy" version="10.0" vendor="HP"/>
        <target name="UCMDB" version="10.20" vendor="HP"/>
    </info>

    <!--
    This mapping converts the Root entities received from the population connector to a "node" item in UCMDB,
    The output of this mapping must match the indicated query (TQL), "Dummy Query"
    -->

    <target_entities>
        <!--The query name must match the one selected in the UI-->
        <source_instance query-name="SM_Incident" root-element-name="incident_12">
            <!-- need to match case in UCMDB TQL -->
            <target_entity name="incident_12">
                <target_mapping name="name" datatype="STRING" value="incident_12['name']"/>
                <target_mapping name="description" datatype="STRING" value="incident_12['description']"/>
                <target_mapping name="reference_number" datatype="STRING" value="incident_12['reference_number']"/>
            </target_entity>
        </source_instance>
    </target_entities>

</integration>
```

- Incident linked to Node

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resource:XmlResourceWrapper xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRuleDefinition" xmlns:resou
    <resource xsi:type="tql:Query" name="SM_Node" is-active="false" priority="low" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<tql:node class="node" name="node_10050" id="10050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="name"/>
                    <tql:list type="string">
                        <tql:string>mynode</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
            <tql:links>
                <tql:link-ref name="composition_30050" min-occurs="0"/>
                <tql:link-ref name="connection_1"/>
            </tql:links>
        </tql:where>
        <tql:content>
            <tql:properties...>
        </tql:content>
    </tql:node>
    <tql:node class="node" name="node_1000050" id="1000050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="global_id"/>
                    <tql:list type="string">
                        <tql:string>9b360a1f42eaf7cff793feb57c88f098</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content...>
    </tql:node>
    <tql:node class="incident" name="incident_12" id="12">
        <tql:where>
            <tql:ids>
                <tql:id>GA%0Aincident%0A3%0Adescription%3DSTRING%3Dtest_incident2%0Aname%3DSTRING%3DIncident2%0Areference_number%3DSTRING%3D101%0A</tql:id>
                <tql:id>GA%0Aincident%0A3%0Adescription%3DSTRING%3Dtest_incident%0Aname%3DSTRING%3DIncident%0Areference_number%3DSTRING%3D100%0A</tql:id>
            </tql:ids>
            <tql:links>
                <tql:or>
                    <tql:link-ref name="connection_1"/>
                    <tql:link-ref name="connection_2"/>
                </tql:or>
            </tql:links>
        </tql:where>
    </tql:node>
    <tql:node class="datacenter" name="datacenter_20050" id="20050"/>
    <tql:link from="incident_12" to="node_10050" class="managed_relationship" name="connection_1" id="1"/>
    <tql:link from="incident_12" to="node_1000050" class="managed_relationship" name="connection_2" id="2"/>
    <tql:link from="datacenter_20050" to="node_10050" class="composition" name="composition_30050" id="30050"/>
    </resource>
    <resourceBundle>integration_tqls_bundle</resourceBundle>
</resource:XmlResourceWrapper>
```

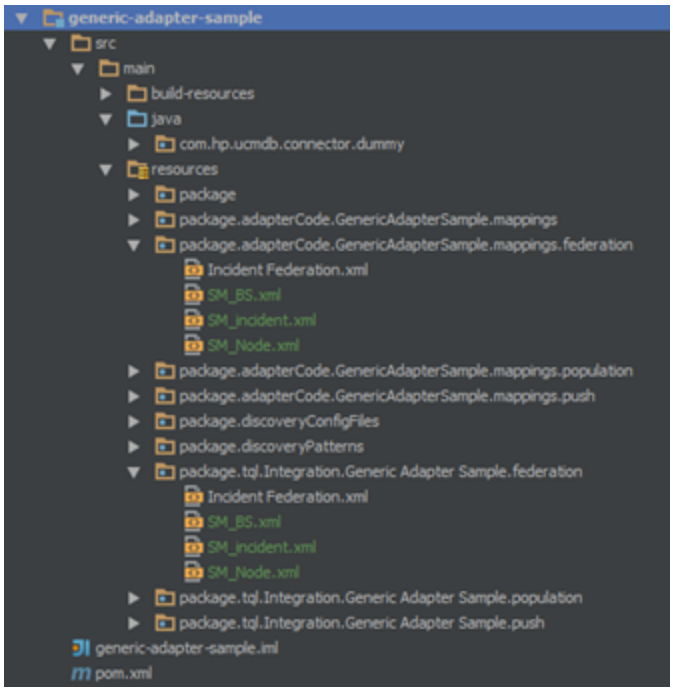
```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../generic-adapter.xsd">
    <!-- add scheme reference -->
    <info>
        <source name="Dummy" version="10.0" vendor="HP"/>
        <target name="UCMDB" version="10.20" vendor="HP"/>
    </info>
    <target_entities>
        <!--The query name must match the one selected in the UI-->
        <source_instance query-name="SM_Node" root-element-name="Computer">
            <!-- need to match case in UCMDB TQL -->
            <target_entity name="node_1000050">
                <target_mapping name="name" datatype="STRING" value="Computer['name']"/>
            </target_entity>


            <for-each-source-entity count-index="i" source-entities="Computer.incident_12" var-name="currIP">
                <target_entity name="incident_12">
                    <target_mapping name="name" datatype="STRING" value="currIP['name']"/>
                    <target_mapping name="description" datatype="STRING" value="currIP['description']"/>
                    <target_mapping name="reference_number" datatype="STRING" value="currIP['reference_number']"/>
                </target_entity>
            </for-each-source-entity>


        </source_instance>
    </target_entities>
</integration>
```

- Incident to business_service

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<resource:XmlResourceWrapper xmlns:ns4="http://www.hp.com/ucmdb/1-0-0/ViewDefinition" xmlns:ns3="http://www.hp.com/ucmdb/1-0-0/PolicyRu
    <resource xsi:type="tql:Query" name="SM_BS" is-active="false" priority="low" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <tql:node class="incident" name="incident_16" id="16">
        <tql:where>
            <tql:links>
                <tql:or>
                    <tql:link-ref name="connection_1"/>
                    <tql:link-ref name="connection_2"/>
                </tql:or>
            </tql:links>
        </tql:where>
    </tql:node>
    <tql:node class="business_service" name="business_service_10050" id="10050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="name"/>
                    <tql:list type="string">
                        <tql:string>MyBusServ</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="name"/>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
    <tql:node class="business_service" name="business_service_1000050" id="1000050">
        <tql:where>
            <tql:properties>
                <tql:in>
                    <tql:property-ref name="global_id"/>
                    <tql:list type="string">
                        <tql:string>183ad8038405644e67aba201334714ea</tql:string>
                    </tql:list>
                </tql:in>
            </tql:properties>
        </tql:where>
        <tql:content>
            <tql:properties>
                <tql:property name="global_id"/>
                <tql:property name="TenantOwner"/>
                <tql:property name="TenantsUses"/>
            </tql:properties>
        </tql:content>
    </tql:node>
        <tql:link from="incident_16" to="business_service_10050" class="managed_relationship" name="connection_1" id="1"/>
        <tql:link from="incident_16" to="business_service_1000050" class="managed_relationship" name="connection_2" id="2"/>
    </resource>
    <resourceBundle>integration_tqls_bundle</resourceBundle>
</resource:XmlResourceWrapper>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../generic-adapter.xsd">
    <!-- add scheme reference -->
    <info>
        <source name="Dummy" version="10.0" vendor="HP"/>
        <target name="UCMDB" version="10.20" vendor="HP"/>
    </info>
    <target_entities>
        <!--The query name must match the one selected in the UI-->
        <source_instance query-name="SM_BS" root-element-name="BS">
            <!-- need to match case in UCMDB TQL -->
            <target_entity name="business_service_10050">
                <target_mapping name="name" datatype="STRING" value="BS['name']"/>
            </target_entity>
            <for-each-source-entity count-index="i" source-entities="BS.incident_16" var-name="currIP">
                <target_entity name="incident_16">
                    <target_mapping name="name" datatype="STRING" value="currIP['name']"/>
                    <target_mapping name="description" datatype="STRING" value="currIP['description']"/>
                    <target_mapping name="reference_number" datatype="STRING" value="currIP['reference_number']"/>
                </target_entity>
            </for-each-source-entity>
        </source_instance>
    </target_entities>
</integration>
```

Эти TQL-запросы и файлы сопоставления должны быть представлены в адаптере, как показано ниже:



Выверка

Когда Generic Adapter Framework используется для заполнения или объединения данных, ЭК всегда должны иметь необходимые данные выверки для приема в UCMDB. При заполнении таких типов ЭК, как Running Software, для которых требуется ЭК контейнера, всегда следует заполнять поля контейнера (например, **root_container_name** и **product_name**) и ЭК контейнера (например, **Node**). Для заполнения ЭК, которые зависят от корневого контейнера, ЭК, его корневой контейнер и связь между ними должны создаваться на одном и том же этапе (явным заполнением связей или автоматическим заполнением связей между двумя ЭК).

Кроме того, при сопоставлении заполненных/объединенных ЭК следует рассмотреть возможность сопоставления атрибута **global_id**, поскольку это существенно поможет механизму выверки UCMDB и гарантирует точную выверку ЭК.

API общего адаптера

API-интерфейс, разворачиваемый Generic Adapter Framework:

```
<UCMDB_Server>\lib\push-interfaces.jar
<UCMDB_Server>\lib\integrationFramework\GenericAdapter\generic-adapter-api-factory.jar
```

Для разработки экземпляра общего адаптера также может потребоваться API объединения:

```
<UCMDB_Server>\lib\federation-api.jar
```

API-интерфейсы указателей ресурсов

API-интерфейсы указателей ресурсов могут применяться при изменении заданий общих адаптеров. Используйте API-интерфейсы указателей ресурсов (общих и заполнения) для поиска ресурсов адаптера, которые связаны с TQL-запросом выбранного задания.

Ниже представлено изображение API-интерфейса указателя общих ресурсов в Java-интерфейсе GenericConnector:

```
/**
 * This methods reports general resources used by the adapter.<br>
 * This allows editing these resources directly from the Integration Studio.
 *
 *
 * @param env the Adapter's environment
 * @param input  the requested information
 * @param output the returned resources
 * @see com.hp.ucmdb.federationspi.adapter.resource.GeneralResourcesLocator
 */
void locateGeneralResources(DataAdapterEnvironment env, LocateGeneralResourcesInput input, LocateGeneralResourcesOutput output);
```

Ниже представлено изображение API-интерфейса указателя ресурсов заполнения в Java-интерфейсе PopulationAdapterConnector:

```
/**
 * This methods reports population queries resources used by the adapter.<br>
 * This allows editing these resources directly from the Integration Studio.
 *
 * @param input  the requested information
 * @param output the returned resources
 * @see com.hp.ucmdb.federationspi.adapter.resource.PushQueriesResourceLocator
 */
void locatePopulationQueriesResources(DataAdapterEnvironment env, LocatePopulationQueriesResourcesInput input,
                                      LocatePopulationQueriesResourcesOutput output);
```

Чтобы просмотреть ресурсы, связанные с TQL-запросом задания:

1. В Студии интеграции выберите точку интеграции.
2. На панели "Задания интеграции" выберите задание и нажмите **Изменить ресурсы запроса**  .

Создание пакета общего адаптера

Пакет общего адаптера схож с пакетом расширенного адаптера принудительной отправки. Для создания ZIP-архива начальной структуры рекомендуется скопировать существующий пакет общего адаптера и настроить его надлежащим образом. Подробнее о пакете адаптера см. в разделе "Принудительная отправка данных с помощью общего адаптера" на странице 280.

Различия между пакетом расширенного адаптера принудительной отправки и пакетом общего адаптера:

- Различия в XML адаптера
  - Класс адаптера изменен с **PushAdapter** на **GenericAdapter**:

```
<className>com.hp.ucmdb.adapters.push.PushAdapter</className>
```

```
<className>com.hp.ucmdb.adapters.GenericAdapter</className>
```

  - Возможности адаптера включают в себя объединение

```
<support-replicatioin-data>
    <source>
        <push-back-ids/>
        <instance-based-data/>
        <population-queries-resources-locator/>
    </source>
```

  - Наряду с определением соединителя заполнения, выполняемым параметром адаптера:

```
<adapter-setting name="PopulationConnector.class.name">com.hp.ucmdb.connector.dummy.DummyPopulationConnector</adapter-setting>
```

    общий адаптер (использующий функцию объединения) также требует определения класса соединителя принудительной отправки:

```
<adapter-setting name="PushConnector.class.name">com.hp.ucmdb.connector.dummy.DummyPushConnector</adapter-setting>
```

- Папки файлов сопоставления

  В отличие от расширенного адаптера принудительной отправки (который требует, чтобы его файлы сопоставления располагались в папке **<adapter_package_zip>/adapterCode/<*adapter_name*>/mappings**) сопоставления общего адаптера должны находиться в трех отдельных папках (для принудительной отправки, для заполнения и для объединения). Требуемые папки:

  <adapter_package_zip>/adapterCode/<*adapter_name*>/mappings/push
  <adapter_package_zip>/adapterCode/<*adapter_name*>/mappings/population
  <adapter_package_zip>/adapterCode/<*adapter_name*>/mappings/federation

  где **<adapter_package_zip>** относится к ZIP-архиву, который будет создан для пакета общего адаптера.

Примечание. Несмотря на то, что общие адаптеры поддерживают все три типа синхронизации данных (принудительную отправку, заполнение и объединение), отдельный общий адаптер может выбрать предоставление только подмножества этих типов.

Что необходимо учесть при создании нового адаптера из существующего адаптера

- **TestAdapter\discoveryPatterns\TestAdapter.xml**
- Измените файл **TestAdapter.xml**:

- `<pattern xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="TestAdapter" xsi:noNamespaceSchemaLocation="../../Patterns.xsd" description="…" schemaVersion="9.0" displayName="…">`
- `<adapter-id>TestAdapter</adapter-id>`

- ZIP-файл, содержащий новый адаптер, должен иметь то же имя, что и сам адаптер — TestAdapter.

Создание пакета адаптера

Пакет адаптера должен содержать следующие папки:

- **adapterCode.** Создайте внутри этой папки папку **PushExampleAdapter**, в которой будет содержаться файл .jar, созданный из PushExampleAdapter.java. В папке также будет содержаться папка **mappings**, в которую можно поместить созданный ранее файл сопоставлений, **computerIPMapping.xml**. Там же будет находиться папка **scripts** с файлом **PushFunctions.groovy**.
- **discoveryConfigFiles.** Для файлов конфигурации, например кодов ошибок при сообщении об ошибках с помощью UpdateResult. В этом примере папка пуста.
- **discoveryPatterns.** Для **push_example_adapter.xml**.
- **tql.** Для TQL-запроса, созданного в этом примере. Эта папка не является обязательной, однако TQL-запрос автоматически создается при развертывании пакета.

Включение/отключение проверки атрибутов и связей на уровне адаптера

Проверку атрибутов и связей на уровне адаптера для общих адаптеров можно включать или отключать добавлением следующего параметра:

```
<adapter-settings>

        <adapter-setting
name="enable.attributes.links.validation">true</adapter-setting>

</adapter-settings>
```

Чтобы включить проверку атрибута и связи на уровне адаптера, установите для параметра адаптера **enable.attributes.links.validation** значение **true**.

Чтобы отключить проверку атрибута и связи на уровне адаптера, установите для параметра адаптера **enable.attributes.links.validation** значение **false**.

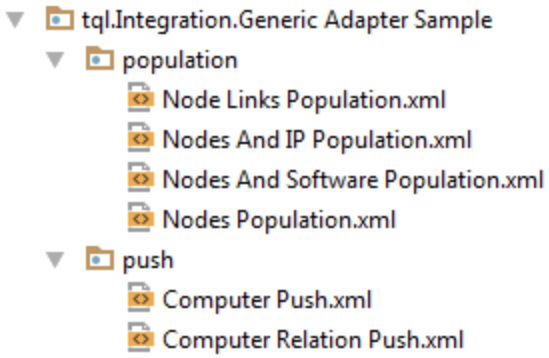
Примечание: Если этот параметр отсутствует, его значением по умолчанию будет **true**, и это означает, что проверка атрибута и связи по умолчанию включена.

TQL-запросы заполнения

TQL-запросы, используемые для заданий заполнения, должны быть включены в ZIP-архив общего адаптера и развернуты вместе с адаптером в UCMDB. Указанный TQL-запрос

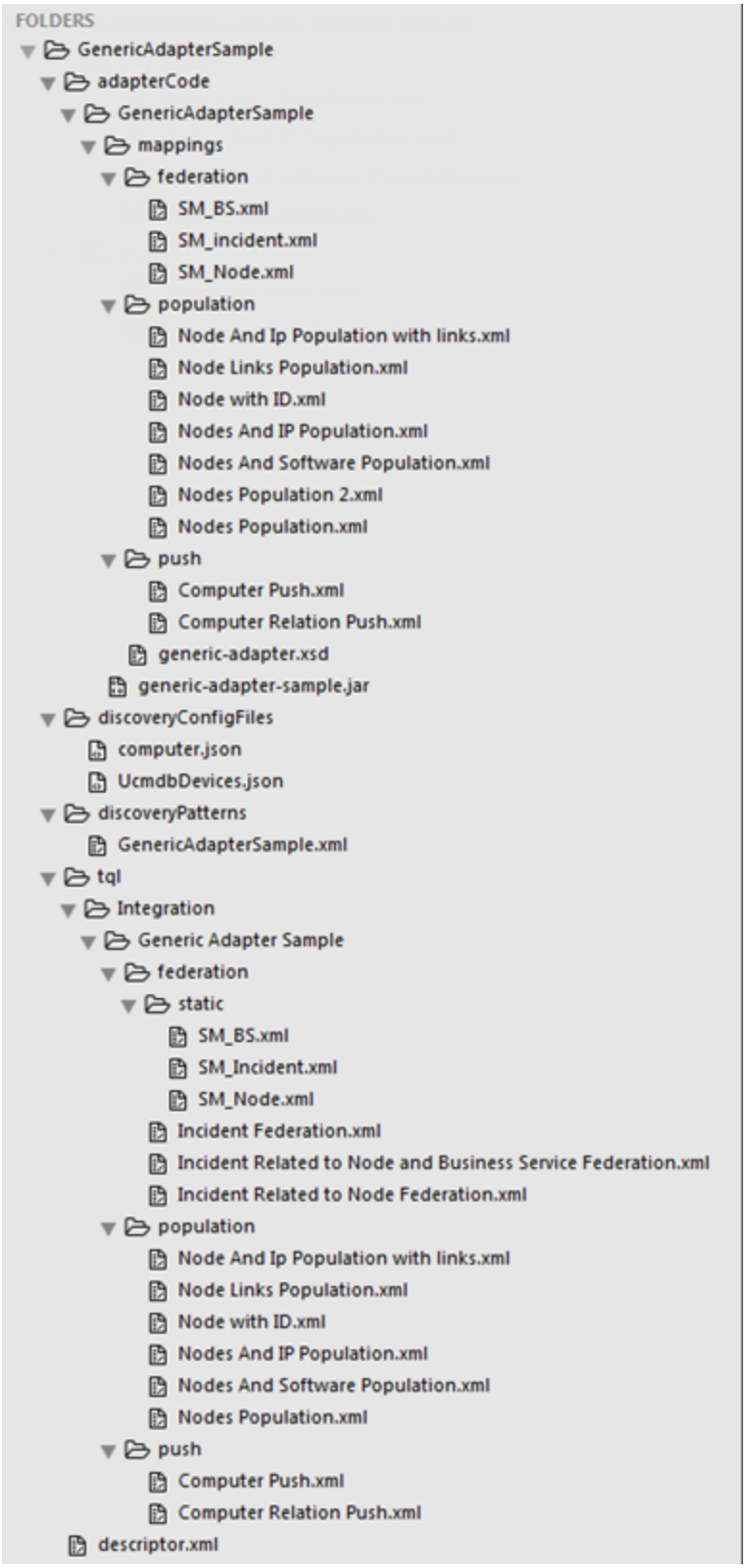
должен существовать в UCMDB, когда выполняется запрос объединения в потоке объединения.

Эти TQL-запросы должны быть включены в **<zip>/tql/<folder_1>/../<folder_n>**. Ниже приведен пример структуры папок:



Несмотря на то, что TQL-запросы заполнения располагаются в папке, указанной выше, соединитель заполнения также должен подтвердить поддерживаемые TQL-запросы заполнения в соответствующем методе из интерфейса Java. Подробнее см. в разделе "Соединитель заполнения" на странице 296.

Пример пакета


FOLDERS
GenericAdapterSample
adapterCode
GenericAdapterSample
mappings
federation
SM_BS.xml
SM_incident.xml
SM_Node.xml
population
Node And Ip Population with links.xml
Node Links Population.xml
Node with ID.xml
Nodes And IP Population.xml
Nodes And Software Population.xml
Nodes Population 2.xml
Nodes Population.xml
push
Computer Push.xml
Computer Relation Push.xml
generic-adapter.xsd
generic-adapter-sample.jar
discoveryConfigFiles
computer.json
UcmdbDevices.json
discoveryPatterns
GenericAdapterSample.xml
tql
Integration
Generic Adapter Sample
federation
static
SM_BS.xml
SM_Incident.xml
SM_Node.xml
Incident Federation.xml
Incident Related to Node and Business Service Federation.xml
Incident Related to Node Federation.xml
population
Node And Ip Population with links.xml
Node Links Population.xml
Node with ID.xml
Nodes And IP Population.xml
Nodes And Software Population.xml
Nodes Population.xml
push
Computer Push.xml
Computer Relation Push.xml
descriptor.xml

Различия в сопоставлении принудительной отправкой и заполнением

Хотя файлы сопоставления принудительной отправкой и заполнением имеют одну и ту же XML-схему, сами файлы интерпретируются несколько по-разному. Подробнее см. в разделе "Справочные материалы по XML-схеме общего адаптера" на следующей странице.

В следующем примере сопоставления принудительной отправкой представлена такая интерпретация: взять результаты TQL-запроса "Computer Push" (выполненного в UCMDB), представить в древовидной структуре Root и создать объект amComputer, который затем будет отправлен в AM.

```
<target_entities>
    <!--The query name must match the one selected in the UI-->
    <source_instance query-name="Computer Push" root-element-name="Root">
        <target_entity name="amComputer">
            <target_mapping name="TcpIpHostName" datatype="STRING" value="Root['name']"/>
            <target_mapping name="ComputerDesc" datatype="STRING" value="Root['os_description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

В следующем примере сопоставления заполнением представлена такая интерпретация: взять результаты TQL-запроса "Nodes Population" (выполненного во внешней системе), представить в древовидной структуре PC и создать объект UCMDB Root (типа Node, как указано TQL-запросом), который затем будет отправлен в UCMDB.

```
<target_entities>
    <!--The query name must match the one selected in the UI-->
    <source_instance query-name="Nodes Population" root-element-name="PC">
        <!-- need to match case in UCMDB class model-->
        <target_entity name="Root">
            <target_mapping name="name" datatype="STRING" value="PC['name']"/>
            <target_mapping name="description" datatype="STRING" value="PC['description']"/>
        </target_entity>
    </source_instance>
</target_entities>
```

Файлы журнала общего адаптера

Для устранения неполадок и отладки используйте следующее:

- Отрегулируйте уровни ведения журналов в следующих файлах (установите для переменной *loglevel* значение TRACE, чтобы получать самые подробные результаты):
  - **<UCMDB_DataFlowProbe>\conf\log\fcmdb.push.properties**

    <UCMDB_DataFlowProbe> — это каталог установки зонда потока данных UCMDB.

- **<UCMDB_Server>\conf\log\reconciliation.properties**

  <UCMDB_Server> — это каталог установки UCMDB Server.

- Проанализируйте следующие файлы журнала общего адаптера:
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.all.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.configuration.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.connector.all.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.connector.configuration.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.mapping.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\fcmdb.push.all.log**
- Проанализируйте следующие общие файлы журнала:
  - **<UCMDB_DataFlowProbe>\runtime\log\probe-error.log**
  - **<UCMDB_DataFlowProbe>\runtime\log\WrapperProbeGw.log**
  - **<UCMDB_Server>\runtime\log\error.log**
  - **<UCMDB_Server>\runtime\log\cmdb.reconciliation.log**

Адаптеры, использующие Generic Adapter Framework

В качестве образцов для разработки пользовательского общего адаптера следует использовать следующие адаптеры, которые предоставляются вместе с UCMDB в рекомендациях по внедрению:

- Адаптер Asset Manager
- Адаптер Service Manager

Справочные материалы по XML-схеме общего адаптера

XML-схема общего адаптера находится в файле **cmdb.jar** в директории **schema**. Ссылка на файл схемы должна быть прописана при записи файлов сопоставления общего адаптера во внешних редакторах. Полный путь для XSD-файла:

<UCMDB_Server_Install_dir>/lib/cmdb.jar/schema/generic-adapter.xsd

Часть II: Использование API-интерфейсов

Глава 9: Введение в API-интерфейсы

Данная глава включает:

- Обзор API-интерфейсов 328

Обзор API-интерфейсов

В HP Universal CMDB доступны следующие API-интерфейсы:

- **UCMDB Java API.** Описание извлечения данных и вычислений сторонними средствами с помощью Java API, а также записи данных в UCMDB (Universal Configuration Management Database). Дополнительные сведения см. в разделе "API-интерфейс HP Universal CMDB" на странице 329.
- **API-интерфейс веб-службы UCMDB.** Запись определений элементов конфигурации и топологических связей в UCMDB, а также получение этой информации с помощью TQL-запросов и специальных запросов. Подробнее см. "API-интерфейс веб-служб HP Universal CMDB" на странице 337.
- **Управление потоком данных — API Java** Обеспечивает управление зондами, заданиями, триггерами и учетными данными для Управления потоком данных. Дополнительные сведения см. в разделе "Управление потоком данных — API Java" на странице 371.
- **API веб-службы управления потоком данных** Обеспечивает управление зондами, заданиями, триггерами и учетными данными для Управления потоком данных. Дополнительные сведения см. в разделе "API веб-службы управления потоком данных" на странице 373.

Примечание. Для наиболее эффективного использования API-интерфейсов рекомендуется изучить онлайн-документацию. В PDF-версии отсутствуют ссылки на документацию по API, которая существует в формате html.

Глава 10: API-интерфейс HP Universal CMDB

Данная глава включает:

- Обозначения 329
- Использование API-интерфейса HP Universal CMDB 329
- Общая структура приложения 330
- Копирование JAR-файла API-интерфейса в каталог Classpath 333
- Создание пользователя интеграции 333
- API-интерфейс UCMDB — сценарии использования 335
- Примеры 336

Обозначения

В этой главе используются следующие условные обозначения:

- UCMDB — это сама универсальная база данных управления конфигурациями. HP Universal CMDB означает само приложение.
- Элементы и аргументы методов UCMDB приводятся, если указаны в интерфейсах.

Полную документацию по доступным API-интерфейсам см. в разделе HP UCMDB API Reference.

Эти файлы находятся по следующему пути:

\\<UCMDB root directory>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Использование API-интерфейса HP Universal CMDB

Примечание. Эту главу следует использовать в сочетании с документом API Javadoc, доступным в библиотеке документации.

API-интерфейс HP Universal CMDB используется для интеграции приложений с Universal CMDB (CMDB). API-интерфейс предоставляет методы для решения следующих задач:

- Добавление, удаление и обновление ЭК и связей в CMDB;
- Получение информации о модели классов;
- Извлечение информации из истории UCMDB;

- Выполнение сценариев «что если»;
- Получение информации об ЭК и связях.

Как правило, для методов получения информации об ЭК и связях используется язык TQL. Подробнее см. в разделе Topology Query Language в документе *Руководство по моделированию в HP Universal CMDB*.

Пользователи API-интерфейса HP Universal CMDB должны обладать следующими знаниями:

- Язык программирования Java.
- HP Universal CMDB

Этот раздел охватывает следующие темы:

- "Сценарии использования API-интерфейса" ниже
- "Права доступа" ниже

Сценарии использования API-интерфейса

API-интерфейс используется для выполнения ряда бизнес-требований. Например, сторонняя система может запрашивать информацию о доступных ЭК в модели классов. См. дополнительные сценарии использования в разделе "API-интерфейс UCMDB — сценарии использования" на странице 335.

Права доступа

Администратор предоставляет учетные данные для подключения к API-интерфейсу. Клиент API-интерфейса должен иметь имя и пароль, заданные для пользователя интеграции в CMDB. Такие учетные записи представляют не пользователей CMDB (людей), а приложения, которые подключаются к CMDB.

Помимо этого, для входа в систему пользователю необходимо право **доступа к SDK**.

> **Внимание!** Кроме того, клиент API-интерфейса может работать и с обычными пользователями, если у них есть право аутентификации через API. Однако использовать этот вариант не рекомендуется.

Подробнее см. в разделе "Создание пользователя интеграции" на странице 333.

Общая структура приложения

Существует только одна статическая фабрика — UcmdbServiceFactory. Эта фабрика является точкой входа в приложение. Фабрика UcmdbServiceFactory предоставляет доступ к методам getServiceProvider. Эти методы возвращают экземпляр интерфейса **UcmdbServiceProvider**.

Клиент создает другие объекты с помощью методов интерфейса. Например, чтобы создать новое определение запроса, клиент выполняет следующие действия:

1. Получает службу запроса от главного объекта службы CMDB.
2. Получает объект фабрики запросов от объекта службы.
3. Получает новое определение запроса от фабрики.

```
UcmdbServiceProvider provider =
   UcmdbServiceFactory.getServiceProvider(HOST_NAME, PORT);
UcmdbService ucmdbService =
   provider.connect(provider.createCredentials(USERNAME,
   PASSWORD), provider.createClientContext("Test"));
TopologyQueryService queryService = ucmdbService.getTopologyQueryService();
TopologyQueryFactory factory = queryService.getFactory();
QueryDefinition queryDefinition = factory.createQueryDefinition("Test
Query");
queryDefinition.addNode("Node").ofType("host");
Topology topology = queryService.executeQuery(queryDefinition);
System.out.println("There are " + topology.getAllCIs().size() + " hosts in
uCMDB");
```

Службы, доступные в **UcmdbService**:

Методы служб	Использование
getAuthorizationModelService	Операции авторизации (Создание пользователей, групп пользователей, а также назначение ролей пользователям и группам, и т.д.).
getClassModelService	Информация о типах ЭК и связей
getConfigurationService	Управление настройками инфраструктуры, для конфигурации сервера
getDataStoreMgmtService	Сведения о хранении данных запроса, в том числе об объединении ЭК и атрибутов.
getDDMConfigurationService	Настройка системы Управления потоком данных
getDDMManagementService	Анализ и отображение хода выполнения, результатов и ошибок системы Управления потоком данных
getDDMZoneService	Импорт и экспорт зон управления (а также операций).
getHistoryService	Информация об истории отслеживаемых ЭК (изменение, удаление и т.д.)
getImpactAnalysisService	Выполнение сценария анализа влияния (другое название: **корреляция**).
getLicensingService	Запрос сведений о лицензиях, установленных в системе.
getMultipleCMDBService	Преобразование глобальных идентификаторов в идентификаторы UCMDB.

Методы служб	Использование
getMultiTenancyService	Создание, чтение, обновление и удаление владельцев.
getPersistencyService	Преобразование двоичных данных в пары ключ-значение.
getQueryManagementService	Управление доступом к запросам — сохранение, удаление и вывод существующих. Также предоставляет проверку запросов и обнаружение зависимостей запросов.
getReconciliationService	Возможности идентификации и слияния.
getResourceBundleManagementService	Добавление тегов к ресурсам (объединение служб в пакеты). Обеспечивает явное создание новых тегов и их удаление из всех отмеченных ресурсов.
getResourceManagementService	Развертывание пакетов ресурсов (TQL-запросов, представлений, пользователей и т.д.) в системе.
getSecurityService	Проверка достоверности учетных данных.
getServerService	Запрос базовых сведений о системе.
getSnapshotService	Службы для управления снимками (получение, сохранение, сравнение и т.д.)
getSoftwareSignatureService	Указывает элементы ПО для обнаружения системой управления потоками данных
getStateService	Службы для управления состояниями (список, добавление, удаление и т.д.)
getSystemHealthService	Службы контроля работоспособности системы (базовые индикаторы производительности, метрики емкости и доступности)
getTopologyQueryService	Получение информации об IT Universe
getTopologyUpdateService	Изменение информации в IT Universe
getUcmdbVersion	Запрос сведений о версиях и сборках UCMDB и Content Pack.
getViewArchiveService	Службы архивирования результатов представления. Обеспечивает сохранение текущего результата представления и возвращение ранее сохраненных результатов.
getViewService	Служба выполнения представлений (выполнение определения, выполнение сохраненных) и служба управления (сохранение, удаление, вывод

Методы служб	Использование
	существующих). Также предоставляет проверку представлений и обнаружение зависимостей.

Клиент обменивается данными с сервером по протоколу HTTP (S).

Копирование JAR-файла API-интерфейса в каталог Classpath

Для использования этого набора API-интерфейсов необходим файл **ucmdb-api.jar**. Чтобы загрузить файл, введите в адресную строку браузера `http://<localhost>:8080`, где `localhost` — машина, на которой установлена UCMDB, а затем нажмите на ссылку **API Client Download**.

Скопируйте **JAR**-файл в каталог classpath перед компиляцией или запуском приложения.

Примечание. Для использования JAR-файла UCMDB Java API необходимо установить JRE версии 6 и выше.

Создание пользователя интеграции

Для интеграции UCMDB с другими продуктами можно создать специальную учетную запись пользователя. Этот пользователь позволяет продукту, использующему клиентский SDK UCMDB, проходить аутентификацию в SDK сервера и выполнять API-интерфейсы. Приложения, созданные с помощью этого API-интерфейса, должны войти в систему, используя учетные данные пользователя интеграции.

Внимание! Кроме того, подключиться можно и от имени обычного пользователя UCMDB (например, admin). Однако использовать этот вариант не рекомендуется. Чтобы подключиться от имени пользователя UCMDB, необходимо дать этому пользователю право аутентификации через API.

Создание пользователя интеграции:

1. Запустите веб-браузер и введите адрес сервера:
   `http://localhost:8080/jmx-console`
   Возможно, потребуется ввести имя пользователя и пароль для входа в систему.
2. В разделе `UCMDB` нажмите **service=UCMDB Authorization Services**.
3. Найдите операцию **createUser**. Данный метод принимает следующие параметры:
   - **customerId**. Идентификатор заказчика.
   - **username**. Имя пользователя интеграции.
   - **userDisplayName**. Отображаемое имя пользователя интеграции.

- **userLoginName**. Имя для входа пользователя интеграции.
- **password**. Пароль пользователя интеграции.

  Политика паролей по умолчанию требует, чтобы пароль UCMDB содержал хотя бы по одному символу из следующих четырех типов:
  - Буква в верхнем регистре
  - Буква в нижнем регистре
  - Цифра
  - Символы ,\:/. _?&%=+-[]()|

  Также необходимо, чтобы пароль соответствовал требованиям к длине, которые устанавливаются настройкой **Минимальной длины пароля**.

4. Нажмите кнопку **Invoke**.
5. В окружении без множественной аренды найдите метод **setRolesForUser** и укажите следующие параметры:
   - **userName**. Имя пользователя интеграции.
   - **роли.** SuperAdmin.

   Нажмите кнопку **Invoke**.
6. В системе с множественной арендой найдите метод **grantRolesToUserForAllTenants** и введите следующие параметры для назначения роли, связанной со всеми владельцами:
   - **userName**. Имя пользователя интеграции.
   - **роли.** SuperAdmin.

   Нажмите **Вызов**.

   Чтобы назначить роль, связанную лишь с определенными владельцами, вызовите метод **grantRolesToUserForTenants** с теми же значениями параметров userName и roles. В качестве значения параметра **tenantNames** укажите необходимых владельцев.
7. Создайте других пользователей или закройте консоль JMX.
8. Войдите в UCMDB как администратор.
9. Во вкладке **Администрирование** запустите **Диспетчер пакетов**.
10. Нажмите на значок **Создать пользовательский пакет**.
11. Введите имя нового пакета и нажмите кнопку **Далее**.
12. На вкладке «Выбор ресурсов», в разделе **Настройки**, нажмите **Пользователи**.
13. Выберите пользователя или пользователей, созданных в консоли JMX.
14. Нажмите **Далее**, затем **Готово**. Новый пакет появится в списке имен пакетов диспетчера пакетов.
15. Разверните пакет для пользователей, которые будут запускать приложения с поддержкой API-интерфейса.

    Дополнительные сведения см. в разделе "Развертывание пакета" (*Руководство по администрированию HP Universal CMDB*).

Примечание. Пользователи интеграции создаются на уровне заказчика. Чтобы создать пользователя интеграции с более широким набором прав для использования несколькими заказчиками, используйте метод **systemUser** с флагом **isSuperIntegrationUser** = **true**. Используйте методы **systemUser** (**removeUser**, **resetPassword**, **UserAuthenticate** и т.д.).

Существует два встроенных системных пользователя: Рекомендуется изменить их пароли после установки с помощью метода **resetPassword**.

- **sysadmin/sysadmin**
- **UISysadmin/UISysadmin** (Этот пользователь также является суперпользователем интеграции **SuperIntegrationUser**).

  При изменении пароля UISysadmin с помощью **resetPassword** необходимо выполнить следующие действия:

  i. В JMX Console найдите службу **UCMDB-UI:name=UCMDB Integration**.

  ii. Выполните метод **setCMDBSuperIntegrationUser** с именем пользователя и новым паролем пользователя интеграции.

API-интерфейс UCMDB — сценарии использования

Сценарии использования, описанные в этом разделе, относятся к двум типам систем:

- Сервер HP Universal CMDB
- Сторонняя система, содержащая репозиторий ЭК

Этот раздел охватывает следующие темы:

- "Наполнение CMDB" ниже
- "Выполнение запросов к CMDB" ниже
- "Запрос модели классов" на следующей странице
- "Анализ влияния изменений " на следующей странице

Наполнение CMDB

Сценарии использования:

- Сторонняя система управления ресурсами наполняет CMDB данными, доступными только в системе управления ресурсами.
- Несколько сторонних систем наполняют CMDB, создавая централизованную базу CMDB для отслеживания изменений и анализа влияния.
- Сторонняя система создает элементы конфигурации и связи в соответствии со сторонней бизнес-логикой для доступа к возможностям запросов CMDB.

Выполнение запросов к CMDB

Сценарии использования:

- Сторонняя система получает ЭК и связи, представляющие систему SAP, посредством получения результатов TQL-запроса SAP.
- Сторонняя система получает список серверов Oracle, добавленных или измененных за последние 5 часов.
- Сторонняя система получает список серверов, имена которых содержат строку **lab**.
- Сторонняя система находит элементы, связанные с указанным ЭК, путем получения его соседей.

Запрос модели классов

Сценарии использования:

- Сторонняя система дает пользователям возможность указать набор данных для получения из CMDB. Интерфейс пользователя может быть создан на основе модели классов для представления возможных свойств и запроса необходимых данных. Затем пользователь может выбрать информацию для получения.
- Сторонняя система анализирует модель классов, когда пользователь не может получить доступ к интерфейсу пользователя CMDB.

Анализ влияния изменений

Сценарий использования:

- Сторонняя система выводит список бизнес-услуг, которые могут быть затронуты изменением указанного хоста.

Примеры

См. следующие примеры кода:

- 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

Эти файлы находятся в следующем каталоге:

\\<UCMDB root directory>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples

Глава 11: API-интерфейс веб-служб HP Universal CMDB

Данная глава включает:

- Правила 337
- Обзор API-интерфейса веб-службы HP Universal CMDB 338
- Вызов веб-службы HP Universal CMDB 341
- Запрос в CMDB 341
- Обновление UCMDB 344
- Запросы к модели классов UCMDB 345
- Запрос анализа влияния 347
- Общие параметры UCMDB 347
- Выходные параметры UCMDB 350
- Методы запросов UCMDB 351
- Методы обновления UCMDB 362
- Методы анализа влияния в UCMDB 365
- API-интерфейс веб-службы фактического состояния. 367
- API-интерфейс веб-службы UCMDB — сценарии использования 369
- Примеры 370

Правила

В этой главе используются следующие условные обозначения:

- UCMDB — это сама универсальная база данных управления конфигурациями. HP Universal CMDB означает само приложение.
- Элементы и аргументы методов UCMDB приводятся в том виде, в котором они указаны в схеме. Для элементов и аргументов методов используется нижний регистр. Например, `relation` — это элемент типа `Relation`, переданный методу.

См. полную документацию по структурам запросов и ответов в документе HP UCMDB Web Service API Reference. Эти файлы находятся по следующему пути:

<UCMDB root directory>\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\CMDB_Schema\webframe.html

Обзор API-интерфейса веб-службы HP Universal CMDB

Примечание. Эту главу следует использовать в сочетании с документацией по схеме UCMDB, доступной в библиотеке документации.

API-интерфейс веб-службы HP Universal CMDB используется для интеграции приложений с HP Universal CMDB (UCMDB). API-интерфейс предоставляет методы для решения следующих задач:

- Добавление, удаление и обновление ЭК и связей в CMDB;
- Получение информации о модели классов;
- Получение анализа влияния;
- Получение информации об ЭК и связях.
- Управление учетными данными: просмотр, добавление, обновление и удаление;
- Управление заданиями: просмотр статуса, активация и деактивация;
- Управление диапазонами зондов: просмотр, добавление и обновление;
- Управление триггерами: добавление или удаление ЭК-триггера, а также добавление, удаление и отключение TQL-запроса триггера;
- Просмотр стандартных данных по доменам и зондам

Как правило, для методов получения информации об ЭК и связях используется язык TQL. Подробнее см. в разделе "Язык запросов топологии" в документе *Руководство по моделированию в HP Universal CMDB*.

Пользователи API-интерфейса веб-службы HP Universal CMDB должны обладать следующими знаниями:

- Спецификация SOAP.
- Объектно-ориентированный язык программирования, например C++, C# или Java.
- HP Universal CMDB
- Управление потоком данных

Этот раздел охватывает следующие темы:

- "Сценарии использования API-интерфейса" ниже
- "Права доступа" на следующей странице

Сценарии использования API-интерфейса

API-интерфейс веб-служб UCMDB используется для выполнения ряда бизнес-требований. Пример:

- Сторонняя система может запрашивать информацию о доступных ЭК в модели классов.
- Стороннее средство управление активами может обновлять CMDB с использованием данных, доступных только этому средству, объединяя свои данные с данными,

собранными приложениями HP.

- Несколько сторонних систем наполняют CMDB, создавая централизованную базу CMDB для отслеживания изменений и анализа влияния.
- Сторонняя система может создавать объекты и связи в соответствии со своей бизнес-логикой, а затем записывать данные в CMDB, чтобы воспользоваться возможностями запросов CMDB.
- Другие системы, такие как Release Control (CCM), могут использовать методы анализа влияния для анализа изменений.

Права доступа

Чтобы открыть файл WSDL веб-службы, перейдите по адресу: **http://localhost:8080/axis2/services/UcmdbService?wsdl**. Для просмотра файла WSDL необходимо указать учетные данные администратора.

Примечание. Консоль администрирования Axis2 недоступна.

Для входа в систему у пользователя должно быть право доступа **Запуск устаревшей API-функции**.

В таблице ниже описаны дополнительные необходимые права для каждой команды API-интерфейса веб-службы:

Команда API-интерфейса веб-службы	**Необходимые права доступа**
addCIsAndRelations deleteCIsAndRelations updateCIsAndRelations	Общее действие: **Обновление данных**
executeTopologyQueryByName(AdHoc) executeTopologyQueryByNameWithParameters(AdHoc) executeTopologyQueryWithParameters(AdHoc)	Общее действие: **Запуск запроса по определению** Для каждого запроса: Право доступа **Просмотр**
getTopologyQueryExistingResultByName getTopologyQueryResultCountByName releaseChunks pullTopologyMapChunks getCINeighbours getFilteredCIsByType getCIsByID getCIsByType getRelationsById	Общее действие: **Просмотр ЭК** Для каждого запроса: Право доступа **Просмотр**
getQueryNameOfView	Общее действие: **Просмотр ЭК** Для каждого

Команда API-интерфейса веб-службы	Необходимые права доступа
	представления: Право доступа **Просмотр**
getChangedCIs	Общее действие: **Просмотр истории**, **Просмотр ЭК**
calculateImpact getImpactPath getImpactRulesByGroupName getImpactRulesByNamePrefix	Общее действие: **Выполнение анализа влияния**
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Нет

Примечание. При изменении корневого контекста в UCMDB выполните следующие шаги, чтобы включить доступ к API веб-службы:

1. Откройте файл конфигурации **\UCMDB\UCMDBServer\deploy\axis2\WEB-INF\web.xml** и найдите раздел

   ```
   <servlet-class>
   org.apache.axis2.transport.http.AxisServlet
   </servlet-class>
   ```

   Добавьте следующие строки:

   ```
   <init-param>
   <param-name>axis2.find.context</param-name>
   <param-value>false</param-value>
   </init-param>
   ```

2. Откройте файл конфигурации **UCMDB\UCMDBServer\deploy\axis2\WEB-INF\conf\axis2.xml** и найдите строку

   ```
   <parameter name="enableSwA" locked="false">false</parameter>
   ```

   Добавьте после нее:

   ```
   <parameter name="contextRoot" locked="false">test1/setup1/axis2
   </parameter>
   ```

   где **test1/setup1** — корневой контекст.

   (Чтобы удалить корневой контекст, следует удалить текст, добавленный в путь.)

3. Перезапустите сервер UCMDB.

Вызов веб-службы HP Universal CMDB

В веб-службе HP Universal CMDB для вызова серверных методов используются стандартные методы программирования SOAP. Если инструкция не может быть обработана или если при вызове метода возникает проблема, методы API создают исключение `SoapFault`. При создании исключения `SoapFault` UCMDB заполняет одно или несколько полей сообщения об ошибке, кода ошибки или сообщения об исключении. Если ошибка отсутствует, возвращается результат вывода.

Программисты SOAP могут получить доступ к WSDL по адресу:

http://<server>[:port]/axis2/services/UcmdbService?wsdl

Указание порта требуется только в нестандартных средах. Правильный номер порта можно получить у системного администратора.

URL-адрес для вызова службы:

http://<server>[:port]/axis2/services/UcmdbService

См. примеры подключения к CMDB в разделе "API-интерфейс веб-службы UCMDB — сценарии использования" на странице 369.

Запрос в CMDB

Для выполнения запросов в CMDB используются API-интерфейсы, описанные в разделе " Методы запросов UCMDB" на странице 351. Запросы и возвращенные элементы CMDB всегда содержат реальные идентификаторы UCMDB. Примеры использования одного из методов запросов см. в разделе Query Example.

Этот раздел охватывает следующие темы:

- "Динамический расчет ответа (JIT)" ниже
- "Обработка крупных ответов" на следующей странице
- "Выбор свойств для возврата" на следующей странице
- "Конкретные свойства" на странице 343
- "Производные свойства" на странице 343
- "Свойства именования" на странице 343
- "Другие элементы спецификации свойств" на странице 343

Динамический расчет ответа (JIT)

Для всех методов запроса сервер UCMDB рассчитывает значения, которые запрашивает метод, и возвращает результат на основе последних данных. Результат всегда рассчитывается в момент получения запроса, даже если TQL-запрос активен и существует предыдущий рассчитанный результат. Поэтому результаты выполнения запроса, возвращенные клиентскому приложению, могут отличаться от результатов того же запроса, отображаемых в пользовательском интерфейсе.

Совет. Если приложение использует результаты запроса несколько раз и данные существенно не меняются между применениями данных результатов, производительность можно улучшить путем сохранения данных в клиентском приложении вместо повторного выполнения запроса.

Обработка крупных ответов

Ответ на запрос всегда включает структуры данных, запрошенные методом запроса, даже если фактические данные не передаются. Для многих методов, в которых данные представляют собой коллекцию или карту, ответ также включает структуру `ChunkInfo`, состоящую из `chunksKey` и `numberOfChunks`. Поле `numberOfChunks` обозначает число блоков, содержащих данные для получения.

Максимальный объем передаваемых данных задается администратором. Если данные, возвращенные запросом, превышают максимальный объем, структуры данных в первом ответе не будут содержать значимой информации, а значением поля `numberOfChunks` будет `2` или более. Если данные не превышают максимальный объем, поле `numberOfChunks` будет иметь значение 0, и данные будут переданы в первом ответе. Поэтому при обработке запроса сначала следует проверить значение `numberOfChunks`. Если оно меньше `1`, отбросьте передаваемые данные и запросите блоки данных. В противном случае используйте данные в ответе.

Дополнительные сведения об обработке данных, разделенных на блоки, см. в разделах "pullTopologyMapChunks" на странице 360 и "releaseChunks" на странице 362.

Выбор свойств для возврата

ЭК и связи обычно имеют много свойств. Некоторые методы, возвращающие коллекции или графы этих элементов, принимают входные параметры, которые определяют значения свойств для возврата с каждым элементом, соответствующим запросу. CMDB не возвращает пустые свойства. Поэтому ответ на запрос может содержать меньше свойств, чем указано в запросе.

В этом разделе описываются типы параметров, используемые при настройке свойств для возврата.

Ссылки на свойства можно указывать двумя способами:

- По именам.
- По именам предварительно настроенных правил свойств. Предварительно настроенные правила свойств используются базой CMDB для создания списка реальных имен свойств.

Если приложение ссылается на свойства по имени, оно передает элемент `PropertiesList`.

Совет. Если возможно, указывайте имена свойств в `PropertiesList`, а не в наборе на основе правил. При использовании настроенных правил свойств почти всегда будет возвращаться больше свойств, чем требуется, и производительность системы может ухудшиться.

Существует два типа настроенных свойств: свойства квалификатора и простые свойства.

- **Свойства квалификатора.** Используются, если клиентское приложение должно передать элемент `QualifierProperties` (список квалификаторов, которые могут быть применены к свойствам). CMDB преобразует список квалификаторов, переданных клиентским приложением, в список свойств, к которым применяется хотя бы один квалификатор. Значения этих свойств возвращаются с элементами `CI` или `Relation`.
- **Простые свойства.** При использовании свойств на основе простых правил клиентское приложение передает элемент `SimplePredefinedProperty` или `SimpleTypedPredefinedProperty`. Эти элементы содержат имя правила, по которому CMDB создает список свойств для возврата. Правила, которые могут быть указаны в элементе `SimplePredefinedProperty` или `SimpleTypedPredefinedProperty` могут иметь тип `CONCRETE`, `DERIVED` и `NAMING`.

Конкретные свойства

Конкретные свойства — это набор свойств, заданных для указанного типа ЭК. Свойства, добавленные производными классами, не возвращаются для экземпляров этих производных классов.

Набор экземпляров, возвращенный методом, может состоять из экземпляров типа ЭК, указанных в вызове метода, и экземпляров типов ЭК, которые наследуют у этого типа ЭК. Производные типы ЭК наследуют свойства у указанных типов ЭК. Кроме того, производные типы ЭК расширяют родительский тип ЭК путем добавления свойств.

Пример конкретных свойств:

Тип ЭК `T1` имеет свойства `P1` и `P2`. Тип ЭК `T11` наследует у `T1` и расширяет `T1`, добавляя свойства `P21` и `P22`.

Коллекция ЭК с типом `T1` включает экземпляры `T1` и `T11`. Конкретные свойства всех экземпляров в коллекции: `P1` и `P2`.

Производные свойства

Производные свойства — это набор свойств, заданных для определенных типов ЭК. Свойства каждого производного ЭК добавляются производным типом ЭК.

Пример производных свойств:

Продолжая пример конкретных свойств, производными свойствами экземпляров `T1` будут `P1` и `P2`. Производные свойства экземпляров `T11`: `P1`, `P2`, `P21` и `P22`.

Свойства именования

Свойства именования — это `display_label` и `data_name`.

Другие элементы спецификации свойств

- **PredefinedProperties**

PredefinedProperties может содержать элемент QualifierProperties и SimplePredefinedProperty для каждого из других доступных правил. Набор PredefinedProperties не обязательно должен содержать все типы списков.

- **PredefinedTypedProperties**

  PredefinedTypedProperties используется для применения другого набора свойств к каждому ЭК. PredefinedTypedProperties может содержать элемент QualifierProperties и SimpleTypedPredefinedProperty для каждого из других применимых правил. Поскольку PredefinedTypedProperties применяется к каждому типу ЭК по отдельности, производные свойства не имеют значения. Набор PredefinedProperties не обязательно должен содержать все применимые типы списков.

- **CustomProperties**

  CustomProperties может содержать любое сочетание базовых списков PropertiesList и списков свойств на основе правил. Фильтр свойств — это объединение всех свойств, возвращенных всеми списками.

- **CustomTypedProperties**

  CustomTypedProperties может содержать любое сочетание базовых списков PropertiesList и применимых списков свойств на основе правил. Фильтр свойств — это объединение всех свойств, возвращенных всеми списками.

- **TypedProperties**

  TypedProperties используется для передачи другого набора свойств для каждого ЭК. TypedProperties — это коллекция пар, состоящих из имен типов и наборов свойств всех типов. Каждый набор свойств применяется только к соответствующему типу.

Обновление UCMDB

Для обновления CMDB используются API-интерфейсы обновления. Дополнительные сведения о методах API см. в разделе " Методы обновления UCMDB" на странице 362.

Эта задача включает следующие шаги:

- "Параметры обновления UCMDB" ниже
- "Использование типов идентификаторов с методами обновления" на следующей странице

Параметры обновления UCMDB

В этом разделе описываются параметры, используемые только методами обновления службы.

- **CIsAndRelationsUpdates**

  Тип CIsAndRelationsUpdates включает CIsForUpdate, relationsForUpdate, referencedRelations и referencedCIs. Экземпляр CIsAndRelationsUpdates необязательно должен содержать все три элемента.

  CIsForUpdate — это коллекция ЭК. relationsForUpdate — это коллекция связей. Элементы CI и relation в коллекциях включают элемент props. При создании ЭК или связи свойства с атрибутом required или key в определении типа ЭК должны быть

заполнены значениями. Все элементы в этих коллекциях обновляются или создаются методом.

`referencedCIs` и `referencedRelations` — это коллекции ЭК, уже заданных в CMDB. Элементы в коллекции определяются по временному идентификатору в сочетании со всеми тремя ключевыми свойствами. Эти элементы используются для разрешения идентификаторов ЭК и связей для обновления. Они никогда не создаются и не обновляются методом.

Каждый из элементов `CI` и `relation` в этих коллекциях включает коллекцию свойств. Новые элементы создаются со значениями свойств в этих коллекциях.

Использование типов идентификаторов с методами обновления

Далее описываются типы ЭК идентификаторов, ЭК и связи. Если идентификатор не является настоящим идентификатором CMDB, требуется тип и ключевые атрибуты.

- **Удаление и обновление ЭК**

  Временный или пустой идентификатор может использоваться клиентом при вызове метода для обновления или удаления элемента. В этом случае необходимо настроить типы ЭК и " Ключевые атрибуты", которые идентифицируют ЭК.

- **Удаление и обновление связей**

  При удалении или обновлении обновлений идентификатор связи должен быть пустым, временным или реальным.

  Если идентификатор ЭК является временным, ЭК должен быть передан в коллекции `referencedCIs` с указанием его ключевых атрибутов. Дополнительные сведения см. в разделе referencedCIs ("CIsAndRelationsUpdates" на предыдущей странице).

- **Вставка новых элементов конфигурации в CMDB**

  При вставке нового ЭК можно использовать пустой или временный ЭК. Однако если идентификатор пуст, сервер не сможет вернуть реальный идентификатор CMDB в элементе `createIDsMap` структуры, поскольку элемент `clientID` ОТСУТСТВУЕТ. Дополнительные сведения см. в разделах "addCIsAndRelations" на странице 363 и " Методы запросов UCMDB" на странице 351.

- **Вставка новых связей в CMDB**

  Идентификатор связи может быть временным или пустым. Однако если связь является новой, но ЭК на любой стороне связи уже определены в CMDB, значит эти ЭК уже существуют и должны иметь реальные идентификаторы CMDB или указываться в коллекции `referencedCIs`.

Запросы к модели классов UCMDB

Методы модели классов возвращают сведения о типах ЭК и связях. Модель классов настраивается с помощью Диспетчера типов ЭК. Подробнее см. в разделе Диспетчер типов ЭК в документе *Руководство по моделированию в HP Universal CMDB*.

В этом разделе представлены сведения о следующих методах, возвращающих сведения о типах ЭК и связях:

- "getClassAncestors" ниже
- "getAllClassesHierarchy" ниже
- "getCmdbClassDefinition" ниже

getClassAncestors

Метод `getClassAncestors` возвращает путь между указанным типом ЭК и его корнем, включая корень.

Входящие параметры

Параметр	Комментарий
cmdbContext	Дополнительные сведения см. в разделах "CmdbContext" на странице 348
className	Имя типа. Дополнительные сведения см. в разделе "Имя типа" на странице 349.

Исходящие параметры

Параметр	Комментарий
classHierarchy	Коллекция пар имен классов и имен родительских классов.
comments	Только для внутреннего использования.

getAllClassesHierarchy

Метод `getAllClassesHierarchy` извлекает все дерево модели классов.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.

Исходящие параметры

Параметр	Комментарий
classesHierarchy	Коллекция пар имен классов и имен родительских классов.
comments	Только для внутреннего использования.

getCmdbClassDefinition

Метод `getCmdbClassDefinition` получает информацию об указанном классе.

При использовании `getCmdbClassDefinition` для получения ключевых атрибутов также необходимо запросить родительские классы базового класса. `getCmdbClassDefinition` относит к ключевым только атрибуты со значением `ID_ATTRIBUTE` в определении класса, указанном в `className`. Унаследованные ключевые атрибуты не распознаются как ключевые атрибуты указанного класса. Таким образом, полный список ключевых атрибутов указанного класса — это объединение всех ключей в классе и всех их родителей вплоть до корня.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на следующей странице.
className	Имя типа. Дополнительные сведения см. в разделе "Общие параметры UCMDB " ниже.

Исходящие параметры

Параметр	Комментарий
cmdbClass	Определение класса, включающее элементы `name`, `classType`, `displayLabel`, `description`, `parentName`, квалификаторы и атрибуты.
comments	Только для внутреннего использования.

Запрос анализа влияния

Элемент `Identifier` в методах анализа влияния указывает на данные ответа службы. Он уникален для текущего запроса и удаляется из кэша сервера, если не используется в течение 10 минут.

Примеры использования методов анализа влияния см. в разделе Impact Analysis Example.

Общие параметры UCMDB

В этом разделе описываются распространенные параметры методов службы.

Этот раздел охватывает следующие темы:

- "CmdbContext" на следующей странице
- "ID" на следующей странице
- " Ключевые атрибуты" на следующей странице
- "Типы идентификаторов" на следующей странице
- "CIProperties" на следующей странице
- "Имя типа" на странице 349
- "Элемент конфигурации (ЭК)" на странице 349
- "Связь" на странице 349

CmdbContext

Все вызовы API-интерфейса веб-службы UCMDB требуют аргумента `CmdbContext`. `CmdbContext` — это строка `callerApplication`, которая служит для идентификации приложения, которое вызвало службу. `CmdbContext` используется для ведения журналов и устранения неполадок.

ID

Каждый `ЭК` и связь включает поле `ID`. Оно состоит из строки идентификатора, чувствительной к регистру, и необязательного флага `temp`, который обозначает, что идентификатор является временным.

Ключевые атрибуты

Для идентификации элементов `CI` и `Relation` в некоторых контекстах можно использовать ключевые атрибуты вместо идентификатора CMDB. Ключевые атрибуты — это атрибуты со значением ID_ATTRIBUTE в определении класса.

В интерфейсе пользователя рядом с ключевыми атрибутами в списке атрибутов типов ЭК отображается значок ключа. Подробнее см. в разделе "Диалоговое окно "Добавить/изменить атрибут" в документе *Руководство по моделированию в HP Universal CMDB*. См. дополнительные сведения об идентификации ключевых атрибутов в API-интерфейсе клиентского приложения в разделе "getCmdbClassDefinition" на странице 346.

Типы идентификаторов

Элемент `ID` может содержать реальный идентификатор, временный идентификатор или быть пустым.

Реальный идентификатор — это строка, назначенная CMDB и идентифицирующая объект в базе данных. Временный идентификатор может быть любой строкой, уникальной для текущего запроса.

Временный идентификатор может быть назначен клиентом и часто представляет идентификатор ЭК, сохраненный клиентом. Он не обязательно должен представлять объект, уже созданный в CMDB. Если временный идентификатор передается клиенту и CMDB может идентифицировать существующий ЭК с помощью ключевых свойств ЭК, этот ЭК используется в соответствии с контекстом, как если бы он был определен реальным идентификатором.

CIProperties

Элемент `CIProperties` состоит из коллекций, каждая из которых содержит последовательность элементов имя-значение, которые определяют свойства типа, указанного именем коллекции. Ни одна из коллекций не является обязательной, поэтому элемент `CIProperties` может содержать любое сочетание коллекций.

Элементы `CIProperties` используются элементами `CI` и `Relation`. См. дополнительные сведения в разделах "Элемент конфигурации (ЭК)" на следующей странице и "Связь" на следующей странице.

Свойства коллекций:

- dateProps - коллекция элементов DateProp
- doubleProps - коллекция элементов DoubleProp
- floatProps - коллекция элементов FloatProp
- intListProps - коллекция элементов intListProp
- intProps - коллекция элементов IntProp
- strProps - коллекция элементов StrProp
- strListProps - коллекция элементов StrListProp
- longProps - коллекция элементов LongProp
- bytesProps - коллекция элементов BytesProp
- xmlProps - коллекция элементов XmlProp

Имя типа

Имя типа — имя класса типа ЭК или связи. Имя типа используется в коде для вызова класса. Его не следует путать с отображаемым именем, которое отображается в интерфейсе пользователя при обращении к классу, но не имеет значения в коде.

Элемент конфигурации (ЭК)

Элемент CI включает ID, type и коллекции props.

При использовании " Методы обновления UCMDB" для обновления ЭК элемент ID может содержать реальный идентификатор CMDB или временный клиентский идентификатор. Если используется временный идентификатор, установите для флага temp значение true. При удаление элемента значение ID может быть пустым. " Методы запросов UCMDB" используют реальные значения ID в качестве входных параметров и возвращают реальные значения ID в результатах запросов.

Значение type может быть любым именем типа, заданным в диспетчере типов ЭК. Подробнее см. в разделе Диспетчер типов ЭК в документе Руководство по моделированию в HP Universal CMDB.

Элемент props — это коллекция CIProperties. Подробнее см. в разделе "Общие параметры UCMDB " на странице 347.

Связь

Связь — это объект, который связывает два элемента конфигурации. Элемент Связь состоит из значений ID, type, идентификаторов связанных элементов (end1ID и end2ID) и коллекции props.

При использовании " Методы обновления UCMDB" для обновления элемента Relation значение идентификатора Relation может быть реальным идентификатором CMDB или временным идентификатором. При удалении элемента поле ID может быть пустым. " Методы запросов UCMDB" используют реальные значения ID в качестве входных параметров и возвращают реальные значения ID в результатах запросов.

Тип связи — это имя типа класса UCMDB, из которого инициирована связь. Может использоваться любой тип связей, настроенный в CMDB. Дополнительные сведения о классах и типах см. в разделе " Запросы к модели классов UCMDB " на странице 345.

Подробнее см. в разделе Диспетчер типов ЭК в документе *Руководство по моделированию в HP Universal CMDB*.

Два идентификатора конечных элементов не должны быть пустыми, поскольку используются для создания идентификатора текущей связи. Однако они могут иметь временные идентификаторы, назначенные клиентом.

Элемент `props` — это коллекция CIProperties. Дополнительные сведения см. в разделе "CIProperties" на странице 348.

Выходные параметры UCMDB

В этом разделе описываются распространенные выходные параметры методов службы. Подробнее см. online schema documentation..

Этот раздел охватывает следующие темы:

- "CIs" ниже
- "ShallowRelation" ниже
- "Topology" ниже
- "CINode" ниже
- "RelationNode" ниже
- "TopologyMap" на следующей странице
- "ChunkInfo" на следующей странице

CIs

`CIs` — это коллекция ЭК.

ShallowRelation

Элемент `ShallowRelation` связывает два ЭК и состоит из значений `ID`, `type` и идентификаторов двух связанных элементов (`end1ID` и `end2ID`). Тип связи — это `имя типа` класса CMDB, из которого инициирована связь. Может использоваться любой тип связей, настроенный в CMDB.

Topology

`Topology` — это граф элементов `CI` и связей. `Topology` состоит из коллекции `CIs` и коллекции `Relations`, содержащей один или несколько элементов `Relation`.

CINode

`CINode` состоит из коллекции `CIs` и элемента `label`. Элемент `label` в `CINode` — это метка, заданная для узла TQL-запроса.

RelationNode

`RelationNode` состоит из наборов коллекций `Relation` и элемента `label`. Элемент `label` в `RelationNode` — это метка, заданная для узла TQL-запроса.

TopologyMap

TopologyMap — это выходной параметр вычисления запроса в соответствии с TQL-запросом. Элементы label в TopologyMap — это метки узлов, заданные в TQL-запросе.

Данные TopologyMap возвращаются в следующей форме:

- CINodes. Это один или несколько CINode (см. раздел "CINode" на предыдущей странице).
- relationNodes. Это один или несколько RelationNode (см. раздел "RelationNode" на предыдущей странице).

Элементы label в этих двух структурах упорядочивают списки ЭК и связей.

ChunkInfo

Если запрос возвращает большой объем данных, сервер сохраняет данные, разделенные на сегменты, которые называются блоками. Информация, используемая клиентом для получения разделенных данных, находится в структуре ChunkInfo, возвращенной запросом. ChunkInfo состоит из значения numberOfChunks (количество блоков для возвращения) и chunksKey. chunksKey — это уникальный идентификатор данных на сервере для этого вызова запроса.

Дополнительные сведения см. в разделе "Обработка крупных ответов" на странице 342.

Методы запросов UCMDB

Данный раздел содержит сведения о следующих методах:

- "executeTopologyQueryByNameWithParameters" ниже
- "executeTopologyQueryWithParameters " на следующей странице
- "getChangedCIs" на странице 353
- "getCINeighbours" на странице 354
- "getCIsByID" на странице 354
- "getCIsByType" на странице 355
- "getFilteredCIsByType " на странице 356
- "getQueryNameOfView" на странице 359
- "getTopologyQueryExistingResultByName" на странице 359
- "getTopologyQueryResultCountByName" на странице 360
- "pullTopologyMapChunks" на странице 360
- "releaseChunks" на странице 362

executeTopologyQueryByNameWithParameters

Метод executeTopologyQueryByNameWithParameters получает элемент topologyMap, соответствующий указанному параметризованному запросу.

Значения параметров запросов передаются в аргументе parameterizedNodes. Указанный TQL-запрос должен иметь уникальные метки для каждого узла CINode и relationNode. В противном случае вызов метода закончится неудачей.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
queryName	Имя параметризованного TQL-запроса в CMDB, по которому извлекается карта.
parameterizedNodeList	Условия, которым каждый узел должен соответствовать для включения в результаты поиска.
queryTypedProperties	Коллекция наборов свойств для извлечения в элементах определенного типа ЭК.

Исходящие параметры

Параметр	Комментарий
topologyMap	Подробнее см. в разделе "TopologyMap" на предыдущей странице.
chunkInfo	Дополнительные сведения см. в разделах "ChunkInfo" на предыдущей странице и "Обработка крупных ответов" на странице 342.

executeTopologyQueryWithParameters

Метод executeTopologyQueryWithParameters получает элемент topologyMap, соответствующий параметризованному запросу.

Запрос передается в аргументе queryXML. Значения параметров запросов передаются в аргументе parameterizedNodeList. TQL-запрос должен иметь уникальные метки для каждого узла CINode и relationNode.

Метод executeTopologyQueryWithParameters используется для передачи специальных запросов без обращения к запросам, заданным CMDB. Этот метод можно использовать, если доступ к интерфейсу пользователя UCMDB для формирования запроса отсутствует или сохранение запроса в базе данных не требуется.

Для использования экспортированного TQL-запрос в качестве входящего параметра для данного метода выполните следующие действия:

1. Запустите веб-браузер и введите следующий адрес: **http://localhost:8080/jmx-console**.

   Возможно, потребуется ввести имя пользователя и пароль для входа в систему.
2. Нажмите **UCMDB:service=TQL Services**.
3. Найдите операцию **exportTql**.

- В окне параметров **customerId** введите **1** (значение по умолчанию).
- В окне параметров **patternName** введите действительное имя TQL.

4. Нажмите кнопку **Invoke**.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
queryXML	Строка XML, представляющая TQL-запрос без тегов ресурсов.
parameterizedNodeList	Условия, которым каждый узел должен соответствовать для включения в результаты поиска.

Исходящие параметры

Параметр	Комментарий
topologyMap	Подробнее см. в разделе "TopologyMap" на странице 351.
chunkInfo	Дополнительные сведения см. в разделах "ChunkInfo" на странице 351 и "Обработка крупных ответов" на странице 342.

getChangedCIs

Метод `getChangedCIs` возвращает данные об изменении для всех ЭК, связанных с указанными ЭК.

Входящие параметры

Параметр	Комментарий
cmdbContext	Дополнительные сведения см. в разделе "CmdbContext" на странице 348.
ids	Список идентификаторов корневых ЭК, связанные ЭК которых выбраны для изменения. В этой коллекции допускаются только реальные идентификаторы CMDB.
fromDate	Начало периода, за который нужно проверить изменения ЭК.
toDate	Конец периода, за который нужно проверить изменения ЭК.

Исходящие параметры

Параметр	Комментарий
getChangedCIsResponseList	Ноль или более коллекций элементов ChangedDataInfo.

getCINeighbours

Метод getCINeighbours возвращает ближайших соседей указанного ЭК.

Например, если в запросе вызываются соседи ЭК A и ЭК A содержит ЭК B, который использует ЭК C, ЭК B возвращается, а ЭК C — нет. То есть возвращаются только соседи указанного типа.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
ID	Идентификатор ЭК, для которого необходимо получить соседей. Это должен быть настоящий идентификатор CMDB или глобальный идентификатор.
neighbourType	Имя типа ЭК соседей для извлечения. Будут возвращены соседи указанного типа и всех типов, производных от него. Дополнительные сведения см. в разделе "Имя типа" на странице 349.
CIProperties	Данные, которые будут возвращены для каждого ЭК, вызвавшего макет запроса в интерфейсе пользователя. Подробнее см. в разделе "TypedProperties" на странице 344.
relationProperties	Данные, которые будут возвращены для каждой связи (в интерфейсе пользователя это называется схемой запроса). Подробнее см. в разделе "TypedProperties" на странице 344.

Исходящие параметры

Параметр	Комментарий
topology	Подробнее см. в разделе "Topology" на странице 350.
comments	Только для внутреннего использования.

getCIsByID

Метод getCIsByID извлекает ЭК по идентификаторам в CMDB или глобальным идентификаторам.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
ClsTypedProperties	Коллекция свойств с определенным типом. Подробнее см. в разделе "Другие элементы спецификации свойств" на странице 343.
IDs	В этой коллекции допускаются только реальные идентификаторы CMDB или глобальные идентификаторы.

Исходящие параметры

Параметр	Комментарий
Cls	Коллекция ЭК.
chunkInfo	Дополнительные сведения см. в разделах "ChunkInfo" на странице 351 и "Обработка крупных ответов" на странице 342.

getClsByType

Метод `getClsByType` возвращает коллекцию ЭК указанного типа и всех типов, которые наследуют у указанного типа.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
type	Имя класса. Дополнительные сведения см. в разделе "Имя типа" на странице 349.
свойства	Данные, возвращаемые для каждого ЭК. Подробнее см. в разделе "CustomProperties" на странице 344.

Исходящие параметры

Параметр	Комментарий
Cls	Коллекция ЭК.
chunkInfo	См. дополнительные сведения в разделах: "ChunkInfo" на странице 351 и "Обработка крупных ответов" на странице 342.

getFilteredCIsByType

Метод `getFilteredCIsByType` извлекает ЭК указанного типа, соответствующие условиям, которые используются методом. Условие включает следующее:

- Поле имени, содержащее имя свойства;
- Поле оператора с оператором сравнения;
- Необязательное поле значения, содержащее значение или список значений.

Вместе они формируют логическое выражение:
`<item>.property.value [operator] <condition>.value`

Например, если имя условия — `root_actualdeletionperiod`, значения условия — `40`, а оператор — `Equal`, логическое выражение примет следующий вид:

`<item>.root_actualdeletionperiod.value = = 40`

Запрос возвращает все элементы, значение `root_actualdeletionperiod` которых равняется `40`, при условии что другие условия отсутствуют.

Если в качестве аргумента `conditionsLogicalOperator` используется `AND`, запрос вернет элементы, соответствующие всем условиям в коллекции `conditions`. Если в качестве аргумента `conditionsLogicalOperator` используется `OR`, запрос вернет элементы, соответствующие хотя бы одному условию в коллекции `conditions`.

В следующей таблице перечислены операторы сравнения:

Оператор	Тип условия/Комментарии
ChangedDuring	Дата Это проверка диапазона. Значение условия указывается в часах. Если значение свойства date находится в интервале, для которого вызван метод (с учетом значения условия), условие примет значение true. Например, если в качестве значения условия используется 24, условие примет значение true, если значение свойства date находится между текущим временем вчерашнего дня и настоящим моментом. **Примечание.** Имя `ChangedDuring` сохранено для обратной совместимости. В предыдущих версиях оператор использовался только при создании и изменении свойств.
Equal	Строка и число
EqualIgnoreCase	Строка
Больше	Число
GreaterEqual	Число
In	Строка, число и список.

Оператор	Тип условия/Комментарии
	Значение условия — список. Условие принимает значение true, если значение свойства соответствует одному из значений списка.
InList	Список Значение условия и значение свойства — список. Условие принимает значение true, если все значения в списке условий также отображаются в списке свойств элемента. Может существовать больше значений свойств, чем указано в условии. Это не повлияет на его истинность.
IsNull	Строка, число и список. Свойство элемента не имеет значения. При использовании оператора `IsNull` значение условия игнорируется и в некоторых случаях может иметь значение nil.
Less	Число
LessEqual	Число
Like	Строка Значение условия — часть строки значения свойства. Значение условия должно быть отделено знаками процента (%). Например, `%Bi%` соответствует `Bismark` и `Bay of Biscay`, но не `biscuit`.
LikeIgnoreCase	Строка Оператор `LikeIgnoreCase` используется так же, как оператор `Like`. Регистр не учитывается при сопоставлении. Таким образом, `%Bi%` соответствует `biscuit`.
NotEqual	Строка и число
UnchangedDuring	Дата Это проверка диапазона. Значение условия указывается в часах. Если значение свойства date находится в интервале, для которого вызван метод с учетом значения условия, условие примет значение false. Если значение находится вне этого диапазона, условие примет значение true. Например, если в качестве значения условия используется `24`, условие примет значение true, если значение свойства date находится до текущего времени вчерашнего дня или после текущего времени завтрашнего дня. **Примечание.** Имя `UnchangedDuring` сохранено для обратной совместимости. В предыдущих версиях оператор использовался только при создании и изменении свойств.

Пример настройки условия:

```
FloatCondition fc = new FloatCondition();
FloatProp fp = new FloatProp();
fp.setName("attr_name");
fp.setValue(11f);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.FloatOperator.EQUAL);
```

Пример запроса унаследованных свойств:

Если в качестве целевого используется ЭК `sample`, который имеет два атрибута (`name` и `size`), `sampleII` добавляет в ЭК два дополнительных атрибута — `level` и `grade`. В этом примере задается запрос свойств `sampleII`, унаследованных у `sample`, посредством ввода их имен.

```
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);
```

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
type	Имя класса. Дополнительные сведения см. в разделе "Имя типа" на странице 349. Может использоваться любой тип, указанный с помощью диспетчера типов ЭК. Подробнее см. в разделе Диспетчер типов ЭК в документе *Руководство по моделированию в HP Universal CMDB*.
свойства	Данные, которые будут возвращены для каждого ЭК, вызвавшего макет запроса в интерфейсе пользователя. Подробнее см. в разделе "CustomProperties" на странице 344.
conditions	Коллекция пар имен/значений и операторов, которые связывают их друг с другом. Например, `host_hostname like QA`.
conditionsLogicalOperator	• **AND.** Должны выполняться все условия. • **OR**. Должно выполняться хотя бы одно условие.

Исходящие параметры

Параметр	Комментарий
CIs	Коллекция ЭК.
chunkInfo	См. дополнительные сведения в разделах "ChunkInfo" на странице 351 и "Обработка крупных ответов" на странице 342.

getQueryNameOfView

Метод `getQueryNameOfView` извлекает имя TQL-запроса, на котором основывается указанное представление.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
viewName	Имя представления, которое является подмножеством модели классов в CMDB.

Исходящие параметры

Параметр	Комментарий
queryName	Имя TQL-запроса в CMDB, на котором основывается представление.

getTopologyQueryExistingResultByName

Метод `getTopologyQueryExistingResultByName` получает последний результаты выполнения указанного TQL-запроса. Вызов не приводит к выполнению TQL-запроса. Если результаты предыдущего выполнения отсутствуют, результат не возвращается.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
queryName	Имя TQL-запроса.
queryTypedProperties	Коллекция наборов свойств для извлечения в элементах определенного типа ЭК.

Исходящие параметры

Параметр	Комментарий
topologyMap	Дополнительные сведения см. в разделе "TopologyMap" на странице 351.
chunkInfo	Подробнее см. разделы "ChunkInfo" на странице 351 и "Обработка крупных ответов" на странице 342.

getTopologyQueryResultCountByName

Метод `getTopologyQueryResultCountByName` извлекает количество экземпляров каждого узла, соответствующего указанному запросу.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
queryName	Имя TQL-запроса.
countInvisible	Если выбрано значение true, выходные данные включают ЭК, указанные как невидимые в запросе.

Исходящие параметры

Параметр	Комментарий
getTopologyQueryResultCountByNameResponse	Количество экземпляров, совпадающих с условиями запроса.

pullTopologyMapChunks

Метод `pullTopologyMapChunks` извлекает один из блоков данных, содержащих ответ на метод.

Каждый блок содержит элемент `topologyMap`, который является частью ответа. Первый блок имеет номер `1`, поэтому цикл извлечения повторяется от 1 до *<объект запроса>*`.getChunkInfo().getNumberOfChunks()`.

См. дополнительные сведения в разделах "ChunkInfo" на странице 351 и "Запрос в CMDB" на странице 341.

Клиентское приложение должно поддерживать частичные карты.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
ChunkRequest	Номер блока для извлечения и элемент `ChunkInfo`, возвращенный методом запроса.
queryTypedProperties	Коллекция наборов свойств для извлечения в элементах определенного типа ЭК.

Исходящие параметры

Параметр	Комментарий
topologyMap	Подробнее см. в разделе "TopologyMap" на странице 351.
comments	Только для внутреннего использования.

Пример обработки блоков:

```
GetCIsByType request =
     new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
     ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1; j<=response.getChunkInfo().getNumberOfChunks(); j++){
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req =new PullTopologyMapChunks
(cmdbContext,chunkRequest);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().sizeCINodeList();
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINode(m).getCIs();
            for(int i=0 ; i < cis.sizeCIList() ; i++) {
                // ваш код для обработки ЭК
            }
        }
}

GetCIsByType request =
     new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
```

```
     ucmdbService.getCIsByType(request);
ChunkRequest chunkRequest = new ChunkRequest();
chunkRequest.setChunkInfo(response.getChunkInfo());
for(int j=1 ; j <= response.getChunkInfo().getNumberOfChunks() ; j++) {
    chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req = new PullTopologyMapChunks(cmdbContext,
chunkRequest);
    PullTopologyMapChunksResponse res =
        ucmdbService.pullTopologyMapChunks(req);
        for(int m=0 ;
            m < res.getTopologyMap().getCINodes().getCINodes().size();
            m++) {
            CIs cis =
            res.getTopologyMap().getCINodes().getCINodes().get(m).getCIs();
            for(int i=0 ; i < cis.getCIs().size(); i++) {
                // ваш код для обработки ЭК
            }
        }
}
```

releaseChunks

Метод `releaseChunks` освобождает память из блоков, содержащих данные из запроса.

Совет. Сервер отбрасывает данные через 10 минут. Вызов этого метода позволяет отбросить данные сразу после прочтения для экономии ресурсов сервера.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
chunksKey	Идентификатор данных, разделенных на блоки, на сервере. Ключ — элемент `ChunkInfo`.

Методы обновления UCMDB

Данный раздел содержит сведения о следующих методах:

- "addCIsAndRelations" на следующей странице
- "addCustomer" на странице 364
- "deleteCIsAndRelations" на странице 364
- "removeCustomer" на странице 364
- "updateCIsAndRelations" на странице 364

addCIsAndRelations

Метод addCIsAndRelations добавляет или обновляет ЭК и связи.

Если ЭК и связи не существуют в CMDB, они добавляются, а их свойства устанавливаются в соответствии с содержимым аргумента CIsAndRelationsUpdates.

Если ЭК или связи существуют в CMDB, они обновляются, если для элемента updateExisting установлено значение **true**.

Если элемент updateExisting имеет значение **false**, CIsAndRelationsUpdates не может ссылаться на существующие ЭК или связи. Любые попытки использовать существующие элементы, когда updateExisting = false приведут к исключению.

Если updateExisting имеет значение **true**, операция добавления и обновления выполняется без проверки ЭК независимо от значения ignoreValidation.

Если updateExisiting = **false**, а ignoreValidation = **true**, операция добавления выполняется без проверки ЭК.

Если updateExisiting = **false**, а ignoreValidation = **false**, ЭК проверяются перед операцией добавления.

Отношения никогда не проверяются.

CreatedIDsMap — это карта или словарь типа ClientIDToCmdbID, который связывает временные идентификаторы клиента с соответствующими реальными идентификаторами в CMDB.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
updateExisting	При использовании значения **true** обновляет элементы, уже присутствующие в CMDB. При использовании значения **false** выдает исключение, если любой из элементов уже существует.
CIsAndRelationsUpdates	Элементы для обновления или создания. Дополнительные сведения см. в разделе "CIsAndRelationsUpdates" на странице 344.
ignoreValidation	При использовании значения true проверка перед обновлением CMDB не выполняется.
dataStore	Сведения об инициаторе изменения.

Исходящие параметры

Параметр	Комментарий
createdIDsMapList	Список идентификаторов клиентов, сопоставленных с

Параметр	Комментарий
	идентификаторами CMDB. Дополнительные сведения см. в приведенном выше описании.
comments	Только для внутреннего использования.

addCustomer

Метод `addCustomer` добавляет заказчика.

Входящие параметры

Параметр	Комментарий
CustomerID	Цифровой идентификатор заказчика.

deleteCIsAndRelations

Метод `deleteCIsAndRelations` удаляет указанные ЭК и связи из CMDB.

Если удаляемый ЭК является конечным для одного или несколько элементов `Relation`, эти элементы `Relation` также удаляются.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
CIsAndRelationsUpdates	Элементы для удаления. Дополнительные сведения см. в разделе "CIsAndRelationsUpdates" на странице 344.
dataStore	Сведения об инициаторе изменения.

removeCustomer

Метод `removeCustomer` удаляет запись заказчика.

Входящие параметры

Параметр	Комментарий
CustomerID	Цифровой идентификатор заказчика.

updateCIsAndRelations

Метод `updateCIsAndRelations` обновляет указанные ЭК и связи.

При обновлении используются значения свойств из аргумента `CIsAndRelationsUpdates`. Если любой из ЭК или связей отсутствует в CMDB, выдается исключение.

`CreatedIDsMap` — это карта или словарь типа `ClientIDToCmdbID`, который связывает временные идентификаторы клиента с соответствующими реальными идентификаторами в CMDB.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
CIsAndRelationsUpdates	Элементы для обновления. Дополнительные сведения см. в разделе "CIsAndRelationsUpdates" на странице 344.
ignoreValidation	При использовании значения true проверка перед обновлением CMDB не выполняется.
dataStore	Сведения об инициаторе изменения.

Исходящие параметры

Параметр	Комментарий
createdIDsMapList	Список идентификаторов клиентов, сопоставленных с идентификаторами CMDB. Дополнительные сведения см. в разделе "addCIsAndRelations" на странице 363.

Методы анализа влияния в UCMDB

Данный раздел содержит сведения о следующих методах:

- "calculateImpact" ниже
- "getImpactPath" на следующей странице
- "getImpactRulesByNamePrefix" на странице 367

calculateImpact

Метод `calculateImpact` определяет, какие ЭК затрагиваются указанным ЭК в соответствии с правилами, заданными в CMDB.

Здесь показан эффект события, инициирующего правило. Выходной аргумент `identifier` метода `calculateImpact` используется в качестве входного элемента для "getImpactPath" на следующей странице.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
impactCategory	Тип события, которое инициирует моделирование правила.
IDs	Коллекция элементов CMDB или глобальных идентификаторов.
impactRulesNames	Коллекция элементов `ImpactRuleName`.
severity	Серьезность инициирующего события.

Исходящие параметры

Параметр	Комментарий
impactTopology	Подробнее см. в разделе "Topology" на странице 350.
identifier	Ключ ответа сервера.

getImpactPath

Метод `getImpactPath` извлекает топологический граф путь между затронутым и затрагивающим ЭК.

Выходной элемент `identifier` метода "calculateImpact" на предыдущей странице используется как входной элемент `identifier` метода `getImpactPath`.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
identifier	Ключ ответа сервера, возвращенный методом calculateImpact.
relation	Связь на основании одного из элементов "ShallowRelation", возвращенных методом calculateImpact в элементе `impactTopology`.

Исходящие параметры

Параметр	Комментарий
impactPathTopology	Коллекции `CIs` и `ImpactRelations`.
comments	Только для внутреннего использования.

Элемент `ImpactRelations` включает `ID`, `type`, `end1ID`, `end2ID`, `rule` и `action`.

getImpactRulesByNamePrefix

Метод `getImpactRulesByNamePrefix` получает правила с использованием фильтра префиксов.

Этот метод относится к правилам влияния, имя которых включает префикс, обозначающий контекст, к которому применяется правило, например `SAP_myrule`, `ORA_myrule` и т.д. Этот метод фильтрует все имена правил влияния и выводит только те из них, которые включают префикс, указанный в аргументе `ruleNamePrefixFilter`.

Входящие параметры

Параметр	Комментарий
cmdbContext	Дополнительные сведения см. в разделе "CmdbContext" на странице 348.
ruleNamePrefixFilter	Строка, содержащая первые буквы имен правил для вывода.

Исходящие параметры

Параметр	Комментарий
impactRules	`impactRules` включает ноль или более элементов `impactRule`. Элемент `impactRule`, который определяет влияние изменения, состоит из `ruleName`, `description`, `queryName` и `isActive`.

API-интерфейс веб-службы фактического состояния.

API-интерфейс веб-службы фактического состояния используется, в основном, приложением Service Manager, которое извлекает сведения о фактическом состоянии для определенного идентификатора CMDB или глобального идентификатора, а также определенного ID заказчика. API-интерфейс находит соответствующий запрос в папке **Integration/SM Query** и выполняет TQL-запрос, указав в качестве условия ID CMDB или глобальный ID, а затем возвращает результат запроса.

URL-адрес веб-службы: http://[machine_name]:8080/axis2/services/ucmdbSMService

Схема веб-службы: http://[machine_name]:8080/axis2/services/ucmdbSMService?xsd=xsd0

Поток

После вызова метода API он пытается найти соответствующий запрос в папке **Integration/SM Query**. Он пытается сопоставить тип запрошенного CMDBID/GlobalID с одним из запросов в папке. Для этого сначала выполняется поиск **QueryElement** с именем **Root**, а если таковой не найден, метод пытается найти любой **QueryNode** того же типа, как запрошенный CMDBID/GlobalID. Когда найдены соответствующий запрос и QueryNode,

CMDBID/GlobalID указывается в качестве условия для QueryNode, после чего выполняется запрос. Результат возвращается программе, вызвавшей API.

Обработка результатов с помощью преобразований

В некоторых случаях необходимо дополнительно преобразовать итоговый XML-код (например, сложить объем всех дисков и указать полученное значение как дополнительный атрибут ЭК). Чтобы выполнить дополнительное преобразование результатов TQL-запроса, поместите ресурс **[tql_name].xslt** в файл конфигурации адаптера следующим образом: **Управление адаптерами > ServiceDeskAdapter7-1 >Файлы конфигурации > [tql_name].xslt.**

Журналы API-интерфейса веб-службы фактического состояния

Настройки журналов UCMDB хранятся в директории: **UCMDBServer/Conf/log** в различных файлах с расширением ***.properties**.

Чтобы просмотреть журналы потока фактического состояния SM:

1. Откройте файл **cmdb_soaapi.properties** и установите уровень ведения журнала DEBUG: **loglevel=DEBUG**.
2. Откройте файл **fcmdb.properties** и установите уровень ведения журнала DEBUG: **loglevel=DEBUG**.
3. Подождите 1 минуту, пока сервер получит изменения.
4. Запустите фактическое состояние в SM.
5. Просмотрите следующие файлы журнала в директории **UCMDBServer/Runtime/log**:
   - cmdb.soaapi.log
   - fcmdb.log

Включение фактического состояния реплицированных ЭК после изменения корневого контекста

В случае изменения корневого контекста, используемого для доступа в UCMDB, для включения фактического состояния реплицированных ЭК необходимо внести следующие изменения в конфигурацию:

1. В папке **UCMDBServer\deploy\axis2\WEB-INF** откройте файл **web.xml**.
2. Добавьте следующий параметр **servlet init** в AxisServlet (вставьте эти четыре строки после строки 28):

   ```
   <init-param>

   <param-name>axis2.find.context</param-name>

   <param-value>false</param-value>

   </init-param>
   ```

   Эта строка предотвращает попытку Axis2 рассчитать корень контекста, указывая, что его необходимо найти в файле **axis2.html**.
3. В папке **UCMDBServer\deploy\axis2\WEB-INF\conf** откройте файл **axis2.xml**.

4. В строке 58 удалите символ комментария с параметра **contextRoot** и измените его следующим образом:

   ```
   <parameter name="contextRoot" locked="false">test/axis2</parameter>
   ```

   (где **test** — новый корневой контекст в **cmdb.xml**).

   Примечание. Обратите внимание на отсутствие наклонной черты в начале **test/axis2**.

API-интерфейс веб-службы UCMDB — сценарии использования

В следующих сценариях использования предполагается наличие двух систем:

- HP Universal CMDB Сервер
- Сторонняя система, содержащая репозиторий ЭК

Этот раздел охватывает следующие темы:

- "Заполнение CMDB " ниже
- "Выполнение запросов к CMDB " ниже
- "Запрос модели классов" на следующей странице
- "Анализ влияния изменений" на следующей странице

Заполнение CMDB

Сценарии использования:

- Сторонняя система управления ресурсами наполняет CMDB данными, доступными только в системе управления ресурсами.
- Несколько сторонних систем наполняют CMDB, создавая централизованную базу CMDB для отслеживания изменений и анализа влияния.
- Сторонняя система создает элементы конфигурации и связи в соответствии со сторонней бизнес-логикой для доступа к возможностям запросов CMDB.

Выполнение запросов к CMDB

Сценарии использования:

- Сторонняя система получает ЭК и связи, представляющие систему SAP, посредством получения результатов TQL-запроса SAP.
- Сторонняя система получает список серверов Oracle, добавленных или измененных за последние 5 часов.
- Сторонняя система получает список серверов, имена которых содержат строку *lab*.
- Сторонняя система находит элементы, связанные с указанным ЭК, путем получения его соседей.

Запрос модели классов

Сценарии использования:

- Сторонняя система дает пользователям возможность указать набор данных для получения из CMDB. Интерфейс пользователя может быть создан на основе модели классов для представления возможных свойств и запроса необходимых данных. Затем пользователь может выбрать информацию для получения.
- Сторонняя система анализирует модель классов, когда пользователь не может получить доступ к интерфейсу пользователя CMDB.

Анализ влияния изменений

Сценарий использования:

Сторонняя система выводит список бизнес-услуг, которые могут быть затронуты изменением указанного хоста.

Примеры

См. следующие примеры кода:

- The Example Base Class
- Query Example
- Update Example
- Class Model Example
- Impact Analysis Example

Эти файлы находятся в следующем каталоге:

\\<UCMDB root directory>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\WebServiceAPI_Samples

Глава 12: Управление потоком данных — API Java

Данная глава включает:

- Управление потоком данных — использование API Java 371

Управление потоком данных — использование API Java

Примечание. Эту главу следует использовать в сочетании с документом API DFM Javadoc, доступным в библиотеке документации.

В данной главе описывается, как сторонние инструменты могут контролировать потоки данных через API веб-службы HP Data Flow Management. API-интерфейс обеспечивает выполнение следующих функций:

- **Управление учетными данными**. Просмотр, добавление, обновление и удаление.
- **Управление заданиями**. Просмотр статуса, активация и деактивация.
- **Управление диапазонами зондов**. Просмотр, добавление и обновление.
- **Управление триггерами**. Добавление или удаление ЭК триггера, а также добавление, удаление и отключение TQL-запроса триггера.
- **Просмотр общих данных**. Данные по доменам и зондам.

Приведенные ниже службы доступны в пакете Discovery Services:

- **DDMConfigurationService.** Службы настройки зондов Data Flow Probe, кластеров, диапазонов IP-адресов, а также учетных записей. Сервер Universal Discovery можно настраивать с помощью XML-файла или зонда Data Flow Probe.
- **DDMManagementService.** Анализ и отображение хода выполнения, результатов и ошибок Universal Discovery.
- **DDMSoftwareSignatureService.** Указывает элементы ПО для обнаружения компонентами зонда Data Flow Probe. Определения работают в рамках всей системы. Если задано более одного компонента зонда Data Flow Probe, определения применяются к ним всем.
- **DDMZoneService.** Управление обнаружением по зонам.

Службы дополнены API-интерфейсами клиента управления потоком данных, которые используются для создания адаптеров Jython. Подробнее см. "Разработка адаптеров Jython" на странице 37.

Права доступа

Администратор предоставляет учетные данные для подключения к API-интерфейсу.

Клиент API-интерфейса должен иметь имя и пароль, заданные для пользователя интеграции в CMDB. Такие учетные записи представляют не пользователей CMDB (людей), а приложения, которые подключаются к CMDB.

Помимо этого, для входа в систему пользователю необходимо право **доступа к SDK**.

Внимание! Кроме того, клиент API-интерфейса может работать и с обычными пользователями, если у них есть право аутентификации через API. Однако использовать этот вариант не рекомендуется.

Дополнительные сведения см. в разделе "Создание пользователя интеграции" на странице 333.

Глава 13: API веб-службы управления потоком данных

Данная глава включает:

- API веб-службы управления потоком данных — обзор 373
- Обозначения 374
- Вызов веб-службы HP Data Flow Management 374
- Структуры данных и методы управления потоком данных 374
- Пример кода 385
- Пример добавления учетных данных 388

API веб-службы управления потоком данных — обзор

В данной главе описывается, как сторонние инструменты могут контролировать потоки данных через API веб-службы HP Data Flow Management.

API-интерфейс HP Data Flow Management — используется для интеграции приложений с HP Universal CMDB. API-интерфейс предоставляет методы для решения следующих задач:

- **Управление учетными данными**. Просмотр, добавление, обновление и удаление.
- **Управление заданиями**. Просмотр статуса, активация и деактивация.
- **Управление диапазонами зондов**. Просмотр, добавление и обновление.
- **Управление триггерами**. Добавление или удаление ЭК триггера, а также добавление, удаление и отключение TQL-запроса триггера.
- **Просмотр общих данных**. Данные по доменам и зондам.

Пользователи веб-службы HP Data Flow Management должны обладать следующими знаниями:

- Спецификация SOAP
- Объектно-ориентированный язык программирования, например C++, C# или Java.
- HP Universal CMDB
- Управление потоком данных

Примечание.

- Для входа в систему у пользователя должно быть право доступа **Запуск устаревшей API-функции**.
- Вошедший пользователь должен иметь право на общее действие **Запуск процессов обнаружения и интеграции** для доступа к любому из методов.

Полную документацию по доступным операциям см. в документе *HP Universal Discovery Schema Reference*. Эти файлы находятся по следующему пути:

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

Обозначения

В этой главе используются следующие условные обозначения:

- Стиль `элемент` указывает, что элемент является сущностью в базе данных или схеме. Этим же стилем обозначаются структуры, передаваемые методам и возвращаемые ими. Обычный текст указывает, что элемент рассматривается в общем контексте.
- Элементы и аргументы методов Управления потоком данных приводятся в том виде, в каком они указаны в схеме. Как правило это обозначает, что имена классов и общие ссылки на экземпляры классов пишутся с большой буквы. Для элементов и аргументов методов используется нижний регистр. Например, `credential` — это элемент типа `Credential`, переданный методу.

Вызов веб-службы HP Data Flow Management

API веб-службы HP Data Flow Management позволяет использовать для вызова серверных методов стандартные приемы программирования SOAP. Если инструкция не может быть обработана или если при вызове метода возникает проблема, методы API создают исключение `SoapFault`. При возникновении исключения `SoapFault` служба заполняет одно или несколько полей сообщения об ошибке, кода ошибки или сообщение об исключении. Если ошибка отсутствует, возвращается результат вывода.

Для вызова службы используются:

- `Протокол: http или https (в зависимости от настроек сервера)`
- `URL: <UCMDB server>:8080/axis2/services/DiscoveryService`
- `Пароль по умолчанию: "admin"`
- `Имя пользователя по умолчанию: "admin"`

Программисты SOAP могут получить доступ к WSDL по адресу:

- `axis2/services/DiscoveryService?wsdl`

Структуры данных и методы управления потоком данных

В этом разделе перечисляются методы управления потоком данных, а также структуры данных, применяемые API веб-службы. Кроме того здесь содержатся краткие примеры их использования. Полные сведения о запросах и ответах для каждой операции см. в документе *HP Universal Discovery Schema Reference*.

Этот раздел охватывает следующие темы:

- "Структуры данных" ниже
- "Методы управления заданиями обнаружения" ниже
- "Управление методами триггеров" на странице 377
- "Методы обработки данных зонда и домена" на странице 379
- "Методы учетных данных" на странице 381
- "Методы обновления данных" на странице 383

Структуры данных

API веб-службы управления потоком данных использует определенные структуры данных.

CIProperties

`CIProperties` — это коллекция коллекций. В каждой коллекции содержатся свойства определенного типа данных. Примеры коллекций: `dateProps`, `strListProps`, `xmlProps` и т.д.

В каждой из коллекций содержатся индивидуальные свойства определенного типа данных. Имена элементов свойств совпадают с именами контейнеров, но в единственном, а не множественном числе. К примеру, в коллекции `dateProps` содержатся элементы `dateProp`. Каждое свойство представляет собой пару из имени и значения.

См. CIProperties в документе *HP Universal Discovery Schema Reference*.

IPList

Список элементов `IP`, каждый из которых содержит адрес IPv4 или IPv6.

См. IPList (*Документация по схеме для HP Discovery and Dependency Mapping*).

IPRange

`IPRange` состоит из двух элементов — `Start` и `End`. В каждом из них содержится элемент `Address`, представляющий собой адрес IPv4 или IPv6.

См. IPRange в документе *HP Universal Discovery Schema Reference*.

Scope

Два элемента `IPRange`. `Exclude` — это коллекция `IPRanges`, которые необходимо исключить из задания. `Include` — это коллекция `IPRanges`, которые необходимо включить в задание.

См. Scope в документе *HP Universal Discovery Schema Reference*.

Методы управления заданиями обнаружения

activateJob

Активация определенного задания.

См. раздел "Пример кода" на странице 385.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя задания.

deactivateJob

Деактивация определенного задания.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя задания.

dispatchAdHocJob

Отправка задания на зонд по запросу. Задание должно быть активным и содержать указанный ЭК-триггер.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя задания.
CIID	Идентификатор ЭК-триггера.
ProbeName	Имя зонда.
Время ожидания	В мс

getDiscoveryJobsNames

Возврат списка имен заданий.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.

Исходящие параметры

Параметр	Комментарий
strList	Список имен заданий.

isJobActive

Проверка активности задания.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя проверяемого задания.

Исходящие параметры

Параметр	Комментарий
JobState	Указывает, является ли задание активным.

Управление методами триггеров

addTriggerCI

Добавление нового ЭК-триггера в указанное задание.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя задания.
CIID	Идентификатор ЭК-триггера.

addTriggerTQL

Добавление нового TQL-запроса в указанное задание.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя задания.
TqlName	Имя добавляемого TQL-запроса.

disableTriggerTQL

Предотвращение инициации задания TQL-запросом без его окончательного удаления из списка запросов, которые вызывают задание.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя задания.

removeTriggerCI

Удаление указанного ЭК из списка ЭК, которые инициируют задание.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя задания.
CIID	Идентификатор ЭК-триггера.

removeTriggerTQL

Удаление указанного TQL-запроса из списка запросов, которые инициируют задание.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Коллекция имен проверяемых заданий.
CIID	Идентификатор TQL-запроса, подлежащего удалению.

setTriggerTQLProbesLimit

Ограничение зондов, в которых активен TQL-запрос в задании из указанного списка.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
JobName	Имя задания.
tqlName	Имя TQL-запроса.
probesLimit	Список зондов, для которых активен TQL-запрос.

Методы обработки данных зонда и домена

getDomainType

Возврат типа домена.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Имя домена.

Исходящие параметры

Параметр	Комментарий
domainType	Тип домена.

getDomainsNames

Возврат имен текущих доменов.

См. раздел "Пример кода" на странице 385.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.

Исходящие параметры

Параметр	Комментарий
domainNames	Список доменных имен.

getProbeIPs

Возврат IP-адресов указанного зонда.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Проверяемый домен.
probeName	Имя зонда, используемого в этом домене.

Исходящие параметры

Параметр	Комментарий
probeIPs	"IPList" адресов на зонде.

getProbesNames

Возврат имен текущих зондов в указанном домене.

См. раздел "Пример кода" на странице 385.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Проверяемый домен.

Исходящие параметры

Параметр	Комментарий
probesName	Список зондов в домене.

getProbeScope

Возврат определения охвата указанного зонда.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Проверяемый домен.
probeName	Имя зонда.

Исходящие параметры

Параметр	Комментарий
probeScope	"Scope" зонда.

isProbeConnected

Проверка подключения указанного зонда.

См. раздел "Пример кода" на странице 385.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Проверяемый домен.
probeName	Проверяемый зонд.

Исходящие параметры

Параметр	Комментарий
isConnected	Указывает, подключен ли зонд.

updateProbeScope

Установка охвата указанного зонда с переопределением существующего охвата.

Входные

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Домен.
probeName	Обновляемый зонд.
newScope	Установка "Scope" зонда.

Методы учетных данных

addCredentialsEntry

Добавляет запись учетных данных с указанным протоколом в указанный домен.

См. раздел "Пример кода" на странице 385.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Обновляемый домен.
protocolName	Имя протокола.
credentialsEntryParameters	Коллекция "CIProperties" новых учетных данных.

Исходящие параметры

Параметр	Комментарий
credentialsEntryID	Идентификатор ЭК новых учетных данных.

getCredentialsEntriesIDs

Возврат идентификаторов учетных данных, заданных для указанного протокола.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Домен, для которого необходимо получить учетные данные.
protocolName	Имя протокола, используемого в этом домене.

Исходящие параметры

Параметр	Комментарий
credentialsEntryIDs	Список идентификаторов учетных данных для протокола в данном домене.

getCredentialsEntry

Возврат учетных данных, заданных для указанного протокола. Зашифрованные атрибуты возвращаются пустыми.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Домен, для которого необходимо получить учетные данные.
protocolName	Имя протокола, используемого в этом домене.
credentialsEntryID	Идентификатор получаемых учетных данных.

Исходящие параметры

Параметр	Комментарий
credentialsEntryParameters	Коллекция "CIProperties" новых учетных данных.

removeCredentialsEntry

Удаление выбранной записи учетных данных из протокола.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Домен.
protocolName	Имя протокола, используемого в этом домене.
credentialsEntryID	Идентификатор учетных данных, подлежащих удалению.

updateCredentialsEntry

Установка новых значений свойств указанной записи учетных данных.

Эти свойства устанавливаются вместо текущих. Свойства, значения которых не установлены данным вызовом, остаются неопределенными.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
domainName	Домен, в котором необходимо обновить учетные данные.
protocolName	Имя протокола, используемого в этом домене.
credentialsEntryID	Идентификатор обновляемых учетных данных.
credentialsEntryParameters	Коллекция свойств "CIProperties", которые необходимо установить как свойства учетных данных.

Методы обновления данных

rediscoverCIs

Поиск триггеров, обнаруживших указанные объекты ЭК, и повторное выполнение этих триггеров. Метод **rediscoverCIs** выполняется асинхронно. Вызовите метод **checkDiscoveryProgress**, чтобы определить, когда повторное обнаружение будет выполнено.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
CmdbIDs	Коллекция идентификаторов объектов, подлежащих повторному обнаружению.

Исходящие параметры

Параметр	Комментарий
isSucceed	Указывает, успешно ли выполнено повторное обнаружение ЭК.

checkDiscoveryProgress

Возврат хода выполнения последнего вызова **rediscoverCIs** для указанных идентификаторов. Ответ — это значение от 0 до 1. Ответ 1 означает, что вызов **rediscoverCIs** завершен.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
CmdbIDs	Коллекция идентификаторов отслеживаемых объектов в вызове повторного обнаружения.

Исходящие параметры

Параметр	Комментарий
progress	Для завершенного задания имеет значение 1. Незавершенные задания имеют значение менее 1.

rediscoverViewCIs

Поиск триггеров, сформировавших данные для заполнения указанного представления, и повторное выполнение этих триггеров. Метод **rediscoverViewCIs** выполняется асинхронно. Вызовите метод **checkViewDiscoveryProgress**, чтобы определить, когда будет выполнено повторное обнаружение.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
viewName	Проверяемые представления.

Выходной

Параметр	Комментарий
isSucceed	Указывает, успешно ли выполнено повторное обнаружение ЭК.

checkViewDiscoveryProgress

Возврат хода выполнения последнего вызова **rediscoverViewCIs** для указанного представления. Ответ — это значение от 0 до 1. Ответ 1 означает, что вызов **rediscoverCIs** завершен.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 348.
viewName	Коллекция проверяемых представлений.

Исходящие параметры

Параметр	Комментарий
progress	Для завершенного задания имеет значение 1. Незавершенные задания имеют значение менее 1.

Пример кода

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.*;
import com.hp.ucmdb.generated.types.*;
public class test {
    static final String HOST_NAME = "<my_hostname>";
    static final int PORT = 8080;
    private static final String PROTOCOL = "http";
    private static final String FILE = "/axis2/services/DiscoveryService";

    private static final String PASSWORD = "<my_password>";
    private static final String USERNAME = "<my_username>";

    private static CmdbContext cmdbContext = new CmdbContext("ws tests");

    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest(
            "Range IPs by ICMP", cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protcol
        addNTCMDCredentialsEntry();
    }

    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();
```

```
        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(
                new GetDomainsNamesRequest(cmdbContext)).
                getDomainNames();
        if (domains.sizeStrValueList() == 0) {
            System.out.println("No domains were found, can't create credentials");
            return;
        }
        String domainName = domains.getStrValue(0);
        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user", newCredsProperties);
        setStringProperties("ntadminprotocol_ntdomain",
            "test doamin", newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(
            new AddCredentialsEntryRequest(domainName,
            "ntadminprotocol",  newCredsProperties, cmdbContext));
        System.out.println("new credentials craeted for domain: " + domainName + "
in ntcmd protocol");
    }

    private static void setPasswordProperty(CIProperties newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }

    private static void setStringProperties(String propertyName, String value,
CIProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }


    private static void getProbesInfo(DiscoveryService discoveryService) throws
Exception {
```

```
        GetDomainsNamesResponse result = discoveryService.getDomainsNames(new
GetDomainsNamesRequest(cmdbContext ));
        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName =
            result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(
                    new GetProbesNamesRequest(domainName, cmdbContext));
            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames().sizeStrValueList(); i++)
{
                String probeName = probesResult.getProbesNames().getStrValue(i);
                // Check if connected
                IsProbeConnectedResponce connectedRequest =
                    discoveryService.isProbeConnected(
                        new IsProbeConnectedRequest(
                        domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected();
                // Do something ...
                System.out.println("probe " + probeName + " isconnect=" +
isConnected);
            }
        }
    }

    private static DiscoveryService getDiscoveryService() throws Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub =
                new DiscoveryServiceStub(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth =
                new HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty(
                HTTPConstants.AUTHENTICATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }
        return discoveryService;
    }
}
```

Пример добавления учетных данных

```
import java.net.URL;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
import com.hp.ucmdb.generated.params.discovery.*;
import com.hp.ucmdb.generated.services.DiscoveryService;
import com.hp.ucmdb.generated.services.DiscoveryServiceStub;
import com.hp.ucmdb.generated.types.BytesProp;
import com.hp.ucmdb.generated.types.BytesProps;
import com.hp.ucmdb.generated.types.CIProperties;
import com.hp.ucmdb.generated.types.CmdbContext;
import com.hp.ucmdb.generated.types.StrList;
import com.hp.ucmdb.generated.types.StrProp;
import com.hp.ucmdb.generated.types.StrProps;

public class test {
    static final String HOST_NAME = "hostname";
static final int PORT = 8080;
    private static final String PROTOCOL = "http";
    private static final String FILE = "/axis2/services/DiscoveryService";

    private static final String PASSWORD = "admin";
    private static final String USERNAME = "admin";

    private static CmdbContext cmdbContext = new CmdbContext("ws tests");

    public static void main(String[] args) throws Exception {
        // Get the stub object
        DiscoveryService discoveryService = getDiscoveryService();

        // Activate Job
        discoveryService.activateJob(new ActivateJobRequest("Range IPs by ICMP",
cmdbContext));

        // Get domain & probes info
        getProbesInfo(discoveryService);
        // Add credentilas entry for ntcmd protcol
        addNTCMDCredentialsEntry();
    }

    public static void addNTCMDCredentialsEntry() throws Exception {
        DiscoveryService discoveryService = getDiscoveryService();

        // Get domain name
        StrList domains =
            discoveryService.getDomainsNames(new GetDomainsNamesRequest
(cmdbContext)).getDomainNames();
        if (domains.sizeStrValueList() == 0) {
```

```
            System.out.println("No domains were found, can't create credentials");
            return;
        }
        String domainName = domains.getStrValue(0);
        // Create propeties with one byte param
        CIProperties newCredsProperties = new CIProperties();

        // Add password property - this is of type bytes
        newCredsProperties.setBytesProps(new BytesProps());
        setPasswordProperty(newCredsProperties);

        // Add user & domain properties - these are of type string
        newCredsProperties.setStrProps(new StrProps());
        setStringProperties("protocol_username", "test user", newCredsProperties);
        setStringProperties("ntadminprotocol_ntdomain", "test doamin",
newCredsProperties);

        // Add new credentials entry
        discoveryService.addCredentialsEntry(new AddCredentialsEntryRequest
(domainName,  "ntadminprotocol",  newCredsProperties, cmdbContext));
        System.out.println("new credentials craeted for domain: " + domainName + "
in ntcmd protocol");
    }

    private static void setPasswordProperty(CIProperties newCredsProperties) {
        BytesProp bProp = new BytesProp();
        bProp.setName("protocol_password");
        bProp.setValue(new byte[] {101,103,102,104});
        newCredsProperties.getBytesProps().addBytesProp(bProp);
    }

    private static void setStringProperties(String propertyName, String value,
CIProperties newCredsProperties) {
        StrProp strProp = new StrProp();
        strProp.setName(propertyName);
        strProp.setValue(value);
        newCredsProperties.getStrProps().addStrProp(strProp);
    }

    private static void getProbesInfo(DiscoveryService discoveryService) throws
Exception {
        GetDomainsNamesResponse result = discoveryService.getDomainsNames(new
GetDomainsNamesRequest(cmdbContext ));
        // Go over all the domains
        if (result.getDomainNames().sizeStrValueList() > 0) {
            String domainName = result.getDomainNames().getStrValue(0);
            GetProbesNamesResponse probesResult =
                discoveryService.getProbesNames(new GetProbesNamesRequest
(domainName, cmdbContext));
            // Go over all the probes
            for (int i=0; i<probesResult.getProbesNames().sizeStrValueList(); i++)
```

```
{
                String probeName = probesResult.getProbesNames().getStrValue(i);
                // Check if connected
                IsProbeConnectedResponce connectedRequest =
                    discoveryService.isProbeConnected(new IsProbeConnectedRequest
(domainName, probeName, cmdbContext));
                Boolean isConnected = connectedRequest.getIsConnected();
                // Do something ...
                System.out.println("probe " + probeName + " isconnect=" +
isConnected);
            }
        }
    }

    private static DiscoveryService getDiscoveryService() throws Exception {
        DiscoveryService discoveryService = null;
        try {
            // Create service
            URL url = new URL(PROTOCOL,HOST_NAME,PORT, FILE);
            DiscoveryServiceStub serviceStub = new DiscoveryServiceStub
(url.toString());

            // Authenticate info
            HttpTransportProperties.Authenticator auth = new
HttpTransportProperties.Authenticator();
            auth.setUsername(USERNAME);
            auth.setPassword(PASSWORD);
            serviceStub._getServiceClient().getOptions().setProperty
(HTTPConstants.AUTHENTICATE,auth);

            discoveryService = serviceStub;
        } catch (Exception e) {
            throw new Exception("cannot create a connection to service ", e);
        }
        return discoveryService;
    }
} // End class
```

Отправить отзыв о документации

Если у вас есть комментарии к данному документу, обратитесь в отдел документации по электронной почте. Если на вашем компьютере настроен клиент электронной почты, при нажатии на ссылку выше откроется окно нового сообщения, в теме которого будет указана следующая информация:

Отзывы о Справочное руководство для разработчиков (Universal CMDB 10.20)

Напишите в сообщении свой отзыв и отправьте его нам.

Если клиент электронной почты не настроен, скопируйте приведенную выше информацию в окно нового сообщения вручную, а затем отправьте свой отзыв по адресу cms-doc@hp.com.

Благодарим за отзыв!