HP Universal CMDB

Для операционных систем Windows и Red Hat Enterprise Linux

Версия программного обеспечения: 10.00

Справочное руководство для разработчиков

Дата выпуска документа: Июнь 2012 г.

Дата выпуска программного обеспечения: Июнь 2012 г.

Правовые уведомления

Гарантия

Гарантии на продукты и услуги HP формулируются только в заявлениях о прямой гарантии, сопровождающих эти продукты и услуги. Никакая часть настоящего документа не может быть истолкована как дополнительная гарантия. Компания HP не несет ответственности за содержащиеся здесь технические или редакционные ошибки.

Приводимые в настоящем документе сведения могут быть изменены без предварительного уведомления.

Пояснение об ограниченных правах

Конфиденциальное компьютерное программное обеспечение. Для обладания, использования или копирования необходима действующая лицензия от компании HP. В соответствии с нормами FAR 12.211 и 12.212, коммерческое компьютерное программное обеспечение, документация на компьютерное программное обеспечение и технические данные для коммерческих позиций лицензируются государственным организациям США на условиях стандартной коммерческой лицензии поставщика.

Заявление об авторских правах

Заявления о товарных знаках

Adobe™ является товарным знаком компании Adobe Systems Incorporated.

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

UNIX® является зарегистрированным товарным знаком группы The Open Group.

Обновления документации

На титульном листе настоящего документа приведены следующие идентификационные данные.

- Номер версии программного обеспечения для указания версии ПО.
- 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

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

Чтобы воспользоваться этим сайтом, необходимо зарегистрировать идентификатор HP Passport и войти в систему. Регистрация HP Passport ID производится на сайте

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

или по ссылке **New users - please register** на странице входа в HP Passport.

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

Используйте веб-сайт технической поддержки программного обеспечения компании HP по адресу

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

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 средствам 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 ПО;
- управление договорами на техническую поддержку;
- поиск контактов технической поддержки HP;
- проверка сведений о доступных услугах;
- участие в обсуждениях различных вопросов с другими заказчиками ПО;
- исследование определенных проблем и регистрация для обучения работе с программным обеспечением.

В большинстве случаев для получения поддержки требуется регистрация HP Passport, а также договор на услуги технической поддержки. Чтобы зарегистрироваться для получения идентификатора HP Passport ID, перейдите на веб-сайт

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

Дополнительные сведения об уровнях доступа представлены на сайте

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

Заявление об отказе от ответственности для 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.

Примечание. Некоторые разделы неправильно преобразуются в формат PDF, что приводит к проблемам с форматированием. Некоторые элементы интерактивной справки полностью удалены из PDF-версии. Эти разделы можно распечатать из интерактивной справки.

Содержание

Создание адаптеров обнаружения и интеграции

Глава 1

Разработка и написание адаптеров

Данная глава включает:

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

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

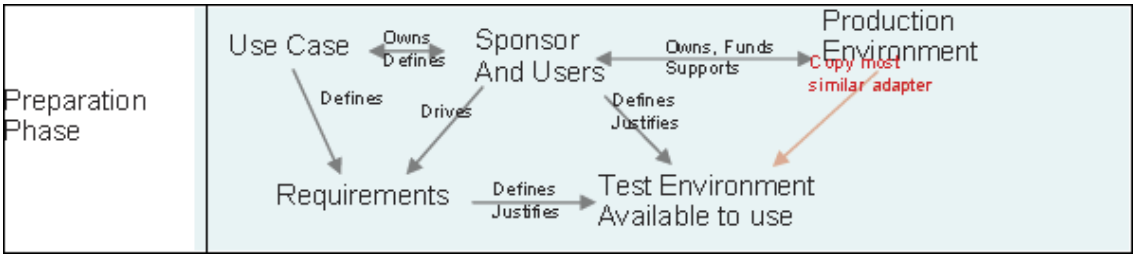
Каждый этап разработки адаптера основывается на предыдущем этапе.

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. См. дополнительные сведения об упаковке в разделе "Диспетчер пакетов" на странице 1 документа *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 17

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 "Resources Pane" (on page 1) (*Руководство по управлению потоками данных в HP Universal CMDB*).
2. Исследование методов сбора данных адаптером:
   - Использование внешних средств 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 упаковки и коммерческого внедрения 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" на странице 1 (*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" на странице 1 (*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	HP ServiceCenter

Тип интеграции	Пример, доступный для использования
экспорте	
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. Подробнее см. в разделе "Обзор разработки и написания адаптеров" на странице 14.

Этот раздел охватывает следующие темы:

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 по следующему пути: **Управление потоком данных > Настройка зонда потока данных > Домены и зонды**. Подробнее см. в разделе "Панель "Домены и зонды"" на странице 1 (*Руководство по управлению потоками данных в HP Universal CMDB*).

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?

  Необходимо **наполнение**.

- 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?

  Необходимо **объединение**.

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

  Необходима **принудительная отправка данных**.

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

См. сведения о различных типах интеграции в разделе "Студия интеграции" на странице 1 документа *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 40.
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 соединение одного источника или целевого объекта.

   Подробнее см. в разделе "Разработка адаптеров Java" на странице 156.
3. Общий адаптер БД

- Абстрактный адаптер, основанный на адаптере Java и использующий платформу Federation SDK Framework.
- Обеспечивает создание адаптеров для подключения внешних репозиториев данных.
- Поддерживает объединение и наполнение (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 соединение одного источника данных.

Подробнее см. в разделе "Разработка общих адаптеров БД" на странице 84.

4. Общий адаптер принудительной отправки
   - 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 соединение одного целевого объекта.
   - Используется для принудительной отправки данных.

   Подробнее см. в разделе "Разработка адаптеров принудительной отправки" на странице 186.

В таблице ниже приводятся возможности каждого адаптера:

Поток/Адаптер	Адаптер Jython	Адаптер Java	Общий адаптер БД	Адаптер принудительной отправки
Заполнение	X	X	X	
Объединение		X	X	
Принудительная отправка данных		X		X

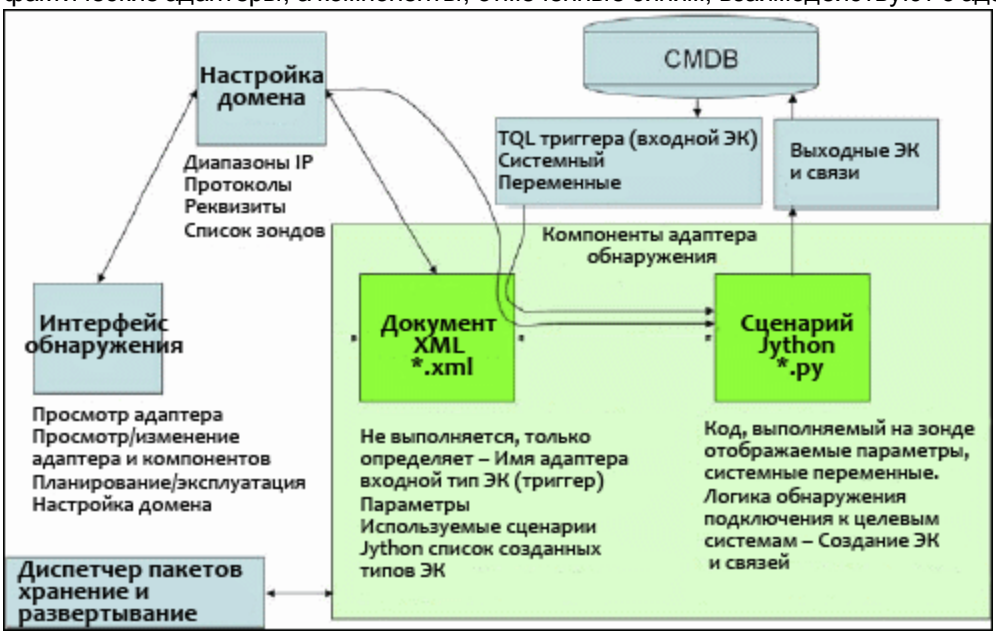
Разработка содержимого обнаружения

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. Вы можете использовать страницы «Панель управления обнаружением», «Управление адаптерами» и «Настройка зонда для потока данных» при создании адаптеров.

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

С технической точки зрения 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. Подробнее см. в разделе "Шаг 1: Создание адаптера" на странице 29.

2. **Создание задания обнаружения.**

   Настройка задания с данными планирования и запроса-триггера. Подробнее см. в разделе "Шаг 2: Назначение задания адаптеру" на странице 36.

3. **Изменение кода обнаружения.**

   Изменение кода Jython или Java, который содержится в файлах адаптера и ссылается на платформу DFM. Подробнее см. в разделе "Шаг 3: Создание кода Jython" на странице 38.

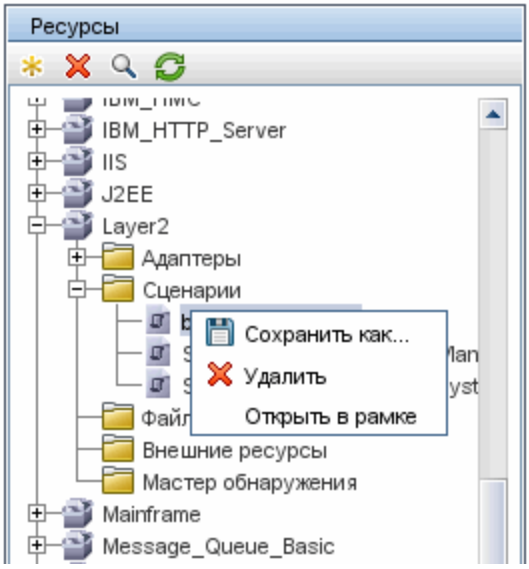
Для создания новых адаптеров создаются все компоненты, описанные выше, и каждый из них автоматически привязывается к компонентам в предыдущем шаге. Например, после создания задания и выбора соответствующего адаптера файл адаптера привязывается к заданию.

Код адаптера

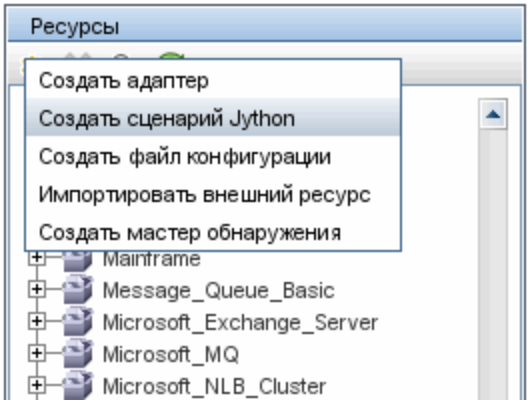
Фактическая реализация подключения к удаленной системе, запроса ее данных и их сопоставление с данными CMDB основывается на коде Jython. Например, код содержит логику подключения к базе данных и извлечения данных из нее. В этом случае код ожидает URL-адрес JDBC, имя пользователя, пароль и т. д. Эти параметры 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.

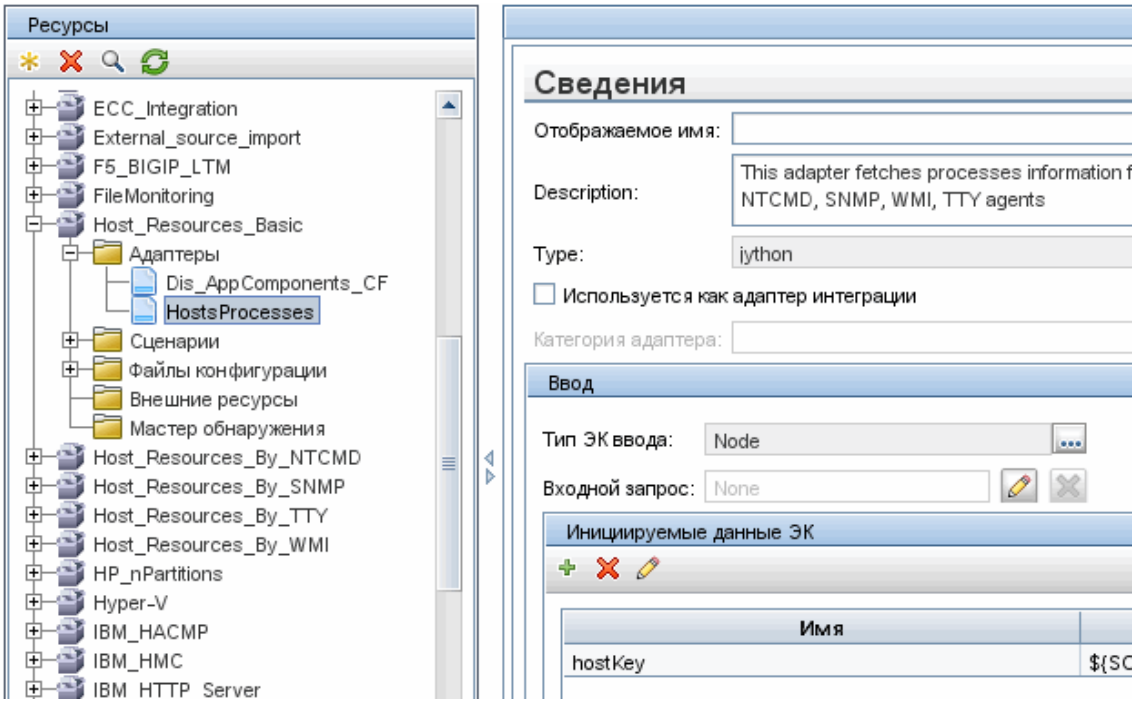
Адаптер может содержать список сценариев, которые будут использоваться при выполнении обнаружения. При создании адаптера обычно создается новый сценарий, который назначается адаптеру. Новый сценарий включает базовые шаблоны, но вы можете использовать один из прочих сценариев в качестве шаблона, щелкнув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



См. дополнительные сведения о создании новых сценариев Jython в "Шаг 3: Создание кода Jython" на странице 38. 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 "Related CIs Window" документа *Руководство по управлению потоками данных в HP Universal CMDB*.) Код адаптера является стандартным за исключением этих входных параметров ЭК-триггера, которые передаются в код.

См. дополнительные сведения о входных компонентах в разделе "Data Flow Management Concepts" документа *Руководство по управлению потоками данных в HP Universal CMDB*.

Этот раздел охватывает следующие темы:

- "Определение входных данных адаптера (тип ЭК-триггера и входной запрос)" на следующей странице

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 атрибута `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. Подробнее см. в разделах "Пример определения входного запроса" на следующей странице ниже и "Add Query Nodes and Relationships to a TQL Query" (*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.

   Пример определения типа ЭК-триггера:

   В этом примере тип ЭК-триггера указывает, что ЭК IP разрешены в адаптере.

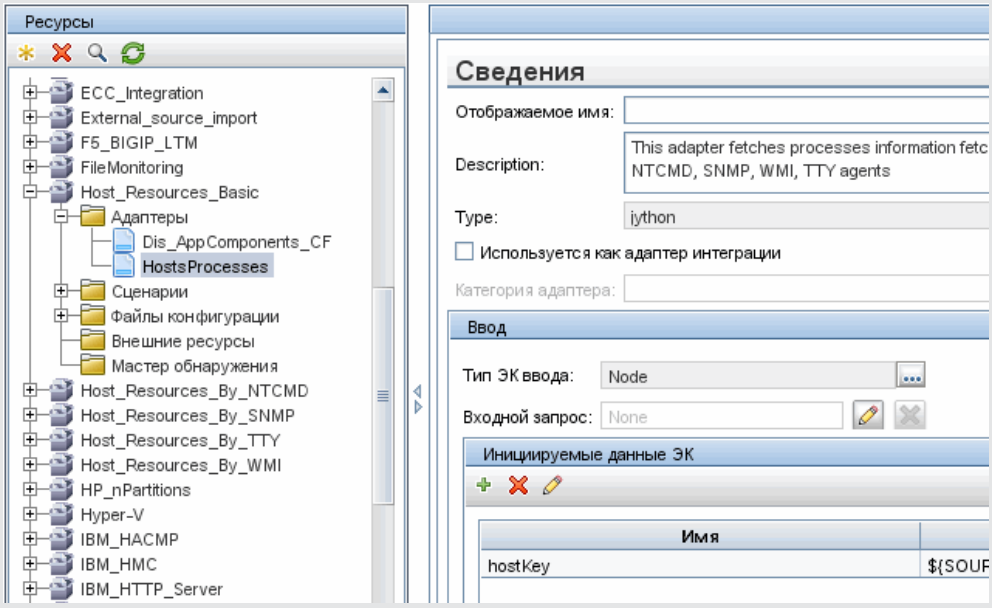
   a. Откройте раздел **Управление потоком данных > Управление адаптерами**. Выберите адаптер **HostProcesses** (**Пакеты > Host_Resources_Basic > Адаптеры > HostProcesses**).

   b. Найдите поле «Тип ЭК ввода». Подробнее см. в разделе "Adapter Definition Tab" (*Руководство по управлению потоками данных в HP Universal CMDB*).

   c. Нажмите кнопку, чтобы открыть диалоговое окно «Выберите класс обнаружения». Подробнее см. в разделе "Choose Discovered Class Dialog Box" (*Руководство по управлению потоками данных в HP Universal CMDB*

   d. Выберите типы ЭК.

В этом примере тип ЭК IP (хост) разрешен в адаптере.



Пример определения входного запроса

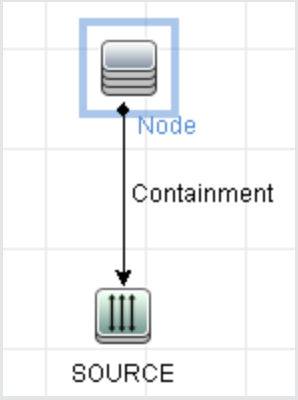
В этом примере входной TQL-запрос указывает, что ЭК `IpAddress` (настроенный в предыдущем примере как тип ЭК-триггера) должен быть подключен к ЭК `Node`.

a. Откройте раздел **Управление потоком данных > Управление адаптерами**. Найдите поле «Входной запрос». Нажмите кнопку **Изменить**, чтобы открыть редактор входного запроса. Подробнее см. в разделе "Input Query Editor Window" (*Руководство по управлению потоками данных в HP Universal CMDB*

b. В редакторе входного запроса назначьте узлу ЭК триггера имя **SOURCE**: щелкните узел правой кнопкой мыши и выберите команду **Свойства узла запроса**. В поле **Имя элемента** измените имя на **SOURCE**.

c. Добавьте ЭК `Node` и связь `Containment` в ЭК `IpAddress`. См. дополнительные сведения об использовании редактора TQL в разделе "Input Query Editor Window" документа *Руководство по управлению потоками данных*

в HP Universal CMDB.



ЭК **IpAddress** соединен с ЭК **Node**. Входной запрос состоит из двух узлов, **Node** и **IpAddress**, со связью между ними. ЭК **IpAddress** имеет имя **SOURCE**.

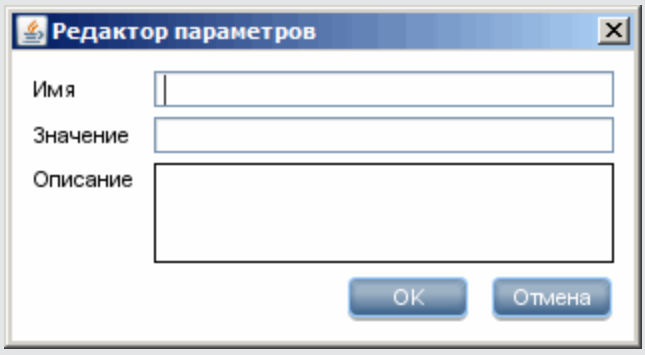
Пример добавления переменных к TQL-запросу ввода:

В этом примере вы добавите переменные DIRECTORY и CONFIGURATION_FILE к TQL-запросу ввода, созданному в предыдущем примере. Эти переменные помогают определить компоненты для обнаружения. В этом примере необходимо найти файлы конфигурации на хостах, связанных с IP-адресами для обнаружения.

a. Откройте входной TQL-запрос созданный в предыдущем примере.

   Откройте раздел **Управление потоком данных > Управление адаптерами**. Найдите панель «Инициируемые данные ЭК». Подробнее см. в разделе "Adapter Definition Tab" (*Руководство по управлению потоками данных в HP Universal CMDB*).

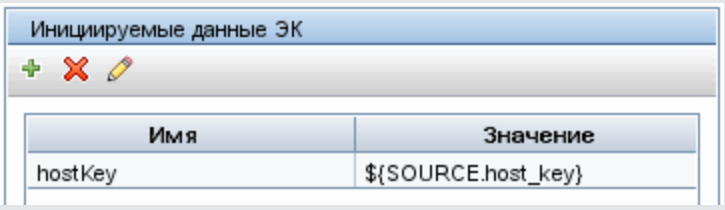
b. Добавление переменных к входному TQL-запросу. Подробнее см. в разделе меню **Управление потоком данных > Управление адаптерами**. Найдите панель «Инициируемые данные ЭК». Подробнее см. в описании поля «Переменные» в разделе "Adapter Definition Tab" (*Руководство по управлению потоками данных в HP Universal CMDB*).



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:



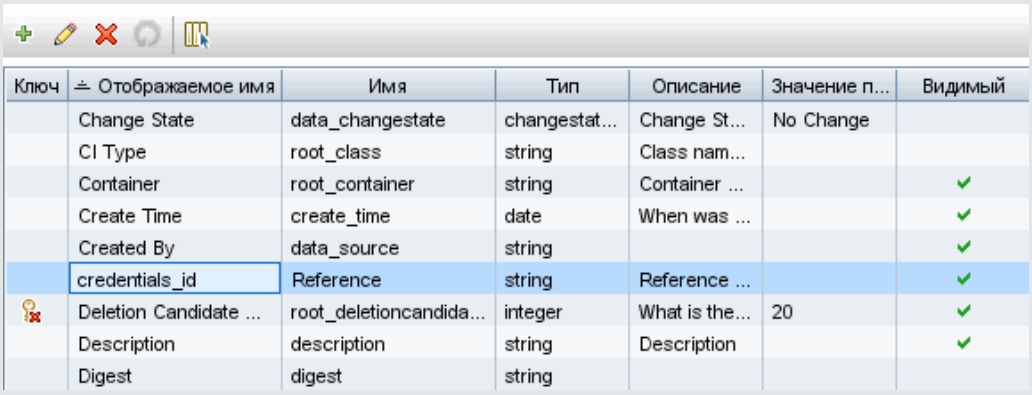
Инициируемые данные ЭК передаются зонду, причем все переменные заменяются фактическими значениями. Сценарий адаптера включает команду, которая позволяет извлечь фактические значения указанных переменных с помощью DFM Framework:

```
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).

Щелкните здесь, чтобы увидеть иллюстрацию.

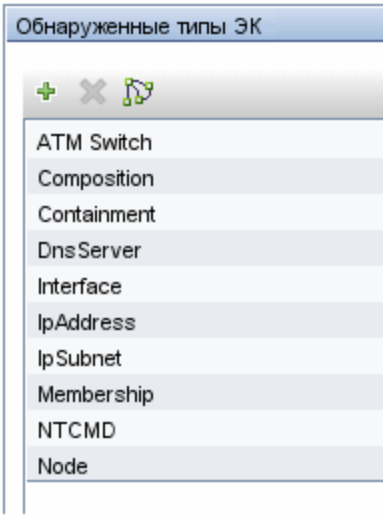
Переменные `Protocol`, `credentialsId` и `ip_address` используют атрибуты `root_class`, `credentials_id` и `application_ip`:



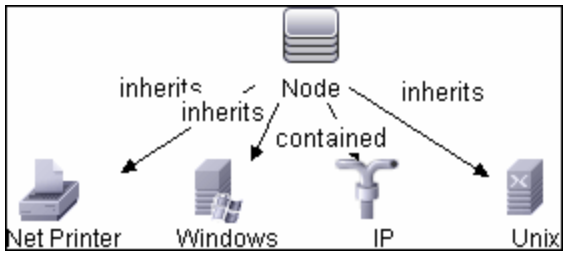
Ключ	Отображаемое имя	Имя	Тип	Описание	Значение п...	Видимый
	Change State	data_changestate	changestat...	Change St...	No Change	
	CI Type	root_class	string	Class nam...		
	Container	root_container	string	Container ...		✔
	Create Time	create_time	date	When was ...		✔
	Created By	data_source	string			✔
	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 45.

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» во 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 "Adapter Definition Tab" (*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`.

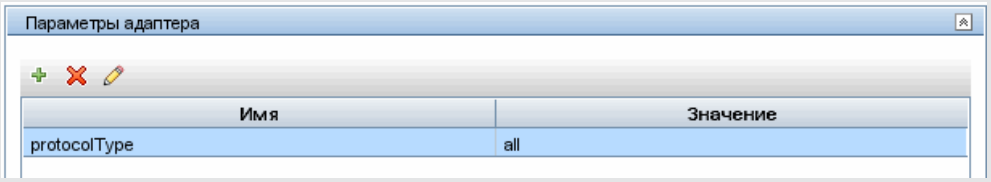
Пример переопределения параметров адаптера:

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**.

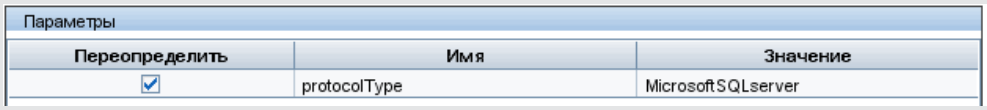
b. На панели ресурсов выберите **Database_Basic > Адаптеры > SQL_NET_Dis_Connection**.

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.

Сведения о добавлении, удалении и изменении параметров см. в разделе "Adapter Definition Tab" (*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

В 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 типа ЭК запущенного ПО:** использовать атрибуты **application_ip** и **application_ip_domain** и выбрать зонд, указанный для IP-адреса в соответствующем домене.
- **Для других типов ЭК:** взять IP-адрес узла, связанного с ЭК (если он есть).

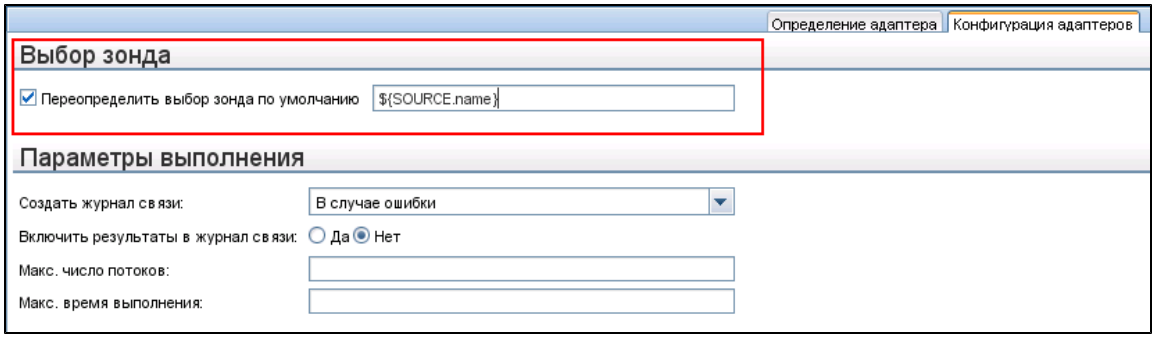
Автоматический выбор зонда выполняется для узла, связанного с ЭК. После получения связанного ЭК узла механизм диспетчеризации выбирает один из IP-адресов узла, а затем выбирает зонд в соответствии с определениями охвата сети зонда.

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, сделав следующее:

a. В разделе "Выбор зонда", выберите **Переопределить выбор зонда по умолчанию**, как показано ниже.



b. В поле "Зонд" укажите зонд, который следует использовать для данной задачи.

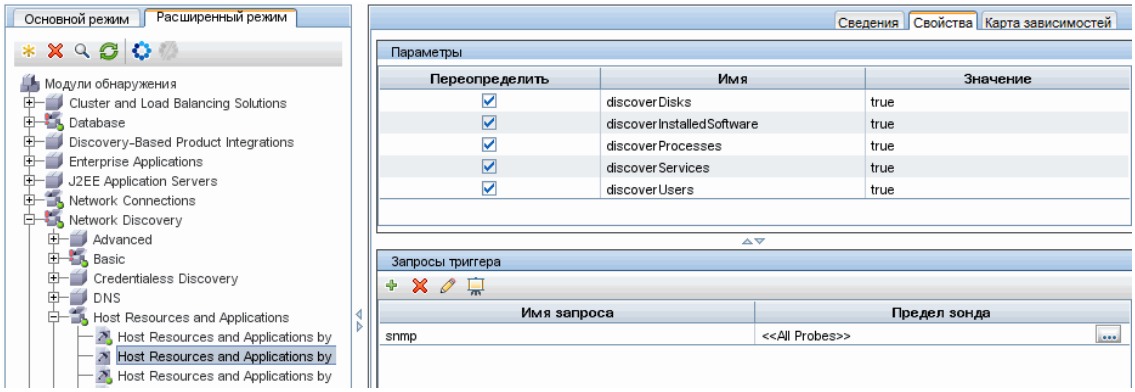
5. **Настройте каталог classpath для удаленного процесса (необязательно).**

   Подробнее см. в разделе "Настройка удаленного выполнения процессов" на странице 38.

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

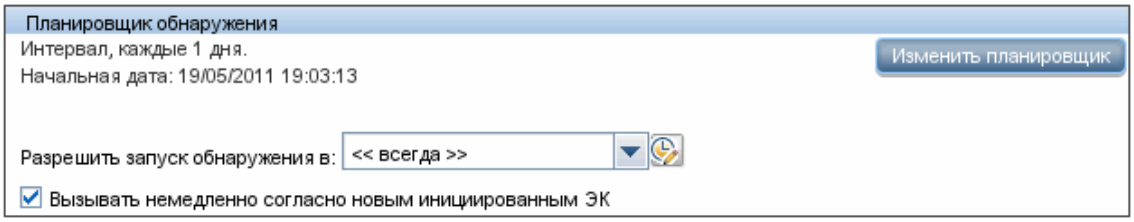
Каждое задание связывается с TQL-запросом триггера. Эти TQL-запросы триггера публикуют результаты, которые используются как входные ЭК-триггеры для этого задания.

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 (*Руководство по управлению потоками данных в HP Universal CMDB*).

Переопределение параметров адаптера

При настройке задания можно переопределить параметры адаптера. Подробнее см. в разделе "Переопределение параметров адаптера" на странице 34.

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 вебсайтах:

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

Подробнее см. в разделе "Создание кода Jython" на странице 40.

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_**

Параметр	Описание

	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

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

C:\hp\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 вебсайтах:

- 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 DFM, должны быть совместимы с Jython версии 2.1.
- См. полную документацию по доступным API-интерфейсам в документе *HP Universal CMDB Data Flow Management API Reference*.

Этот раздел охватывает следующие темы:

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 Resources Pane (*Руководство по управлению потоками данных в HP Universal CMDB*). После добавления сценария в пакет его также можно будет использовать для других заданий. См. дополнительные сведения об использовании диспетчера пакетов в разделе "Диспетчер пакетов" на странице 1 документа *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. Основная 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:

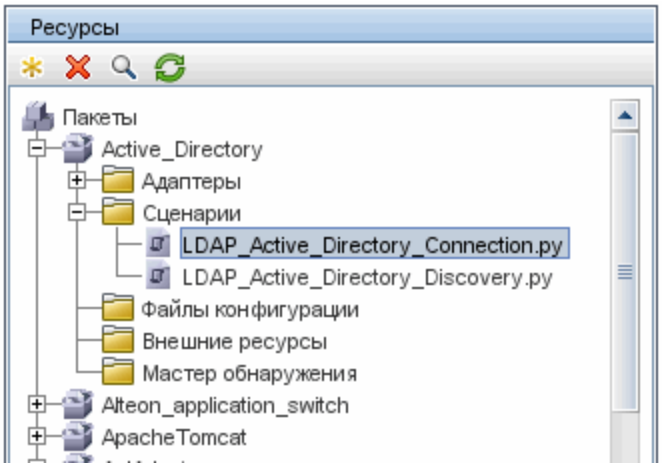
```
def DiscoveryMain(Framework):
```

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.getClientFactory(ClientsConsts.SNMP_
PROTOCOL_NAME).createClient()
    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`. См. дополнительные сведения об End 1 и End 2 в разделе "Связь" на странице 1 документа *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" на странице 1 (*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)
```

Подробнее 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 подразделе "modeling.py" на странице 77 раздела "Средства и библиотеки Jython" на странице 76.

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 "Справка по API-интерфейсу Управления потоком данных" на странице 40.

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:

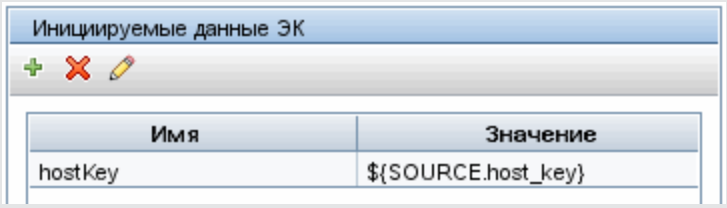
- "Framework.getTriggerCIData(String attributeName)" ниже
- "Framework.createClient(credentialsId, props)" ниже
- "Framework.getParameter (String parameterName)" на странице 49
- "Framework.reportError(String message) и Framework.reportWarning(String message)" на странице 49

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 обнаружению и интеграции в HP Universal CMDB*.

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:

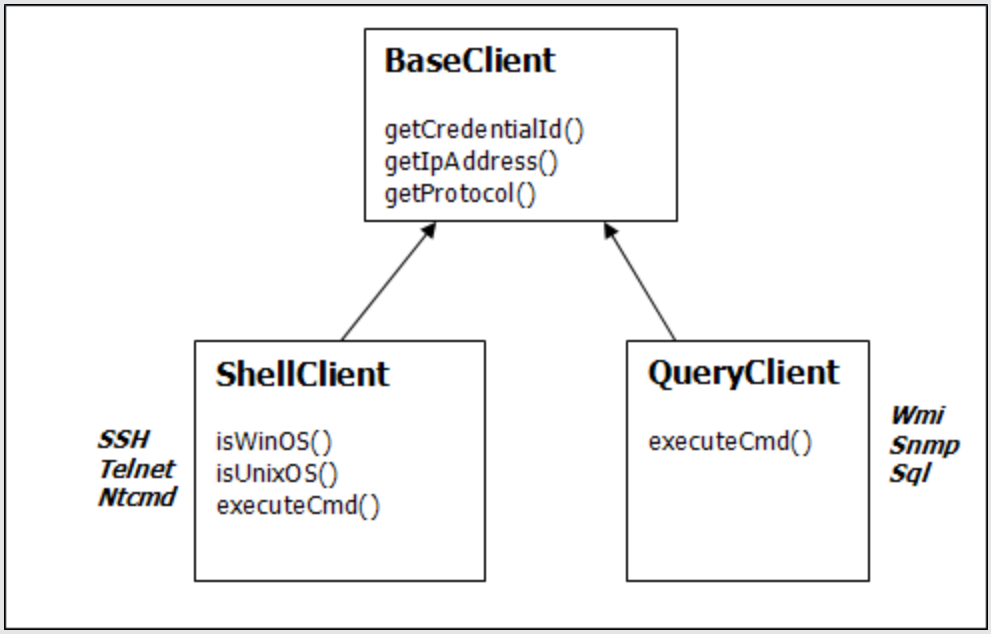
Чтобы создать клиент WMI и выполнить запрос WMI с помощью клиента:

```
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:

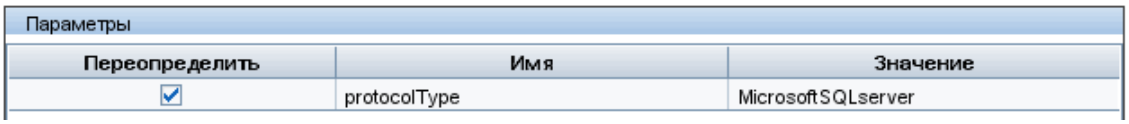


Дополнительные сведения о клиентах и API-интерфейсах, которые они поддерживают, см. в разделах BaseClient, ShellClient и QueryClient в документе *HP Discovery and Dependency Mapping Schema Reference*. Эти файлы находятся по следующему пути:

<UCMDB root directory>\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>)`:

```
try:
        client = Framework.getClientFactory(ClientsConsts.SNMP_
PROTOCOL_NAME)
        createClient()
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 79.

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 (через ClientFactory) — это идентификатор учетных данных. Сценарий подключения получает доступ к имеющимся наборам учетных данных и применяет их один за другим с помощью метода `clientFactory.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).

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 26.

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

        connected = 0
        # Go over all the protocol credentials
        for credentials_id in protocols:
            client = None
            try:
                # try to connect to the snmp agent
                client = clientFactory.createClient(credentials_id)
                // Query the agent
                ....
                # connection succeed
                connected = 1
            except:
```

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

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:

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

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

Поддержка локализации в адаптерах Jython

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

До выпуска Content Pack 3.00 в DFM использовалась статическая кодировка для обработки вывода всех целевых объектов в сети. Однако этот подход не подходит для многоязычной ИТ-сети: для обнаружения хостов с разными языками ОС администраторы зондов должны повторно запускать задания DFM вручную, используя разные параметры заданий для каждого выполнения. Эта процедура приводила к существенной нагрузке на сеть и препятствовала использованию ряда ключевых возможностей DFM, таких как немедленный вызов заданий для ЭК-триггера или автоматическое обновление данных в UCMDB с помощью диспетчера планирования.

Следующие языки поддерживаются по умолчанию: японский, русский и немецкий. Язык по умолчанию: английский.

Этот раздел охватывает следующие темы:

- "Добавление поддержки нового языка" на следующей странице
- "Изменение языка по умолчанию" на странице 53
- "Определение набора символов для кодировки" на странице 53

Добавление поддержки нового языка

В этом разделе описывается добавление поддержки нового языка.

Эта задача включает следующие шаги:

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 55.

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"" на странице 1 (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 56. См. дополнительные сведения об объекте Class Locale 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. Подробнее см. в разделе "Добавление поддержки нового языка" на предыдущей странице.

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**. Подробнее см. в разделе " Обзор API-интерфейса веб-службы HP Universal CMDB" на странице 234.

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. Подробнее см. в шаге "Получите ключевое слово из пакета ресурсов, подходящего для определенной команды:" ("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**.

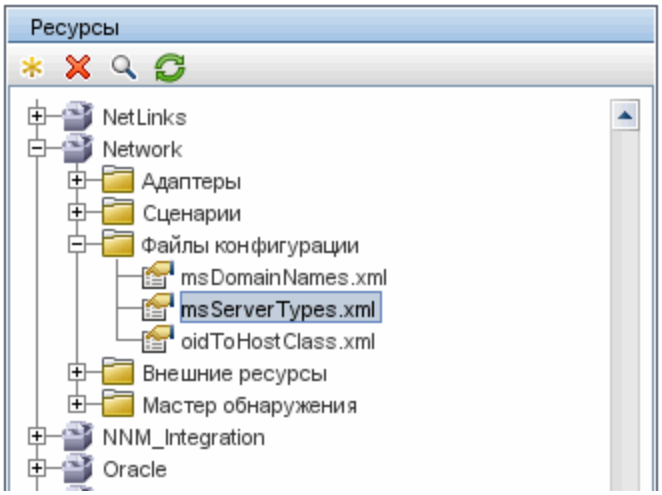
Обычный сценарий использования описывается далее.

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 Configuration File Pane (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:

- "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.

Работа с Discovery Analyzer

Discovery Analyzer — это приложение, предназначенное для отладки при разработке пакетов, сценариев и других материалов. Приложение выполняет задание для удаленного

объекта и возвращает журнал с информацией, сведениями о предупреждениях и ошибках и результатами обнаружения ЭК.

Обратите внимание, что результаты не всегда отображаются в интерфейсе. Это связано с тем, что результаты передаются двумя способами, и только один из них поддерживается. Кроме того, журнал обмена данными не поддерживается для Eclipse.

При запуске приложения из Eclipse в файле **DataFlowProbe.properties** file (**C:\hp\UCMDB\DataFlowProbe\conf\DataFlowProbe.properties**) для следующего параметра должно быть установлено значение **true**:

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

Подробнее см. в разделе "Запуск Discovery Analyzer" на странице 65.

Во всех остальных случаях (если приложение выполняется из файла **cmd** или если запущен зонд) для этого флага должно быть указано значение **false**:

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

Задачи и записи

Файл задачи содержит сведения о выполняемой задаче. Задача включает такие сведения, как имя задания и обязательные параметры ЭК-триггера, например удаленный адрес назначения.

Файл записей содержит сведения о задаче, а также результаты определенного выполнения, т.е. подробный журнала обмена данными (включая ответ) между зондом и Discovery Analyzer (в зависимости от того, какой модуль выполнил задачу) и удаленным целевым объектом.

Задача, указанная в файле задачи, может быть выполнена для удаленного целевого объекта, а задача в файле записи (содержащем дополнительные данные о выполнении) может быть выполнена и воспроизведена (т. е. может воспроизвести выполнение, задокументированное в файле записи).

Журналы

Журналы предоставляют информацию о последнем выполнении следующим образом:

- **Общий журнал**. Этот журнал включает все данные, ошибки и предупреждения, связанные с выполнением.
- **Журнал связи**. Этот журнал содержит подробный обмен данными между Discovery Analyzer и удаленным целевым объектом (включая его ответ). После выполнения журнал можно сохранить как файл записи.
- **Журнал результатов**. Отображает список обнаруженных ЭК. Время появления каждого ЭК содержит от структуры адаптеров и сценариев.

Журналы можно сохранять вместе и по отдельности. При сохранении всех журналов они записываются под одним именем.

При воспроизведении файла записи те же данные будут отображаться в журнале связи. Единственная разница — время выполнения.

Ограничение: Журналы связи и результатов недоступны при запуске Discovery Analyzer через Eclipse.

Этот раздел включает следующие шаги.

1. ## Необходимые условия

   - Зонд должен быть установлен. (Discovery Analyzer устанавливается в рамках установки зонда и использует ресурсы совместно с ним.)
   - Зонд не должен быть запущен при использовании Discovery Analyzer.

     Однако если зонд уже выполнялся для сервера UCMDB, все необходимые ресурсы уже загружены в файловую систему. Если зонд не выполнялся, можно передать ресурсы, необходимые Discovery Analyzer, с помощью меню «Параметры». Подробнее см. в разделе "Импорт параметров" на странице 64.
   - Установка сервера CMDB не требуется.

2. ## Запуск Discovery Analyzer

   Существует несколько способов доступа к Discovery Analyzer:

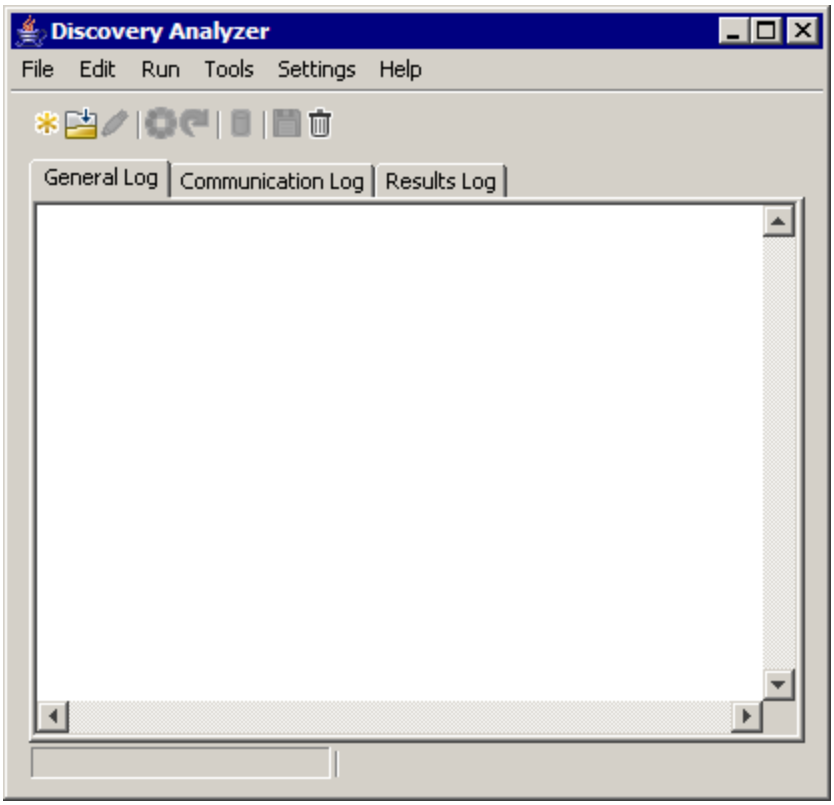
   - При использовании Eclipse.

     Установка зонда включает рабочую область Eclipse по умолчанию, которая находится в каталоге **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzerWorkspace**. Эта рабочая область включает сценарий Jython для запуска Discovery Analyzer (**startDiscoveryAnalyzerScript.py**), а также ссылки на все сценарии DFM. Запустив

приложение таким способом, вы можете найти точки останова в сценариях Jython для отладки.

- Напрямую, дважды щелкнув файл в следующей папке: **C:\hp\UCMDB\DataFlowProbe\tools\discoveryAnalyzer.cmd**. См. дополнительные сведения в следующем разделе.

Откроется окно Discovery Analyzer.



3. **Определение задачи**

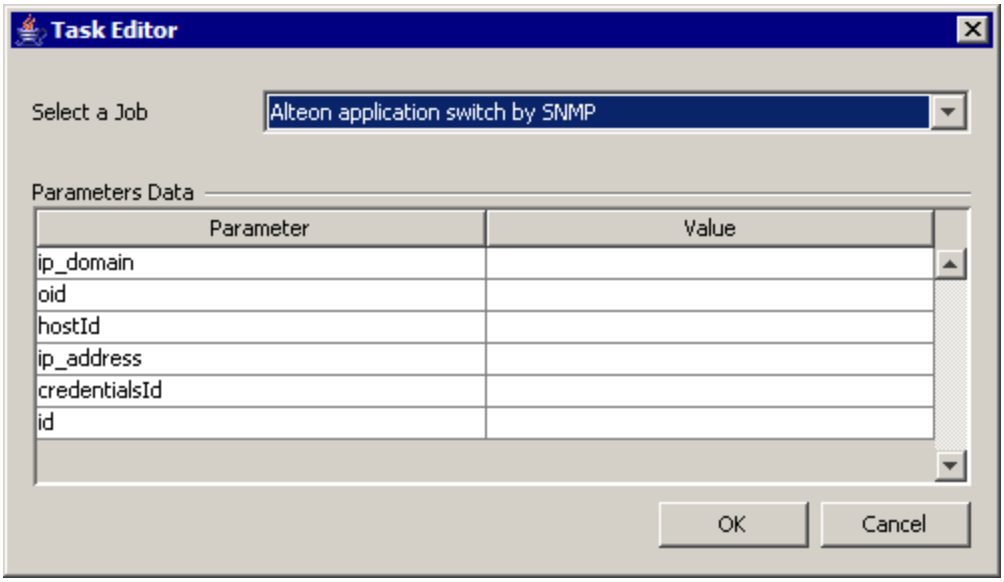
Для определения задачи можно использовать один из следующих методов:

- Определение новой задачи. Подробнее см. в разделе "Определение новой задачи" ниже.
- Импорт задачи из файла записи. Подробнее см. в разделе "Получение записи" на следующей странице.
- Импорт сохраненной задачи из файла задачи. Подробнее см. в разделе "Открытие файла задачи" на странице 63.
- Путем извлечения задания из внутренней базы данных зонда. Подробнее см. в разделе "Импорт задачи из базы данных" на странице 63.

4. **Определение новой задачи**

a. Откройте редактор задач: нажмите кнопку **Создать задачу** .

В редакторе задач отображается список заданий, существующих в файловой системе. Этот список обновляется каждый раз, когда зонд получает задачи с сервера, или пакеты развертываются вручную из меню «Параметры».



b. Выберите задание.

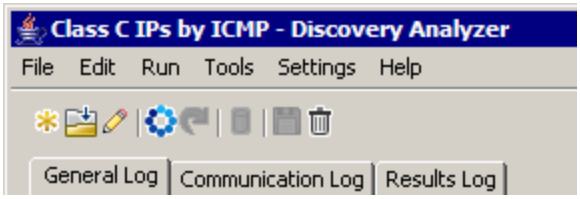
c. Введите значения всех параметров.

Здесь отображаются параметры адаптера DFM. Их можно просмотреть на панели Discovery Pattern Parameters вкладки Pattern Signature. Подробнее см. в разделе "Adapter Definition Tab" (*Руководство по управлению потоками данных в HP Universal CMDB*).

Все поля обязательны (если сценарий задания не требует, чтобы поле оставалось пустым).

Для параметров, которые требуют ввода идентификатора или идентификатора учетных данных можно использовать случайные идентификаторы: щелкните поле правой кнопкой и выберите **Generate random CMDB ID** или **Credential Chooser**.

Задача активна, и имя открытой задачи отображается на панели заголовка:



d. Продолжайте процедуру определения задачи. Подробнее см. в разделе "Сохранение задачи" на следующей странице.

5. ## Получение записи

Для определения задачи можно открыть файл записи с данными об определенном выполнении. Если задача определена таким способом, вы можете воспроизвести ее выполнение, выбрав функцию воспроизведения. (При воспроизведении задачи ответы

получаются из данных, сохраненных в файле записи, а не из удаленного целевого объекта.)

Выберите **File > Open Record**. Перейдите к папке, в которую сохранили запись. Запись становится активной, и имя задачи отображается на панели заголовка.

См. дополнительные сведения о получении файла записи в разделе "Запись кода DFM" на странице 74.

6. ### Открытие файла задачи

   Задачу можно определить из файла задачи: Выберите **File > Open Task**.

7. ### Импорт задачи из базы данных

   Задачу можно извлечь из базы данных зонда, при условии, что зонд уже запускался и содержит активные задачи во внутренней базе данных. Вы можете использовать значения параметров для указания задачи.

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

   b. В открывшемся диалоговом окне выберите задачу для выполнения и нажмите кнопку **OK**.

   c. Продолжайте процедуру определения задачи. Подробнее см. в разделе "Сохранение задачи" ниже.

8. ### Изменение задачи

   После определения задачи ее имя (или имя файла) будет отображаться на панели заголовка. Теперь файл можно изменить.

   a. Выберите **Edit > Edit Task**.

   b. Внесите изменения в задачу и нажмите кнопку **OK**.

9. ### Сохранение задачи

   Также можно сохранить параметры задачи: Выберите **File > Save Task**.

   Следующие параметры доступны только после выполнения задачи.

   - Сохранение записи задачи. Вы можете сохранить параметры задачи и результаты ее выполнения: Выберите **File > Save Record**.
   - Сохранение журнала задачи. Выберите **File > Save General Log**.
   - Сохранение результатов. Выберите **File > Save Results**.

10. ### Выполнение задачи

    Следующий этап процедуры — выполнение созданной задачи.

    a. Импорт файла конфигурации учетных данных и диапазонов. Подробнее см. в разделе "Импорт параметров" на следующей странице.

    b. Чтобы выполнить задачу только для удаленного целевого объекта, нажмите кнопку **Run Task**.

       Discovery Analyzer выполнит задание и отобразит сведения в трех файлах журнала: **General**, **Communication** и **Results**.

c. Журналы можно сохранять вместе и по отдельности. Выберите **File > Save General Log**, **Save Record**, **Save Results** или **Save All Logs**. Дополнительные сведения о файлах журнала см. в разделе "Журналы" на странице 59.

d. Если задача извлекается из файла запись, выполнение, задокументированное в этом файле, можно воспроизвести с помощью кнопки **Playback**. Откроется тот же журнал Communication, но время выполнения будет обновлено.

11. Отправка результата задачи на сервер

Если выполнение задачи завершается с результатами (т.е. во вкладке Results Log отображается список обнаруженных ЭК), результаты можно отправить на сервер UCMDB. Это может быть полезно, если при предыдущем тестировании сценария сервер был недоступен.

Примечание. Результаты можно отправить только на сервер UCMDB, который получает задачи от зонда, установленного на одном компьютере с Discovery Analyzer.

12. Импорт параметров

Для выполнения задач или воспроизведения файла записи необходимо импортировать файл **domainScopeDocument.bin**. Во время импорта необходимо ввести пароль.

a. Запустите веб-браузер и введите следующий URL-адрес: **http://localhost:8080/jmx-console**. Возможно, потребуется ввести имя пользователя и пароль для входа в систему.

b. Нажмите **UCMDB:service=DiscoveryManager**, чтобы открыть страницу JMX MBEAN View.

c. Найдите операцию **exportCredentialsAndRangesInformation**. Выполните следующие действия:

- Введите идентификатор заказчика (значение по умолчанию: **1**).
- Введите имя экспортированного файла.
- Введите пароль.
- Установите значение **False** для параметра **isEncrypted**.

d. Нажмите кнопку **Invoke**, чтобы экспортировать файл **domainScopeDocument.bin**.

После успешного выполнения процесса экспорта файл будет сохранен по следующему пути: **C:\hp\UCMDB\UCMDBServer\conf\discovery\<customer_dir>**.

e. Скопируйте файл **domainScopeDocument.bin** в файловую систему зонда потоков данных и импортируйте его, выбрав **Settings > Import domainScopeDocument**.

Примечание. Во время импорта файла **domainScopeDocument** система предложит ввести пароль. Этот запрос также будет отображаться при каждом перезапуске Discovery Analyzer и до выполнения первой задачи или записи.

13. Точки останова

При выполнении Discovery Analyzer из сценария Python можно добавить в него точки останова.

14. **Настройка Eclipse**

    См. дополнительные сведения о выполнении сценариев Jython в режиме отладки в разделе "Запуск Discovery Analyzer" ниже.

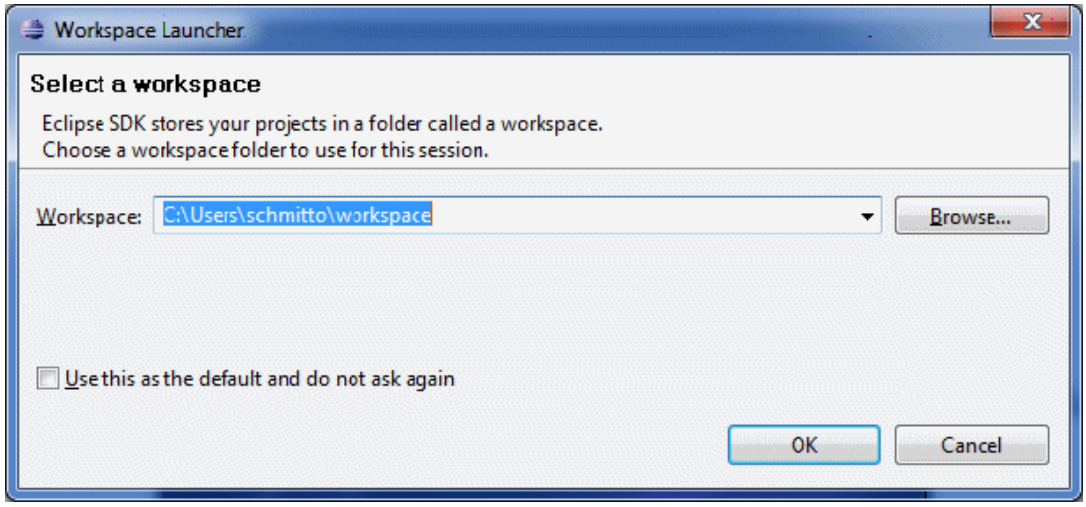
Запуск Discovery Analyzer

В этой задаче описывается настройка Eclipse для выполнения сценариев Jython в режиме отладки, который обеспечивает лучшую визуализацию потоков заданий, ЭК-триггеров и результатов.

Этот раздел включает следующие шаги.

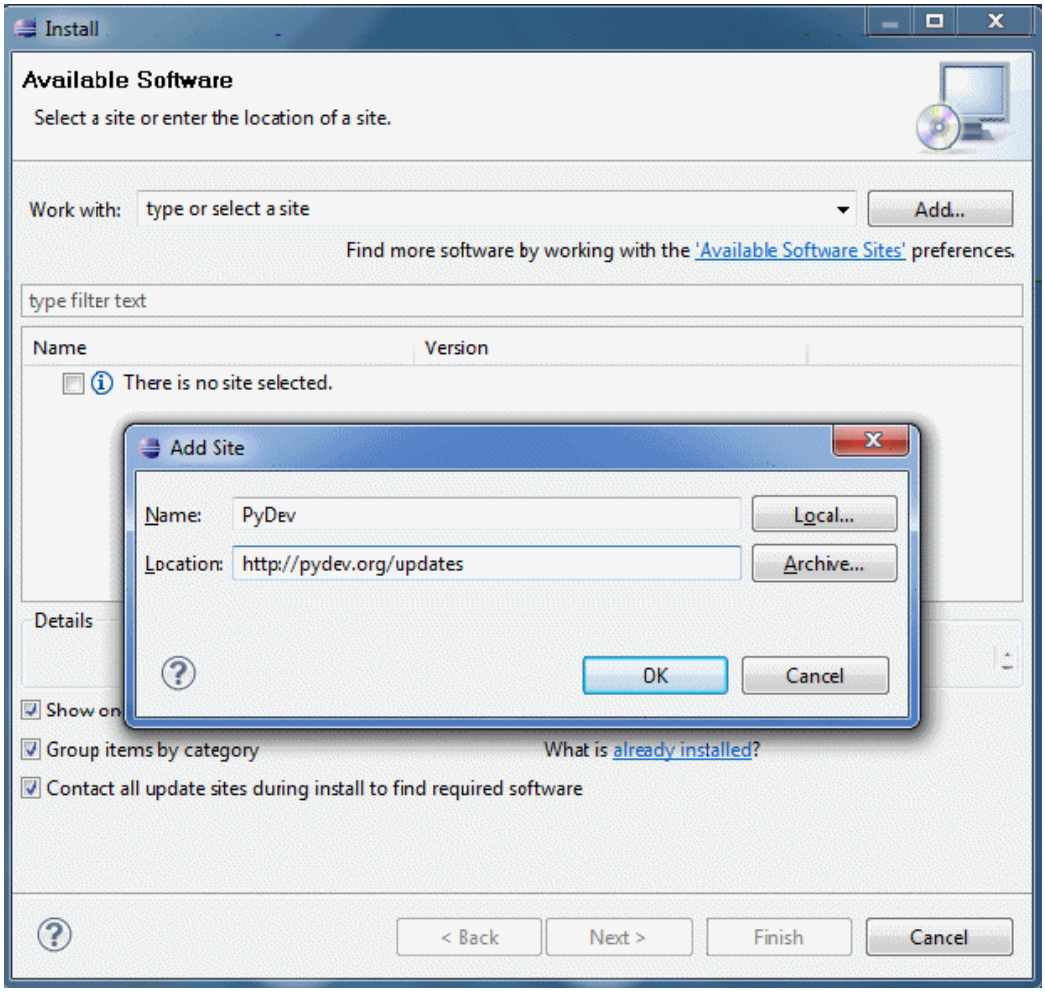
1. **Необходимые условия**
   - Установите последнюю версию Eclipse на компьютере. Приложение доступно по адресу www.eclipse.org.
   - Убедитесь, что компонент зонд потоков данных установлен на компьютере.
   - Убедитесь, что для параметра **appilog.agent.local.discoveryAnalyzerFromEclipse** в файле **DataFlowProbe.properties** установлено значение **true**.
2. Распаковка и запуск приложения Eclipse
3. **Настройка рабочей области по умолчанию**

   Настройте рабочую область по умолчанию, в которой Eclipse сохраняет все проекты и связанные данные.



4. ## Настройка расширений PyDev

   a. Откройте в меню раздел **Help > Install New Software**, нажмите **Add**, введите имя подключаемого модуля PyDev и введите URL-адрес сайта, с которого можно загрузить **pydev**, в поле Location: **http://pydev.org/updates**. Нажмите **OK**.

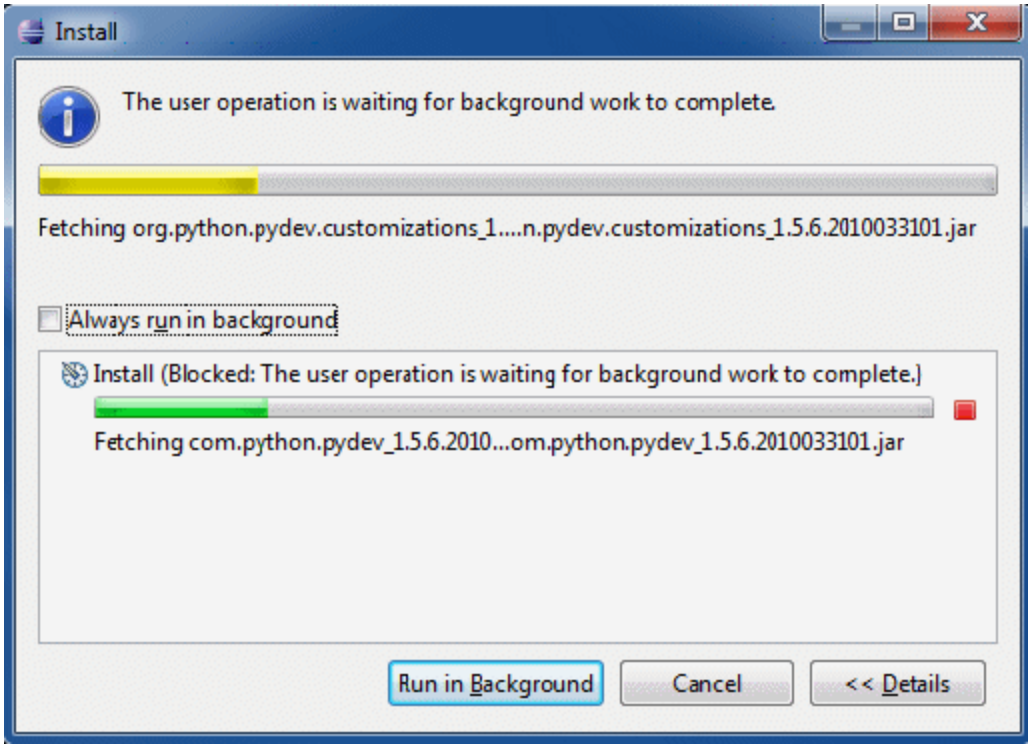


> **Примечание.** PyDev и расширения PyDev объединены в один подключаемый модуль, поскольку исходный код расширений PyDev открыт. См. дополнительные сведения по адресу http://pydev.org.

b. В открывшемся окне выберите **Pydev**. Второй подключаемый модуль используется для интерфейса на основе задач. Нажмите кнопку **Next**, проверьте сведения об установке и нажмите кнопку **Next** еще раз

c. Примите лицензионное соглашение и нажмите кнопку **Next**.

d. Компонент Pydev установлен. Если появится запрос установки неподписанного содержимого, согласитесь, нажав кнопку **OK**.

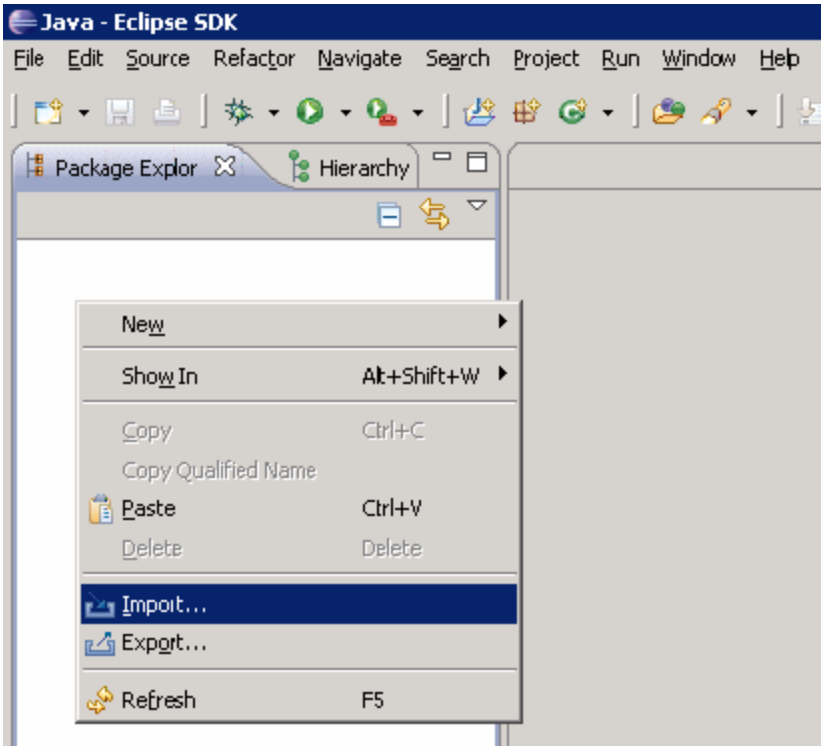


e. Перезапустите Eclipse.

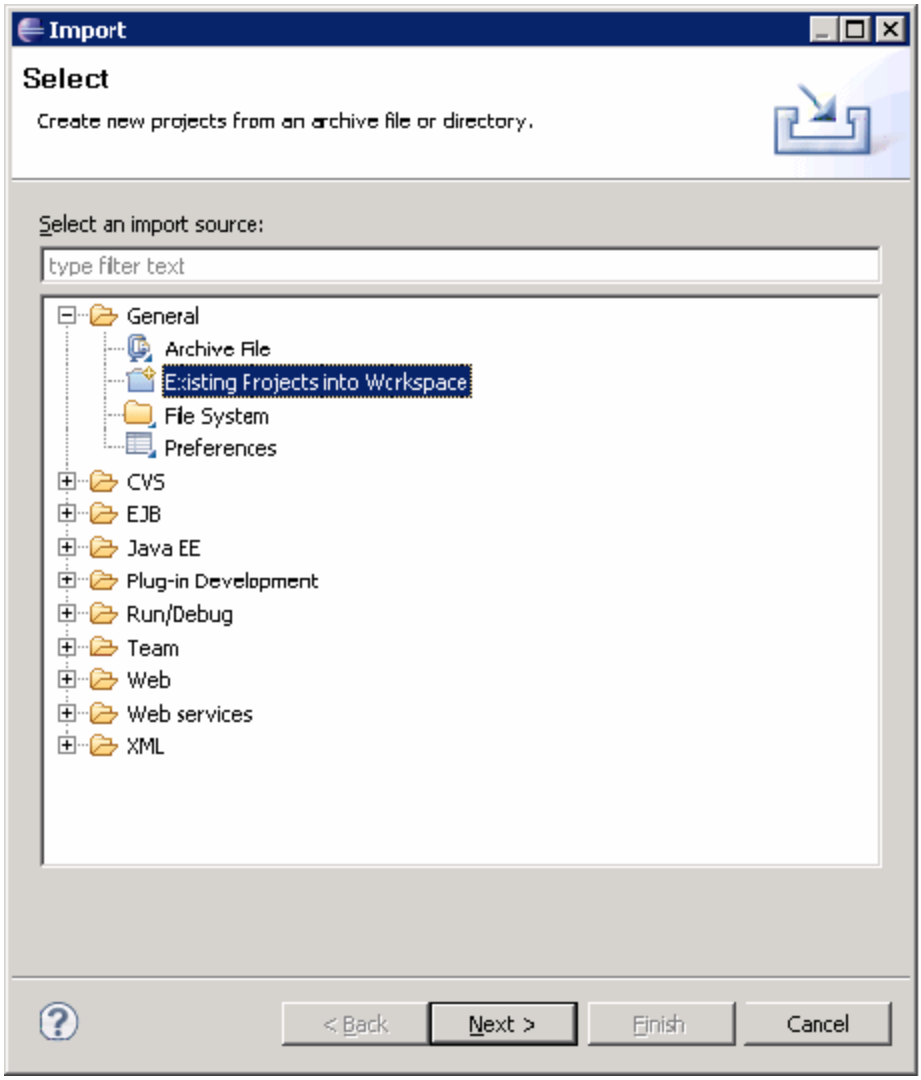
Компонент PyDev установлен в Eclipse IDE. В Eclipse доступны новые перспективы и IDE может интерпретировать сценарии Python (выделение текста, дополнительные параметры конфигурации и др.).

5. Настройка рабочей области Discovery Analyzer

a. Импорт необходимых файлов: щелкните правой кнопкой белую область в Package Explorer и выберите **Import**, чтобы импортировать готовый элемент **discoveryAnalyzerWorkspace**, включенный в установку зонда.



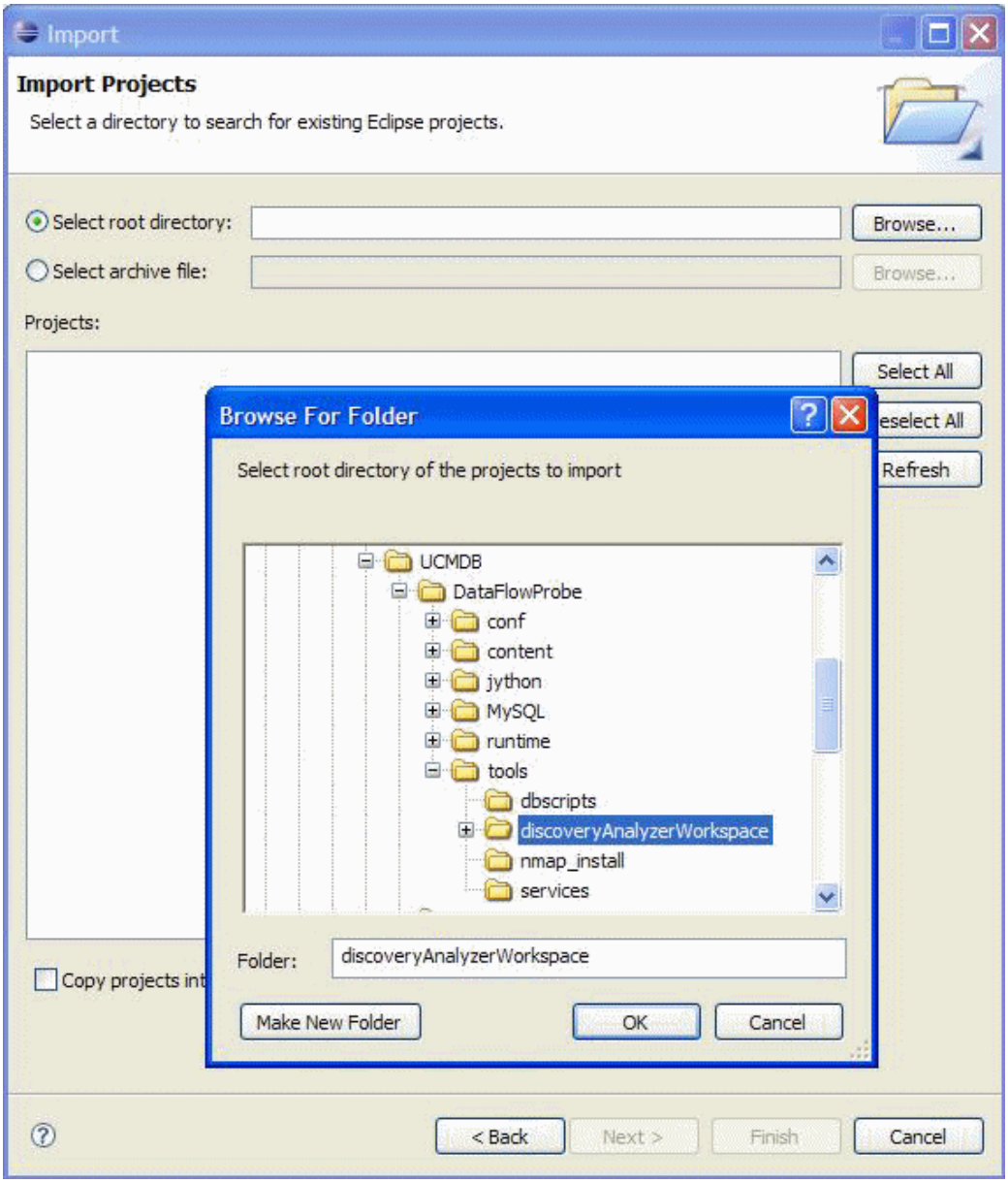
b. В разделе **General** выберите **Existing projects into Workspace**, чтобы импортировать проект в рабочую область Eclipse.



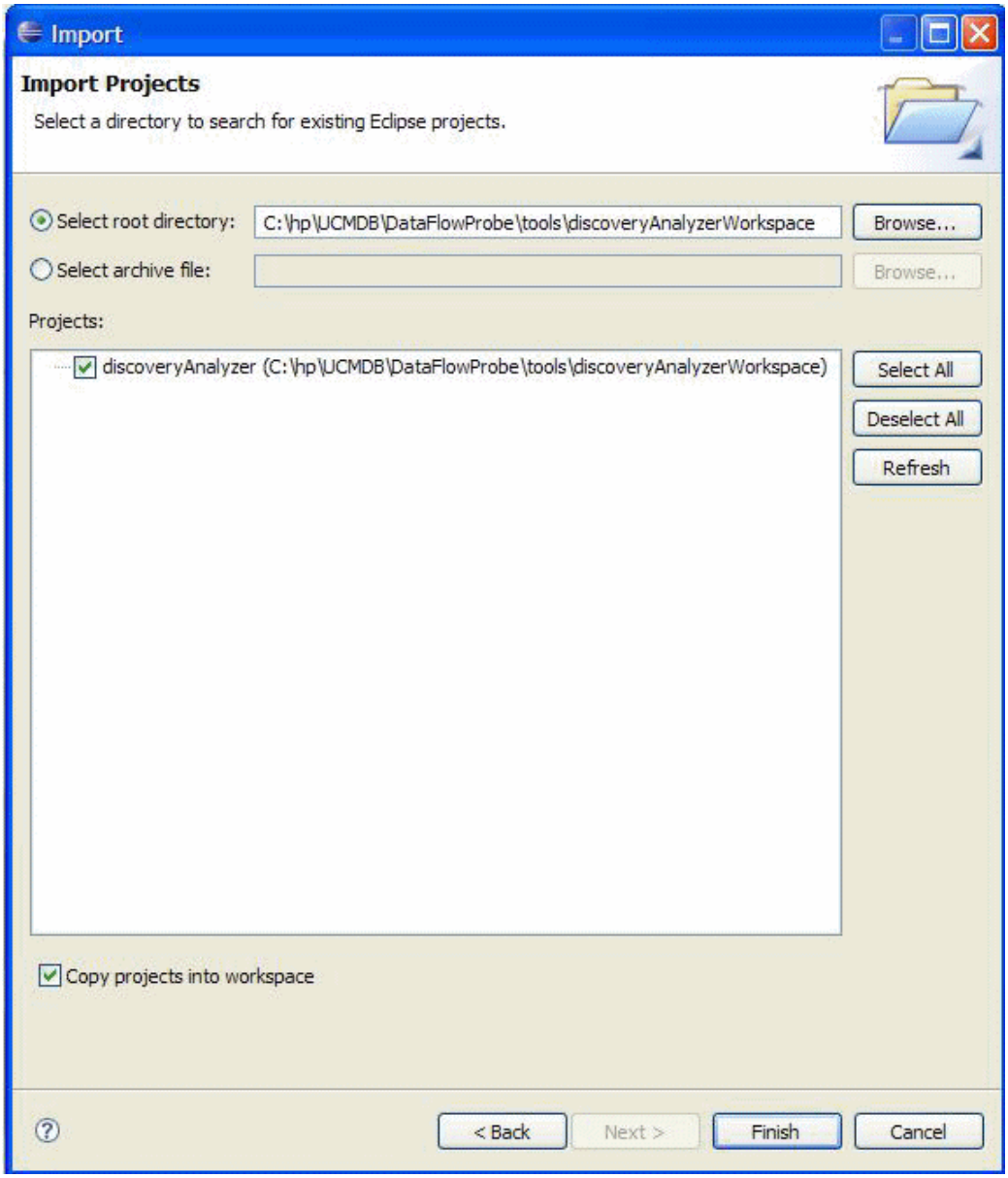
c. В поле **Select root directory** выберите рабочую область Analyzer по умолчанию, обычно она находится в каталоге

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

d. Выберите **Copy projects into workspace**, чтобы создать реальную копию существующей рабочей области. Это важный шаг: в случае ошибки можно будет заново импортировать исходный элемент **discoveryAnalyserWorkspace**.



e. Нажмите кнопку **Finish**, чтобы начать импорт.



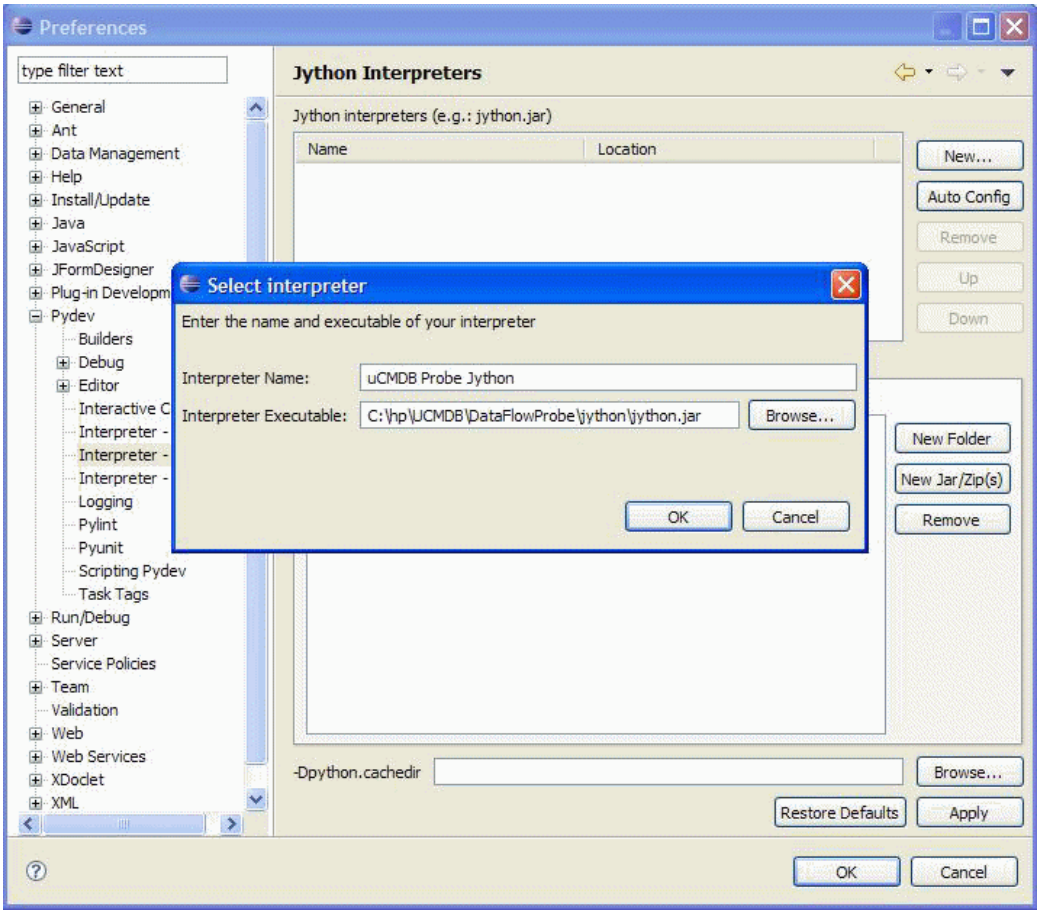
6. ## Настройка каталога classpath и интерпретатора

   a. Щелкните **discoveryAnalyzerWorkspace** правой кнопкой мыши и выберите **Properties**, чтобы открыть параметры проекта.

   b. Последовательно выберите **Pydev > Interpreter/Grammar** и щелкните **Please configure an interpreter in the related preferences before proceeding**.

   Данный шаг предусматривает настройку интерпретатора Jython, который используется зондом. Это позволяет избежать ситуации, когда сценарии интерпретируются разными версиями Jython.

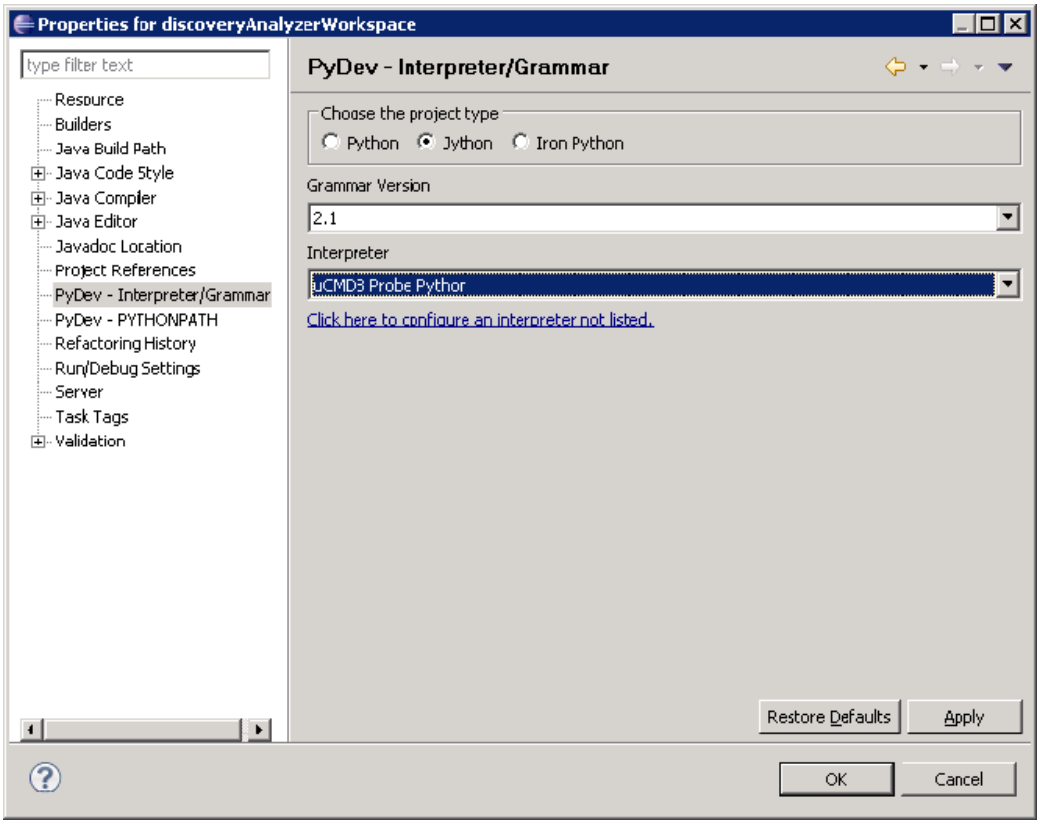
   c. Нажмите кнопку **New**, введите имя интерпретатора и выберите файл из следующей папки: **C:\hp\UCMDB\DataFlowProbe\jython\jython.jar**.



d. Нажмите **OK**. При появлении окна с предложением выбрать папки, которые должны быть импортированы в системный путь Python, не вносите никаких изменений (правильные значения: **C:\hp\UCMDB\DataFlowProbe\jython** и **C:\hp\UCMDB\DataFlowProbe\jython\lib**) и нажмите кнопку **OK**.

e. Нажмите кнопку **Apply**, затем **OK**.

f. Нажмите **Interpreter** и выберите созданный интерпретатор.



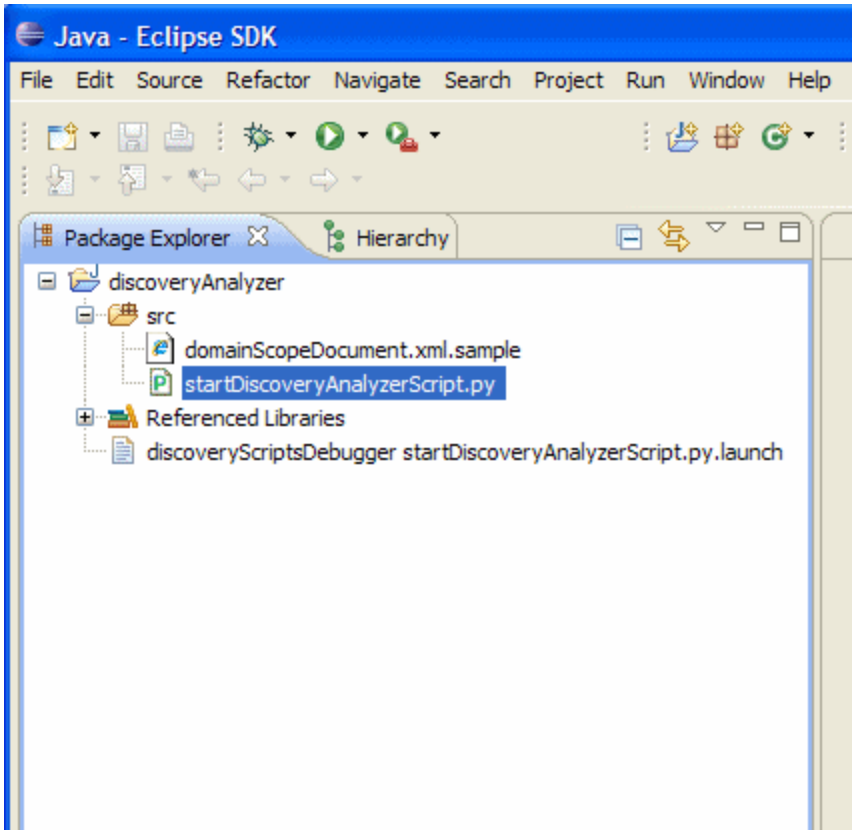
g. Нажмите кнопку **Apply**, затем **OK**.

Теперь интерпретатор Jython соответствует интерпретатору зонда.

7. Запуск Discovery Analyzer

a. Добавьте точку останова в сценарий Jython для отладки.

b. Чтобы запустить Discovery Analyzer, выберите **startDiscoveryAnalyzerScript.py** в проекте **discoveryAnalyzerWorkspace\src**. Щелкните файл правой кнопкой мыши и выберите **Debug as > Jython run**.

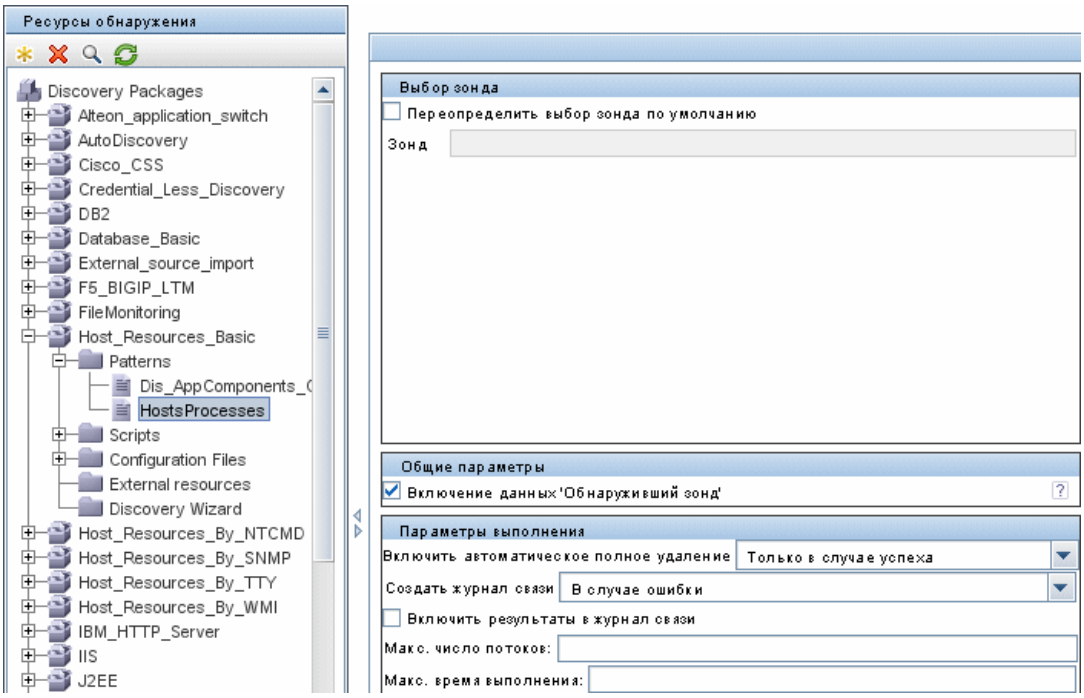


Запись кода DFM

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.

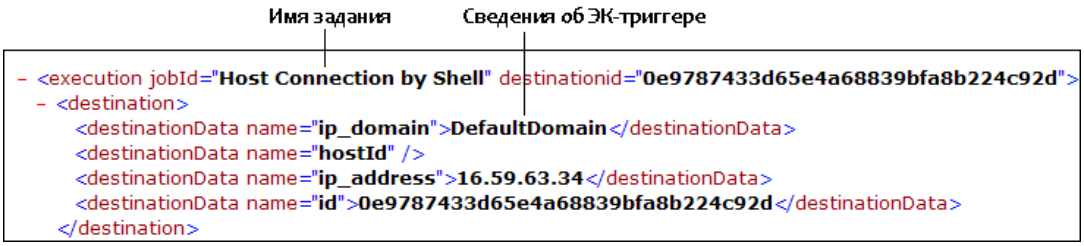
Для записи кода DFM:

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

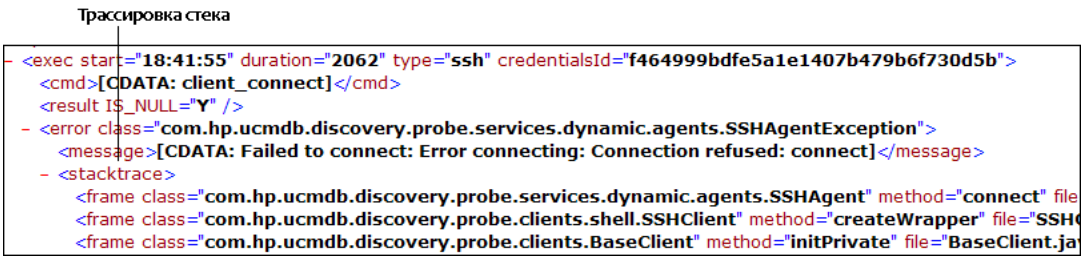


3. Измените значения поля **Создать журнал связи** на **Всегда**. См. дополнительные сведения о настройке параметров журнала в разделе "Панель "Параметры выполнения"" на странице 1 документа Руководство по управлению потоками данных в HP Universal CMDB.

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



   В следующем примере представлены сообщения и параметры трассировки стека:



Средства и библиотеки Jython

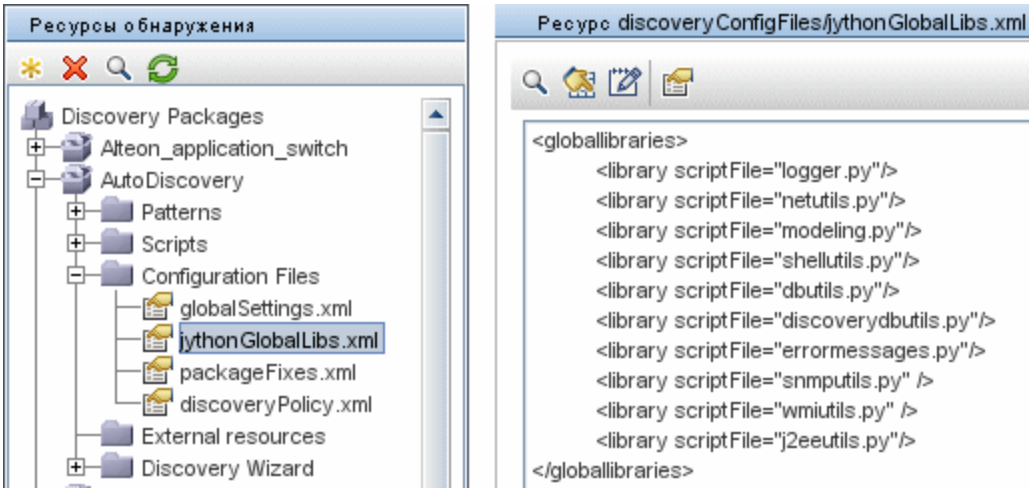
Несколько служебных сценариев широко используются в адаптерах. Эти сценарии являются частью пакета `AutoDiscovery` и находятся по следующему пути: **C:\hp\UCMDB\DataFlowProbe\runtime\probeManager\discoveryScripts** вместе с другими сценариями, загруженными в зонд.

> **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

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`.) Подробнее см. в разделе "Уровни серьезности ошибок" на странице 82.

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

Глава 3

Сообщения об ошибках

Данная глава включает:

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. DFM отображает эти ошибки на панели управления обнаружением как в основном, так и в расширенном режимах, когда обычный поток обнаружения завершается неудачей. 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.

См. дополнительные сведения о поиске ошибок в разделе "Discovery Overview/Status Pane" документа *Руководство по управлению потоками данных в HP Universal CMDB*.

См. дополнительные сведения о настройке журналов связи в разделе "Панель "Параметры выполнения"" на странице 1 документа *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

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 включать 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

Разработка общих адаптеров БД

Данная глава включает:

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.

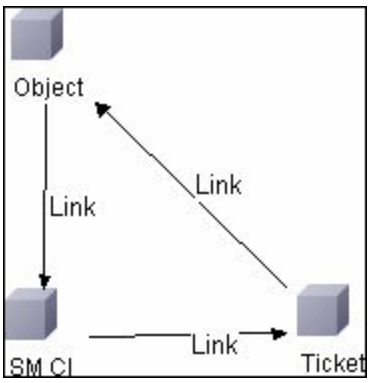
TQL-запросы для общего адаптера БД

Для заданий заполнения необходимо проверить каждую схему ЭК в диалоговом окне Настройки структуры в Студии моделирования. Подробнее см. в разделе "Диалоговое окно "Свойства узла запросов/связи"" на странице 1 (*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:

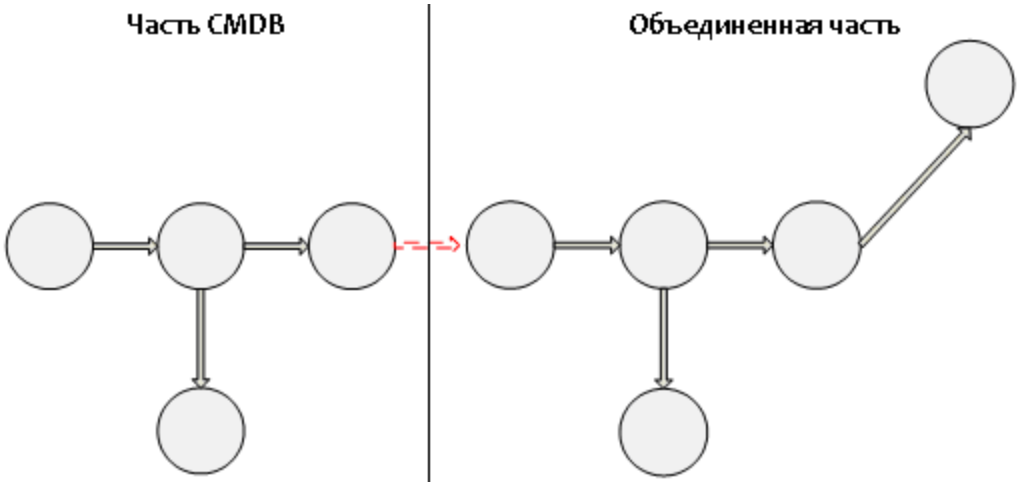
- подграфы не поддерживаются.
- составные связи не поддерживаются.
- циклы и части циклов не поддерживаются

  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. Подробнее см. в разделе "Необходимые условия" на странице 89.
- 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. Подробнее см. в разделе "Функциональные ограничения" на странице 155.

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. Подробнее 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.

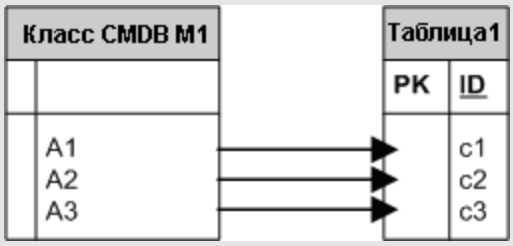
Данный раздел также включает следующие подразделы.

Примеры сопоставления между объектно-ориентированным приложением и реляционной базой данных

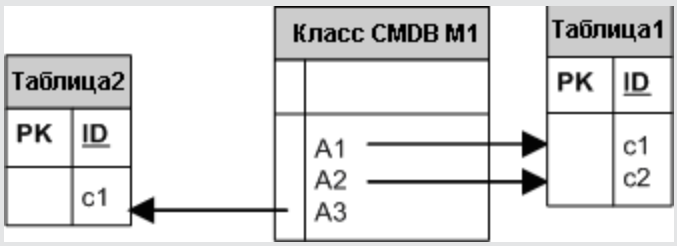
В следующих примерах приводится описание сопоставления между объектно-ориентированным приложением и реляционной базой данных:

Пример сопоставления одного класса CMDB с одной таблицей в БД:

Класс `M1` с атрибутами `A1`, `A2` и `A3` сопоставляется со столбцами таблицы `1c1`, `c2` и `c3`. Это значит, что для любого экземпляра `M1` существует соответствующая строка в таблице `1`.

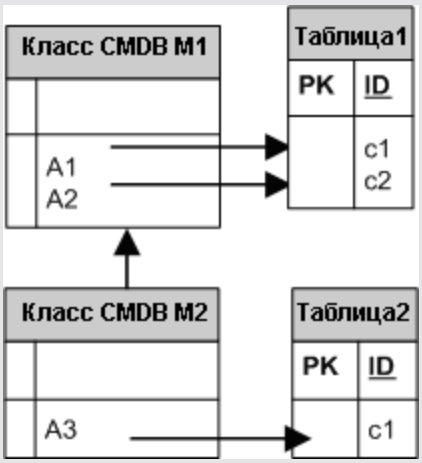


Пример сопоставления одного класса CMDB с двумя таблицами в БД:



Пример наследования:

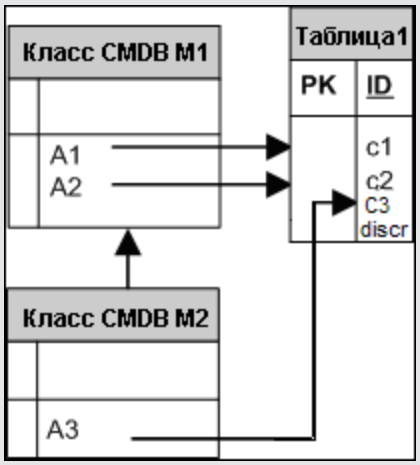
В данном случае используется база CMDB, в которой каждый класс имеет собственную таблицу БД.



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.

При использовании дискриминатора пропуск класса в иерархии невозможен. Таким образом, C3 наследует у C2, а C2 наследует у C1, и пользователь не может просто указать классы C1 и C3, должны быть заданы все три класса.



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, представлен XML-файл для упрощения определений.

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 91
- "Создание связи" на странице 91

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 86.

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 86.

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	Система	Intel ® 631xESB/6321ESB/3100 Chipset LPC
3581	3	Дисплей	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

CARD_ID	CARD_VENDOR
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).

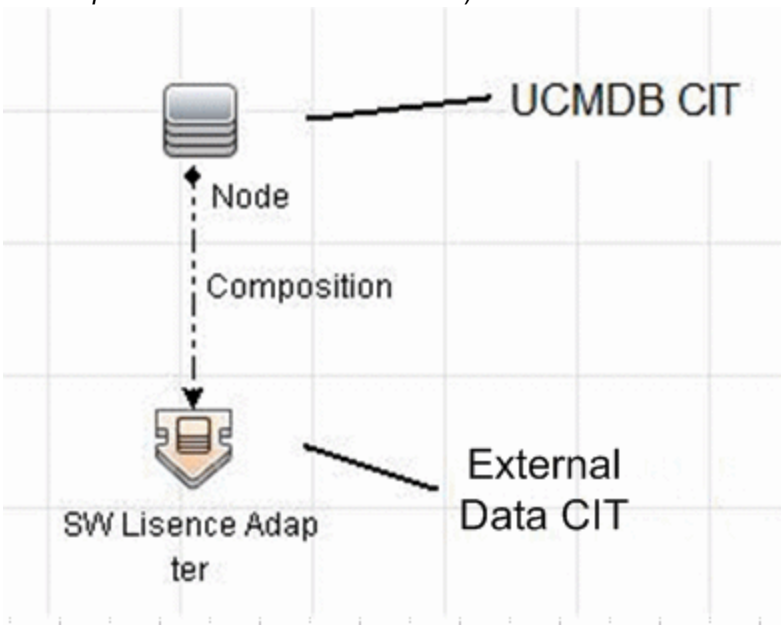
   a. В UCMDB откройте диспетчер типов ЭК и создайте новый тип ЭК. Подробнее см. в разделе "Создание типа ЭК" на странице 1 (*Руководство по моделированию в HP Universal CMDB*).

   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.

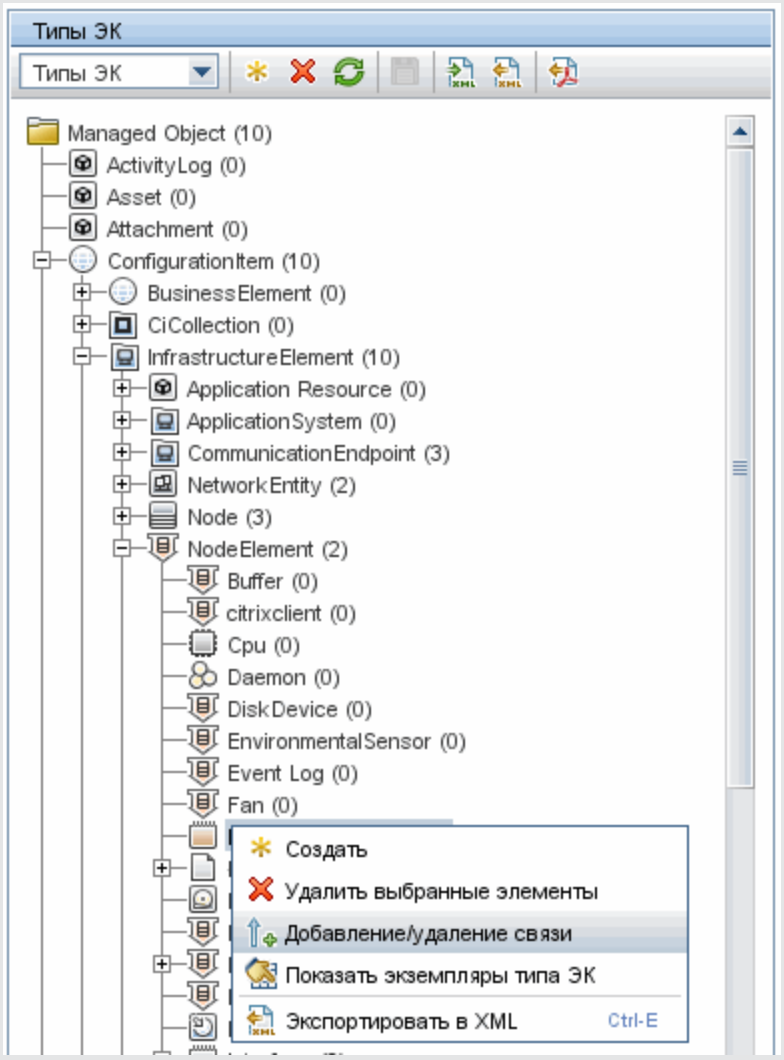
   Добавьте соответствующие связи к новому типу ЭК. Подробнее см. в разделе "Диалоговое окно "Добавить/удалить связь"" на странице 1 (*Руководство по моделированию в HP Universal CMDB*).



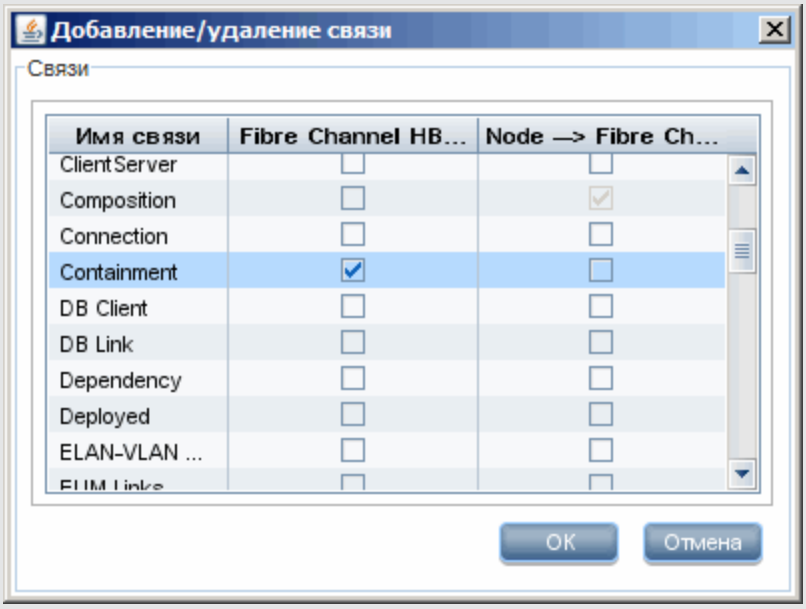
Примечание. На этом этапе еще невозможно просматривать объединенные данные или использовать внешние данные для заполнения, поскольку метод объединения данных еще не определен.

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.

Подробнее о заполнении CMDB данными см.в разделе "Страница Студии интеграции " на странице 1 документа *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 93.

   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

В следующей процедуре описывается метод сопоставления модели классов в CMDB с реляционной СУБД.

Эти файлы конфигурации находятся в пакете **db-adapter.zip** в директориии **C:\hp\UCMDB\UCMDBServer\content\adapters** которая была извлечена во время шага "Извлеките пакет в локальный временный каталог." на странице 93 раздела "Подготовка пакета адаптера" на странице 93.

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.

Эта задача включает следующие шаги:

- "Настройка файла adapter.conf." ниже
- "Настройка файла simplifiedConfiguration.xml" на следующей странице

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**.

Настройка файла simplifiedConfiguration.xml

Во время этого шага вы настроите файл **simplifiedConfiguration.xml**, сопоставив тип ЭК в CMDB с полями в таблице реляционной СУБД.

1. Откройте файл **simplifiedConfiguration.xml** в текстовом редакторе.

   Этот файл включает шаблон, который можно использовать для сопоставления каждого объекта.

   Примечание. Не изменяйте файл **simplifiedConfiguration.xml** в любой версии блокнота от корпорации Microsoft. Используйте Notepad++, UltraEdit или любой другой сторонний редактор.

2. Внесите изменения в следующие атрибуты.

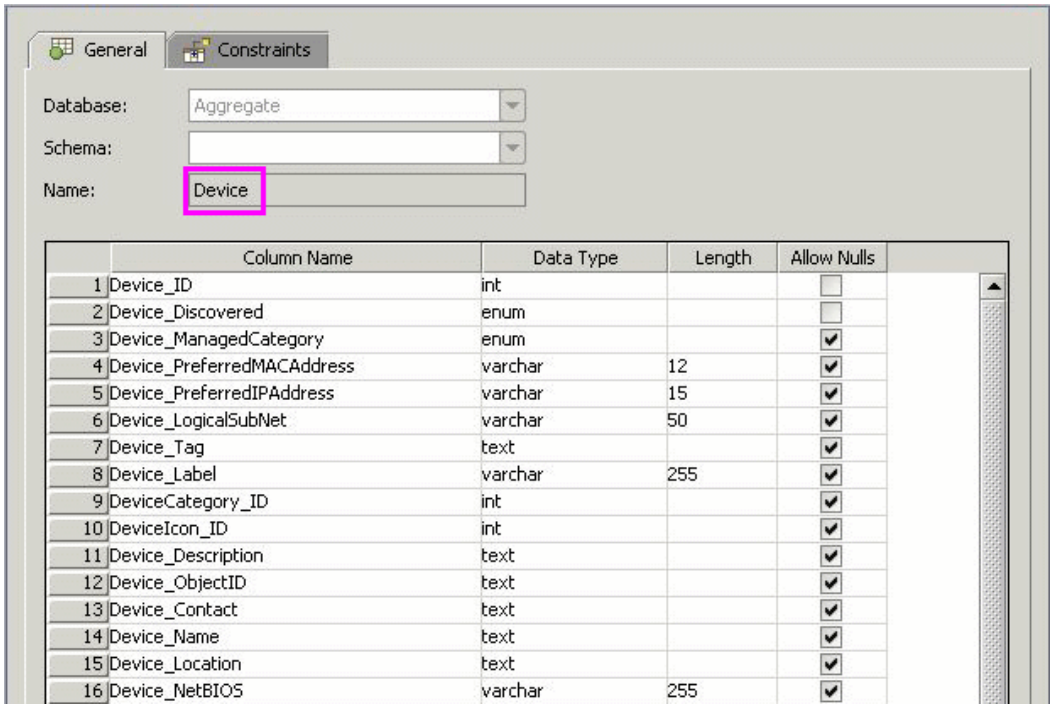
   - Имя типа ЭК в UCMDB (cmdb-class-name) и соответствующее имя таблицы в реляционной СУБД (default-table-name):

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

     Атрибут `cmdb-class-name` получен из типа ЭК узла:

     Атрибут `default-table-name` получен из таблицы устройств:

- Уникальный идентификатор в реляционной СУБД:

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

> **Примечание.** Первичный ключ аналогичен идентификатору объекта в файле orm.xml.

- Правило выверки (reconciliation-by-two-nodes):

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

- Атрибут выверки в UCMDB (`cmdb-attribute-name`) и реляционной СУБД (`column-name`):

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

- Имя типа ЭК (`cmdb-class-name`) и соответствующей таблицы в реляционной СУБД (`default-table-name`). Кроме того, связь CMDB (`connected-cmdb-class-name`) и связь типов ЭК (`link-class-name`):

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

- Основной ключ и внешний ключ:

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

- Уникальный идентификатор в реляционной СУБД:

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

- Сопоставление между атрибутом CMDB (`cmdb-attribute-name`) и именем столбца в реляционной СУБД (`column-name`):

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

3. Сохраните файл.

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:

- "Создание файла simplifiedConfiguration.xml" ниже
- " Создайте TQL-запрос" на следующей странице
- "Создание точки интеграции" на странице 101

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:

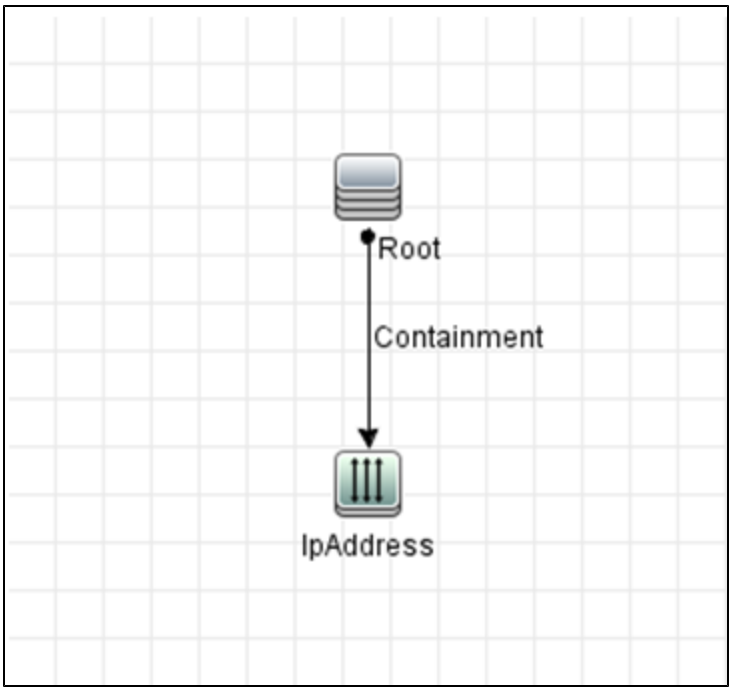
```
</or>

</reconciliation-by-two-nodes>

</cmdb-class>
```

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 атрибутов** 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 запроса».
9. Откройте вкладку **Структура элемента**. В разделе **Условие атрибутов** 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 директориии **C:\hp\UCMDB\UCMDBServer\content\adapters** которая была извлечена во время шага "Извлеките пакет в локальный временный каталог." на странице 93 раздела "Подготовка пакета адаптера" на странице 93.

Эта задача включает следующие шаги:

- "Настройка файла orm.xml" ниже
- "Настройка файла reconciliation_types.txt" на странице 105
- "Настройка файла reconciliation_rules.txt " на странице 105

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.

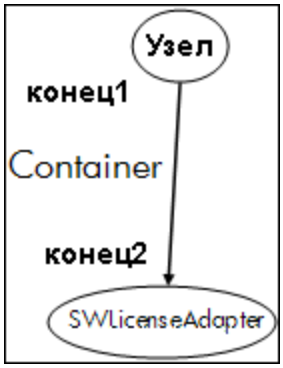
   Примечание. Не изменяйте файл **orm.xml** в любой версии блокнота от корпорации Microsoft. Используйте Notepad++, UltraEdit или любой другой сторонний редактор.

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**:

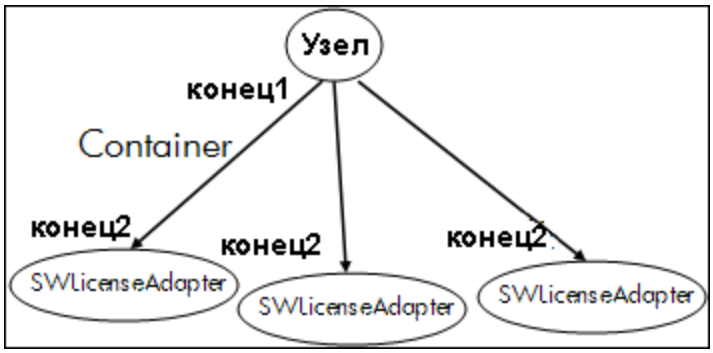
- Один к одному:



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>
```

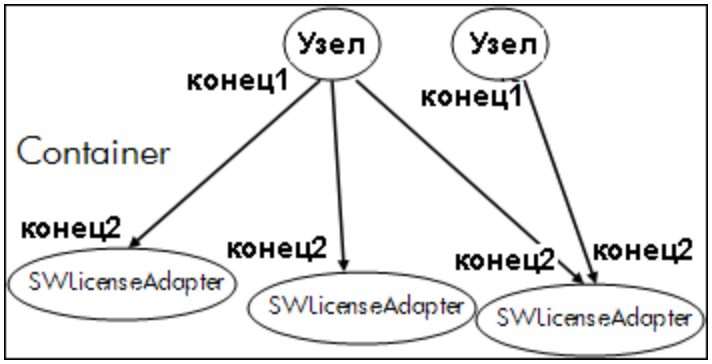
- Многие к одному:



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 125.

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 91:

```
<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_types.txt

Откройте файл **reconciliation_types.txt** в текстовом редакторе.

Подробнее см. в разделе "Файл reconciliation_types.txt" на странице 134.

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 версией8.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.

Подробнее см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 134.

Реализация подключаемого модуля

В этой задаче описывается реализация и развертывание общего адаптера БД с подключаемыми модулями.

Примечание. Перед созданием подключаемого модуля для адаптера убедитесь, что все необходимые шаги из раздела "Подготовка пакета адаптера" на странице 93 выполнены.

1. 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. См. дополнительные сведения о разработке подключаемых модулей в файлах **db-interfaces-javadoc.jar** и **federation-api-javadoc.jar**, а также в интерактивной документации по следующему пути:

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

2. Напишите 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
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>
```

3. 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**.

4. Упакуйте класс в JAR-файл и поместите его в каталог adapterCode\<Your Adapter Name> пакета адаптера перед его развертыванием.

   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.

5. 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"" на странице 1 (*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. Подробнее см. в разделе "Конфигурация адаптеров" (*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 разделеNew Integration Point/Edit Integration Point Dialog Box (*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"" на странице 1 (*Руководство по управлению потоками данных в HP Universal CMDB*).

Создание представления

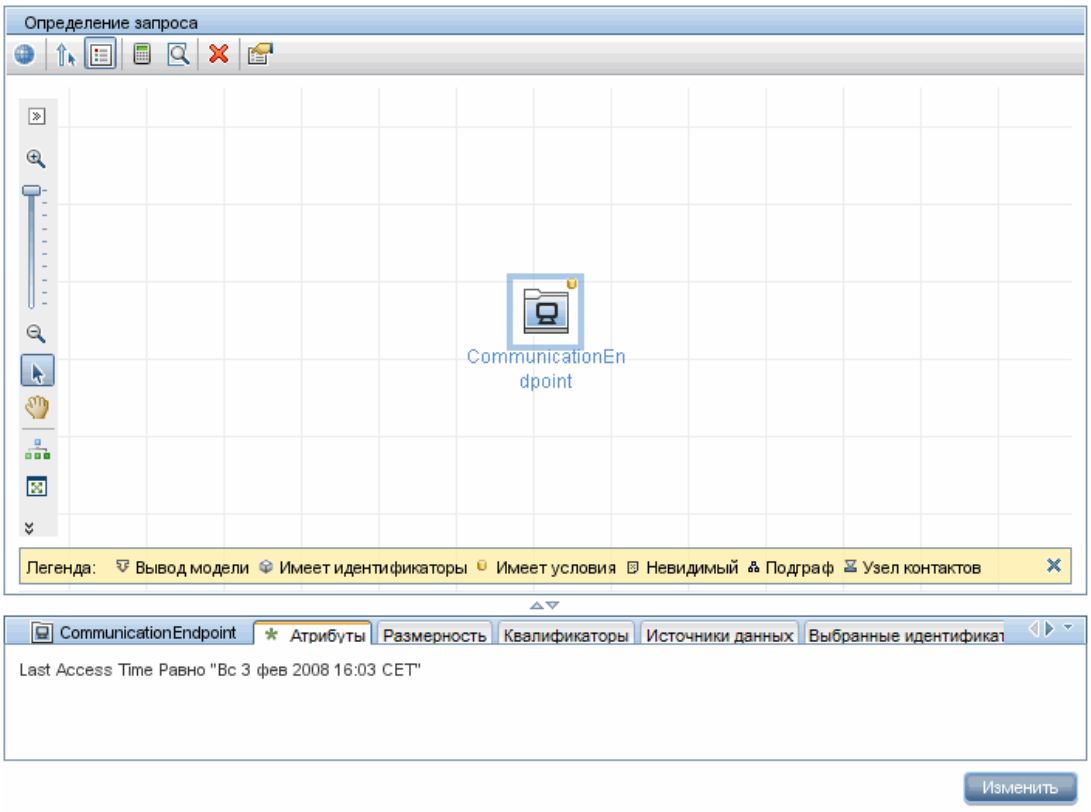
Во время этого шага вы создадите представление для просмотра экземпляров типов ЭК.

1. В UCMDB откройте студию моделирования (**Моделирование > Студия моделирования**).

2. Создайте представление. Подробнее см. в разделе "Создать представление образца" на странице 1 (*Руководство по моделированию в HP Universal CMDB*).

3. В TQL-запрос можно добавить условия, например, время последнего доступа – более

шести месяцев назад:



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 "Отчеты о топологии: обзор" на странице 1 (*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. Подробнее см. в разделе "Файлы журнала адаптера" на странице 152.

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 версии 9.01 или более поздней и Eclipse IDE for Java EE Developers версии 1.2.2.20100217-2310 или более поздней.

Эта задача включает следующие шаги:

- "Необходимые условия" на следующей странице
- "Установка" на следующей странице
- "Подготовка рабочей среды" на следующей странице
- "Создание адаптера" на странице 112
- " Настройка подключаемого модуля CMDB" на странице 112
- "Импорт модели классов UCMDB" на странице 113

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 29.

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 вид: **C:\hp\UCMDB\UCMDBServer\runtime\fcmdb\CodeBase\<adapter name>**
. 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 **ОК**. 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 **ОК**.

   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 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 134.

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 предыдущей странице.

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 как указано в шаге "Построение ORM-файла — сопоставление классов

UCMDB с таблицами базы данных" на странице 113. 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" ниже.

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" на странице 1 (*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 134.

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" на странице 1 (*Руководство по управлению потоками данных в HP Universal CMDB*).

b. Создайте точку интеграции с помощью адаптера, который использовался в шаге "Создание точки интеграции" на предыдущей странице. Подробнее см. в разделеNew Integration Point/Edit Integration Point Dialog Box (*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. Подробнее см. в описанном выше шаге " Настройка подключаемого модуля CMDB" на странице 112.

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 119
- "Файл orm.xml" на странице 121
- "Файл reconciliation_types.txt" на странице 134
- "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 134
- "Файл transformations.txt" на странице 136
- "Файл discriminator.properties" на странице 137
- "Файл replication_config.txt" на странице 138
- "Файл fixed_values.txt" на странице 138
- "Файл persistence.xml" на странице 138

Общая конфигурация

- **adapter.conf.** Файл конфигурации адаптера. Подробнее 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. Подробнее см. в разделе "Файл simplifiedConfiguration.xml" на странице 119.

Расширенная настройка

- **orm.xml**. Файл сопоставлений объектов, в котором задается сопоставление типов ЭК CMDB и таблиц базы данных. Подробнее см. в разделе "Файл orm.xml" на странице 121.
- **reconciliation_types.txt**. Содержит правила, используемые для настройки типов выверки. Подробнее см. в разделе "Файл reconciliation_types.txt" на странице 134.
- **reconciliation_rules.txt**. Содержит правила выверки. Подробнее см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 134.
- **transformations.txt.** Файл преобразований, в котором указываются конвертеры для преобразования значений CMDB в значения БД и наоборот. Подробнее см. в разделе "Файл transformations.txt" на странице 136.
- **Discriminator.properties.** В этом файле выполняется сопоставление всех поддерживаемых типов ЭК со списком возможных соответствующих значений, разделенных запятыми. Подробнее см. в разделе "Файл discriminator.properties" на странице 137.
- **Replication_config.txt.**Этот файл содержит список типов ЭК и связей, условия свойств которых поддерживаются этим подключаемым модулем репликации. Подробнее см. в разделе "Файл replication_config.txt" на странице 138.
- **Fixed_values.txt.**Этот файл обеспечивает настройку фиксированных значений определенных атрибутов определенных типов ЭК. Подробнее см. в разделе "Файл fixed_values.txt" на странице 138.

Конфигурация Hibernate

- **persistence.xml.** Используется для переопределения встроенных конфигураций Hibernate. Подробнее см. в разделе "Файл persistence.xml" на странице 138.

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 (подробнее см. в шаге "Настройка файла reconciliation_types.txt (для системы сопоставления UCMDB 9.0x по умолчанию)" в "Реализация системы сопоставления" на странице 181) 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.

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 121

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

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" ниже
- "Несколько ORM-файлов" на странице 125
- "Правила именования" на странице 125
- "Использование встроенных инструкций SQL вместо имен таблиц" на странице 125
- "Схема orm.xml" на странице 126
- "Пример создания файла orm.xml" на странице 130

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 следовать 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 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

Имя элемента и путь	Описание	Атрибуты
		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. 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	**Имя:** name **Описание:** Имя вторичной таблицы. Если имя таблицы не содержит схему, к которой принадлежит, поиск таблицы

Имя элемента и путь	Описание	Атрибуты
	основной таблице со связью один к одному. Можно настроить несколько вторичных таблиц. Необязательно.	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 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	**Имя:** name **Описание:** Имя столбца дискриминатора. **Обязательно?** Обязательно **Тип:** Строка

Имя элемента и путь	Описание	Атрибуты
	для каждой строки.	
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. **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.

Имя элемента и путь	Описание	Атрибуты
		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 **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 Table**.

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 **класса** объекта 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.

Файл reconciliation_types.txt

Этот файл содержит типы выверки.

Каждая строка в файле представляет тип ЭК UCMDB , который относится к объединенному типу ЭК в TQL-запросе.

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 `nameip_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.

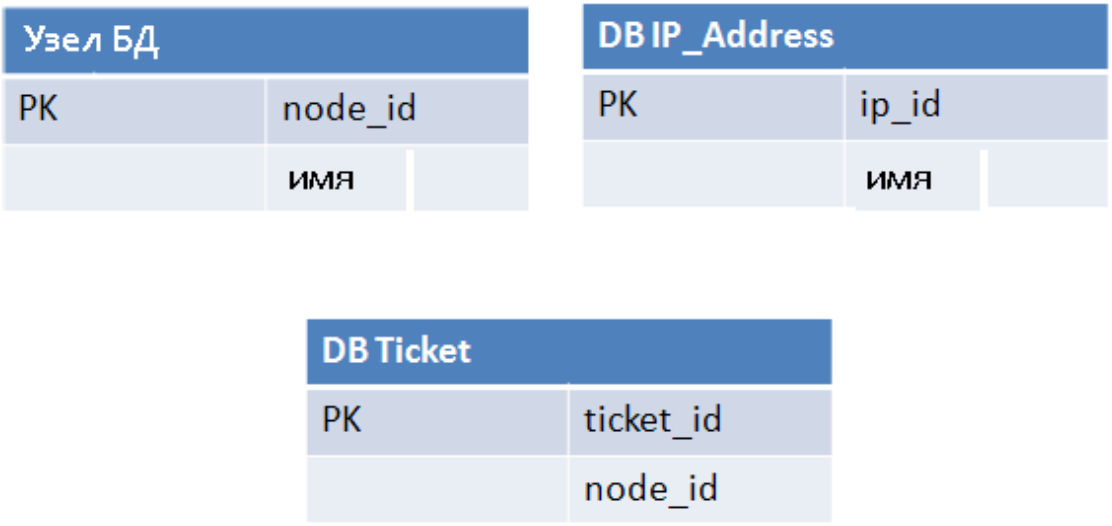
> **Примечание.** В результате будет действовать ограничение — все атрибуты выверки в источнике данных должны находиться в таблицах, которые используют общий первичный ключ.

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.FcmdbDalTransformerTo ExternalDB**. 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.

Подробнее см. в разделе "Встроенные конвертеры" на странице 139.

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. Подробнее см. в разделе "Подключаемые модули" на странице 144.

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. в разделе 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">         <properties>             <!--
Don't change this value  -->            <property
name="hibernate.archive.autodetection" value="class, hbm" />
            <!--The driver class name"/-->             <property
name="hibernate.connection.driver_class" value="com.mercury.
jdbc.MercOracleDriver" />             <!--The connection url"/-->
            <property name="hibernate.connection.url"
value="jdbc:mercury:oracle: //artist:1521;sid=cmdb2" />
            <!--DB login credentials"/-->             <property
name="hibernate.connection.username" value="CMDB" />
            <property name="hibernate.connection.password"
value="CMDB" />             <!--connection pool properties"/-->
            <property name="hibernate.c3p0.min_size" value="5" />
            <property name="hibernate.c3p0.max_size" value="20" />
            <property name="hibernate.c3p0.timeout" value="300" />
            <property name="hibernate.c3p0.max_statements" value="50"
/>            <property name="hibernate.c3p0.idle_test_period"
value="3000" />             <!--The dialect to use-->
            <property name="hibernate.dialect"
value="org.hibernate.dialect. OracleDialect" />         </properties>
    </persistence-unit> </persistence>
```

Встроенные конвертеры

Следующие конвертеры можно использовать для преобразования объединенных запросов и заданий репликации в данные БД и обратно.

Этот раздел охватывает следующие темы:

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>
```

Example of Converting 'sys' Value to 'System' Value:

In this example, `sys` value in the CMDB is transformed into `System` value in the federated database, and `System` value in the federated database is transformed into `sys` value in the CMDB.

If the value does not exist in the XML file (for example, the string `demo`), the converter returns the same input value it receives.

```
<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 '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.**

Подключаемые модули

Общий адаптер БД поддерживает следующие подключаемые модули:

- Подключаемый 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.
- Подключаемый модуль для синхронизации макета.
- Подключаемый модуль для получения поддерживаемых запросов для синхронизации. Если подключаемый модуль не указан, возвращаются все имена TQL-запросов.
- Внутренний подключаемый модуль для изменения определения и результата TQL-запроса.
- Внутренний подключаемый модуль для изменения запроса макета и ЭК в результате.
- Внутренний подключаемый модуль для изменения запроса макета и связей в результате.
- Внутренний подключаемый модуль для изменения действия обратной передачи идентификаторов.

См. дополнительные сведения о реализации и развертывании подключаемых модулей в разделе "Реализация подключаемого модуля" на странице 105.

Примеры конфигурации

В этом разделе приводятся примеры конфигурации.

Этот раздел охватывает следующие темы:

Сценарий использования

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>
            <or>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
            </or>
        </reconciliation-by-single-node>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="composition">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

Расширенное определение

Файл orm.xml

Обратите внимание на добавление сопоставления связей. См. подраздел определений в документе "Файл orm.xml" на странице 121.

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

Подробнее см. в разделе "Файл reconciliation_types.txt" на странице 134.

```
node
```

Файл reconciliation_types.txt

Подробнее см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 134.

```
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">
            <or>
                <attribute CMDB-attribute-name="name" column-
name="Device_Name" />
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
```

```
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

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">
            <or>
                <connected-node-attribute CMDB-attribute-name="name"
column-name="Device_PreferredIPAddress" />
            </or>
        </reconciliation-by-two-nodes>
    </CMDB-class>
    <class CMDB-class-name="card" default-table-name="hwCards"
connected-CMDB-class-name="node" link-class-name="containment">
        <foreign-primary-key column-name="Device_ID" CMDB-class-
primary-key-column="Device_ID" />
        <primary-key column-name="hwCards_Seq" />
        <attribute CMDB-attribute-name="card_class" column-
name="hwCardClass" />
        <attribute CMDB-attribute-name="card_vendor" column-
name="hwCardVendor" />
        <attribute CMDB-attribute-name="card_name" column-
name="hwCardName" />
    </class>
</generic-DB-adapter-config>
```

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

Подробнее см. в разделе "Файл reconciliation_types.txt" на странице 134.

```
node
```

Файл reconciliation_types.txt

Подробнее см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 134.

```
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.

Подробнее см. в разделе "Файл orm.xml" на странице 121.

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 86.

Внешние ссылки

Сведения о спецификации JavaBeans 3.0 см. по адресу http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

Устранение неполадок и ограничения

В данном разделе описываются процедуры поиска и устранения неполадок, а также ограничения общего адаптера базы данных.

Общие ограничения

- Аутентификация NTLM SQL Server не поддерживается.
- 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**.
- Общий адаптер БД не поддерживает аутентификацию Windows для SQL Server.

Глава 5

Разработка адаптеров Java

Данная глава включает:

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

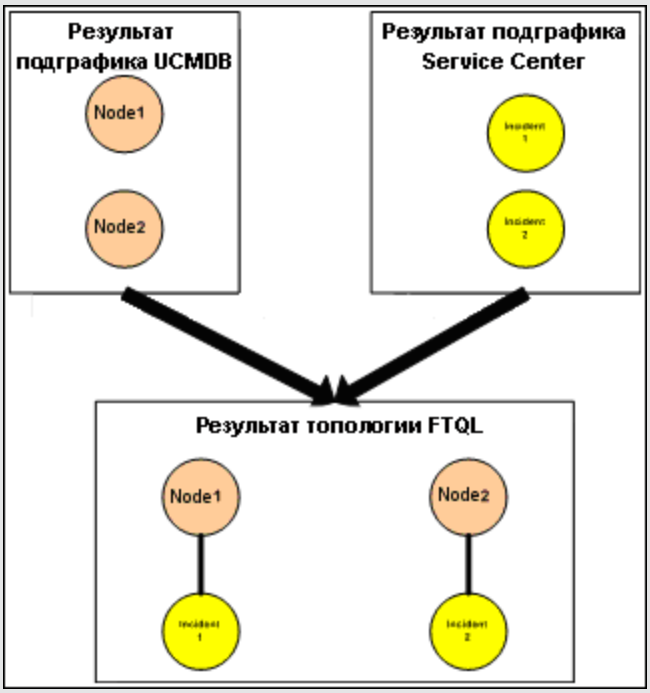
Объединенные TQL-запросы обеспечивают извлечение данных из любого внешнего репозитория без извлечения его данных.

Объединенный TQL-запрос использует адаптеры, представляющие внешние репозитории данных, для создания соответствующих внешних связей между ЭК из других внешних репозиториев и ЭК UCMDB.

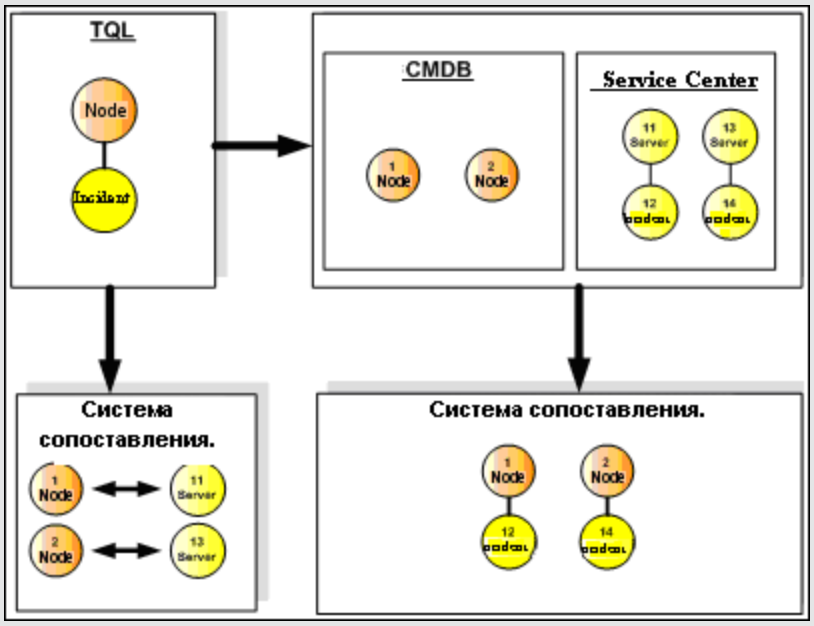
Пример оперативного объединения:

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 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.

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" на следующей странице
- "Система сопоставления." на следующей странице
- "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.

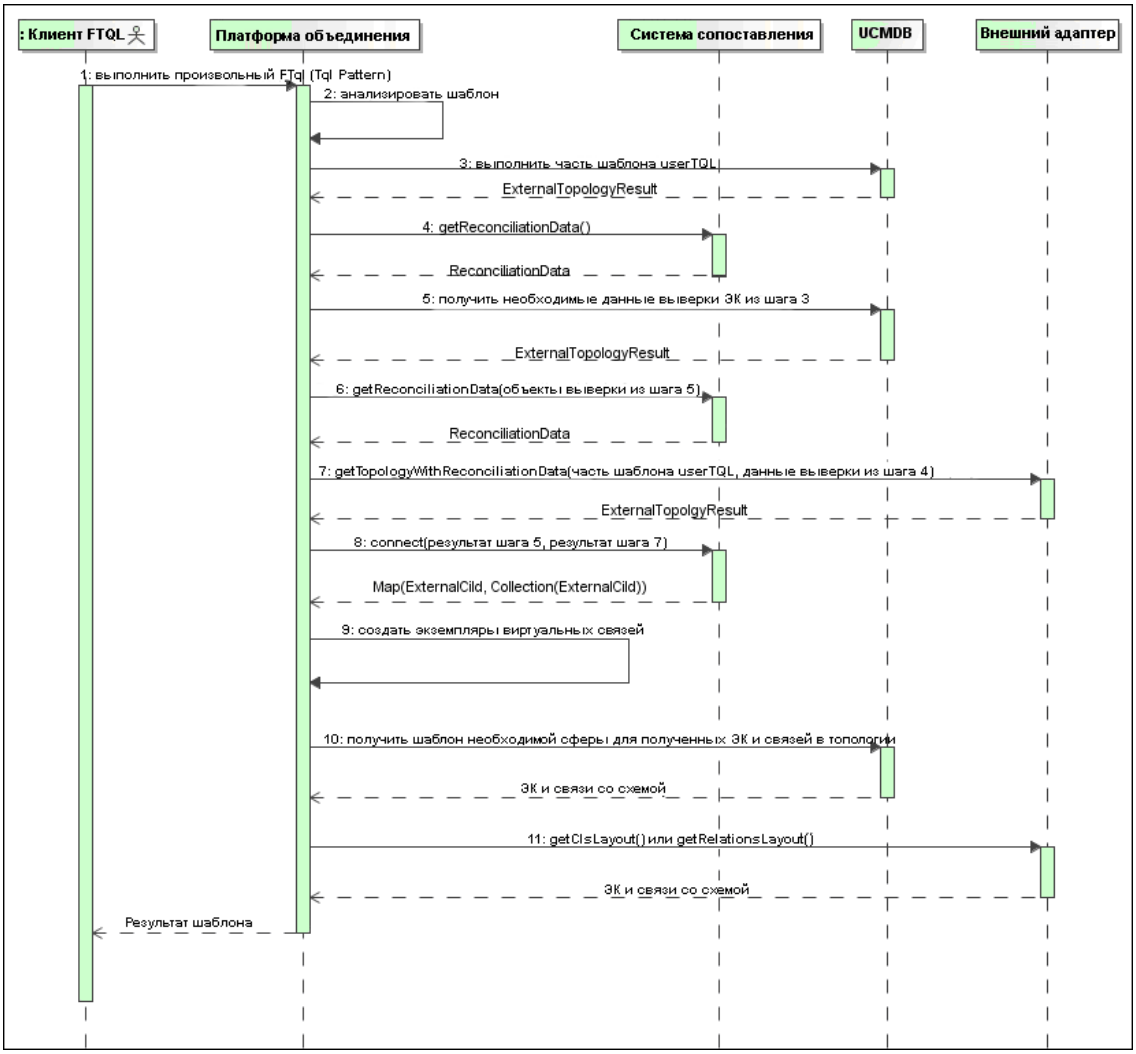
Этот раздел охватывает следующие темы:

- "Вычисление начинается на стороне сервера" ниже
- "Вычисление начинается на стороне внешнего адаптера" на странице 166
- "Пример потока Federation Framework для объединенных TQL-запросов" на странице 167

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.

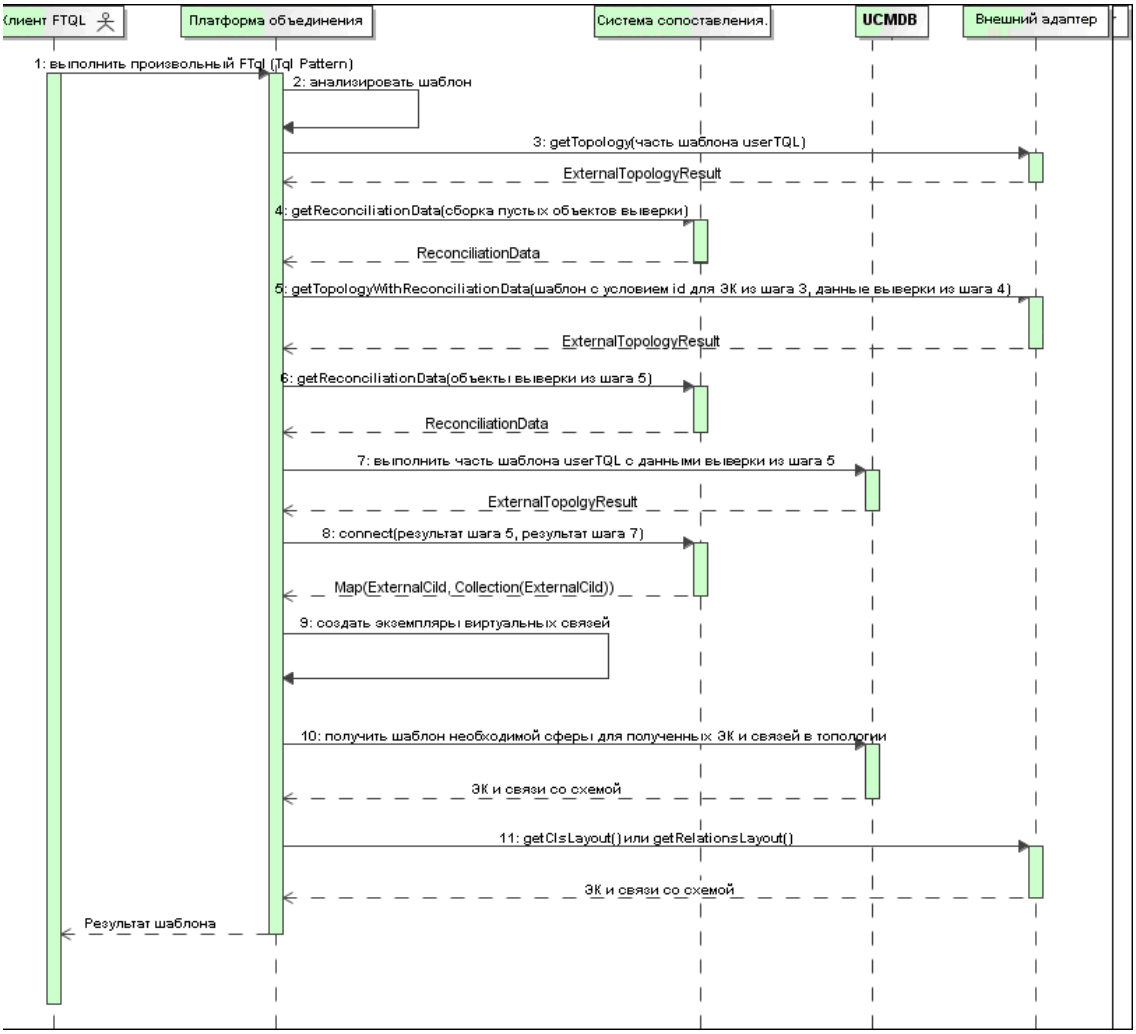


Числа на изображении объясняются ниже:

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, какие данные необходимы для сопоставления ЭК выверки в UCMDB с внешним репозиторием. Существует три типа данных выверки:

Число	Пояснение
	• **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:

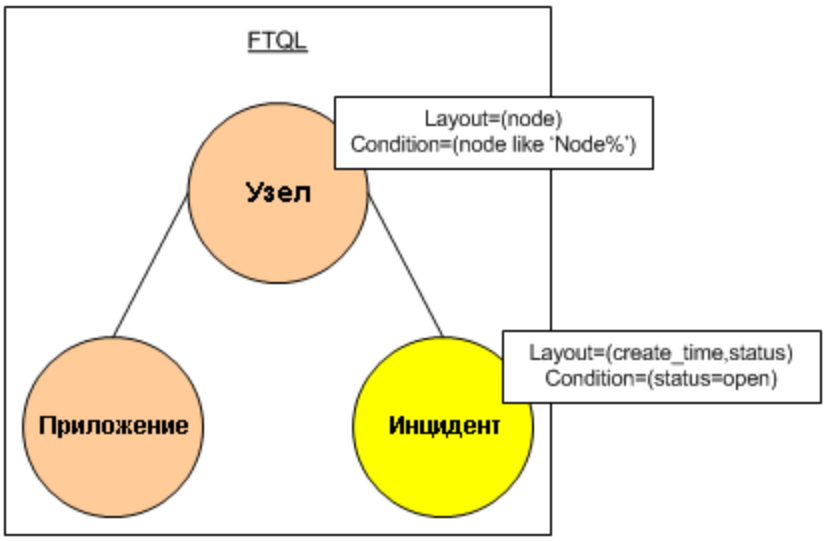
Число	Пояснение
1	Federation Framework получает вызов от объединенного вычисления TQL.
2	Federation Framework анализирует адаптер, находит виртуальное отношение и разделяет исходный TQL-запрос на два пододаптера — один для UCMDB и второй для внешнего репозитория данных.
3	Federation Framework запрашивает топологию подзапроса TQL у внешнего адаптера. Возвращенный элемент `ExternalTopologyResult` не должен содержать объекты выверки, поскольку данные выверки не являются частью запроса.
4	После получения результатов топологии Federation Framework вызывает соответствующую систему сопоставления для текущего виртуального отношения и запрашивает данные выверки. Параметр

Число	Пояснение
	`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

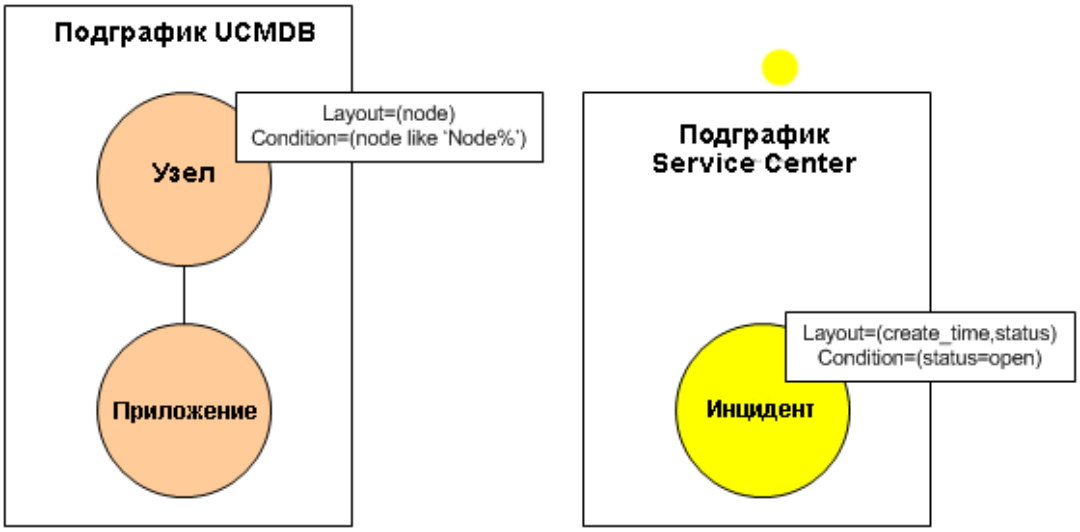
В этом примере описывается способ просмотра всех открытых инцидентов для определенных узлов. Репозиторий данных ServiceCenter — это внешний репозиторий. Экземпляры узлов хранятся в UCMDB, а экземпляры инцидентов хранятся в ServiceCenter.

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.

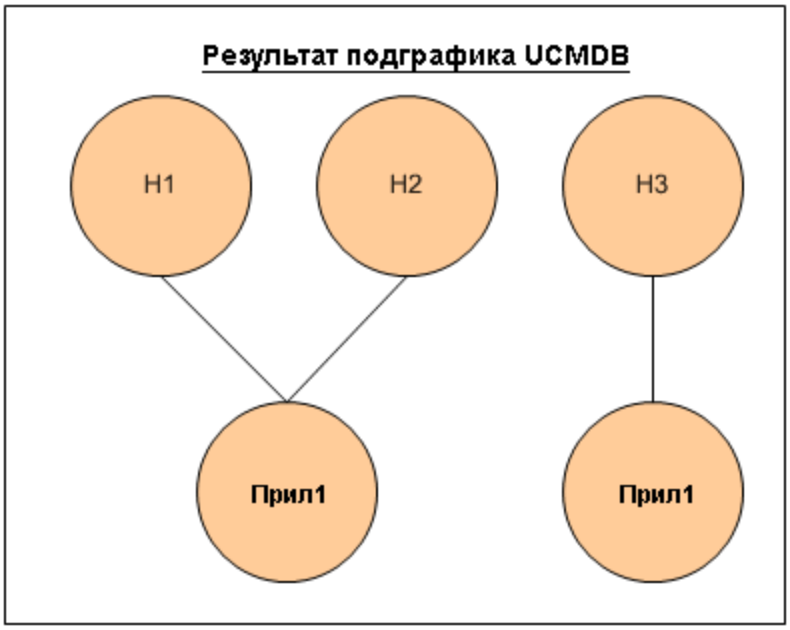


Примечание. Для объединения атрибутов вызывается метод адаптера **getTopology**. Данные выверки адаптируются в TQL-запросе пользователя (в нашем случае элементе ЭК).

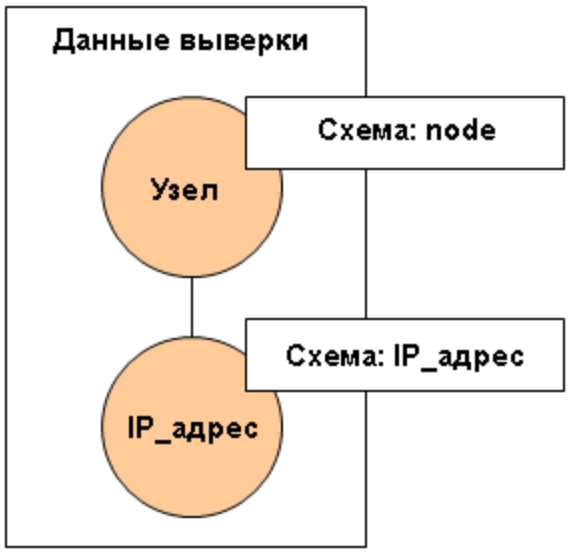
1. После анализа адаптера Federation Framework распознает виртуальное отношение между элементами `Node` и `Incident` и разделяет объединенный TQL-запрос на два подграфа:



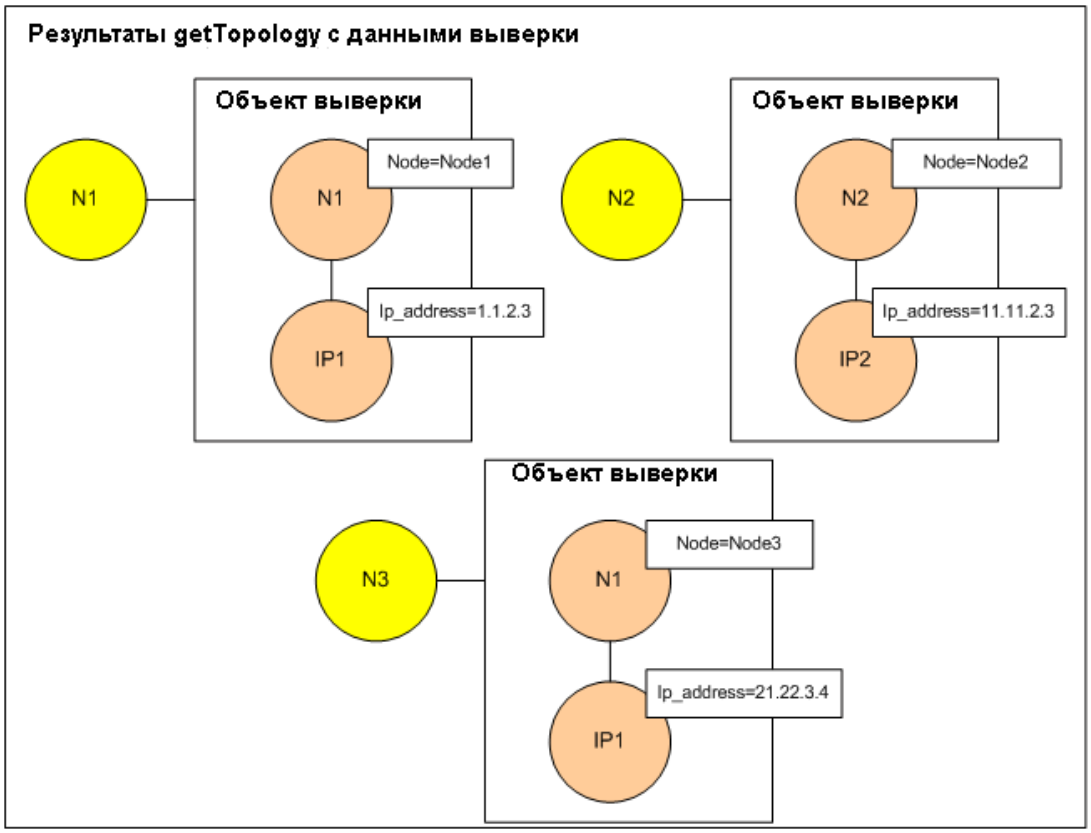
2. Federation Framework выполняет подграф UCMDB для запроса топологии и получает следующие результаты:



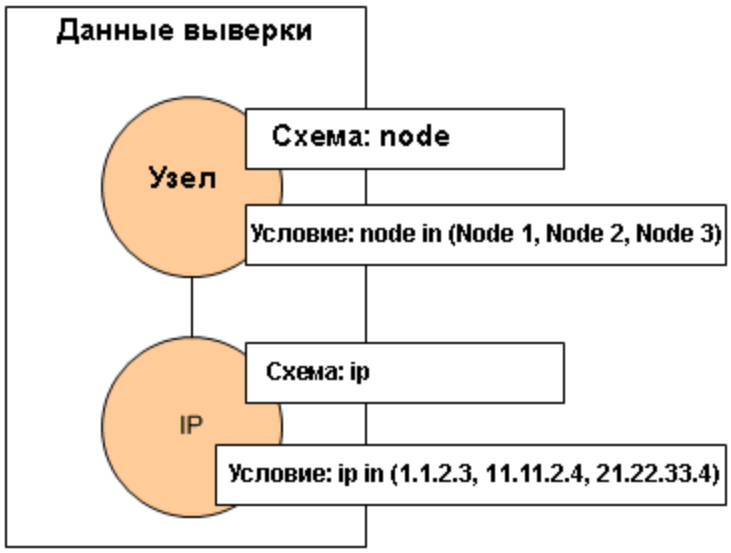
3. Federation Framework запрашивает данные выверки для первого репозитория данных (UCMDB) у соответствующей системы сопоставления, содержащие сведения для соединения полученных данных из двух репозиториев. В этом случае данные выверки примут следующий вид:



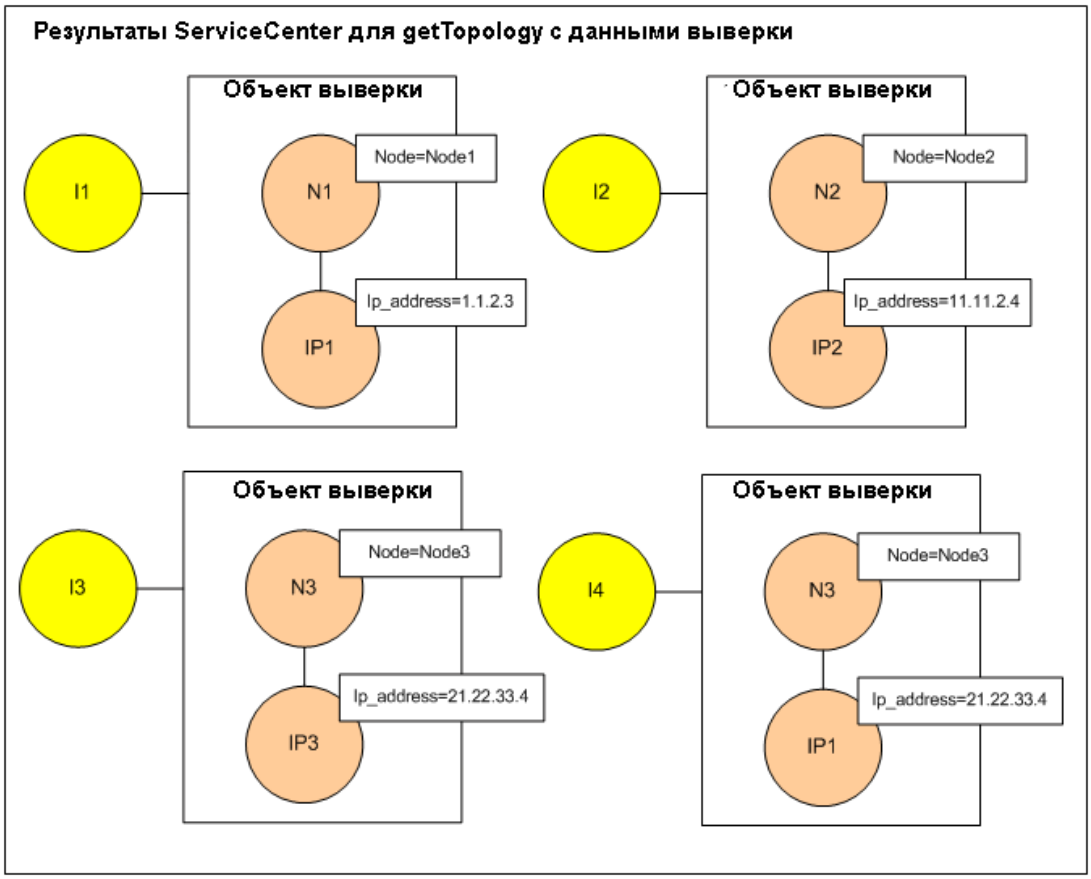
4. Federation Framework создает одноузловой запрос топологии с условиями Node и ID из предыдущего результата (`node` в H1, H2, H3) и выполняет этот запрос с необходимыми данными выверки в UCMDB. Результат включает ЭК узла, связанные с условием идентификатора и соответствующим объектом выверки для каждого ЭК:



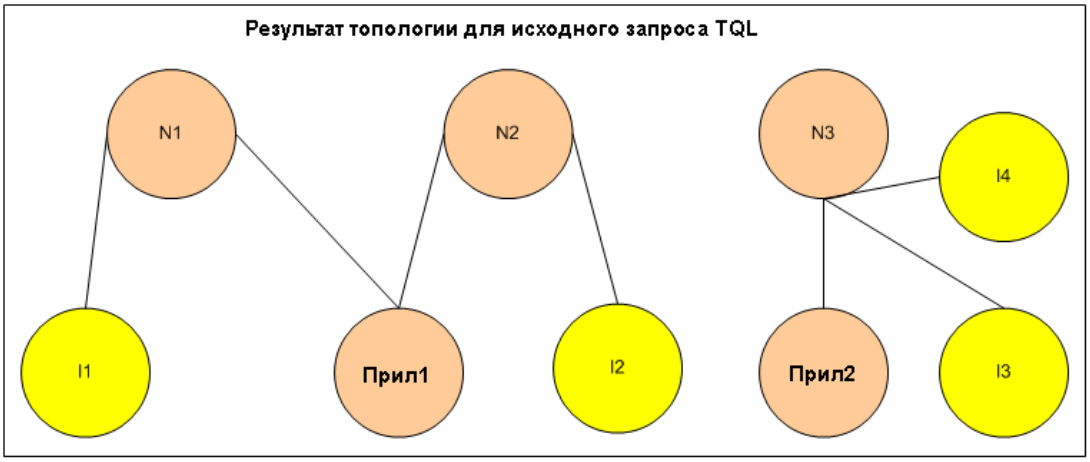
5. Данные выверки для ServiceCenter должны содержать условие для `node` и `ip`, производное от объектов выверки, полученных от UCMDB:



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

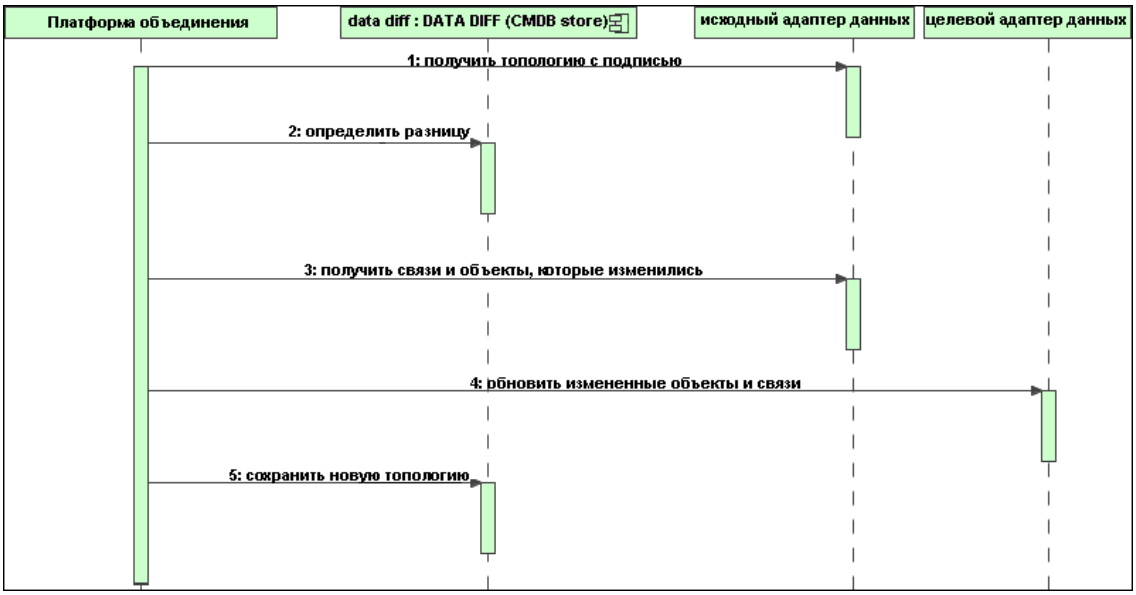
Этот раздел охватывает следующие темы:

Определения и термины

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 one Node** используется, если адаптер не поддерживает внешние связи, т.е. если он не предназначен для получения запросов более чем с одним внешним ЭК. Все интерфейсы OneNode создаются для упрощения рабочего процесса. В случаях, когда необходимы более сложные запросы, используйте интерфейс `DataAdapter`.
- **Интерфейс DataAdapter** используется для настройки адаптеров, поддерживающих сложные объединенные запросы. Запрос выверки в этих адаптерах является частью одного параметра QueryDefinition. Эти адаптеры также могут использоваться для заполнения.

Интерфейсы OneNode

В следующих интерфейсах применяются разные типы данных выверки:

- **OneNodeTopologyIdReconciliationDataAdapter.** Используется, если адаптер поддерживает **single-node TQL** и выверка между репозиториями вычисляется по идентификатору.
- **OneNodeTopologyPropertyReconciliationDataAdapter.** Используется, если адаптер поддерживает **single-node TQL** и выверка между репозиториями вычисляется по свойствам ЭК.
- **OneNodeTopologyDataAdapter.** Используется, если адаптер поддерживает **single-node TQL** и выверка между репозиториями вычисляется по топологии.

Интерфейсы Data Adapter

- **DataAdapter.** Используется для сложных объединенных TQL-запросов. Обеспечивает максимальное разнообразие.
- **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. Подробнее см. в документации Java для конкретного 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 (значения по умолчанию: sysadmin/sysadmin).
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:

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). См. дополнительные сведения о добавлении новых типов ЭК в модель классов UCMDB в разделе "Работа с Селектором ЭК" на странице 1 документа 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 "Диспетчер пакетов" на странице 1 (*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**  .

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.BaseDestinationDat-
a">

<!-- check -->

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

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

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

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

</destinationInfo>

<resultMechanism isEnabled="true">

<autoDeleteCITs isEnabled="true">

<CIT>link</CIT>

<CIT>object</CIT>

</autoDeleteCITs>

</resultMechanism>

</taskInfo>

<adapterInfo>

<adapter-capabilities>

<support-federated-query>

<!--<supported-classes/> <!—see the section about supported
classes-->

<topology>

<pattern-topology /> <!—or <one-node-topology> -->

</topology>
```

```
</support-federated-query>

<!--<support-replicatioin-data>

<source>

<changes-source/>

</source>

<target/>

</adapter-capabilities>

<default-mapping-engine />

<queries />

<removedAttributes />

<full-population-days-interval>-1</full-population-days-
interval>

</adapterInfo>

<inputClass>destination_config</inputClass>

<protocols />

<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 184.

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.

   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.

6. ## Укажите правила выверки или реализуйте систему сопоставления

   Если адаптер поддерживает объединенные TQL-запросы, существует три варианта настройки системы сопоставления:

   - Использовать систему сопоставления CMDB 9.0x по умолчанию, использующую внутренние правила CMDB для сопоставления. Для этого оставьте XML-тег **<default-mapping-engine>** пустым.

     Подробнее см. в разделе "Файл reconciliation_types.txt" на странице 134.

   - Использовать систему сопоставления CMDB 8.0x. Для этого добавьте следующий XML-тег: **<default-mapping-engine>com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine</default-mapping-engine>**

     Подробнее см. в разделе "Файл reconciliation_rules.txt (для обратной совместимости)" на странице 134.

   - 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. См. общие сведения о развертывании пакета в разделе "Диспетчер пакетов" на странице 1 документа *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» (*Руководство по администрированию HP Universal CMDB*).

Реализация системы сопоставления

Конфигурация системы сопоставления зависит от того, какая именно система используется.

Эта задача включает следующие шаги:

- "Настройка файла reconciliation_types.txt (для системы сопоставления UCMDB 9.0x по умолчанию)" ниже
- "Настройте файл reconciliation_rules.txt (для системы сопоставления UCMDB 8.0x)" ниже

1. ### Настройка файла reconciliation_types.txt (для системы сопоставления UCMDB 9.0x по умолчанию)

   Файл, используемый для указания типов ЭК, которые применяются при выверке адаптера.

   Запишите типы ЭК, используемые для выверки, в одну строку следующим образом:

   ```
   node business_application
   ```

   Пометите файл в папку **adapterCode\<AdapterID>\META-INF** пакета адаптера. Для реализации выверки по ID (на основании сопоставления идентификаторов CMDB в CMDB со значениями в удаленной базе данных) необходимо сопоставить особый атрибут CMDB под названием **cmdb_id** со столбцом в базе данных, имеющим формат string (char, varchar) или byte[] (raw/bytes).

2. ### Настройте файл reconciliation_rules.txt (для системы сопоставления UCMDB 8.0x)

   Этот файл используется для настройки правил выверки. Каждая строка файла представляет правило. Пример:

   ```
   reconcilition_type[node] expression[^node.name OR ip_address.name]
   end1_type[node] end2_type[ip_address] link_type[containment]
   ```

   Параметру **reconcilition_type** назначается тип ЭК, по которому выполняется выверка (имя класса UCMDB, связанного с объединенным классом в TQL-запросе).

   Параметр **expression** содержит логику, которая определяет равенство двух объектов выверки (объект выверки со стороны UCMDB и объект выверки со стороны объединенного адаптера).

   Выражение состоит из операторов OR и AND.

   Правило именования атрибутов в части выражения: **[className].[attributeName]**. Например, атрибут **ip_address** в классе **ip** будет иметь имя **ip.ip_address**.

   Вы можете указать сопоставление по порядку. Упорядоченное сопоставление проверяет первое подвыражение OR. Если два объекта выверки имеют значение в атрибутах подвыражения и оно возвращает значение false (объекты выверки неравны), выражение OR не сравнивается.

   Для упорядоченного отбора используйте элемент **ordered expression** вместо **expression**.

   Знак циркумфлекса (**^**) используется для пропуска регистра при сравнении.

   Другие параметры (**end1_type**, **end2_type** и **link_type**) используются, только если данные выверки содержат два узла, а не только узел типа выверки (топологических данных выверки). В этом случае данные выверки примут следующий вид **end1_type -(link_type)> end2_type**.

   Добавление соответствующего макета не требуется, так как он берется из выражения.

Для выполнения выверки по идентификатору UCMDB используйте **cmdb_id** в качестве имени атрибута выражения.

Пометите файл в папку **adapterCode\<AdapterID>\META-INF** в пакете адаптера.

Примеры.

- Добавить правило выверки можно только для типа ЭК node. Это связано с тем, что только типы ЭК node имеют действующие связи с внешними типами ЭК. Например, ЭК node в CMDB связан с ЭК node в ServiceCenter через атрибут `node.name` или `ip_address.name`.
- В этом случае правило выверки — это правило топологии, и выражение упорядочено. Правило выполняет следующие проверки ЭК в рамках сравнения:
  - Если значения атрибута `node.name` равны, правило связывает узлы.
  - Если значения атрибута `node.name` неравны, правило не связывает узлы.
  - Если атрибут `node.name` имеет значение null в одном из сравниваемых ЭК, правило проверяет атрибут `ip_address.name`. Если значения атрибута `ip_address.name` равны, правило связывает узлы.

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.
	<one-node-topology>	Возможность объединения запросов с одним объединенным узлом запроса.
	<pattern-topology>	Возможность объединять сложные запросы.
	<support-replication-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.
	<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 (для системы сопоставления UCMDB 8.0x: **com.hp.ucmdb.federation.mappingEngine.AdapterMappingEngine**)
	<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).

Глава 6

Разработка адаптеров принудительной отправки

Данная глава включает:

Разработка адаптеров принудительной отправки данных

Общий адаптер принудительной отправки предоставляет платформу для быстрой разработки адаптеров интеграции, выполняющих принудительную отправку данных UCMDB 9.0x во внешние репозитории (базы данных и сторонние приложения). Для разработки настраиваемых адаптеров интеграции на основе общего адаптера принудительной отправки требуется следующее:

- XML-файл сопоставления между типами связей ЭК UCMDB и внешними элементами данных.
- Сценарий Jython для принудительной отправки элементов данных во внешний репозиторий.

Разностная синхронизация

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 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 все данные добавляются в CMDB при каждой синхронизации.

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 их 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 **<query name>.xml**.

Подробнее см. в разделе "Схема файла сопоставления" на странице 195.

Эта задача включает следующие шаги:

- "Создание файла сопоставления" ниже
- "Сопоставление ЭК" на следующей странице
- "Сопоставление связей " на следующей странице

1. ### Создание файла сопоставления

   Структура файла сопоставления будет иметь следующий вид:

```
<?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 наследованных 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 потомков 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 таким образом, чтобы сопоставлялись и все типы связей, наследующие от указанного:

```
<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 только указанного типа связей, но не его наследников:

```
<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. Подробнее см. в разделе "Разработка адаптеров Jython" на странице 40.

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, заданные для адаптера (см. дополнительные сведения в шаге "Измените файл discoveryPatterns\push_adapter.xml." раздела "Создание пакета адаптера " на странице 194), воспользуйтесь следующим кодом:

```
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 192

Работа с результатами сопоставления

Адаптер принудительной отправки создает сроки 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 есть `ExternalId` в момент его отправки сценарию. 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 `externaId`. 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`.

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. В случае, 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()`. Региональные настройки сервера и зонда по умолчанию могут различаться. В этом случае 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

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. Подробнее см. в разделе

"Создание пакета адаптера " на следующей странице.

Эта задача позволяет адаптеру принудительной отправки выполнять разностную синхронизацию. Подробнее см. в разделе "Разностная синхронизация " на странице 186.

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.

Создание пакета адаптера

1. Извлеките содержимое файла **C:\hp\UCMDB\UCMDBServer\content\adapters\push-adapter.zip** во временную папку. Файл **sql_queries** в пакете адаптера, расположенный в разделе **adapterCode > PushAdapter > sqlTablesCreation,** содержит запросы, необходимые для создания таблиц в новой схеме Oracle для тестирования адаптера. Таблицы соответствуют файлу **adapterCode\<adapter ID>\mappings\mappings.xml**.

   > **Примечание.** Файл **sql_queries** адаптеру не требуется. Он служит только для примера.

2. Измените файл **discoveryPatterns\push_adapter.xml**.
   a. Измените тег <pattern> на новый идентификатор и отображаемую метку. Замените:

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

   на:

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

   b. Обновите список параметров в соответствии с необходимыми атрибутами подключения. Не удаляйте атрибут **probeName**.

3. Переименуйте папку **adapterCode\PushAdapter** в соответствии с идентификатором адаптера, используемым в предыдущем шаге (например, **adapterCode\MyPushAdapter**).
4. В файле **discoveryScript** есть сценарий **script pushScript.py**, 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 189). Если сценарий переименован, свойство `jythonScript.name` в **adapterCode\<adapter ID>\push.properties** должно быть обновлено соответствующим образом.
5. Файл **adapterCode\<adapter ID>\mappings\mappings.xml** в папке **adapterCode\<adapter ID>\mappings** содержит пример сопоставления:
   - ЭК типа Node и всех наследующих типов ЭК
   - ЭК типа UNIX без наследующих типов ЭК
   - Связи типа Dependency и всех наследующих типов связей
   - Связи типа Talk без наследующих типов связей

   Этот пример соответствует примеру таблиц в ORACLE в файле **sql_queries** (см. шаг 1).

   Замените файл **adapterCode\<adapter ID>\mappings\mappings.xml** на подготовленные файлы сопоставления (см. дополнительные сведения в разделе "Подготовка файлов сопоставления" на странице 187.

   Чтобы использовать файл сопоставления для каждого метода TQL, назначьте имя соответствующего TQL-запроса каждому XML-файлу с расширением **xml**. В этом случае файл **mappings.xml** будет использоваться по умолчанию, если определенный файл сопоставления не будет найден в текущем имени TQL-запроса. Имя файла сопоставления по умолчанию может быть изменено путем изменения свойства `mappingFile.default` в **adapterCode\<adapter ID>\push.properties**.

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	Информация об	1. **Имя:** type

Имя элемента и путь	Описание	Атрибуты
(integration > info)	исходном репозитории.	**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 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:

Имя элемента и путь	Описание	Атрибуты
		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. 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	Основные ключевые	

Имя элемента и путь	Описание	Атрибуты
(integration > targetcis > source_ci_type) -OR- (integration > targetcis > source_ci_type_tree -OR- (integration > targetrelations > link) -OR- (integration > targetrelations > source_link_type_tree)	атрибуты целевого типа ЭК.	
pkey (integration > targetcis> source_ci_type > targetprimarykey -OR- integration > targetcis > source_ci_type_tree > targetprimarykey -OR- (integration > targetrelations > link > targetprimarykey) -OR- integration > targetrelations > source_link_type_tree > targetprimarykey)	Один первичный ключевой атрибут. Требуется только в режиме **update** или **insert_else_update**.	
target_attribute (integration > targetcis> source_ci_type -OR- integration > targetcis	Атрибут целевого типа ЭК.	1. **Имя:** name **Описание:** Имя атрибута целевого типа ЭК. **Обязательно?** Обязательно **Тип:** Строка 2. **Имя:** datatype **Описание:** Тип данных атрибута

Имя элемента и путь	Описание	Атрибуты
> source_ci_type_tree -OR- integration > targetrelations > link -OR- integration > targetrelations > source_link_type_tree)		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.
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 >	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 — это одно или несколько значений 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**

Имя элемента и путь	Описание	Атрибуты
target_attribute)		**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- (integration > targetrelations > link > target_attribute > map -OR- integration > targetrelations > source_link_type_tree > target_attribute > map) Действительно только при использовании типа сопоставления **childattr**	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 число всех включенных значений. • **average** — возвращает среднее всех 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	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

Имя элемента и путь	Описание	Атрибуты
-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**.		дочернего ЭК. **Обязательно?** Обязательно, только если тип агрегации **childAttr** (по тому же пути) не =count. **Тип.** Строка
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 >	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

Имя элемента и путь	Описание	Атрибуты
target_attribute > map) Действительно только при использовании типа сопоставления **childattr**		
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 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

Имя элемента и путь	Описание	Атрибуты
		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 **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

Имя элемента и путь	Описание	Атрибуты
		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- integration > targetrelations > source_link_type_tree)	Тип ЭК End2 целевой связи	1. **Имя:** name **Описание:** Имя типа ЭК End2 целевой связи. **Обязательно?** Обязательно **Тип:** Строка 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

Имя элемента и путь	Описание	Атрибуты
		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: 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 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

Имя элемента и путь	Описание	Атрибуты
		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). **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,

Имя элемента и путь	Описание	Атрибуты
		используется значение **add** по умолчанию. 6. **Имя:** mamId **Описание:** Идентификатор объекта в исходной базе CMDB. **Обязательно?** Обязательно **Тип:** Строка

Глава 7

Разработка общих адаптеров принудительной отправки

Данная глава включает:

Разработка адаптеров принудительной отправки данных - обзор

Расширенный адаптер принудительной отправки данных – это структура данных, представляющая результат TQL-запроса. Все адаптеры, созданные на базе расширенного адаптера принудительной отправки данных, 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 расширенного адаптера принудительной отправки данных, должны быть основаны на корневом элементе, т.е. запрос должен содержать один узел с именем элемента **root**, либо одну или несколько элементов связей, имена которых начинаются с префикса **root**. Этот ЭК или эта связь служит корневым элементом запроса. Подробнее см. в разделе "Принудительная отправка данных" на странице 1 (*Руководство по управлению потоками данных в HP Universal CMDB*).

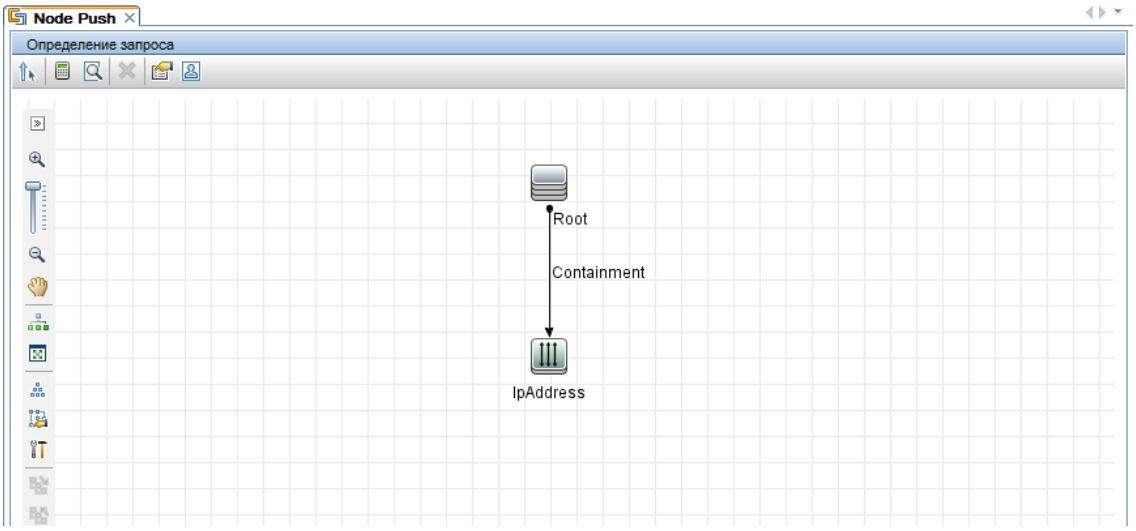
Разработка расширенного адаптера принудительной отправки данных состоит из двух основных шагов:

1. Реализация интерфейса PushConnector – 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:



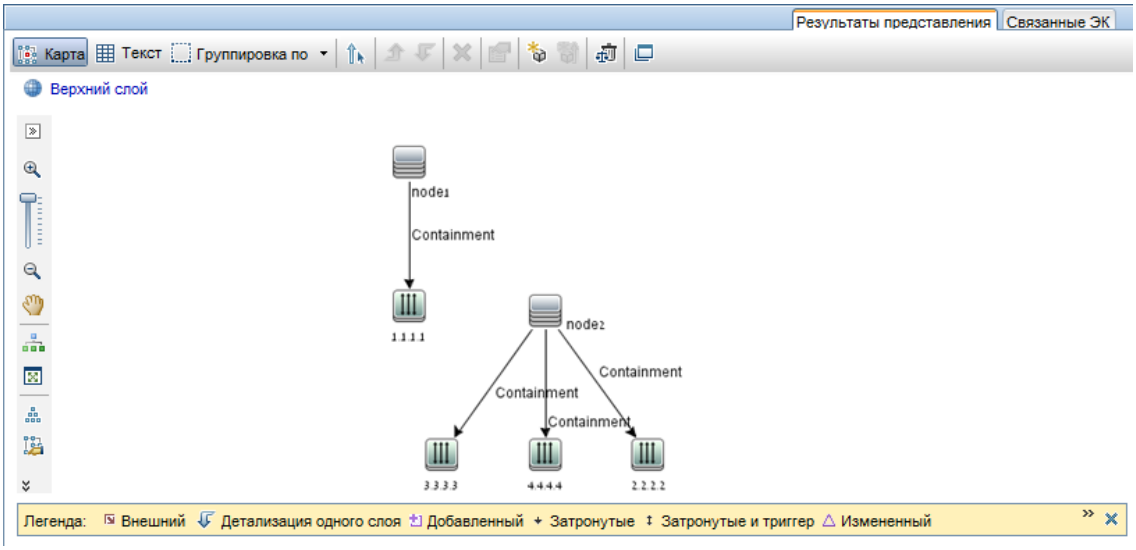
В файле сопоставления создается два целевых типа ЭК: **Computer** и **IP**. Computer имеет одну переменную и два атрибута. IP имеет один атрибут.

XML-файл сопоставления выглядит следующим образом:

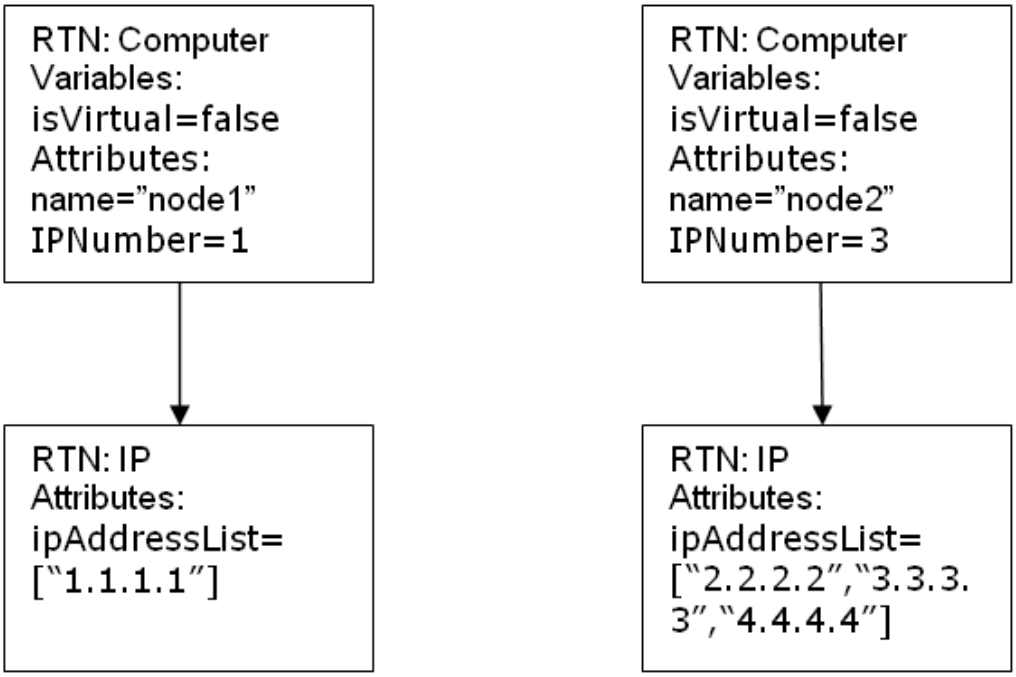
```
<integration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://schemas.hp.com/ucmdb/1/pushAdap
ter">
        <info>
            <source name="UCMDB" versions="10.0" vendor="HP" />
            <target name="PushProduct" versions="9.3" vendor="HP" />
     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

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

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 PushConnector 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_ci_type

Данная метка используется для создания RTN.

Атрибут name соответствует имени target_ci_type: name=Computer

Атрибут **is-valid** – это логическое значение, рассчитываемое в процессе сопоставления и указывающее, является ли допустимым текущий target_ci. Недопустимые значения target_ ci_type не добавляются в RTN. В данном примере мы не создаем экземпляр target_ci_type, для которого значение атрибута **root_iscandidatefordeletion** в UCMDB равно true.

Атрибут target_ci_type 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_ci_type дочерним элементом другого target_ci_type можно следующим образом:

```
<target_ci_type name="Portfolio">

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

<target_mapping name="CMDBId" datatype="STRING" value="globalId"/>

<target_ci_type name=Asset">

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

</target_ci_type>

</target_ci_type>
```

RTN **Portfolio** будет иметь дочернюю RTN **Asset**.

for-each-source-ci

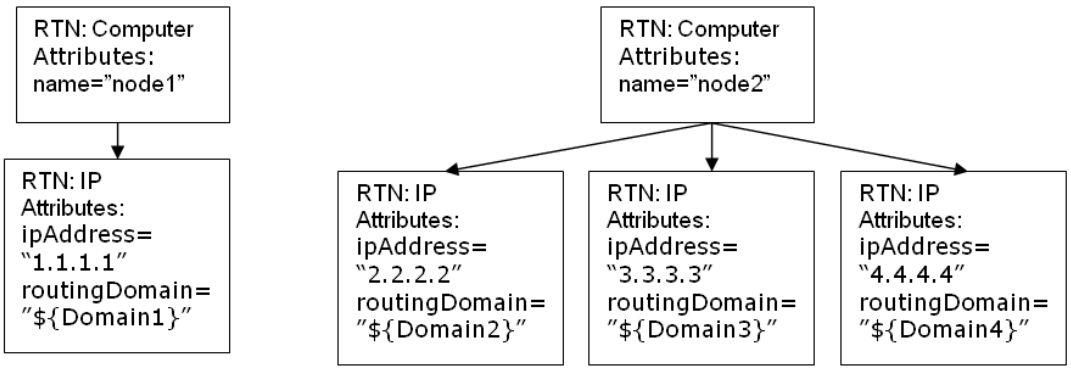
В данной метке перечисляются конкретные ЭК корневого экземпляра. У нее есть следующие поля:

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

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 PushConnector

Реализация должна поддерживать следующие основные шаги:

```
public class PushExampleAdapter implements PushConnector

{

public UpdateResult pushTreeNodes(ResultTreeNodeActionData
resultTreeNodes, QueryDefinition queryDefinition) throws
DataAccessException{

// 1. build an UpdateResult instance - the UpdateResult is used to
return mappings between the sent ids to the actual ids that entered
the data store. // Also has an update status which allows to pass
```

```
errors to the statistics in the UI. // 2. обработка данных: // a.
handle data to add. Can be retrieved by:
resultTreeNodeActionData.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. } }
```

Создание пакета адаптера

Пакет 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.

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 (info)	Исходный продукт	**Имя:** name **Описание:** имя продукта **Обязательно?** Обязательно **Тип:** Строка
		Имя: vendor **Описание:** поставщик продукта **Обязательно?** Обязательно **Тип:** Строка
		Имя: versions **Описание:** версия продукта **Обязательно?** Обязательно **Тип:** десятичное число
target (info)	Целевой продукт	**Имя:** name **Описание:** имя продукта **Обязательно?** Обязательно **Тип:** Строка
		Имя: vendor **Описание:**

Имя элемента и путь	Описание	Атрибуты
		поставщик продукта **Обязательно?** Обязательно **Тип:** Строка
		Имя: versions **Описание:** версия продукта **Обязательно?** Обязательно **Тип:** десятичное число
Import (integration)	Элемент-контейнер для импортированных файлов сценариев.	
scriptFile (integration>import)	Импортируемый файл сценария groovy.	Имя: path Описание: путь к файлу сценария Обязательно?: обязательно Тип: string
Targetcis (integration)	Элемент-контейнер для целевых типов ЭК.	
Source_instance_type (integration>targetcis)	Тип экземпляра исходного ЭК. Должен быть корневым элементом, как указано в TQL-запросе.	**Имя:** query-name. **Описание:** Имя соответствующего TQL-запроса **Обязательно?** Обязательно **Тип:** Строка
		Имя: name **Описание:** Корневой элемент, как указано в TQL-запросе. **Обязательно?** Обязательно **Тип:** Строка
dynamic_mapping	Схема сопоставлений, создаваемая	**Имя:** name

Имя элемента и путь	Описание	Атрибуты
(integration>targetcis> source_instance_type)	один раз для каждого фрагмента.	**Описание:** имя сопоставления **Обязательно?** Обязательно **Тип:** Строка
		Имя: keys-unique **Описание:** Указывает, являются ли ключи уникальными. **Обязательно?** Обязательно **Тип:** логическое
target_ci_type (integration>targetcis> source_instance_type) -OR- (integration>targetcis> source_instance_type> for-each-source-ci)	Целевой тип ЭК, добавляемый RTN.	**Имя:** name **Описание:** имя целевого типа ЭК **Обязательно?** Обязательно **Тип:** Строка
		Имя: is-valid **Описание:** проверяет допустимость текущего целевого ЭК согласно данному сценарию. **Обязательно?** необязательно **Тип:** String (groovy script)
Variable (target_ci_type)	Переменная для целевого типа ЭК.	**Имя:** name **Описание:** имя переменной **Обязательно?** Обязательно **Тип:** Строка

Имя элемента и путь	Описание	Атрибуты
		Имя: datatype **Описание:** тип данных переменной. **Обязательно?** Обязательно **Тип:** type-enum (может иметь следующие значения: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Имя: значение **Описание:** Значение, которое необходимо присвоить переменной **Обязательно?** Обязательно **Тип:** сценарий groovy
target_mapping (target_ci_type)	Атрибут для целевого типа ЭК.	**Имя:** name **Описание:** имя данного атрибута. **Обязательно?** Обязательно **Тип:** Строка
		Имя: datatype **Описание:** тип данных переменной. **Обязательно?** Обязательно **Тип:** type-enum (может иметь

Имя элемента и путь	Описание	Атрибуты
		следующие значения: INTEGER, LONG, FLOAT, DOUBLE, STRING, BYTES, XML, BOOLEAN, DATE, INTEGER_LIST, STRING_LIST)
		Имя: значение **Описание:** Значение, которое необходимо присвоить переменной **Обязательно?** Обязательно **Тип:** сценарий groovy
		Имя: ignore-on-null **Описание:** Если целевое значение сопоставление равно null, а данный атрибут имеет значение TRUE, игнорировать атрибут **Обязательно?** необязательно **Тип:** Логический (сценарий groovy)
before-mapping (target_ci_type)	Сценарий groovy, выполняемый перед сопоставлением целевого типа ЭК.	
after-mapping (target_ci_type)	Сценарий groovy, выполняемый после сопоставления целевого типа ЭК.	
for-each-source-ci (target_ci_type)	Определяет итерацию определенных ЭК корневого экземпляра.	**Имя:** count-index **Описание:** Индекс текущего ЭК

Имя элемента и путь	Описание	Атрибуты
		Обязательно? Обязательно **Тип:** Строка
		Имя: var-name **Описание:** переменная, ссылающаяся на текущий ЭК **Обязательно?** необязательно **Тип:** Строка
		Имя: source-cis **Описание:** Имя ЭК в запросе, который будет подвергнут итерации **Обязательно?** Обязательно **Тип:** String (groovy script)

Использование API-интерфейсов

Глава 8

Введение в API-интерфейсы

Данная глава включает:

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). Подробнее см. в разделе "API-интерфейс HP Universal CMDB" на странице 225.
- **API-интерфейс веб-службы UCMDB.** Обеспечивает запись определений элементов конфигурации и топографических связей в UCMDB (Universal Configuration Management Database) и запрос этой информации с помощью TQL-запросов и специальных запросов. Дополнительные сведения см. в разделе " API-интерфейс веб-службы HP Universal CMDB" на странице 233.
- **API веб-службы управления потоком данных** Обеспечивает управление зондами, заданиями, триггерами и учетными данными для Управления потоком данных. Подробнее см. в разделе "API-интерфейс Управления потоком данных" на странице 290.

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 9

API-интерфейс HP Universal CMDB

Данная глава включает:

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.

См. полную документацию по доступным API-интерфейсам в разделе HP UCMDB API Reference.

Эти файлы находятся по следующему пути:

\\<Корневая директория UCMDB>\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 "Язык запросов топологии" на странице 1 (*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 "Сценарии использования" на странице 230.

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.

> **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 228.

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
getClassModelService	Информация о типах ЭК и связей
getConfigurationService	Управление настройками инфраструктуры, для конфигурации сервера
getDDMConfigurationService	Настройка системы Управления потоком данных
getDDMManagementService	Анализ и отображение хода выполнения, результатов и ошибок системы Управления потоком данных
getHistoryService	Информация об истории отслеживаемых ЭК (изменение, удаление и т.д.)
getImpactAnalysisService	Выполнение сценария анализа влияния (другое название: **корреляция**).
getQueryManagementService	Управление доступом к запросам — сохранение, удаление и вывод существующих. Также предоставляет проверку запросов и обнаружение зависимостей запросов.
getResourceBundleManagementService	Добавление тегов к ресурсам (объединение служб в пакеты). Обеспечивает явное создание новых тегов и их удаление из всех отмеченных ресурсов.
getStateService	Службы для управления состояниями (список,

Методы служб	Использование
	добавление, удаление и т.д.)
getSoftwareSignatureService	Указывает элементы ПО для обнаружения системой обнаружения и управления зависимостями
getSnapshotService	Службы для управления снимками (получение, сохранение, сравнение и т.д.)
getTopologyQueryService	Получение информации об IT Universe
getTopologyUpdateService	Изменение информации в IT Universe
getViewService	Служба выполнения представлений (выполнение определения, выполнение сохраненных) и служба управления (сохранение, удаление, вывод существующих). Также предоставляет проверку представлений и обнаружение зависимостей.
getViewArchiveService	Службы архивирования результатов представления. Обеспечивает сохранение текущего результата представления и возвращение ранее сохраненных результатов.
SystemHealthService	Службы контроля работоспособности системы (базовые индикаторы производительности, метрики емкости и доступности)

Клиент обменивается данными с сервером по протоколу HTTP.

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 делать это не рекомендуется. Чтобы подключиться от имени пользователя UCMDB, необходимо дать этому пользователю право аутентификации через API.

Чтобы создать пользователя интеграции, выполните следующие действия:

1. Запустите веб-браузер и введите адрес сервера:

   `http://localhost:8080/jmx-console`.

   Возможно, потребуется ввести имя пользователя и пароль (значения по умолчанию: sysadmin/sysadmin).

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.

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 кнопку **Invoke**.

   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. Во 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.

    Подробнее 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 следующий метод: В JMX Console найдите службу **UCMDB-UI:name=UCMDB Integration**. Выполните метод **setCMDBSuperIntegrationUser** с именем пользователя и новым паролем пользователя интеграции.

Справка по API-интерфейсу HP Universal CMDB

Эти файлы находятся по следующему пути:

\\<UCMDB root directory>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\UCMDB_JavaAPI\index.html

Сценарии использования

В следующих сценариях использования предполагается наличие двух 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" ниже
- "Анализ влияния изменений " ниже

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 следующей директории:

\\<UCMDB root directory>\hp\UCMDB\UCMDBServer\deploy\ucmdb-docs\docs\eng\APIs\JavaSDK_Samples

Глава 10

API-интерфейс веб-службы HP Universal CMDB

Данная глава включает:

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 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" на странице 1 (*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

API-интерфейс 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.

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**

Команда API-интерфейса веб-службы	Необходимые права доступа
	Для каждого представления: Право доступа **Просмотр**
getChangedCIs	Общее действие: **Просмотр истории**, **Просмотр ЭК**
calculateImpact getImpactPath getImpactRulesByGroupName getImpactRulesByNamePrefix	Общее действие: **Выполнение анализа влияния**
getAllClassesHierarchy getClassAncestors getCmdbClassDefinition	Нет

Справка по API-интерфейсу веб-службы HP Universal CMDB

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

Вызов веб-службы

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 сообщение 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 "Сценарии использования" на странице 263.

Запрос в CMDB

Для выполнения запросов в CMDB используются API-интерфейсы, описанные в разделе "Методы запросов UCMDB" на странице 247.

Запросы и возвращенные элементы CMDB всегда содержат реальные идентификаторы UCMDB.

См. примеры использования одного из методов запросов в документе "Пример запроса" на странице 266.

Этот раздел охватывает следующие темы:

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.

См. дополнительные сведения об обработке данных, разделенных на блоки, в разделах "pullTopologyMapChunks" на странице 255 и "releaseChunks" на странице 257.

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 257. См. примеры использования методов обновления в разделе "Пример обновления" на странице 278.

Эта задача включает следующие шаги:

- "Обновление UCMDB" выши
- "Использование типов идентификаторов с методами обновления" ниже

Параметры обновления UCMDB

В этом разделе описываются параметры, используемые только методами обновления службы. Дополнительные сведения см в документе schema documentation..

- **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 указать тип 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. Подробнее см. в разделе referencedCIs ("CIsAndRelationsUpdates" на предыдущей странице).

- **Вставка новых ЭК 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. Подробнее см. в разделах "addCIsAndRelations" на странице 257 и "Методы запросов UCMDB" на странице 247.

- **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" на странице 1 (*Руководство по моделированию в HP Universal CMDB*).

См. примеры использования методов модели классов в документе "Пример модели классов" на странице 282.

В этом разделе представлены сведения о следующих методах, возвращающих сведения о типах ЭК и связях:

- "getClassAncestors" ниже
- "getAllClassesHierarchy" на следующей странице
- "getCmdbClassDefinition" на следующей странице

getClassAncestors

Метод `getClassAncestors` возвращает путь между указанным типом ЭК и его корнем, включая корень.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на следующей странице.
className	Имя типа. Подробнее см. в разделе "Имя типа" на странице 245.

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 следующей странице.

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` определяет как ключевые 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.

Параметр	Комментарий
className	Имя типа. Подробнее см. в разделе "Общие параметры UCMDB " на следующей странице.

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.

См. примеры использования методов анализа влияния в документе "Пример анализа влияния" на странице 284.

Общие параметры UCMDB

В этом разделе описываются распространенные параметры методов службы. Подробнее см. в разделе schema documentation..

Этот раздел охватывает следующие темы:

- "CmdbContext" ниже
- "ID" ниже
- "Ключевые атрибуты" ниже
- "Типы идентификаторов" на следующей странице
- "CIProperties" на следующей странице
- "Имя типа" на странице 245
- "Элемент конфигурации (ЭК)" на странице 245
- "Связь" на странице 245

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"" на странице 1 (Руководство по моделированию в HP Universal CMDB). См. дополнительные сведения об идентификации ключевых атрибутов в API-интерфейсе клиентского приложения в разделе "getCmdbClassDefinition" на странице 242.

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 CMDBможет 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 значение true для флага `temp`. При удалении 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" на странице 1 (Руководство по моделированию в HP Universal CMDB).

Элемент `props` — это коллекция CIProperties. Подробнее см. в разделе "Общие параметры UCMDB " на странице 243.

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 запроса.

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 241.

Подробнее см. в разделе "Диспетчер типов ЭК" на странице 1 (*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. Подробнее см. в разделе "CIProperties" на предыдущей странице.

Выходные параметры UCMDB

В этом разделе описываются распространенные выходные параметры методов службы. Подробнее см. в разделе schema documentation..

Этот раздел охватывает следующие темы:

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" выши).
- `relationNodes`. Это один или несколько `RelationNode` (см. раздел "RelationNode" выши).

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.

Подробнее см. в разделе "Обработка крупных ответов" на странице 237.

Методы запросов UCMDB

Данный раздел содержит сведения о следующих методах:

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 243.
queryName	Имя параметризованного TQL-запроса в CMDB, по которому извлекается карта.

Параметр	Комментарий
parameterizedNodes	Условия, которым каждый узел должен соответствовать для включения в результаты поиска.
queryTypedProperties	Коллекция наборов свойств для получения в ЭК определенного типа.

Исходящие параметры

Параметр	Комментарий
topologyMap	Подробнее см. в разделе "TopologyMap" на странице 246.
chunkInfo	См. дополнительные сведения в разделах: "ChunkInfo" на предыдущей странице и "Обработка крупных ответов" на странице 237.

executeTopologyQueryWithParameters

Метод `executeTopologyQueryWithParameters` получает элемент `topologyMap`, соответствующий параметризованному запросу.

Запрос передается в аргументе `queryXML`. Значения параметров запросов передаются в аргументе `parameterizedNodes`. 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. Значение по умолчанию – **sysadmin**/**sysadmin**.

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 243.
queryXML	Строка XML, представляющая TQL-запрос без тегов ресурсов.
parameterizedNodes	Условия, которым каждый узел должен соответствовать для включения в результаты поиска.

Исходящие параметры

Параметр	Комментарий
topologyMap	Подробнее см. в разделе "TopologyMap" на странице 246.
chunkInfo	См. дополнительные сведения в разделах "ChunkInfo" на странице 247 и "Обработка крупных ответов" на странице 237.

getChangedCIs

Метод `getChangedCIs` возвращает данные об изменении для всех ЭК, связанных с указанными ЭК.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
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
changeDataInfo	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 243.
ID	Идентификатор ЭК, для которого нужно получить соседей. Это должен быть настоящий идентификатор CMDB.
neighbourType	Имя типа ЭК соседей для извлечения. Будут возвращены соседи указанного типа и всех типов, производных от него. Подробнее см. в разделе "Имя типа" на странице 245.
CIProperties	Данные, которые будут возвращены для каждого ЭК, вызвавшего макет запроса в интерфейсе пользователя. Подробнее см. в разделе "TypedProperties" на странице 240.
relationProperties	Данные, которые будут возвращены для каждой связи (в интерфейсе пользователя это называется схемой запроса). Подробнее см. в разделе "TypedProperties" на странице 240

Исходящие параметры

Параметр	Комментарий
topology	Подробнее см. в разделе "Topology" на странице 246.
comments	Только для внутреннего использования.

getClsByID

Метод `getCIsByID` извлекает ЭК по идентификаторам в CMDB.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
ClsTypedProperties	Коллекция свойств с определенным типом. Подробнее см. в разделе "Другие элементы спецификации свойств" на странице 239.
IDs	В этой коллекции допускаются только реальные идентификаторы CMDB. .

Исходящие параметры

Параметр	Комментарий
CIs	Коллекция ЭК.
chunkInfo	См. дополнительные сведения в разделах: "ChunkInfo" на странице 247 и "Обработка крупных ответов" на странице 237.

getClsByType

Метод `getCIsByType` 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 243.
type	Имя класса. Подробнее см. в разделе "Имя типа" на странице 245.
свойства	Данные, возвращаемые для каждого ЭК. Подробнее см. в разделе "CustomProperties" на странице 239.

Исходящие параметры

Параметр	Комментарий
CIs	Коллекция ЭК.
chunkInfo	См. дополнительные сведения в разделах: "ChunkInfo" на странице 247 и "Обработка крупных ответов" на странице 237.

getFilteredClsByType

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. 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`.

Оператор	Тип условия/Комментарии
	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(11);
fc.setCondition(fp);
fc.setFloatOperator(FloatCondition.floatOperatorEnum.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.addPropertyName("name");
propertiesList.addPropertyName("size");
customProperties.setPropertiesList(propertiesList);
request.setProperties(customProperties)
```

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
type	Имя класса. Подробнее см. в разделе "Имя типа" на странице 245. Может использоваться любой тип, указанный с помощью диспетчера типов ЭК. Подробнее см. в разделе "Диспетчер типов ЭК" на странице 1 (*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 239.
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 247 и "Обработка крупных ответов" на странице 237.

getQueryNameOfView

Метод `getQueryNameOfView` извлекает имя TQL-запроса, на котором основывается указанное представление.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
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 243.
queryName	Имя TQL-запроса.
queryTypedProperties	Коллекция наборов свойств для извлечения в элементах определенного типа ЭК.

Исходящие параметры

Параметр	Комментарий
queryName	Имя TQL-запроса в CMDB, на котором основывается представление.

getTopologyQueryResultCountByName

Метод `getTopologyQueryResultCountByName` извлекает количество экземпляров каждого узла, соответствующего указанному запросу.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
queryName	Имя TQL-запроса.
countInvisible	Если выбрано значение true, выходные данные включают ЭК, указанные как невидимые в запросе.

Исходящие параметры

Параметр	Комментарий
queryName	Имя TQL-запроса в CMDB, на котором основывается представление.

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 247 и "Запрос в CMDB" на странице 237.

Клиентское приложение должно поддерживать частичные карты. См. пример обработки коллекции ЭК и объединений и пример объединения блоков в карту в разделе "Пример запроса" на странице 266.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
ChunkRequest	Номер блока для извлечения и элемент `ChunkInfo`, возвращенный методом запроса.

Исходящие параметры

Параметр	Комментарий
topologyMap	Подробнее см. в разделе "TopologyMap" на странице 246.
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++) {
                // your code to process the CIs             }
        } }

GetCIsByType request =
     new GetCIsByType(cmdbContext, typeName, customProperties);
GetCIsByTypeResponse response =
     ucmdbService.getCIsByType(request); ChunkRequest chunkRequest
```

```
= new ChunkRequest(); chunkRequest.setChunkInfo
(response.getChunkInfo()); for(int j=1 ; j <= response.getChunkInfo
().getNumberOfChunks() ; j++) {     chunkRequest.setChunkNumber(j);
    PullTopologyMapChunks req = new PullTopologyMapChunks
(cmdbContext,     chunkRequest);     PullTopologyMapChunksResponse
res =
        ucmdbService.pullTopologyMapChunks(req);         for(int
m=0 ;
            m < res.getTopologyMap().getCINodes().sizeCINodeList()
;
            m++) {             CIs cis =
            res.getTopologyMap().getCINodes().getCINode(m).getCIs
();             for(int i=0 ; i < cis.sizeCIList() ; i++) {
                // your code to process the CIs             }
        } }
```

releaseChunks

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 243.
chunksKey	Идентификатор данных, разделенных на блоки, на сервере. Ключ — элемент `ChunkInfo`.

Методы обновления UCMDB

Данный раздел содержит сведения о следующих методах:

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 243.
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. Подробнее см. в разделе "CIsAndRelationsUpdates" на странице 240.
ignoreValidation	При использовании значения true проверка перед обновлением CMDB не выполняется.

Исходящие параметры

Параметр	Комментарий
CreatedIDsMap	Сопоставление идентификаторов клиентов и идентификаторов 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 243.
CIsAndRelationsUpdates	Элементы для удаления. Подробнее см. в разделе "CIsAndRelationsUpdates" на странице 240.

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 243.
CIsAndRelationsUpdates	Элементы для обновления. Подробнее см. в разделе "CIsAndRelationsUpdates" на странице 240.

Параметр	Комментарий
ignoreValidation	При использовании значения true проверка перед обновлением CMDB не выполняется.

Исходящие параметры

Параметр	Комментарий
CreatedIDsMap	Сопоставление идентификаторов клиентов и идентификаторов CMDB. Подробнее см. в разделе "addCIsAndRelations" на странице 257.

Методы анализа влияния в UCMDB

Данный раздел содержит сведения о следующих методах:

- "calculateImpact" ниже
- "getImpactPath" на следующей странице
- "getImpactRulesByNamePrefix" на следующей странице

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 243.
impactCategory	Тип события, которое инициирует моделирование правила.
IDs	Коллекция идентификаторов.
impactRulesNames	Коллекция элементов `ImpactRuleName`.
severity	Серьезность инициирующего события.

Исходящие параметры

Параметр	Комментарий
impactTopology	Подробнее см. в разделе "Topology" на странице 246.
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 243.
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	Подробнее см. в разделе "CmdbContext" на странице 243.
ruleNamePrefixFilter	Строка, содержащая первые буквы имен правил для вывода.

Исходящие параметры

Параметр	Комментарий
impactRules	`impactRules` включает ноль или более элементов `impactRule`. Элемент

Параметр	Комментарий
	`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**.

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" ниже
- "Анализ влияния изменений " ниже

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

Этот раздел охватывает следующие темы:

Пример базового класса

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.services.UcmdbService;
import com.hp.ucmdb.generated.services.UcmdbServiceStub;
import com.hp.ucmdb.generated.types.CmdbContext;
import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;

import org.apache.axis2.transport.http.HttpTransportProperties;
import java.net.MalformedURLException;
import java.net.URL;

/**
* User: hbarkai
* Date: Jul 12, 2007
*/
abstract class Demo {

UcmdbService stub;
CmdbContext context;

public void initDemo() {
    try {
        setStub(createUcmdbService("admin", "admin"));
        setContext();
    } catch (Exception e) {
        //handle exception
    }
}

public UcmdbService getStub() {
    return stub;
}

public void setStub(UcmdbService stub) {
    this.stub = stub;
}
```

```
public CmdbContext getContext() {
    return context;
}

public void setContext() {
    CmdbContext context = new CmdbContext();
    context.setCallerApplication("demo");
    this.context = context;
}

//connection to service  - for axis2/jibx client
private static final String PROTOCOL = "http";
private static final String HOST_NAME = "host_name";
private static final int PORT = 8080;
private static final String FILE = "/axis2/services/UcmdbService";
protected UcmdbService createUcmdbService
    (String username, String password) throws Exception{
        URL url;
        UcmdbServiceStub serviceStub;

        try {
             url = new URL
                (Demo.PROTOCOL, Demo.HOST_NAME,
                Demo.PORT, Demo.FILE);
             serviceStub = new UcmdbServiceStub(url.toString());
             HttpTransportProperties.Authenticator auth =
                new HttpTransportProperties.Authenticator();
             auth.setUsername(username);
             auth.setPassword(password);
serviceStub._getServiceClient().getOptions().setProperty
    (HTTPConstants.AUTHENTICATE,auth);

    } catch (AxisFault axisFault) {
            throw new Exception
            ("Failed to create SOAP adapter for "
                + Demo.HOST_NAME , axisFault);

        } catch (MalformedURLException e) {
            throw new Exception
                ("Failed to create SOAP adapter for "
                    + Demo.HOST_NAME, e);
        }
        return serviceStub;
 }
}
```

Пример запроса

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.query.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.services.UcmdbService;
```

```
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.props.*;
import java.rmi.RemoteException;

public class QueryDemo extends Demo{
    UcmdbService stub;
    CmdbContext context;


    public void getCIsByTypeDemo() {
        GetCIsByType request = new GetCIsByType();
        //set cmdbcontext
        CmdbContext cmdbContext = getContext();
        request.setCmdbContext(cmdbContext);
        //set CIs type
        request.setType("anyType");
        //set CIs propeties to be retrieved
        CustomProperties customProperties = new CustomProperties();
        PredefinedProperties predefinedProperties =
            new PredefinedProperties();
        SimplePredefinedProperty simplePredefinedProperty =
            new SimplePredefinedProperty();
        simplePredefinedProperty.setName
            (SimplePredefinedProperty.nameEnum.DERIVED);
        SimplePredefinedPropertyCollection
            simplePredefinedPropertyCollection =
                new SimplePredefinedPropertyCollection();

        simplePredefinedPropertyCollection.addSimplePredefinedProperty
            (simplePredefinedProperty);
        predefinedProperties.setSimplePredefinedProperties
            (simplePredefinedPropertyCollection);
        customProperties.setPredefinedProperties
(predefinedProperties);
        request.setProperties(customProperties);
        try {
            GetCIsByTypeResponse response =
                getStub().getCIsByType(request);
            TopologyMap map =
                getTopologyMapResultFromCIs
                    (response.getCIs(), response.getChunkInfo());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }


    public void getCIsByIdDemo() {
        GetCIsById request = new GetCIsById();
        CmdbContext cmdbContext = getContext();
```

```
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set ids
        ID id1 = new ID();
        id1.setBase("cmdbobjectidCIT1");
        ID id2 = new ID();
        id2.setBase("cmdbobjectidCIT2");
        IDs ids = new IDs();
        ids.addID(id1);
        ids.addID(id2);
        request.setIDs(ids);
        //set CIs properties to be retrieved
        TypedPropertiesCollection properties =
            new TypedPropertiesCollection();

    TypedProperties typedProperties1 =
            new TypedProperties();
        typedProperties1.setType("CIT1");

        CustomTypedProperties customProperties1 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties1 =
            new PredefinedTypedProperties();
        SimpleTypedPredefinedProperty simplePredefinedProperty1 =
            new SimpleTypedPredefinedProperty();
        simplePredefinedProperty1.setName
            (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
        SimpleTypedPredefinedPropertyCollection
            simplePredefinedPropertyCollection1 =
                new SimpleTypedPredefinedPropertyCollection();
        simplePredefinedPropertyCollection1
            .addSimpleTypedPredefinedProperty
                (simplePredefinedProperty1);

predefinedProperties1.
        setSimpleTypedPredefinedProperties
            (simplePredefinedPropertyCollection1);
        customProperties1.
            setPredefinedTypedProperties
                (predefinedProperties1);
        typedProperties1.setProperties(customProperties1);
        properties.addTypedProperties(typedProperties1);

        TypedProperties typedProperties2 =
            new TypedProperties();
        typedProperties2.setType("CIT2");
        CustomTypedProperties customProperties2 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties2 =
            new PredefinedTypedProperties();
        SimpleTypedPredefinedProperty simplePredefinedProperty2 =
            new SimpleTypedPredefinedProperty();
```

```
        simplePredefinedProperty2.setName
            (SimpleTypedPredefinedProperty.nameEnum.NAMING);
        SimpleTypedPredefinedPropertyCollection
            simplePredefinedPropertyCollection2 =
                new SimpleTypedPredefinedPropertyCollection();

        simplePredefinedPropertyCollection2.
            addSimpleTypedPredefinedProperty
                (simplePredefinedProperty2);

        predefinedProperties2.setSimpleTypedPredefinedProperties
            (simplePredefinedPropertyCollection2);
        customProperties2.setPredefinedTypedProperties
            (predefinedProperties2);
        typedProperties2.setProperties(customProperties2);
        properties.addTypedProperties(typedProperties2);

    request.setCIsTypedProperties(properties);
        try {
            GetCIsByIdResponse response =
                getStub().getCIsById(request);
            CIs cis = response.getCIs();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }


    public void getFilteredCIsByTypeDemo() {
        GetFilteredCIsByType request = new GetFilteredCIsByType();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set CIs type
        request.setType("anyType");
        //sets Filter conditions
        Conditions conditions = new Conditions();
        IntConditions intConditions = new IntConditions();
        IntCondition intCondition = new IntCondition();
        IntProp intProp = new IntProp();
        intProp.setName("int_attr1");

        intProp.setValue(100);
        intCondition.setCondition(intProp);
        intCondition.setIntOperator
            (IntCondition.intOperatorEnum.Greater);
        intConditions.addIntCondition(intCondition);

        conditions.setIntConditions(intConditions);
        request.setConditions(conditions);
        //set logical operator for conditions
```

```
        request.setConditionsLogicalOperator
            (GetFilteredCIsByType.conditionsLogicalOperatorEnum.AND);
        //set CIs properties to be retrieved
        CustomProperties customProperties =
            new CustomProperties();
        PredefinedProperties predefinedProperties =
            new PredefinedProperties();
        SimplePredefinedProperty simplePredefinedProperty =
            new SimplePredefinedProperty();
        simplePredefinedProperty.setName
            (SimplePredefinedProperty.nameEnum.NAMING);

        SimplePredefinedPropertyCollection
            simplePredefinedPropertyCollection =
                new SimplePredefinedPropertyCollection();
        simplePredefinedPropertyCollection.
            addSimplePredefinedProperty
                (simplePredefinedProperty);
        predefinedProperties.setSimplePredefinedProperties
            (simplePredefinedPropertyCollection);
        customProperties.setPredefinedProperties
            (predefinedProperties);

        request.setProperties(customProperties);
        try {
            GetFilteredCIsByTypeResponse response =
                getStub().getFilteredCIsByType(request);
            TopologyMap map =
                getTopologyMapResultFromCIs
                    (response.getCIs(), response.getChunkInfo());

        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }


    public void executeTopologyQueryByNameDemo() {
        ExecuteTopologyQueryByName request = new
ExecuteTopologyQueryByName();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set query name
        request.setQueryName("queryName");

        try {
            ExecuteTopologyQueryByNameResponse response =
                getStub().executeTopologyQueryByName(request);
            TopologyMap map =
```

```
                getTopologyMapResult
                    (response.getTopologyMap(), response.getChunkInfo
());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }

    // assume the follow query was defined at UCMDB
    // Query Name: exampleQuery
    // Query sketch:
    //                                Host
    //                                /  \
    //                              ip    Disk
    // Query Parameters:
    //      Host-
    //          host_os (like)
    //      Disk-
    //          disk_failures (equal)
    public void executeTopologyQueryByNameWithParametersDemo() {
        ExecuteTopologyQueryByNameWithParameters request =
            new ExecuteTopologyQueryByNameWithParameters();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set query name
        request.setQueryName("queryName");
        //set parameters
        ParameterizedNode hostParametrizedNode =
            new ParameterizedNode();
        hostParametrizedNode.setNodeLabel("Host");
        CIProperties parameters = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp = new StrProp();
        strProp.setName("host_os");
        strProp.setValue("%2000%");
        strProps.addStrProp(strProp);
        parameters.setStrProps(strProps);
        hostParametrizedNode.setParameters(parameters);
        request.addParameterizedNodes(hostParametrizedNode);
        ParameterizedNode diskParametrizedNode =
            new ParameterizedNode();

        diskParametrizedNode.setNodeLabel("Disk");
        CIProperties parameters1 = new CIProperties();
        IntProps intProps = new IntProps();
```

```
    IntProp intProp = new IntProp();
        intProp.setName("disk_failures");
        intProp.setValue(30);
        intProps.addIntProp(intProp);
        parameters1.setIntProps(intProps);
        diskParametrizedNode.setParameters(parameters1);

        request.addParameterizedNodes(diskParametrizedNode);
        try {
            ExecuteTopologyQueryByNameWithParametersResponse
                response =
                getStub().executeTopologyQueryByNameWithParameters
                    (request);
            TopologyMap map =
                getTopologyMapResult
                    (response.getTopologyMap(), response.getChunkInfo
());
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }

/    // assume the follow query was defined at UCMDB
    // Query Name: exampleQuery
    // Query sketch:
    //                                Host
    //                                /  \
    //                              ip    Disk
    // Query Parameters:
    //      Host-
    //          host_os (like)
    //      Disk-
    //          disk_failures (equal)


    public void executeTopologyQueryWithParametersDemo() {
        ExecuteTopologyQueryWithParameters request =
            new ExecuteTopologyQueryWithParameters();
        CmdbContext cmdbContext = getContext();
        //set cmdbcontext
        request.setCmdbContext(cmdbContext);
        //set query definition
        String queryXml = "<xml that represents the query above>";
        request.setQueryXml(queryXml);
        //set parameters
        ParameterizedNode hostParametrizedNode =
            new ParameterizedNode();
```

```
    hostParametrizedNode.setNodeLabel("Host");
        CIProperties parameters = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp = new StrProp();
        strProp.setName("host_os");
        strProp.setValue("%2000%");
        strProps.addStrProp(strProp);
        parameters.setStrProps(strProps);
        hostParametrizedNode.setParameters(parameters);
        request.addParameterizedNodes(hostParametrizedNode);
        ParameterizedNode diskParametrizedNode =
            new ParameterizedNode();
        diskParametrizedNode.setNodeLabel("Disk");
        CIProperties parameters1 = new CIProperties();
        IntProps intProps = new IntProps();
        IntProp intProp = new IntProp();
        intProp.setName("disk_failures");
        intProp.setValue(30);
        intProps.addIntProp(intProp);
        parameters1.setIntProps(intProps);
        diskParametrizedNode.setParameters(parameters1);
        request.addParameterizedNodes(diskParametrizedNode);

        try {
            ExecuteTopologyQueryWithParametersResponse
            response = getStub().executeTopologyQueryWithParameters
                (request);
            TopologyMap map =
                getTopologyMapResult
                    (response.getTopologyMap(), response.getChunkInfo
()));

         } catch (RemoteException e) {
             //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }


    public void getCINeighboursDemo() {
        GetCINeighbours request = new GetCINeighbours();
        //set cmdbcontext
        CmdbContext cmdbContext = getContext();
        request.setCmdbContext(cmdbContext);
        // set CI id
        ID id = new ID();
        id.setBase("cmdbobjectidCIT1");
        request.setID(id);
        //set neighbour type
        request.setNeighbourType("neighbourType");
```

```
        //set Neighbours CIs propeties to be retrieved
        TypedPropertiesCollection properties =
            new TypedPropertiesCollection();
        TypedProperties typedProperties1 = new TypedProperties();
        typedProperties1.setType("neighbourType");
        CustomTypedProperties customProperties1 =
            new CustomTypedProperties();
        PredefinedTypedProperties predefinedProperties1 =
            new PredefinedTypedProperties();

        QualifierProperties qualifierProperties =
            new QualifierProperties();
        qualifierProperties.addQualifierName("ID_ATTRIBUTE");
        predefinedProperties1.setQualifierProperties
(qualifierProperties);
        customProperties1.setPredefinedTypedProperties
            (predefinedProperties1);
        typedProperties1.setProperties(customProperties1);
        properties.addTypedProperties(typedProperties1);
        request.setCIProperties(properties);

        TypedPropertiesCollection relationsProperties =
            new TypedPropertiesCollection();
        TypedProperties typedProperties2 = new TypedProperties();
        typedProperties2.setType("relationType");
        CustomTypedProperties customProperties2 =
            new CustomTypedProperties();

        PredefinedTypedProperties predefinedProperties2 =
            new PredefinedTypedProperties();
        SimpleTypedPredefinedProperty simplePredefinedProperty2 =
            new SimpleTypedPredefinedProperty();
        simplePredefinedProperty2.setName

            (SimpleTypedPredefinedProperty.nameEnum.CONCRETE);
        SimpleTypedPredefinedPropertyCollection
            simplePredefinedPropertyCollection2 =
                new SimpleTypedPredefinedPropertyCollection();
        simplePredefinedPropertyCollection2.
            addSimpleTypedPredefinedProperty
                (simplePredefinedProperty2);
        predefinedProperties2.
            setSimpleTypedPredefinedProperties
                (simplePredefinedPropertyCollection2);
        customProperties2.setPredefinedTypedProperties
            (predefinedProperties2);
        typedProperties2.setProperties(customProperties2);
        relationsProperties.addTypedProperties(typedProperties2);
        request.setRelationProperties(relationsProperties);

        try {
            GetCINeighboursResponse response =
```

```
                getStub().getCINeighbours(request);
            Topology topology = response.getTopology();
        } catch (RemoteException e) {
            //handle exception
         } catch (UcmdbFaultException e) {
             //handle exception
         }
    }

//get Topology Map for chunked/non-chunked result
    private TopologyMap getTopologyMapResult(TopologyMap topologyMap,
ChunkInfo chunkInfo) {
        if(chunkInfo.getNumberOfChunks() == 0) {
            return topologyMap;
        } else {

            topologyMap = new TopologyMap();
            for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
                ChunkRequest chunkRequest = new ChunkRequest();
                chunkRequest.setChunkInfo(chunkInfo);
                chunkRequest.setChunkNumber(i);
                PullTopologyMapChunks req =
                    new PullTopologyMapChunks();
                req.setChunkRequest(chunkRequest);
                req.setCmdbContext(getContext());
                PullTopologyMapChunksResponse res = null;

                try  {
                    res = getStub().pullTopologyMapChunks(req);
                    TopologyMap map = res.getTopologyMap();
                    topologyMap = mergeMaps(topologyMap, map);
                } catch (RemoteException e) {
                    //handle exception
                } catch (UcmdbFaultException e) {
                    //handle exception
                }
            }
        }
        return topologyMap;
    }


    private TopologyMap getTopologyMapResultFromCIs(CIs cis, ChunkInfo
chunkInfo) {
        TopologyMap topologyMap = new TopologyMap();
        if(chunkInfo.getNumberOfChunks() == 0) {
            CINode ciNode = new CINode();
            ciNode.setLabel("");
            ciNode.setCIs(cis);
            CINodes ciNodes = new CINodes();
            ciNodes.addCINode(ciNode);
```

```
            topologyMap.setCINodes(ciNodes);
        } else {

            for(int i=1 ; i <= chunkInfo.getNumberOfChunks() ; i++) {
                ChunkRequest chunkRequest =
                    new ChunkRequest();
                chunkRequest.setChunkInfo(chunkInfo);
                chunkRequest.setChunkNumber(i);
                PullTopologyMapChunks req =
                    new PullTopologyMapChunks();
                req.setChunkRequest(chunkRequest);
                req.setCmdbContext(getContext());
                PullTopologyMapChunksResponse res = null;

                try  {
                    res = getStub().pullTopologyMapChunks(req);
                } catch (RemoteException e) {
                    //handle exception
                } catch (UcmdbFaultException e) {
                    //handle exception
                }
                TopologyMap map = res.getTopologyMap();
                topologyMap = mergeMaps(topologyMap, map);
            }

            //release chunks
            ReleaseChunks req = new ReleaseChunks();
            req.setChunksKey(chunkInfo.getChunksKey());
            req.setCmdbContext(getContext());

            try {
                getStub().releaseChunks(req);
            } catch (RemoteException e) {
                //handle exception
            } catch (UcmdbFaultException e) {
                //handle exception
            }
        }
        return topologyMap;
    }

//=================================================
/*  WARNING merge will be correct only if a each node is given
    a unique name. This applies to both CI and Relation nodes .*/
//=================================================
    private TopologyMap mergeMaps(TopologyMap topologyMap, TopologyMap
newMap) {
        for(int i=0 ; i < newMap.getCINodes().sizeCINodeList() ; i++ )
{
            CINode ciNode = newMap.getCINodes().getCINode(i);
            boolean alreadyExist = false;
            if(topologyMap.getCINodes() == null) {
```

```
                topologyMap.setCINodes(new CINodes());
            }


            for(int j=0 ; j < topologyMap.getCINodes().sizeCINodeList
() ; j++) {
                CINode ciNode2 = topologyMap.getCINodes().getCINode
(j);
                if(ciNode2.getLabel().equals(ciNode.getLabel())){
                    CIs cisTOAdd = ciNode.getCIs();
                    CIs cis =
                        mergeCIsGroups
                        (topologyMap.getCINodes().getCINode(j).getCIs
(),
                            cisTOAdd);
                    topologyMap.getCINodes().getCINode(j).setCIs(cis);
                    alreadyExist = true;
                }
            }
            if(!alreadyExist) {
                topologyMap.getCINodes().addCINode(ciNode);
            }
        }
        for(int i=0 ; i < newMap.getRelationNodes
().sizeRelationNodeList() ; i++ ) {
            RelationNode relationNode =
                newMap.getRelationNodes().getRelationNode(i);
            boolean alreadyExist = false;
            if(topologyMap.getRelationNodes() == null) {
                topologyMap.setRelationNodes(new RelationNodes());
            }
            for(int j=0 ;
                    j < topologyMap.getRelationNodes
().sizeRelationNodeList() ;
                    j++) {
                RelationNode relationNode2 =
                    topologyMap.getRelationNodes().getRelationNode(j);
                if(relationNode2.getLabel().equals
(relationNode.getLabel())){
                    Relations relationsTOAdd =
relationNode.getRelations();
                    Relations relations =
                        mergeRelationsGroups
                        (topologyMap.getRelationNodes().
                            getRelationNode(j).getRelations(),
                            relationsTOAdd);
                    topologyMap.getRelationNodes().
                        getRelationNode(j).setRelations(relations);
                    alreadyExist = true;
```

```
                }
            }

            if(!alreadyExist) {
                topologyMap.getRelationNodes().addRelationNode
(relationNode);
            }
        }
        return topologyMap;
    }

    private Relations mergeRelationsGroups(Relations relations1,
Relations relations2) {
        for(int i=0 ; i < relations2.sizeRelationList() ; i++) {
            relations1.addRelation(relations2.getRelation(i));
        }
        return relations1;
    }

    private CIs mergeCIsGroups(CIs cis1, CIs cis2) {
        for(int i=0 ; i < cis2.sizeCIList() ; i++) {
            cis1.addCI(cis2.getCI(i));
        }
        return cis1;
    }
}
```

Пример обновления

```
import com.hp.ucmdb.generated.params.update.AddCIsAndRelations;

import
com.hp.ucmdb.generated.params.update.AddCIsAndRelationsResponse;

import com.hp.ucmdb.generated.params.update.UpdateCIsAndRelations;

import com.hp.ucmdb.generated.params.update.DeleteCIsAndRelations;

import com.hp.ucmdb.generated.services.UcmdbFault;

import com.hp.ucmdb.generated.types.*;

import com.hp.ucmdb.generated.types.update.CIsAndRelationsUpdates;

import com.hp.ucmdb.generated.types.update.ClientIDToCmdbID;

import java.rmi.RemoteException;

import java.util.ArrayList;

import java.util.List;

public class UpdateDemo extends Demo{

   public void getAddCIsAndRelationsDemo() {

      AddCIsAndRelations request = new AddCIsAndRelations();

      request.setCmdbContext(getContext());
```

```
        request.setUpdateExisting(true);
        CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        List<CI> listCI = new ArrayList<CI>();
        CI ci = new CI();
        ID id = new ID();
        id.setString("temp1");
        id.setTemp(true);
        ci.setID(id);
        ci.setType("host");
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp = new StrProp();
        strProp.setName("host_key");
        String value = "blabla";
        strProp.setValue(value);
        strProps.getStrProps().add(strProp);
        props.setStrProps(strProps);
        ci.setProps(props);
        listCI.add(ci);
        cis.setCIs(listCI);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
        try {
          AddCIsAndRelationsResponse response  = getStub().addCIsAndRelations
(request);
          for(int i = 0  ;  i < response.getCreatedIDsMaps().size() ; i++) {
             ClientIDToCmdbID idsMap = response.getCreatedIDsMaps().get(i);
             //do something
          }
        } catch (RemoteException e) {
          //handle exception
        } catch (UcmdbFault e) {
          //handle exception
```

```
    }
}
public void getUpdateCIsAndRelationsDemo() {
    UpdateCIsAndRelations request = new UpdateCIsAndRelations();
    request.setCmdbContext(getContext());
    CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
    CIs cis = new CIs();
    List<CI> listCI = new ArrayList<CI>();
    CI ci = new CI();
    ID id = new ID();
    id.setString("temp1");
    id.setTemp(true);
    ci.setID(id);
    ci.setType("host");
    CIProperties props = new CIProperties();
    StrProps strProps = new StrProps();
    StrProp hostKeyProp = new StrProp();
    hostKeyProp.setName("host_key");
    String hostKeyValue = "blabla";
    hostKeyProp.setValue(hostKeyValue);
    strProps.getStrProps().add(hostKeyProp);
    StrProp hostOSProp = new StrProp();
    hostOSProp.setName("host_os");
    String hostOSValue = "winXP";
    hostOSProp.setValue(hostOSValue);
    strProps.getStrProps().add(hostOSProp);
    StrProp hostDNSProp = new StrProp();
    hostDNSProp.setName("host_dnsname");
    String hostDNSValue = "dnsname";
    hostDNSProp.setValue(hostDNSValue);
    strProps.getStrProps().add(hostDNSProp);
    props.setStrProps(strProps);
    ci.setProps(props);
```

```
        listCI.add(ci);
        cis.setCIs(listCI);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
        try {
          getStub().updateCIsAndRelations(request);
        } catch (RemoteException e) {
          //handle exception
        } catch (UcmdbFault e) {
          //handle exception
        }
    }
    public void getDeleteCIsAndRelationsDemo() {
        DeleteCIsAndRelations request = new DeleteCIsAndRelations();
        request.setCmdbContext(getContext());
        CIsAndRelationsUpdates updates = new CIsAndRelationsUpdates();
        CIs cis = new CIs();
        List<CI> listCI = new ArrayList<CI>();
        CI ci = new CI();
        ID id = new ID();
        id.setString("stam");
        id.setTemp(true);
        ci.setID(id);
        ci.setType("host");
        CIProperties props = new CIProperties();
        StrProps strProps = new StrProps();
        StrProp strProp1 = new StrProp();
        strProp1.setName("host_key");
        String value1 = "for_delete";
        strProp1.setValue(value1);
        strProps.getStrProps().add(strProp1);
        props.setStrProps(strProps);
        ci.setProps(props);
```

```
        listCI.add(ci);
        cis.setCIs(listCI);
        updates.setCIsForUpdate(cis);
        request.setCIsAndRelationsUpdates(updates);
        try {
          getStub().deleteCIsAndRelations(request);
        } catch (RemoteException e) {
          //handle exception
        } catch (UcmdbFault e) {
          //handle exception
        }
    }
    public static void main(String[] args) {
      try{
        UpdateDemo demo = new UpdateDemo();
        demo.initDemo();
        demo.getAddCIsAndRelationsDemo();
       } catch(Exception e){
        System.out.println(e.getMessage());
        e.printStackTrace();
       }
    }
}
```

Пример модели классов

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.classmodel.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import
com.hp.ucmdb.generated.types.classmodel.UcmdbClassModelHierarchy;
import com.hp.ucmdb.generated.types.classmodel.UcmdbClass;
import java.rmi.RemoteException;

public class ClassmodelDemo extends Demo{

    public void getClassAncestorsDemo() {
```

```
        GetClassAncestors request =
            new GetClassAncestors();
        request.setCmdbContext(getContext());
        request.setClassName("className");

        try {
            GetClassAncestorsResponse response  =
                getStub().getClassAncestors(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassHierarchy();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }

    public void getAllClassesHierarchyDemo() {
        GetAllClassesHierarchy request =
            new GetAllClassesHierarchy();
        request.setCmdbContext(getContext());
        try {
            GetAllClassesHierarchyResponse response  =
                getStub().getAllClassesHierarchy(request);
            UcmdbClassModelHierarchy hierarchy =
                response.getClassesHierarchy();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }

    public void getCmdbClassDefinitionDemo() {
        GetCmdbClassDefinition request =
            new GetCmdbClassDefinition();
        request.setCmdbContext(getContext());
        request.setClassName("className");

        try {
            GetCmdbClassDefinitionResponse response  =
            getStub().getCmdbClassDefinition(request);
            UcmdbClass ucmdbClass = response.getUcmdbClass();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
    }
}
```

Пример анализа влияния

```
package com.hp.ucmdb.demo;
import com.hp.ucmdb.generated.params.impact.*;
import com.hp.ucmdb.generated.services.UcmdbFaultException;
import com.hp.ucmdb.generated.types.*;
import com.hp.ucmdb.generated.types.impact.*;
import java.rmi.RemoteException;

/**
 * Date: Jul 17, 2007
 */
public class ImpactDemo extends Demo{

//Impact Rule Name : impactExample
//Impact Query:
//              Network
//                  |
//               Host
//                  |
//                IP
//Impact Action: network affect on ip ;severity 100% ; category:
change
//
public void calculateImpactAndGetImpactPathDemo() {
    CalculateImpact request = new CalculateImpact();
    request.setCmdbContext(getContext());
    //set root cause ids
    IDs ids = new IDs();
    ID id = new ID();
    id.setBase("rootCauseCmdbID");
    ids.addID(id);

    request.setIDs(ids);
    //set impact category
    request.setImpactCategory("change");
    //set rule Names
    ImpactRuleNames impactRuleNames = new ImpactRuleNames();
    ImpactRuleName impactRuleName = new ImpactRuleName();
    impactRuleName.setBase("impactExample");
    impactRuleNames.addImpactRuleName(impactRuleName);
    request.setImpactRuleNames(impactRuleNames);
    //set severity
    request.setSeverity(100);
    CalculateImpactResponse response =
        new CalculateImpactResponse();

    request.setIDs(ids);
    //set impact category
    request.setImpactCategory("change");
    //set rule Names
```

```
    ImpactRuleNames impactRuleNames = new ImpactRuleNames();
    ImpactRuleName impactRuleName = new ImpactRuleName();
    impactRuleName.setBase("impactExample");
    impactRuleNames.addImpactRuleName(impactRuleName);
    request.setImpactRuleNames(impactRuleNames);
    //set severity
    request.setSeverity(100);
    CalculateImpactResponse response =
        new CalculateImpactResponse();

    try {
        response  = getStub().calculateImpact(request);
    } catch (RemoteException e) {
        //handle exception

    } catch (UcmdbFaultException e) {
        //handle exception
    }
    Identifier identifier= response.getIdentifier();
    Topology topology = response.getImpactTopology();
    Relation relation = topology.getRelations().getRelation(0);
    GetImpactPath request2 = new GetImpactPath();
    //set cmdb context
    request2.setCmdbContext(getContext());
    //set impact identifier
    request2.setIdentifier(identifier);
    //set shallowRelation
    ShallowRelation shallowRelation = new ShallowRelation();
    shallowRelation.setID(relation.getID());
    shallowRelation.setEnd1ID(relation.getEnd1ID());
    shallowRelation.setEnd2ID(relation.getEnd2ID());
    shallowRelation.setType(relation.getType());
    request2.setRelation(shallowRelation);

    try {
        GetImpactPathResponse response2 =
            getStub().getImpactPath(request2);
        ImpactTopology impactTopology =
            response2.getImpactPathTopology();
    } catch (RemoteException e) {
        //To change body of catch statement
        // use File | Settings | File Templates.
        e.printStackTrace();
    } catch (UcmdbFaultException e) {
        //To change body of catch statement
        // use File | Settings | File Templates.
        e.printStackTrace();
    }
}

public void getImpactRulesByGroupName() {
```

```
    GetImpactRulesByGroupName request =
        new GetImpactRulesByGroupName();
    //set cmdb context
    request.setCmdbContext(getContext());
    //set group names list
    request.addRuleGroupNameFilter("groupName1");
    request.addRuleGroupNameFilter("groupName2");

    try {
        GetImpactRulesByGroupNameResponse response  =
            getStub().getImpactRulesByGroupName(request);
        ImpactRules impactRules = response.getImpactRules();
    } catch (RemoteException e) {
        //handle exception
    } catch (UcmdbFaultException e) {
        //handle exception
    }
}


public void getImpactRulesByNamePrefix() {
        GetImpactRulesByNamePrefix request =
            new GetImpactRulesByNamePrefix();
        //set cmdb context
        request.setCmdbContext(getContext());
        //set prefixes list
        request.addRuleNamePrefixFilter("prefix1");

        try {
            GetImpactRulesByNamePrefixResponse response  =
                getStub().getImpactRulesByNamePrefix(request);
            ImpactRules impactRules = response.getImpactRules();
        } catch (RemoteException e) {
            //handle exception
        } catch (UcmdbFaultException e) {
            //handle exception
        }
}
}
```

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
```

Глава 11

API-интерфейс Управления потоком данных

Данная глава включает:

Обзор API-интерфейса Управления потоком данных

В данной главе описывается, как сторонние инструменты могут контролировать модуль управления потоком данных через веб-службу HP Data Flow Management.

Полную документацию по доступным операциям см. в документе *HP Discovery and Dependency Mapping Schema Reference*. Эти файлы находятся по следующему пути:

<UCMDB root directory>\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.

Веб-служба управления потоком данных

Веб служба управления потоком данных – это API-интерфейс 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 управления потоком данных должны обладать следующими знаниями:

- Спецификация SOAP.
- Объектно-ориентированный язык программирования, например C++, C# или Java.
- HP Universal CMDB
- Управление потоком данных

Права доступа

Администратор предоставляет учетные данные для подключения к веб-службе. Необходимы уровни прав доступа «Просмотр», «Обновление» и «Выполнение». Сведения о правах доступа, необходимых для выполнения каждой операции, см. в документе *HP Discovery and Dependency Mapping Schema Reference*.

Вызов веб-службы

Веб-служба обнаружения и отображение зависимостей 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`

Методы управления потоком данных

В данном разделе содержится список операций веб-службы и краткий обзор их использования. Полные сведения о запросах и ответах для каждой операции см. в документе *HP Discovery and Dependency Mapping Schema Reference*.

Этот раздел охватывает следующие темы:

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 Discovery and Dependency Mapping Schema Reference*.

IPList

Список элементов `IP`, каждый из которых содержит адрес IPv4.

См. IPList в документе *HP Discovery and Dependency Mapping Schema Reference*.

IPRange

`IPRange` состоит из двух элементов – `Start` и `End`. В каждом из них содержится элемент `Address`, представляющий собой адрес IPv4.

См. IPRange в документе *HP Discovery and Dependency Mapping 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 Discovery and Dependency Mapping Schema Reference*

Методы управления заданиями обнаружения

activateJob

Активация определенного задания.

См. "Пример кода" на странице 301

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
JobName	Имя задания.

deactivateJob

Деактивация определенного задания.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
JobName	Имя задания.

dispatchAdHocJob

Отправка задания в зонд по запросу. Задание должно быть активным и содержать указанный ЭК-триггер.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
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 243.

Исходящие параметры

Параметр	Комментарий
strList	Список имен заданий.

isJobActive

Проверка активности задания.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
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 243.
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 243.
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 243.

Параметр	Комментарий
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 243.
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 243.
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 243.
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 243.
domainName	Имя домена.

Исходящие параметры

Параметр	Комментарий
domainType	Тип домена.

getDomainsNames

Возврат имен текущих доменов.

См. "Пример кода" на странице 301

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.

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 243.
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. "Пример кода" на странице 301

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
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 243.
domainName	Проверяемый домен.
probeName	Имя зонда.

Исходящие параметры

Параметр	Комментарий
probeScope	"Scope" зонда.

isProbeConnected

Проверка подключения указанного зонда.

См. "Пример кода" на странице 301

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
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 243.
domainName	Домен.
probeName	Обновляемый зонд.
newScope	"Scope" для зонда (новое значение).

Методы учетных данных

addCredentialsEntry

Добавляет запись учетных данных с указанным протоколом в указанный домен.

См. "Пример кода" на странице 301

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
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 243.
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 243.
domainName	Домен, для которого необходимо получить учетные данные.
protocolName	Протокол, используемый в этом домене.
credentialsEntryID	Идентификатор получаемых учетных данных.

Исходящие параметры

Параметр	Комментарий
credentialsEntryParameters	Коллекция "CIProperties" учетных данных.

removeCredentialsEntry

Удаление выбранной записи учетных данных из протокола.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
domainName	Домен.
protocolName	Протокол, используемый 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 243.
domainName	Домен, в котором необходимо обновить учетные данные.
protocolName	Протокол, используемый в этом домене.
credentialsEntryID	Идентификатор обновляемых учетных данных.
credentialsEntryParameters	Коллекция "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 243.
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 — значение от `0` до `1`. Ответ `1` означает, что вызов **rediscoverCIs** завершен.

Входящие параметры

Параметр	Комментарий
cmdbContext	Подробнее см. в разделе "CmdbContext" на странице 243.
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 наполнения 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 243.
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 243.
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;
    }
}
```


Java™
COMPATIBLE