

IBM Content
Manager for Multiplatforms/IBM Information
Integrator for Content



Руководство по прикладному программированию для рабочих станций

Версия 8 Выпуск 2

IBM Content
Manager for Multiplatforms/IBM Information
Integrator for Content



Руководство по прикладному программированию для рабочих станций

Версия 8 Выпуск 2

Примечание

Перед тем как использовать данный документ и продукты, описанные в нем, прочтите сведения под заголовком “Замечания” на стр. 629.

Второе издание (март 2003)

Данным изданием можно пользоваться при работе с Версией 8 Выпуском 2 IBM Enterprise Information Portal for Multiplatforms, IBM Content Manager for Multiplatforms (номер продукта 5724-B43, 5724-B19) и со всеми последующими выпусками и модификациями, пока в новых изданиях не будет иных указаний.

На эти продукты распространяется частичное авторское право: Copyright © 1990-2000 ActionPoint, Inc. and/or its licensors, 1299 Parkmoor Drive, San Jose, CA 95126 U.S.A. Все права защищены.

CUP Parser Generator Copyright Notice, License, and Disclaimer

© Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian

Настоящим дается разрешение на бесплатное использование, копирование, модификацию и распространение программного обеспечения CUP Parser Generator и документации для любой цели при условии, что данное замечание об авторских правах будет присутствовать во всех копиях, а замечание об авторских правах, данное разрешение и отказ от гарантий будут включены в сопровождающую документацию, и что имена авторов или их работодателей не будут использоваться для рекламы, связанной с распространением этого программного обеспечения, без их предварительного письменного разрешения.

Авторы и их работодатели не предоставляют никаких гарантий, связанных с данным программным обеспечением, включая любые подразумеваемые гарантии рыночной пригодности и соответствия целям. Ни при каких обстоятельствах авторы или их работодатели не несут ответственности за прямой или косвенный ущерб, связанный с потерей пригодности, информации или упущенной выгодой, из-за действия или бездействия по причине использования или в связи с использованием этого программного обеспечения.

© Copyright International Business Machines Corporation 1996, 2003. Все права защищены.

Содержание

Об этом руководстве ix

Кто должен использовать это руководство . . . ix

Где найти дополнительную информацию . . . x

Информация, включенная в пакет продукта . . . x

Поддержка в Web xi

Как послать ваши отзывы. xi

Что нового в Enterprise Information Portal

Версии 8? xii

Что нового в Content Manager Версии 8? . . . xiv

Введение в Enterprise Information Portal . . . 1

Поиск информации о клиентах 1

Потребности 2

Решение. 2

Решение Enterprise Information Portal. 3

Компоненты IBM Enterprise Information Portal

for Multiplatforms 4

Что нового в API Версии 8.2 7

Что нового в API Версии 8.1 8

Новые и измененные классы Java. 9

Новые и измененные классы C++ 10

Основные понятия прикладного программирования в Enterprise Information Portal 13

Как происходит обращение к данным через контент-серверы. 13

Основные понятия динамических объектов данных. 14

Динамические объекты данных (DDO) . . . 14

Расширенные объекты данных (XDO) . . . 15

Представление мультимедийного

содержимого. 16

Контент-серверы и объекты DDO 16

Сравнение DDO/XDO со значениями

атрибутов и частями элементов. 17

Постоянные идентификаторы (PID) 17

Работа с контент-сервером объединения и с объединенным поиском. 19

Отображение схемы объединения 22

Использование компонентов отображения

контент-сервера объединения 22

Выполнение запросов объединения. 23

Синтаксис запроса объединения 25

Сохранение результатов поиска в папках

объединения (только Java) 28

Работа с управлением системой. 28

Настройка клиента администратора

системы EIP 28

Создание программ с использованием интерфейсов прикладного программирования (API) 31

Различия между API Java и C++. 31

Архитектура клиент-сервер (только Java). . . 32

Пакеты для среды Java. 32

Советы по программированию 33

Настройка среды Java (только Java) 33

Задание переменных среды Java для

Windows 34

Задание переменных среды Java для AIX . . 34

Задание переменных среды Java для Solaris . 34

Использование вызова удаленного метода

(RMI) с контент-серверами 35

Настройка среды C++ (только C++) 35

Задание переменных среды C++ для

Windows 37

Задание переменных среды C++ для AIX . . 37

Построение программ C++ 38

Задание подсистемы консоли для

преобразования кодовых страниц в Windows 39

Опции множественного поиска 39

Трассировка 40

Трассировка текстовых запросов,

использующих механизм текстового поиска. 40

Трассировка параметрических запросов . . 41

Обработка исключительных ситуаций. . . 41

Константы 43

Соединение с контент-серверами 44

Установление соединения. 44

Соединение клиента с контент-сервером и

отсоединение от него 45

Задание и получение опций контент-сервера 46

Получение списка контент-серверов . . . 46

Вывод списка объектов и атрибутов для

контент-сервера 47

Работа с динамическими объектами данных

(DDO) 50

Создание DKDDO 51

| | | | |
|---|-----|--|------------|
| Добавление свойств в DDO | 53 | Различие между dkResultSetCursor и DKResults | 126 |
| Создание постоянного идентификатора (PID) | 53 | Использование параметрических запросов | 126 |
| Работа с элементами данных и свойствами | 54 | Использование текстового запроса | 134 |
| Получение свойств DKDDO и атрибутов | 58 | Использование указателя набора результатов | 151 |
| Как вывести весь DDO | 59 | Открытие и закрытие указателя набора результатов для повторного выполнения запроса | 151 |
| Удаление DDO (только C++). | 61 | Задание и получение позиций в указателе набора результатов | 151 |
| Работа с расширенными объектами данных (XDO) | 61 | Создание собрания по указателю набора результатов | 154 |
| Использование постоянного идентификатора (PID) XDO | 62 | Запросы собраний | 155 |
| Свойства XDO | 62 | Получение результатов запроса | 155 |
| Строки конфигурации DB2, ODBC и DataJoiner (только C++) | 63 | Оценка (evaluate) нового запроса | 155 |
| Советы по программированию на Java | 64 | Использование запрашиваемого собрания вместо комбинированного запроса | 157 |
| Советы по программированию на C++ | 64 | | |
| XDO как часть DDO | 65 | | |
| Отдельный XDO | 67 | | |
| Примеры работы с XDO | 71 | | |
| Работа с XML (только Java) | 100 | Работа с Content Manager Версии 8.2 | 159 |
| Расширение возможностей импорта и экспорта XML | 100 | Что такое система Content Manager | 159 |
| Импорт документов XML | 101 | Основные понятия системы Content Manager | 160 |
| Экспорт XML | 106 | Элементы | 161 |
| Создание документов и использование атрибута DKPARTS | 107 | Атрибуты | 161 |
| Создание папок и использование атрибута DKFOLDER | 111 | Типы элементов | 162 |
| Использование объектов DKAny (только C++) | 114 | Корневые и дочерние компоненты | 162 |
| Использование кода типа | 114 | Объекты | 164 |
| Управление памятью в DKAny | 114 | Связи и ссылки | 164 |
| Использование конструкторов | 114 | Документы | 165 |
| Получение кода типа | 115 | Папки | 165 |
| Присваивание нового значения объекту DKAny | 115 | Поддержка версий | 165 |
| Присваивание значения из DKAny | 115 | Управление доступом | 166 |
| Вывод DKAny | 116 | Планирование прикладной программы Content Manager | 172 |
| Уничтожение DKAny | 117 | Определение свойств прикладной программы | 172 |
| Советы по программированию | 117 | Обработка ошибок | 173 |
| Использование собраний и итераторов | 117 | Работа с примерами Content Manager | 174 |
| Использование методов последовательных собраний | 118 | Пример сценария со страховой компанией | 175 |
| Использование последовательного итератора | 118 | Создание прикладной программы Content Manager | 175 |
| Управление памятью в собраниях (только C++) | 121 | Программные компоненты | 175 |
| Сортировка собрания | 123 | Представление элементов с помощью объектов DDO | 176 |
| Собрание и итератор объединения | 123 | Соединение с системой Content Manager | 177 |
| Запрос контент-сервера | 125 | Работа с элементами | 179 |
| | | Работа с папками | 206 |
| | | Определение связей между элементами | 214 |
| | | Работа с управлением доступом | 216 |
| | | Создание привилегии | 216 |
| | | Создание набора привилегий | 218 |

| | | | |
|--|-----|--|------------|
| Как вывести свойства набора привилегий | 220 | Что надо учитывать при проектировании транзакций в собственных прикладных программах | 269 |
| Определение списка управления доступом (ACL) | 221 | Что надо учитывать при использовании явных транзакций | 269 |
| Как получить и вывести информацию ACL | 223 | Применение резервирования и активирования в транзакциях | 270 |
| Назначение ACL для типа элементов | 224 | Обработка транзакций | 270 |
| Назначение ACL для элемента | 226 | Маршрутизация документа по процессу | 272 |
| Язык запросов | 227 | Процесс маршрутизации документов | 272 |
| Запрос сервера Content Manager | 227 | Задание процесса маршрутизации документов | 273 |
| Применение языка запросов к модели данных Content Manager | 228 | Работа с другими контент-серверами | 299 |
| Понятие параметрического поиска | 230 | Работа с ранними версиями Content Manager | 302 |
| Понятие текстового поиска | 231 | Обработка больших объектов | 302 |
| Поиск содержимого объектов | 232 | Использование объектов DDO для содержания ранних версий Content Manager | 302 |
| Поиск документов | 232 | Создание, изменение и удаление документов и папок | 304 |
| Как объявить пользовательские атрибуты допускающими текстовый поиск | 232 | Получение документа или папки | 314 |
| Понятие синтаксиса текстового поиска | 232 | О текстовом поиске (Механизм текстового поиска) | 319 |
| Создание комбинированного параметрического и текстового поиска | 234 | Поиск изображений по их содержимому | 346 |
| Примеры запросов | 236 | Использование программ поиска изображений | 350 |
| Примеры запросов | 240 | Установление соединения в QBIC | 354 |
| Язык запросов | 247 | Получение списка серверов поиска изображений | 355 |
| Применение эскейп-последовательностей в операторах сравнения ("=", "!", ">", "<", "BETWEEN" и прочих) | 247 | Получение списка баз данных, каталогов и характеристик поиска изображений | 357 |
| Применение эскейп-последовательностей с оператором LIKE | 248 | Представление информации поиска изображений при помощи объекта DDO | 359 |
| Применение эскейп-последовательностей со сложным поиском (функции "contains" и "score") | 249 | Работа с запросами изображений | 360 |
| Применение эскейп-последовательностей с основным текстовым поиском (функции "contains-text-basic" и "score-basic") | 250 | Использование механизма поиска изображений | 365 |
| Применение эскейп-последовательностей в Java и C++ | 252 | Индексирование существующего XDO при помощи механизмов поиска | 366 |
| Грамматика языка запросов | 253 | Использование комбинированного запроса | 369 |
| Работа с менеджером ресурсов | 255 | О функциях рабочего потока и рабочего комплекта в ранних версиях Content Manager | 373 |
| Работа с объектами менеджера ресурсов | 256 | Работа с OnDemand | 384 |
| Управление документами в Content Manager | 257 | Представление серверов и документов OnDemand | 385 |
| Создание модели данных управления документами | 259 | Соединение с сервером OnDemand и отсоединение от него | 385 |
| Создание документного типа элементов | 259 | Получение список для OnDemand | 386 |
| Создание документа | 261 | Получение документа OnDemand | 390 |
| Изменение документа | 264 | Включение режима папок OnDemand | 397 |
| Получение и удаление документа | 266 | | |
| Поддержка версий частей в модели данных управления документами | 267 | | |
| Работа с транзакциями | 268 | | |

| | | | |
|---|-----|--|------------|
| Асинхронный поиск | 398 | Создание пользовательских контент-серверов | 464 |
| Папки OnDemand как шаблоны поиска | 398 | Разработка пользовательских | |
| Папки OnDemand как собственные объекты | 398 | контент-серверов | 464 |
| Создание и изменение комментариев | 399 | Использование класса FeServerDefBase | |
| Трассировка | 399 | (только для Java) | 483 |
| Работа с сервером Content Manager ImagePlus | | Построение программ рабочего потока | |
| for OS/390 | 401 | EIP | 485 |
| Получение списка объектов и атрибутов | 402 | Соединение со службами рабочего потока | 485 |
| Синтаксис запросов ImagePlus for OS/390 | 408 | Запуск рабочего потока | 486 |
| Работа с Content Manager for AS/400 | 410 | Прекращение рабочего потока | 488 |
| Вывод списка объектов (индексных | | Получение списка всех рабочих потоков | 489 |
| классов) и атрибутов | 410 | Приостановка рабочего потока | 490 |
| Выполнение запроса | 411 | Возобновление рабочего потока | 490 |
| Выполнение параметрического запроса | 418 | Получение списка всех рабочих списков | 491 |
| Работа с Domino.Doc | 419 | Доступ к рабочему списку | 492 |
| Получение списка объектов и подобъектов | 422 | Доступ к рабочим элементам | 493 |
| Получение списка атрибутов шкафа | 425 | Перемещение элементов в рабочем потоке | 494 |
| Построение запросов в Domino.Doc | 426 | Получение списка всех шаблонов рабочих | |
| Использование синтаксиса запроса | 427 | потоков | 495 |
| Работа с расширенным поиском - Extended | | Создание ваших собственных действий | |
| Search (ES) Domino | 428 | (только Java) | 496 |
| Получение списка серверов Extended Search | 428 | Построение прикладных программ с | |
| Получение списка объектов (баз данных) и | | невизуальными и визуальными | |
| атрибутов (полей) | 429 | JavaBeans. | 499 |
| Использование языка GQL (Generalized | | Основные понятия, связанные с функциями | |
| Query Language - обобщенный язык | | bean | 499 |
| запросов) | 433 | Использование JavaBeans в построителях | 501 |
| Идентификация типа элемента объекта | | Использование IBM Websphere Studio | |
| DDO в Extended Search | 435 | Application Developer | 501 |
| Создание PID в Extended Search | 436 | Вызов Java bean EIP | 502 |
| Обработка содержимого документа | | Невизуальные функции bean | 503 |
| Extended Search. | 436 | Конфигурации невидзуальных функций bean | 503 |
| Получение документа | 444 | Возможности невидзуальных функций bean | 504 |
| Получение двоичного большого объекта | 444 | Категории невидзуальных функций bean | 504 |
| Связывание типов MIME с документами | 446 | Особенности использования невидзуальных | |
| Использование поиска в объединении в | | функций bean | 510 |
| Extended Search. | 446 | Трассировка и регистрация в функциях | |
| Работа с Panagon Image Services (только для | | bean | 511 |
| Java) | 447 | Что такое свойства и события для | |
| Моделирование данных | 447 | невидзуальных функций bean. | 511 |
| Документы и страницы в Panagon Image | | Построение прикладной программы с | |
| Services | 448 | использованием невидзуальных функций | |
| Получение списка объектов и атрибутов | 449 | bean | 512 |
| Запросы | 451 | Работа с визуальными функциями bean | 512 |
| Работа с реляционными базами данных. | 455 | Функция bean CMBLogonPanel | 514 |
| Соединение с реляционными базами | | Функция bean CMBSearchTemplateList | 515 |
| данных | 455 | Функция bean CMBSearchTemplateViewer | 516 |
| Получение списка объектов и списка | | | |
| атрибутов объекта | 457 | | |
| Выполнение запроса | 460 | | |

| | |
|---|------------|
| Проверка или редактирование полей в CMBSearchTemplateViewer | 517 |
| Функция bean CMBSearchPanel | 517 |
| Функция bean CMBSearchResultsViewer | 517 |
| Переопределение всплывающих меню | 519 |
| Функция bean CMBFolderViewer | 519 |
| Функция bean CMBDocumentViewer | 521 |
| Как задать программу просмотра | 521 |
| Программы просмотра по умолчанию | 522 |
| Запуск внешних программ просмотра | 522 |
| Функция bean CMBItemAttributesEditor | 522 |
| Блокировка изменений в CMBItemAttributesEditor | 523 |
| Функция bean CMBVersionsViewer | 523 |
| Общие особенности поведения визуальных функций bean | 523 |
| Замена визуальной функции bean | 524 |
| Построение прикладной программы с использованием визуальных функций bean | 525 |
| Работа с комплектом программы просмотра документов Java | 529 |
| Структура программы просмотра | 530 |
| Механизмы документов | 531 |
| Механизм аннотирования | 531 |
| Создание общей программы просмотра документов | 531 |
| Настройка общей программы просмотра документов | 532 |
| Примеры прикладных программ | 534 |
| Автономная программа просмотра | 534 |
| Программа Java | 535 |
| Минимальный клиент | 536 |
| Апплет или сервлет | 536 |
| Двойной режим, апплет или сервлет | 537 |
| Работа со службами аннотирования | 538 |
| Использование интерфейсов со службами аннотирования | 539 |
| Как работает поддержка редактирования аннотации | 540 |
| Построение прикладной программы с использованием служб аннотирования | 540 |
| Как добавить пользовательский тип аннотации в собственной прикладной программе | 541 |
| Работа с библиотекой тегов и сервлетом контроллера Enterprise Information Portal | 543 |
| Установка библиотеки тегов и сервлета | 543 |
| Использование библиотеки тегов | 543 |
| Соглашения, используемые в библиотеке тегов | 544 |
| Сводка тегов | 545 |
| Теги, связанные с соединениями | 545 |
| Теги, связанные с схемой | 545 |
| Теги, связанные с поиском | 547 |
| Теги, связанные с элементом | 547 |
| Теги, связанные с папками | 549 |
| Теги, связанные с документами | 549 |
| Сервлет контроллера EIP | 549 |
| Что может делать этот сервлет | 550 |
| Обращение к сервлету | 551 |
| Матрица функций сервлетов комплекта инструментов | 558 |
| Работа с исследованием информации | 561 |
| Построение прикладной программы исследования информации при помощи функций bean | 561 |
| Положение файлов примеров | 566 |
| Каталогизация документов | 567 |
| Составление сводок документов | 577 |
| Извлечение информации | 584 |
| Кластеризация | 591 |
| Импорт документов из пространства Web | 597 |
| Поиск документов по категориям | 606 |
| Построение собственного контент-провайдера | 613 |
| Использование API служб | 614 |
| Соединение с API служб исследования информации | 615 |
| Управление библиотекой, таксономиями и каталогами | 615 |
| Использование инструментов исследования информации | 619 |
| Создание записей и сохранение метаданных в каталогах | 623 |
| Поиск документов | 624 |
| Запуск задачи сервера | 625 |
| Пример прикладной программы Исследование информации на основе программ JSP | 627 |
| Установка JSP | 628 |
| Замечания | 629 |
| Торговые марки | 631 |
| Глоссарий | 633 |

| | |
|------------------|-----|
| Индекс | 647 |
|------------------|-----|

Об этом руководстве

В этом руководстве описывается, как создавать интерфейсы прикладного программирования (API) Java, JavaBeans и C++, предусмотренные в Enterprise Information Portal Версия 8 Выпуск 2 (EIP) и Content Manager (CM) Версия 8 Выпуск 2. Эти API и функции bean представляют собой строительные блоки для создания прикладных программ, обращающихся к содержимому на разнородных контент-серверах.

В ранних версиях для CM и EIP использовались отдельные руководства по прикладному программированию. В Версия 8 Выпуск 2 оба этих продукта совместно используют многие API и приемы программирования. Поскольку, кроме того, большинство новых возможностей Enterprise Information Portal Версия 8 Выпуск 2 используют новый соединитель для CM Версия 8 Выпуск 2, оба руководства объединены в одно.

Это руководство содержит:

- Введение в понятия прикладного программирования Enterprise Information Portal и CM, включая понятия динамических объектов данных в контексте Java и C++
- Описание функций, доступных через соединитель CM Версия 8 Выпуск 2
- Документацию для всех остальных соединителей Enterprise Information Portal с контент-серверами
- Изменения для визуальных и невизуальных JavaBeans
- Изменения в информации программирования для Исследование информации, IBM Web Crawler и рабочий поток

Иллюстрации, описывающие Content Manager, применимы и к ранним версиям, и к Версии 8.1 этого продукта.

Кто должен использовать это руководство

Это руководство предназначено для прикладных программистов, обладающих одним или всеми из следующих навыков:

- Опыт работы с C++, Java, JavaBeans, или HTML
- Знакомы с понятиями реляционных баз данных
- Знают протоколы DDO/XDO

Где найти дополнительную информацию

Пакет продукта содержит полный комплект информации по планированию, установке, использованию системы и управлению ей. Кроме того, документацию по продукту и поддержку можно получить в World Wide Web.

Информация, включенная в пакет продукта

В пакет продукта включен Информационный центр и все публикации в формате .PDF (Portable document format - формат переносимых документов).

Информационный центр

В пакет продукта включен Информационный центр, который можно установить при установке продукта. Сведения об установке Информационного центра смотрите в разделе *Планирование и установка вашей системы Content Management*.

Информационный центр содержит документацию по Content Manager, Enterprise Information Portal и Content Manager VideoCharger. Информация разбита на темы и организована по продуктам и задачам (например, Управление). Кроме механизма навигации и указателей, предусмотрена и возможность поиска.

Публикации в формате PDF

Файлы PDF можно просмотреть с помощью прилагаемой программы Adobe Acrobat Reader для вашей операционной системы. Если у вас не установлена программа Acrobat Reader, ее можно получить на сайте Adobe по адресу www.adobe.com.

В разделе Табл. 1 приводятся публикации по Content Manager, прилагаемые к IBM Content Manager for Multiplatforms.

Таблица 1. Публикации по Content Manager

| Имя файла | Заголовок | Номер публикации |
|-----------|--|------------------|
| install | <i>Планирование и установка вашей системы Content Management¹</i> | GH43-0212-01 |
| migrate | <i>Перенастройка в Content Manager Версии 8</i> | SC43-0241-01 |
| sysadmin | <i>Руководство по управлению системой</i> | SH43-0213-01 |

Заказав IBM Content Manager for Multiplatforms, вы получите и IBM Enterprise Information Portal for Multiplatforms. IBM Enterprise Information Portal for Multiplatforms можно заказать и отдельно. В Табл. 2 перечислены публикации по Enterprise Information Portal, прилагаемые к продукту.

Таблица 2. Публикации по Enterprise Information Portal

| Имя файла | Заголовок | Номер публикации |
|-----------|--|------------------|
| apgwork | <i>Workstation Application Programming Guide¹</i> | SH43-0217-01 |

Таблица 2. Публикации по Enterprise Information Portal (продолжение)

| Имя файла | Заголовок | Номер публикации |
|-----------|---|------------------|
| ecliinst | <i>eClient. Установка, конфигурирование и управление</i> | SH43-0219-02 |
| eipinst | <i>Планирование и установка Enterprise Information Portal</i> | GH43-0215-01 |
| eipmanag | <i>Управление Enterprise Information Portal</i> | SH43-0216-01 |
| messcode | <i>Сообщения и коды²</i> | SH43-0218-01 |

Примечание:

1. В *Workstation Application Programming Guide* приводится информация о создании программ и для Content Manager, и для Enterprise Information Portal.
2. В книге *Сообщения и коды* приводятся сообщения и коды для Content Manager и Enterprise Information Portal.

Поддержка в Web

Поддержка данного продукта доступна в Web. Для этого выберите опцию **Support** (Поддержка) на Web-сайтах продукта по адресам:

www.ibm.com/software/data/cm/

www.ibm.com/software/data/eip/

Документация поставляется в электронной форме вместе с продуктом. Если вам понадобится получить доступ к документации по продукту, имеющейся в Web, перейдите на Web-сайт продукта и щелкните по опции **Library** (Библиотека).

В WWW также имеется интерфейс для работы с документацией в формате HTML, который называется Enterprise Documentation Online (EDO). В настоящее время он содержит справочную информацию по API. Информацию о том, как получить доступ к EDO, смотрите на Web-странице Enterprise Information Portal Library.

Как послать ваши отзывы

Обратная связь поможет фирме IBM поставлять качественную информацию. Пожалуйста, высылайте любые свои замечания об этой книге или какой-либо другой документации по Content Manager или Enterprise Information Portal. Выслать замечания можно:

- Через Web. Зайдите на страницу IBM Data Management Online Reader's Comment Form (RCF) по адресу:

www.ibm.com/software/data/rcf

Эту страницу можно использовать для ввода и отправки замечаний.

- По электронной почте на адрес comments@vnet.ibm.com. Не забудьте указать название продукта, номер версии продукта, название и номер книги (если есть). Если вы шлете замечание к определенному тексту, укажите положение этого текста (например, главу и название раздела, номер таблицы, номер страницы или заголовок темы справки.)

Что нового в Enterprise Information Portal Версии 8?

Версия 8.2: В Версию 8.2 включен большой набор усовершенствований. В Версии 8.2 расширены функции рабочего потока управления системой и добавлена поддержка новейших возможностей технологии баз данных - поддержка DB2 Universal Database Версии 8.1. Вот сводка этих и других усовершенствований продукта Версии 8.2:

Название Enterprise Information Portal изменено на IBM Information Integrator for Content

Продукт Enterprise Information Portal переименован в Information Integrator for Content. Хотя названия книг для Версии 8.2 были изменены, в тексте осталось название продукта Enterprise Information Portal. При поиске дополнительной информации в Web продолжайте использовать Enterprise Information Portal или EIP, пока не завершен переход на новое название.

Поддержка DB2 UDB Версии 8.1

Поддерживается Enterprise Information Portal Версии 8.2. Возможность концентрации соединений в DB2 Версии 8.1 обеспечивает повышенную масштабируемость для двухуровневых программ и клиентов.

Папки объединения

В eClient теперь можно организовать документы и собственные папки из нескольких репозиториях в одну папку объединения и запустить эту папку в рабочем потоке. Кроме того, папки объединения позволяют пользователям постоянно хранить результаты поиска в базе данных объединения EIP, откуда их можно получить в любое время. Для этих папок объединения доступны все операции - создание, получение, изменение и удаление без переиндексации.

Точки сбора расширенного рабочего потока

Рабочий поток теперь полностью поддерживается в AIX и Solaris. Построитель рабочего потока, интерфейсы API, Монитор точек сбора и функции JavaBeans обеспечивают удобство использования и усовершенствованные функции рабочего потока.

Microsoft Visual Studio .NET для построения программ

Интерфейсы API Enterprise Information Portal Версии 8.1 и новее

теперь поддерживают Microsoft Visual Studio .NET, что позволяет писать программы управления содержимым и интегрировать программы, построенные при помощи Microsoft Visual Studio .NET.

Версия 8.1: Начиная с Версии 8.1, наследуется интеграция и универсальность. Одно из основных, а также многих других усовершенствований прежних продуктов Content Manager - это новая структура данных, обеспечивающая более широкие возможности настройки документов. Вот сводка изменений для продукта Content Manager Версии 8.1:

Поддержка Sun Solaris

В системе Solaris можно установить соединители, возможности и базы данных Java.

Общее управление системой

Одна прикладная программа клиента обеспечивает независимый доступ к управлению Content Manager и Enterprise Information Portal.

Новые соединители

- Соединитель ICM для Content Manager Версии 8 Выпуск 1 позволяет использовать преимущества мощной системы хранения документов Content Manager Версии 8.
- Новый соединитель C++ Extended Search Версии 3.7 работает в AIX.

Улучшенные соединители

- Параметрический текстовый поиск поддерживается и на уровне объединения, и при прямом соединении Extended Search.
- В соединитель OnDemand внесены функциональные усовершенствования и улучшения производительности, в том числе:
 - Изменения в структуре DDO OnDemand.
 - Поддерживается асинхронный поиск

IBM Web Crawler

IBM Web Crawler - возможность, которая позволяет пользователям искать информацию в Web и базах данных Lotus Notes и составлять для нее сводки.

Усовершенствования рабочих потоков

Рабочий поток теперь полностью поддерживается в AIX и Solaris. Построитель рабочего потока, API и JavaBeans обеспечивают усовершенствованные функции рабочего потока и удобство использования.

Информационный центр

Информационный центр, доступный через браузер, содержит документацию по Content Manager, Enterprise Information Portal и Content Manager VideoCharger. Информация разбита на темы и организована по

продуктам и задачам (например, Управление). Кроме механизма навигации и указателей, предусмотрена и возможность поиска.

Доступность

Функции доступности помогают пользователю с физическими недостатками, например с ограниченной подвижностью или недостаточным зрением, с успехом пользоваться программными продуктами. Основные функции доступности продукта:

- Возможность использовать клавиатуру вместо мыши для работы с любыми функциями.
- Поддержка улучшенных свойств дисплея.
- Опции зрительных и звуковых оповещений.
- Совместимость с технологиями для людей с физическими недостатками
- Совместимости с возможностями доступности операционной системы
- Удобные форматы документации

Что нового в Content Manager Версии 8?

Версия 8.2: В Версию 8.2 по сравнению с Версией 8.1 включен большой набор усовершенствований. В Версии 8.2 в eClient добавлены дополнительные функции рабочего потока, расширена функция управления ресурсами и добавлена поддержка новейших возможностей технологии баз данных и клиентов, в том числе поддержка DB2 Universal Database Версии 8.1, Oracle Версии 8.1.7.4 и Версии 9.2.0.1 и WebSphere Версии 5. Вот сводка этих и других усовершенствований Версии 8.2:

Название Enterprise Information Portal изменено на IBM Information Integrator for Content

Продукт Enterprise Information Portal переименован в Information Integrator for Content. Хотя названия книг для Версии 8.2 были изменены, в тексте осталось название продукта Enterprise Information Portal. При поиске дополнительной информации в Web продолжайте использовать Enterprise Information Portal или EIP, пока не завершён переход на новое название.

Поддержка Oracle Версии 8.1.7.4, версии 9.2.0.1 или новее

В Content Manager Версии 8.2 добавлена поддержка баз данных Oracle, управляющих метаданными, которые хранятся и на библиотечном сервере, и в менеджере ресурсов. Для пользователей Oracle включены инструменты по перенастройке для Content Manager Версии 7. **Замечание:** Oracle нельзя использовать для управления контент-серверами баз данных Enterprise Information Portal.

Репликация

В Content Manager Версии 8.2 включена репликация менеджера ресурсов, то есть возможность хранения объектов в нескольких местоположениях, управление которым осуществляется менеджерами ресурсов репликации. Для улучшения балансировки загрузки реплики объектов будут вести себя как объекты сетевого кэша.

сетевой кэш

Поддержка сетевого кэша в Content Manager Версии 8.2 обеспечивает невидимое для прикладных программ кэширование при помощи локальных серверов, как определено системным администратором.

Поддержка DB2 UDB Версии 8.1

Content Manager Версии 8.2 и Enterprise Information Portal Версии 8.2 поддерживают DB2/UDB Версии 8.1. Возможность концентрации соединений в DB2 Версии 8.1 обеспечивает повышенную масштабируемость для двухуровневых программ и клиентов (например, для клиентов Content Manager Версии 8). В DB2/UDB Версии 8.1 модуль DB2 Universal Database Text Information Extender (TIE) заменен модулем Net Search Extender (NSE).

Поддержка сервера программ WebSphere Версии 4 и Версии 5

В сервере программ WebSphere Версии 5 вводится внедрение сервера, а также доступ к данным и управление ими из любого браузера.

Папки объединения

В eClient теперь можно организовать документы и собственные папки из нескольких репозиториях в одну папку объединения и запустить эту папку в рабочем потоке. Кроме того, папки объединения позволяют пользователям постоянно хранить результаты поиска в базе данных объединения EIP, откуда их можно получить в любое время. Для этих папок объединения доступны все операции - создание, получение, изменение и удаление без переиндексации.

Точки сбора расширенного рабочего потока

Рабочий поток теперь полностью поддерживается в AIX и Solaris. Построитель рабочего потока, интерфейсы API, Монитор точек сбора и функции JavaBeans обеспечивают удобство использования и усовершенствованные функции рабочего потока.

Microsoft Visual Studio .NET для построения программ

Интерфейсы API Content Manager и Enterprise Information Portal Версии 8.1 и новее теперь поддерживают Microsoft Visual Studio .NET, что позволяет писать программы управления содержимым и интегрировать программы, построенные при помощи Microsoft Visual Studio .NET.

Версия 8.1: Начиная с Версии 8.1, наследуется интеграция и универсальность. Одно из основных, а также многих других усовершенствований прежних продуктов Content Manager - это новая структура данных, обеспечивающая более широкие возможности настройки документов. Вот сводка изменений для продукта Content Manager Версии 8.1:

Улучшенная производительность

Применение на библиотечном сервере и менеджере ресурсов хранимых процедур DB2 позволяет использовать преимущества технологии DB2, значительно снижая нагрузку на сеть и улучшая производительность и масштабируемость.

Поддержка Sun Solaris

И библиотечный сервер, и менеджер ресурсов можно установить в системе Sun Solaris.

Усовершенствованная модель данных

Новая иерархическая модель данных создает основу для пользовательских решений по управлению составными документами.

Улучшенный рабочий поток

Благодаря встроенной маршрутизации документов к возможностям рабочего потока добавлены последовательная маршрутизация, динамическая маршрутизация и поддержка точек сбора.

Встроенный текстовый поиск

Теперь, кроме поиска с использованием атрибутов, пользователи могут выполнять с клиентов полнотекстовый поиск по содержащейся в документе текстовой информации. Функция текстового поиска использует теперь модуль расширения DB2 Universal Database Text Information Extender, который упрощает процесс установки текстового поиска.

Общее управление системой

Одна прикладная программа клиента обеспечивает независимый доступ к управлению Content Manager и Enterprise Information Portal. В Content Manager домены администратора позволяют ограничить доступ администратора к подразделам библиотечного сервера.

Полнофункциональный клиент рабочего стола и усовершенствованный eClient

Усовершенствования в клиенте дают пользователям готовую программу для быстрого развертывания или интеграции бизнес-программ. Клиент для Windows поддерживает встроенный текстовый поиск, маршрутизацию документов, иерархическую модель данных (с одним уровнем дочерних компонентов), работу с несколькими версиями и индексирование при импорте. eClient

содержит встроенный текстовый поиск, расширенные возможности рабочего потока EIP, управление версиями и многозначные атрибуты.

Упрощенная установка

Установка в поддерживаемых операционных системах выполняется сходным образом; информацию о пользовательской установке обеспечивает ассистент по планированию на компакт-диске Начните отсюда. Есть также возможность автоматической установки и установки с консоли.

Информационный центр

Информационный центр, доступный через браузер, содержит документацию по Content Manager, Enterprise Information Portal и Content Manager VideoCharger. Информация разбита на темы и организована по продуктам и задачам (например, Управление). Кроме механизма навигации и указателей, предусмотрена и возможность поиска.

Доступность

Функции доступности помогают пользователю с физическими недостатками, например с ограниченной подвижностью или недостаточным зрением, с успехом пользоваться программными продуктами. Основные функции доступности продукта:

- Возможность использовать клавиатуру вместо мыши для работы с любыми функциями.
- Поддержка улучшенных свойств дисплея.
- Опции зрительных и звуковых оповещений.
- Совместимость с дружелюбными технологиями
- Совместимости с возможностями доступности операционной системы
- Доступные форматы документации

Интеграция PeopleSoft и Siebel

Пользователи программ PeopleSoft и Siebel теперь могут сконфигурировать эти программы, чтобы обращаться к содержимому, хранимому на разнородных контент-серверах, при помощи eClient.

Введение в Enterprise Information Portal

Многим предприятиям, например, страховым компаниям и финансовым учреждениям, приходится иметь дело с очень большим объемом деловых документов. Средства управления и доступа к деловой информации требуются предприятиям в самых разных сферах бизнеса.

Контент-сервер - это программная система, обеспечивающая хранение мультимедийных данных, деловых форм, документов и связанных с ними данных и метаданных и позволяющая сотрудникам работать с этим содержанием. Если не существует способа эффективно соединиться с разнородными контент-серверами, предприятие может впустую тратить время и деньги на дублирование информации или же приучить сотрудников выполнять поиск многократно.

Система Enterprise Information Portal представляет передовую технологию, которая делает доступными все информационные ресурсы предприятия с вашей рабочей станции. Соединяя клиент с разнородными серверами, IBM Enterprise Information Portal for Multiplatforms помогает добиться максимальной доступности информации и активов мультимедиа. С помощью IBM Enterprise Information Portal for Multiplatforms пользователи клиентов могут быстро и одновременно обращаться ко всем подключенным контент-серверам. Пользователи могут также выполнять анализ информации или “интеллектуальный” поиск в данных контент-серверов (включая серверы внутрикорпоративной сети и серверы в Интернете); кроме того, они могут выполнять задачи рабочих потоков, входящие в процессы обработки информации предприятия.

При помощи IBM Enterprise Information Portal for Multiplatforms можно настроить программы для вашего предприятия. Используя комплекты IBM Enterprise Information Portal for Multiplatforms, прикладные программисты могут создавать программы и для настольных систем, и для Web.

В этой главе приводится обзор IBM Enterprise Information Portal for Multiplatforms. Возможности и функции IBM Enterprise Information Portal for Multiplatforms показаны на примере работы вымышленной страховой компании XYZ Insurance.

Поиск информации о клиентах

XYZ Insurance (XYZ) - большая компания по страхованию имущества от несчастных случаев, она хранит много фотографий, страховых исков, полисов, актов оценки страхового имущества, докладов экспертов и других документов.

Компания XYZ хранит все письма держателям полисов, медицинские формы и формы оценки в цифровом виде в файловых картотеках Lotus Domino.Doc. Все условия полисов, уведомления и счета компания XYZ архивирует на сервере Content Manager OnDemand для длительного хранения и быстрого доступа к ним. Все формы страховых исков, фотографии и письма, полученные от держателей полисов, XYZ хранит в папке системы Content Manager for AS/400. Доклады экспертов XYZ хранит в менеджере каталогов данных Data Warehouse Center DB2 Universal Database (DB2 UDB) Кроме того, у компании XYZ есть мультимедийные активы, например, графика высокого разрешения, для отделов рекламы, связей с общественностью и развития, которые AA хранит в системе Content Manager. Информацию о процедурах действий компании XYZ хранит во внутренней сети компании.

Потребности

Для обработки страховых исков, ответов на звонки клиентов и общего обслуживания держателей полисов недостаточно содержания только одного сервера, так как сотрудникам необходим доступ ко всей информации о клиентах. Для обслуживания клиентов сотрудникам требуется одновременный доступ к разнообразным контент-серверам. Компании XYZ необходимо решение, которое позволит объединить различные контент-серверы и внутреннюю сеть для поиска и получения информации. Требуется также расширить использование рабочих потоков.

Доступ к документам нужен многим сотрудникам, в том числе клеркам, специалистам по искам и страховым агентам. Компания XYZ должна ограничить доступ к одним элементам, предоставив неограниченный доступ к другим. Кроме того, компании XYZ требуется простой в использовании интерфейс, не требующий много времени на освоение.

Решение

Компания XYZ Insurance использует IBM Enterprise Information Portal for Multiplatforms, поскольку его мощные технологии поиска позволяют выполнять поиск данных на всех контент-серверах. Теперь когда в Центр обработки запросов компании XYZ поступает запрос, для получения всей необходимой информации о держателе страхового полиса достаточно одной операции объединенного поиска.

Компания XYZ Insurance также использует возможность исследования информации в Enterprise Information Portal, чтобы искать информацию и извлекать ее из внутренней сети компании. Компания намеревается также расширить использование у себя рабочих потоков.

Решение Enterprise Information Portal

IBM Enterprise Information Portal for Multiplatforms - это комплексный продукт, его компоненты работают совместно и обеспечивают решение, которое идеально соответствует потребностям вашего предприятия. На базе многоуровневой архитектуры IBM Enterprise Information Portal for Multiplatforms предоставляет клиент управления для управления поисками, другие клиенты (например, для поиска) и соединители для соединения с разнородными контент-серверами, такими как Content Manager, Content Manager ImagePlus for OS/390, Content Manager OnDemand, Lotus Domino.Doc, DB2 UDB, DB2 DataJoiner, и менеджером каталогов данных DB2 Data Warehouse Center. Для других контент-серверов можно написать дополнительные соединители.

На рис. 1 показан принцип многоуровневой архитектуры IBM Enterprise Information Portal for Multiplatforms.

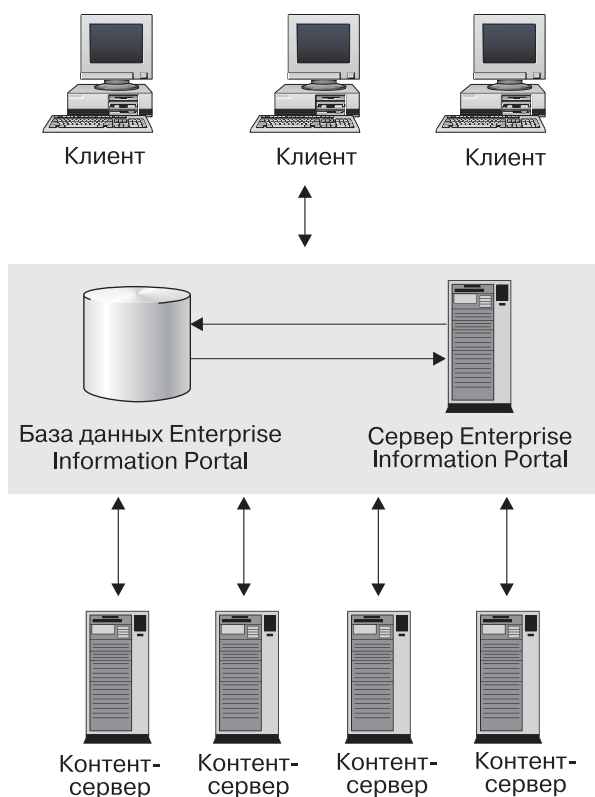


Рисунок 1. Многоуровневая архитектура

Архитектура IBM Enterprise Information Portal for Multiplatforms позволяет клиентским программам в одной операции осуществлять поиск сразу по

нескольким контент-серверам. Чтобы осуществить поиски, клиент использует шаблоны поиска, определенные администратором IBM Enterprise Information Portal for Multiplatforms.

Используя шаблоны поиска, клиент выполняет объединенный поиск. Это поиск, который выполняется одновременно на нескольких контент-серверах; собственные атрибуты этих серверов были отображаются на атрибуты объединения, используемые в шаблоне поиска. Шаблоны поиска IBM Enterprise Information Portal for Multiplatforms содержат критерии поиска, использующие атрибуты объединения, которые отображаются на собственные атрибуты каждого контент-сервера. Шаблоны поиска создает администратор IBM Enterprise Information Portal for Multiplatforms. IBM Enterprise Information Portal for Multiplatforms обеспечивает соединители в качестве общего интерфейса для взаимодействия с разнородными интерфейсами контент-серверов. Затем контент-серверы возвращают объекты данных клиенту.

Архитектура IBM Enterprise Information Portal for Multiplatforms обеспечивает следующие преимущества:

- Обращение по одному запросу сразу к нескольким различным контент-серверам, которые поддерживают транзакции электронной коммерции и программы обслуживания пользователей.
- Возможность исследование информации сразу с нескольких контент-серверов, в том числе и из Web.
- Доступ процесса рабочего потока к данным на нескольких различных контент-серверах.
- Поддержка разработки клиентских программ, не зависящих от расположения данных на определенных контент-серверах, благодаря разделению клиентских программ, индексов и данных.

Компоненты IBM Enterprise Information Portal for Multiplatforms

В этом разделе описаны все компоненты IBM Enterprise Information Portal for Multiplatforms. Эти компоненты поставляются как часть продукта IBM Enterprise Information Portal for Multiplatforms.

Управляющая база данных

База данных IBM Enterprise Information Portal for Multiplatforms - это база данных DB2 UDB. В ней хранится вся информация, необходимая для управления EIP и его компонентами.

Перенастройка вашей базы данных Enterprise Information Portal версия 7.1: до использования управляющей базы данных Enterprise Information Portal Версии 8.1 надо перенастроить ваши данные из Enterprise Information Portal Версии 7.1.

Клиент администратора

Системный администратор использует клиент администратора IBM Enterprise Information Portal for Multiplatforms чтобы:

- Определять контент-серверы для объединенного поиска.
- Указывать собственные элементы и атрибуты контент-серверов и их отображение на элементы объединения.
- Создавать шаблоны поиска.
- Определять пользователей, которые будут иметь доступ к шаблонам поиска, возможностям исследования информации и процессам рабочего потока, и управлять ими.
- Определять бизнес-процессы рабочего потока.

Эта информация хранится в базе данных IBM Enterprise Information Portal for Multiplatforms.

Для удобства рекомендуется установить клиент администратора на той же рабочей станции или сервере, на которой установлена база данных IBM Enterprise Information Portal for Multiplatforms.

Кроме того, у вас может быть сколько угодно клиентов администратора на других рабочих станциях. Чтобы установить такую конфигурацию, надо:

- Установить поддержку клиента DB2 и при помощи ассистента конфигурирования клиента DB2 сконфигурировать доступ к управляющей базе данных системы на каждой рабочей станции, на которой установлен клиент администратора.
- Использовать конфигурирование RMI (Remote Method Invocation - удаленный вызов метода), запустив сервер RMI, на котором установлена база данных IBM Enterprise Information Portal for Multiplatforms. Убедитесь, что в вашем файле `\CMBROOT\cmbclient.ini` указан этот сервер. Положение этого INI-файла задается ключевым словом `CMCOMMON` в файле `cmbsmenv.properties`. Этот файл можно также найти по адресу URL, заданному ключевым словом `CMCOMMON_URL` в файле `cmbsmenv.properties`.

Соединители

Классы соединителя позволяют программам клиента обращаться к контент-серверам и базе данных IBM Enterprise Information Portal for Multiplatforms. С IBM Enterprise Information Portal for Multiplatforms поставляются следующие соединители:

- Соединители реляционных баз данных (DB2, JDBC, ODBC)
- Соединитель объединения (с базой данных IBM Enterprise Information Portal for Multiplatforms)
- Content Manager Версия 8 Выпуск 2
- Соединитель Content Manager Версия 7 Выпуск 1
- Соединитель Content Manager OnDemand
- Соединитель Content Manager ImagePlus for OS/390
- Соединитель Content Manager for AS/400

- Соединитель Lotus Domino.Doc
- Соединитель Extended Search

Соединитель объединения содержит класс соединителя для базы данных IBM Enterprise Information Portal for Multiplatforms. Каждый соединитель контент-сервера содержит соответствующие классы соединителей.

В среде Java существуют и локальные, и удаленные версии Java соединителей. В C++ есть только локальные соединители. Локальные соединители - это набор классов соединителей, используемых для прямого соединения с различными серверами данных. Локальные соединители могут находиться на клиенте рабочего стола IBM Enterprise Information Portal for Multiplatforms или на сервере RMI IBM Enterprise Information Portal for Multiplatforms. Удаленные соединители используются для связи с контент-сервером через сервер RMI или элемент пула серверов RMI. При использовании удаленного соединителя отпадает потребность в прямом соединении с контент-сервером.

IBM eClient

eClient - это работающий через браузер пользовательский интерфейс для доступа к документам, находящимся в хранилищах Content Manager (все платформы), Content Manager OnDemand (все платформы) и Content Manager ImagePlus for OS/390.

Исследование информации

Компонент Исследование информации поддерживает лингвистические службы для поиска скрытой информации в текстовых документах на контент-серверах. При обработке текстовых документов создаются метаданные (информация о данных), для которых можно выполнять составление сводок, категоризацию и поиск. IBM Enterprise Information Portal for Multiplatforms содержит примеры, показывающие, как использовать возможности исследования информации в минимальном клиенте. Вы можете сконструировать свой собственный клиент рабочего стола или минимальный клиент, чтобы использовать исследование информации.

IBM Web Crawler

IBM Web Crawler позволяет искать данные в Web и импортировать их. После этого результаты поиска и метаданные можно проанализировать и категоризировать с помощью инструментов исследования информации. IBM Web Crawler может выполнять поиск на серверах http, ftp, ntp, lotus и domino.

Рабочий поток

С помощью Enterprise Information Portal вы можете управлять потоком работы вашего предприятия. Используя возможности рабочего потока Enterprise Information Portal, можно определять и выполнять процесс рабочего потока для рабочей группы, отдела или всего предприятия. С помощью графических средств построения в построителе рабочего потока Enterprise Information Portal можно

сконструировать всестороннее и простое для понимания представление процесса рабочего потока. После этого пользователи смогут использовать определенные рабочие потоки для выполнения своих задач, используя разработанный вами клиент или минимальные клиенты из примера Enterprise Information Portal.

Механизм текстового поиска Content Manager Версия 7

Этой возможностью можно пользоваться, чтобы выполнять автоматическую индексацию, поиск и вывод документов, хранящихся в Content Manager версии 7. Пользователи могут искать документы по словам или словосочетаниям.

Ограничение: сервер и клиент текстового поиска - это необязательная возможность Content Manager, которую вы можете сконфигурировать и выполнять только с серверами Content Manager до версии 8.1. Если вы не используете серверы Content Manager до версии 8.1, не устанавливайте эту возможность.

Сервер и клиент поиска изображений Content Manager Версия 7

Для этой возможности используется технология запроса по содержанию изображения (QBIC) IBM, которая позволяет искать объекты по их наглядным свойствам, таким как цвет и текстура.

Ограничение: Возможности сервера поиска изображений не поддерживаются ни в Enterprise Information Portal версии 8, ни в Content Manager версии 8. Его возможности доступны при использовании Content Manager до версии 8.1.

Что нового в API Версии 8.2

Всеобъемлющие примеры кодов

Примеры кодов программ были обновлены с учетом возможностей Content Manager Версии 8.2. Кроме того, для улучшения читаемости теперь все примеры кодов появляются внутри подокон с пометками соответствующего языка.

ограничения DB2 DataJoiner

Если контент-серверу требуется DB2 Universal Database Версии 8.1, соединитель EIP DataJoiner не будет работать из-за проблем связывания. Это причина, по которой DataJoiner 2.1 не распознает операторы SQL в файлах связывания DB2 Версии 7.

Решение этой проблемы можно найти в часто задаваемых вопросах (FAQ) на сайте DataJoiner (<http://www.software.ibm.com/data/datajoiner/>) : "Why am I having problems with binding on DB2 Universal Database Version 7 when connecting to a DataJoiner server?" . В этом решении для связи с сервером DataJoiner Версии 2.1.1 предлагается выполнить следующие действия:

1. Загрузите два файла связывания и переименуйте их в db2cliws_dj.bnd и db2clprt_dj.bnd.

2. Выполните следующие команды DB2:

```
db2cmd
db2 connect to user using
db2 bind db2cliws_dj.bnd grant public
db2 bind db2cliprr_dj.bnd grant public
```

Ограничения менеджера каталогов данных DB2 Data Warehouse Manager

Соединитель IC требует DB2 Universal Database Версии 7.2 и не будет работать с DB2 UDB Версии 8.1.

Папки объединения (только Java)

Enterprise Information Portal Версия 8 Выпуск 2 предоставляет специальные объекты объединения, которые можно назвать *папки объединения*. Папки объединения могут хранить собранные результаты запроса объединения, например, документ из Content Manager и связанные с ним документы с OnDemand. Потом эти результаты можно послать непосредственно в рабочий поток.

Поддержка Microsoft Visual Studio .NET

Интерфейсы API Enterprise Information Portal и Content Manager версии 8.2 (и новее) теперь поддерживают Microsoft Visual Studio .NET.

Что нового в API Версии 8.1

Enterprise Information Portal версии 8.1 обеспечивает беспрецедентный доступ к разнородным контент-серверам. Новые функции и компоненты этой версии:

- Возможности импорта XML:

Теперь XML можно использовать для импорта и экспорта содержимого в Content Manager через объекты DDO и XDO (с использованием вызовов API Java).

- Улучшенные процедуры установки
- Дополнительные соединители для реляционных баз данных:

Enterprise Information Portal предоставляет соединители реляционных баз данных для DB2 UDB, DB2 DataJoiner, менеджера каталогов данных DB2 Data Warehouse Manager и других баз данных через драйверы JDBC или ODBC.

- Расширенные возможности исследования информации и поиска:

Исследование информации предоставляет расширенные возможности текстового поиска с помощью гибкого запроса, который можно ограничить документами определенных категорий.

- Возможности рабочего потока:

Используя свойства рабочего потока Enterprise Information Portal, можно определять и выполнять процесс рабочего потока для рабочей группы, отдела или всего предприятия.

- Управление доступом на уровне объединения:

С помощью наборов привилегий и списков управления доступом можно управлять доступом к процессам исследования информации и рабочего потока Enterprise Information Portal. Дополнительное управление доступом к данным можно осуществить с помощью возможностей управления доступом каждого из контент-серверов.

- Дополнительная поддержка для Content Manager:
 - Получение списков, добавление, получение, изменение и удаление класса содержимого.
 - Асинхронное получение содержимого объекта.

Новые и измененные классы Java

Общие классы Java используются всеми соединителями. Этот пакет содержит интерфейсы (такие как `dkDatastore`) и абстрактные классы (такие как `dkAbstractDatastore` и `dkXDO`), а также конкретные классы (такие как `DKDDO`), используемые соединителями.

Новые классы:

- `dkAbstractAccessControlList`
- `dkAbstractAttrGroupDef`
- `dkAbstractAuthorizationMgmt`
- `dkAbstractConfigurationMgmt`
- `dkAbstractDatastoreAdmin`
- `dkAbstractDataObjectBase`
- `dkAbstractPrivilege`
- `dkAbstractPrivilegeGroup`
- `dkAbstractPrivilegeSet`
- `dkAbstractResultSetCursor`
- `dkAbstractUserDef`
- `dkAbstractUserMgmt`
- `dkAttrGroupDef`
- `dkAuthorizationMgmt`
- `dkCheckableObject`
- `DKChildCollection`
- `dkConfigurationMgmt`
- `DKLinkCollection`
- `dkPersistentCheckableObject`
- `dkPrivilege`
- `dkPrivilegeGroup`
- `dkUserDef`
- `dkUserGroupDef`

Измененные классы:

- dkAbstractDatastore
- dkAbstractDatastoreDef
- dkAbstractDatastoreExt
- dkAbstractEntityDef
- dkAccessControlList
- dkDataObjectBase
- dkDatastore
- dkDatastoreAdmin
- dkDatastoreDef
- dkDatastoreExt
- DKDDO
- dkEntityDef
- dkPersistentObject
- dkPrivilegeSet
- dkSearchTemplate
- dkSchemaMapping
- dkUserManagement

Изменения в поведении классов для Enterprise Information Portal Версии 8

Некоторые классы или компоненты классов EIP Версии 8.2 изменили свое поведение по сравнению с EIP Версии 7.1. Эти изменения описаны в настоящем разделе. Подробную информацию смотрите в *электронном справочнике по API*.

- dkIterator: next() перемещает итератор вперед в следующий элемент, и получает этот элемент; previous() перемещает итератор в предыдущий элемент и получает его
- dkXDO: getPidObject() и setPidObject(DKPid pid)

Новые и измененные классы C++

Всеми соединителями используется общий пакет классов C++. Этот пакет содержит интерфейсы (такие как dkDatastore) и абстрактные классы (такие как dkAbstractDatastore и dkXDO), а также конкретные классы (такие как DKDDO), используемые соединителями.

Новые классы:

- dkAbstractAccessControlList
- dkAbstractAttrGroupDef
- dkAbstractAuthorizationMgmt
- dkAbstractConfigurationMgmt
- dkAbstractDataObjectBase

- dkAbstractDatastoreAdmin
- dkAbstractPrivilege
- dkAbstractPrivilegeGroup
- dkAbstractPrivilegeSet
- dkAbstactResultSetCursor
- dkAbstractUserDef
- dkAbstractUserMgmt
- dkAttrGroupDef
- dkAuthorizationMgmt
- dkCheckableObject
- DKChildCollection
- dkConfigurationMgmt
- DKLinkCollection
- dkPersistentCheckableObject
- dkPrivilege
- dkPrivilegeGroup
- dkUserDef
- dkUserGroupDef

Измененные классы:

- dkAbstractDatastore
- dkAbstractDatastoreDef
- dkAbstractDatastoreExt
- dkAbstractEntityDef
- dkAccessControlList
- dkDataObjectBase
- dkDatastore
- dkDatastoreAdmin
- dkDatastoreDef
- dkDatastoreExt
- DKDDO
- dkEntityDef
- dkPersistentObject
- dkPrivilegeSet
- dkSchemaMapping
- dkSearchTemplate
- dkUserManagement

Изменения в поведении классов для Enterprise Information Portal Версии 8.2

Некоторые классы или компоненты классов EIP Версии 8.2 изменили свое поведение по сравнению с EIP Версии 7.1. Эти изменения описаны в настоящем разделе. Подробную информацию смотрите в *электронном справочнике по API*.

- Изменено использование DKString с DKAny в AIX. Чтобы получить DKString из DKAny, теперь DKAny нужно получить в переменную DKAny и назначить ее для переменной DKString.

Пример: В этом примере полагается, что объект DDO был получен вызовом API `dkResultSetCursor fetchNext()` с контент-сервера, и была задана переменная DDO, как показано ниже. Этот пример для поиска строчных значений выполняет цикл по элементам данных в объекте DDO.

```
DKDDO *ddo = NULL; DKAny any; DKString strTmp;
unsigned short data_id = 0;
unsigned short count = 0;

count = ddo->dataCount();
for (data_id = 1; data_id <= count; data_id++)
{
    any = ddo->getDataId(data_id);
    if (any.typeCode() == DKAny::tc_string)
    {
        strTmp = any.toString(); // Так можно получить DKString из DKAny
    }
}
```

- Для интерфейсов, реализующих `dkXDO`, изменены методы `getPidObject` и `setPidObject`.

Основные понятия прикладного программирования в Enterprise Information Portal

Enterprise Information Portal поддерживает объектно-ориентированные интерфейсы прикладного программирования (API), которые можно использовать для создания программ, обращающихся к реляционным данным, в том числе к данным мультимедиа, и выводящих эти данные. В этой главе приводится краткий обзор этих API и их интеграции в архитектуру Enterprise Information Portal; описаны основные понятия объектно-ориентированного программирования, на которых основаны эти API.

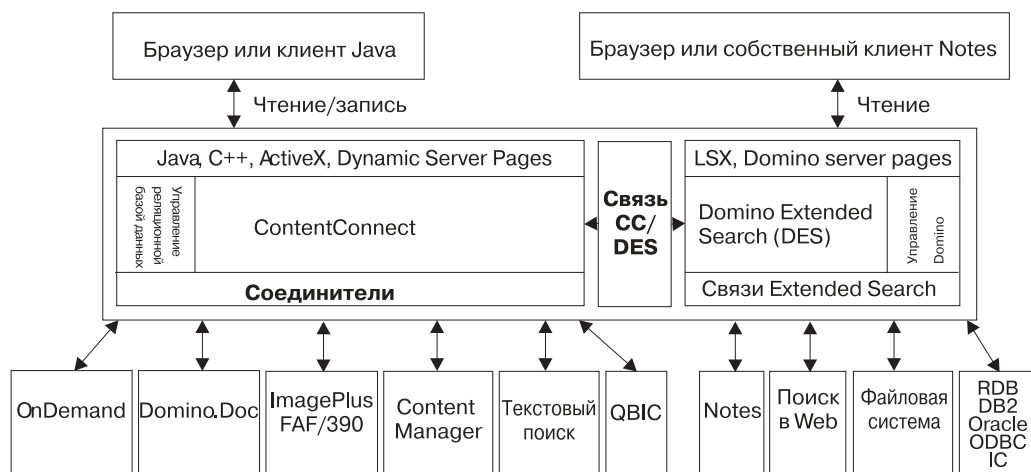


Рисунок 2. Структура Enterprise Information Portal

Как происходит обращение к данным через контент-серверы

Контент-сервер представляет собой систему хранения данных, совместимую с протоколом DDO/XDO. Контент-сервер поддерживает сеансы, соединения, транзакции, указатели и запросы. Прикладные программы, использующие интерфейсы прикладного программирования (API), описанные в этой книге, могут выполнять поддерживаемые на контент-серверах функции, такие как добавление, получение, изменение и удаление DDO. Enterprise Information Portal поддерживает следующие контент-серверы:

- Content Manager Версия 8 Выпуск 2
- Content Manager Версия 7 Выпуск 1
- Domino.Doc
- Extended Search

- ImagePlus for OS/390
- Content Manager OnDemand
- VisualInfo for AS/400
- DB2
- DB2 DataJoiner
- Каталог данных
- Диспетчер информационного каталога DB2 Warehouse Manager
- Серверы JDBC/ODBC

Программы, использующие Enterprise Information Portal, могут создавать контент-сервер объединения, который действует как обычный сервер. Классы объединения Enterprise Information Portal поддерживают объединенный поиск, получение и изменение информации на нескольких контент-серверах.

На контент-сервере объединения Enterprise Information Portal и на каждом из контент-серверов используются различные схемы. Для интеграции нескольких разнотипных контент-серверов в одну систему объединения требуется преобразование и отображение.

Функции отображения схемы предоставляют информацию о схеме каждого контент-сервера. Информация, получаемая из отображения схемы, используется при объединенном поиске, сборании объединения и управлении системой EIP. Enterprise Information Portal хранит эту схему и отображения, а также другую управляющую информацию в своей управляющей базе данных.

Основные понятия динамических объектов данных

В соответствии со спецификациями Object Management Groups's (OMG) CORBA Persistent Object Service и Object Query Service, в Enterprise Information Portal реализованы динамические объекты данных (DDO) и их расширение - расширенные объекты данных (XDO), входящие в протоколы CORBA Persistent Data Service (PDS). Понятия DDO и XDO не являются специфичными для конкретных контент-серверов, их можно использовать для представления объектов данных в любой системе управления базами данных, поддерживаемой Enterprise Information Portal.

Динамический объект данных - это интерфейс для записи данных на контент-сервер и получения их с контент-сервера. DDO существуют в программе и перестают существовать после завершения работы программы.

Динамические объекты данных (DDO)

DDO - это независимое от контент-сервера представление постоянных данных объекта. Его назначение - содержать все данные одного постоянного объекта.

Он обеспечивает также интерфейс для получения постоянных данных с контент-сервера и записи их на контент-сервер.

У DDO есть единый постоянный ID (PID), тип объекта и набор элементов данных, количество которых называется количеством данных. У каждого элемента данных есть имя, значение, ID, одно или несколько свойств данных и количество свойств данных. У каждого свойства данных есть ID, имя и значение.

Например, DDO может представлять строку из таблицы базы данных; элементам данных DDO и их свойствам соответствуют столбцы этой строки. DDO может содержать один или несколько расширенных объектов данных (XDO), представляющих собой нетрадиционные типы данных. На рис. 3 показаны динамические объекты данных и элементы данных.

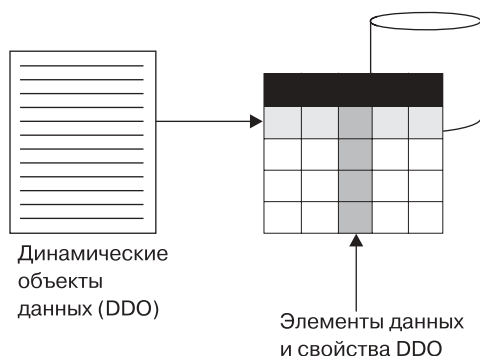


Рисунок 3. Динамические объекты данных и элементы

Расширенные объекты данных (XDO)

XDO - это представление сложных мультимедийных данных, например, ресурсного элемента из Content Manager, в котором хранится изображение или документ, или нового типа данных, поддерживаемого объектно-реляционными возможностями реляционной базы данных, например, модулями расширения IBM DB2.

XDO дополняют DDO, поддерживая хранение мультимедийных данных сложных типов и функции, которые обеспечивают требуемое поведение данных таких типов в программах. XDO могут содержаться в DDO или принадлежать DDO, чтобы представлять объект сложных мультимедийных данных.

У XDO есть набор свойств, содержащих такую информацию, как типы и ID данных. XDO могут быть также отдельными динамическими объектами. На рис. 4 на стр. 16 показан пример XDO.

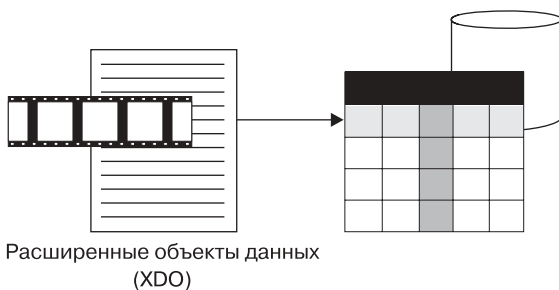


Рисунок 4. Расширенные объекты данных (XDO)

Представление мультимедийного содержимого

DDO и XDO могут представлять объекты данных любого типа и структуры. Например, DDO может представлять фильм. Такой DDO содержит несколько элементов данных, представляющих такие атрибуты фильма, как **Фамилия** режиссера или **Название фильма**, и мультимедийные XDO, представляющие мультимедийные данные фильма - видеоклипы или отдельные кадры.

В Content Manager 8.2 DDO состоит из всех метаданных, описывающих объект, например, документ или изображение. XDO расширяет возможности DDO, поддерживая ресурсное содержимое. Ресурсное содержимое - это содержимое любого типа, начиная от двоичных данных или текста и заканчивая видео- и аудиопотоками. Элемент, реализующий XDO (и, соответственно, являющийся XDO), является также и DDO и поддерживает все функции, предоставляемые DDO, а также дополнительные функции, предоставляемые XDO.

Контент-серверы и объекты DDO

DDO создаются и динамически связываются с контент-сервером. Связь между DDO и контент-сервером устанавливается посредством PID DDO.

Обычно прикладная программа Enterprise Information Portal для помещения данных на контент-сервер или взятия данных с контент-сервера выполняет следующие пять действий:

1. Создает контент-сервер.
2. Устанавливает соединение с этим контент-сервером.
3. Создает DDO, с которыми будет работать, и связывает контент-сервер с этими DDO.
4. Добавляет, получает, изменяет и удаляет эти DDO при помощи соответствующих методов.
5. Закрывает соединение и уничтожает контент-сервер.

Сравнение DDO/XDO со значениями атрибутов и частями элементов

DDO соответствует элементу в Enterprise Information Portal. Тип объекта DDO соответствует типу элемента, связанному с элементом. Элементы данных DDO соответствуют атрибутам элемента. Например, в Content Manager тип элемента создается с некоторым набором атрибутов, а элемент всегда индексируется по типу элемента.

DDO может содержать один или несколько XDO, которые соответствуют частям элемента в Enterprise Information Portal.

Постоянные идентификаторы (PID)

Постоянный идентификатор (PID) однозначно идентифицирует постоянный объект на любом контент-сервере. PID DDO состоит из ID элемента, имени контент-сервера и другой связанной с объектом информации. При добавлении DDO на контент-сервер система назначает DDO уникальный PID.

Поскольку DDO - это динамический интерфейс для постоянных данных, которые записываются на контент-сервер или получаются с контент-сервера, разные DDO могут представлять одни и те же объекты постоянных данных, и у этих DDO может быть один и тот же PID. Например, один DDO можно создать для перемещения объекта данных на контент-сервер для постоянного хранения, а другой DDO - для резервирования того же объекта данных на контент-сервере при его изменении. В таком случае оба DDO будут использовать одно и то же значение PID.

Работа с контент-сервером объединения и с объединенным поиском

Объединенный поиск - это процесс поиска данных на одном или нескольких контент-серверах. Для объединенного поиска используется объект DKDatastoreFed. Объединенный поиск работает с классами - конкретными реализациями dkDatastore, dkDatastoreDef и других связанных с ними классов, поддерживающих объединенный поиск. Конкретные классы объединенного поиска взаимодействуют с другими общими классами, такими как классы для запросов, собраний и других объектов, и являются частью структуры Enterprise Information Portal.

Классы объединенного поиска работают с разными контент-серверами, например с Content Manager ImagePlus for OS/390 или Domino.Doc. Эти классы предоставляют набор общих функций для операций объединенного поиска и доступа к контент-серверам. Это общее представление называется *модель документов объединения*, оно показано на рис. 5.

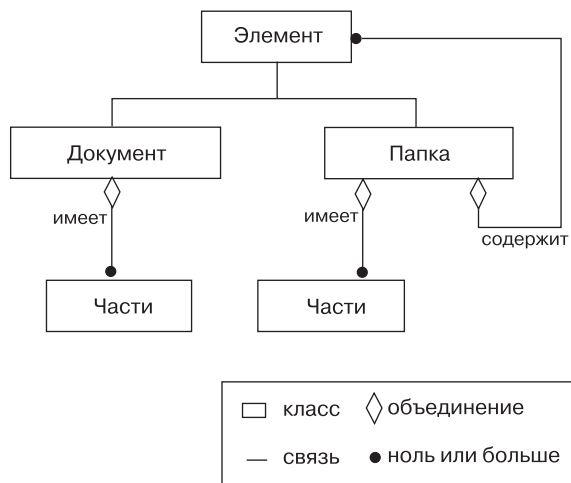


Рисунок 5. Представление документов объединения

Элемент может быть документом или папкой. Документ может не содержать частей или содержать одну или несколько частей. Папка может содержать ноль или более элементов, это могут быть документы или другие папки.

Не все контент-серверы поддерживают модель документов объединения. Например, в базе данных DB2 нет понятия папок или частей. В таблице DB2 или

другой реляционной базы данных элемент отображается на строку; такой вариант используется, если контент-сервер не поддерживает документы или папки.

В общем случае документ представляется в вашей программе динамическим объектом данных (DDO) - объектом данных, содержащим свое описание и предназначенным для передачи данных на сервер данных и обратно. Сам объект DDO имеет обобщенную структуру и поддерживает разнообразные модели. Он не ограничен моделью документов объединения. Такая гибкость позволяет объектам DDO представлять данные различных контент-серверов со своими различными моделями данных.

Объект - это объект контент-сервера, состоящий из атрибутов. *Атрибуты* - это обозначение метаданных на контент-серверах; например, профили, поля и ключевые слова на контент-серверах Domino.Doc - это атрибуты.

На каждом контент-сервере для описания модели, поддерживаемой этим складом данных, используется своя собственная терминология. В Табл. 3 сопоставлена терминология, используемая на различных контент-серверах, с терминологией модели объединения:

Таблица 3. Соответствие терминов для различных контент-серверов

| Контент-сервер | Источник данных | Объект | Атрибут | Представление |
|----------------------------|---|--|---|--|
| Content Manager 8.2 | Библиотечный сервер | тип элемента | атрибут | представление типа элемента или поднабор типа элемента |
| Content Manager 7.1 | Библиотечный сервер | индексный класс | <ul style="list-style-type: none"> • атрибут • ключевой атрибут | представление индексного класса |
| OnDemand | Сервер OnDemand | <ul style="list-style-type: none"> • группа программ • папка | <ul style="list-style-type: none"> • поле • критерий | Нет |
| ImagePlus | Сервер ImagePlus for OS/390 | объект | атрибут | Нет |
| Content Manager for AS/400 | Content Manager for AS/400сервер (server) | индексный класс | атрибут | представление индексного класса |

Таблица 3. Соответствие терминов для различных контент-серверов (продолжение)

| Контент-сервер | Источник данных | Объект | Атрибут | Представление |
|---|--|------------------------------------|-----------------------------------|---------------------|
| Domino.Doc | Сервер Domino | библиотека зал шкаф папка | профиль поле ключевое слово | Нет |
| Extended Search | Сервер Extended Search | имя базы данных | имя базы данных | Нет |
| Реляционная база данных | IBM DB2 UDB, JDBC, ODBC, IBM DB2 DataJoiner | таблица | столбец | производная таблица |
| Каталог данных | Диспетчер информационного каталога DB2 Warehouse Manager | индексный класс | свойство | |
| контент-сервер объединения | сервер отображения | отображенный объект объединения | отображенный атрибут объединения | шаблон поиска |
| Контент-сервер объединения, в котором могут храниться папки объединения | сервер | объект объединения | атрибут объединения | папка объединения |

На рис. 6 на стр. 22 показан объединенный поиск. Объединенный поиск на контент-сервере объединения Enterprise Information Portal выполняется через шаблоны поиска. Затем контент-сервер объединения вызывает поиски по этим шаблонам в отдельных контент-серверах, чтобы выполнить реальный поиск на контент-серверах. При этом используется отображение схемы.

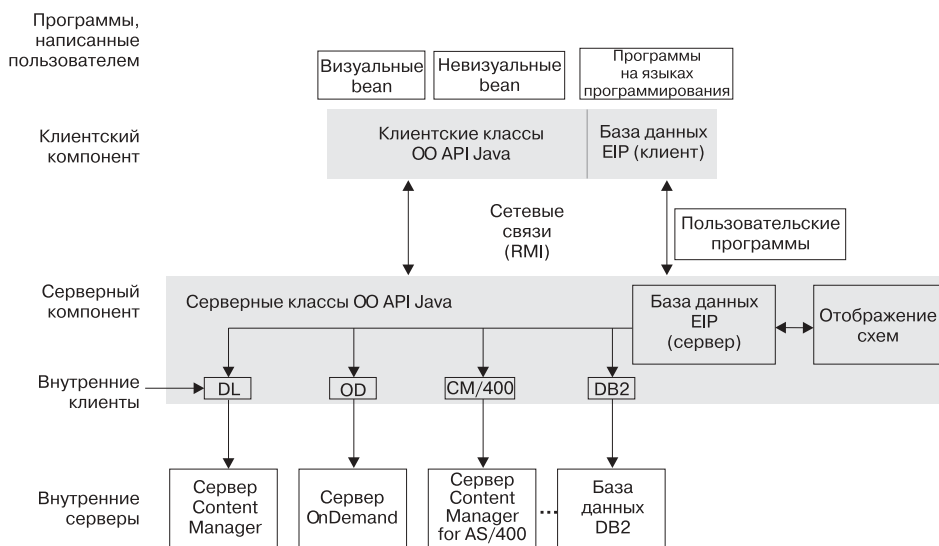


Рисунок 6. Структура объединенного поиска

Контент-сервер объединения может использовать для соединения с контент-серверами локальные или удаленные соединители; для связи может применяться RMI. Можно также разрабатывать прикладные программы на основе классов API.

Отображение схемы объединения

Отображение схемы - это соответствие между схемой на контент-сервере и структурой элементов, которые пользователь хочет обрабатывать в программе. *Схема объединения* - это схема понятий контент-сервера объединения Enterprise Information Portal, она определяет отображение между понятиями на контент-сервере объединения и понятиями каждого определенного контент-сервера. Отображение схемы позволяет снять различия между тем, как физически хранятся данные, и тем, как пользователь хочет их обрабатывать в программе.

Информация отображения представляется в памяти классами отображения схемы.

Использование компонентов отображения контент-сервера объединения

Кроме информации отображения схемы, для отображения объектов и атрибутов контент-сервер объединения должен также иметь доступ к следующей информации:

Отображение ID пользователей и паролей

Для обеспечения однократной регистрации в системе каждый ID

пользователя в Enterprise Information Portal можно отобразить на соответствующий ID пользователя на каждом контент-сервере.

Регистрация контент-сервера

Каждый контент-сервер надо зарегистрировать, чтобы Enterprise Information Portal мог найти его и зарегистрироваться на нем.

ID пользователя и информация о контент-сервере хранится в управляющей базе данных Enterprise Information Portal.

Выполнение запросов объединения

Чтобы выполнить объединенный поиск при помощи API, сначала надо создать строку запроса объединения. Потом этот запрос можно будет создать и выполнить различными способами:

- Можно создать объект запроса объединения DKFederatedQuery, передав ему эту строку запроса; после этого, чтобы обработать запрос, вызвать метод execute или evaluate для этого объекта.
- Можно передать эту строку запроса методу execute или evaluate контент-сервера объединения, чтобы выполнить запрос непосредственно.

Строка запроса преобразуется в форму запроса объединения - независимое от контент-сервера представление запроса. Форма запроса объединения служит входной формой для объединенного поиска.

Если запрос приходит от прикладной программы, использующей графический интерфейс пользователя, синтаксический анализ строки не нужен, и соответствующая форма запроса объединения может быть построена непосредственно.

При выполнении объединенного поиска Enterprise Information Portal выполняет следующие шаги:

- Перевод канонической формы запроса в несколько запросов, которые могут выполняться на каждом контент-сервере. Необходимая для этого преобразования информация берется из отображения схемы.
- Преобразование объектов и атрибутов объединения в объекты и атрибуты каждого контент-сервера. Этот процесс использует механизмы отображения и преобразования, описанные в отображении схемы.
- Фильтрация неподходящих данных при построении запросов для серверов.
- Формирование собственных запросов и передача их конкретным контент-серверам.

Каждый контент-сервер обрабатывает переданный ему запрос. Результаты возвращаются запросу объединения, который может обрабатывать их следующим образом:

- Преобразовать исходные объекты и атрибуты в объединенные объекты и атрибуты в соответствии с информацией отображения.
- Отфильтровать результаты, оставляя только запрошенные данные.
- Объединить результаты, полученные от нескольких контент-серверов, в собрание объединения.

Результат запроса объединения возвращается в виде собрания объединения. Чтобы обращаться к отдельным членам собрания, можно создать итератор. Каждый вызов метода next этого итератора возвращает объект DKDDO - динамический объект, независимый от контент-сервера.

Собрание объединения дает возможность разделить результаты запроса по их контент-серверам. Создайте последовательный итератор, вызвав метод createMemberIterator собрания объединения. С помощью этого последовательного итератора можно обращаться к каждому члену собрания - объекту DKResults - и обрабатывать его отдельно.

Компоненты объединенного поиска и отношения между ними показаны на рис. 7.

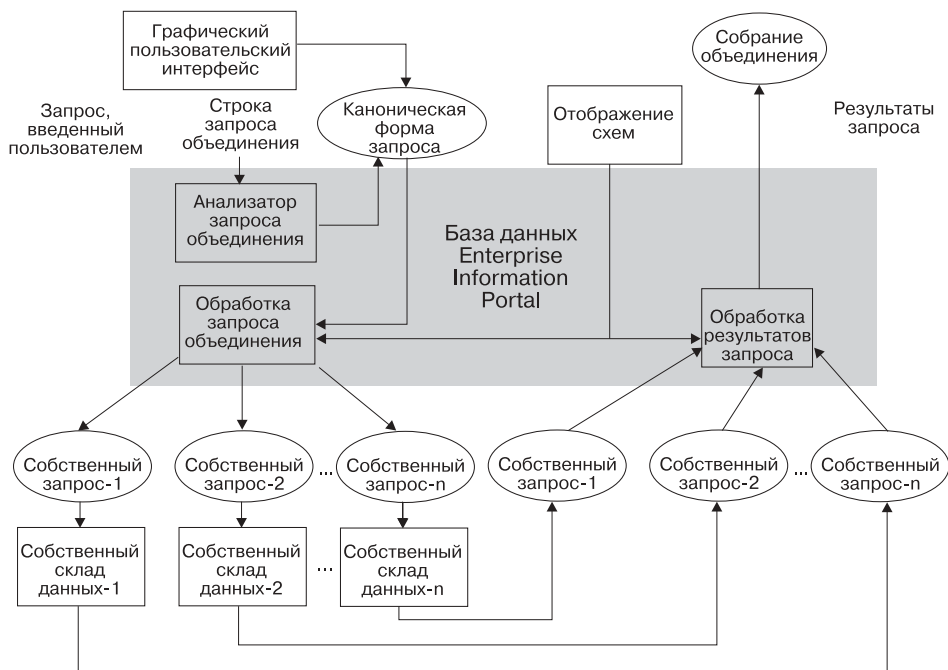


Рисунок 7. Обработка запроса объединения

Синтаксис запроса объединения

Создаваемый запрос объединения должен быть записан синтаксически правильно, как показано ниже. Контент-сервер объединения не поддерживает запросы изображений.

```
PARAMETRIC_SEARCH=( [ENTITY=имя_объекта,]
                     [MAX_RESULTS=число_результатов,]
                     [COND=(условное_выражение)]
                     [; ...]
                     );
[OPTION=( [CONTENT=только_атрибут_yes_no]
          )]

[and

TEXT_SEARCH=(COND=(выражение_текстового_поиска)
              );
[OPTION=( [SEARCH_INDEX={имя_индекса_поиска | (список_индексов) };]
          [ASSOCIATED_ENTITY={связанное_имя_объекта});]
          [MAX_RESULTS=число_результатов;]
          [TIME_LIMIT=предельное_время]
          )]
]
```

Оператор NOT не поддерживается в объединенном поиске.

Примеры строк запросов объединения

Параметрический запрос объединения с использованием операции LIKE

```
"PARAMETRIC_SEARCH = ( ENTITY = F_DGSAMP71, MAX_RESULTS = 5,
COND = (fName LIKE '%'))"
```

Параметрический запрос объединения с использованием операторов LIKE и >

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 5,
COND = (fJTitle LIKE 'Java%' AND fJNumPages > 20) )"
```

Параметрический запрос объединения с использованием операторов LIKE и <

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 5,
COND = (fJTitle LIKE 'Java%' AND fJNumPages < 20) )"
```

Параметрический запрос объединения с использованием операции BETWEEN

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJNumPages BETWEEN 5 200) )"
```

Если для MAX_RESULTS задано значение ноль, возвращаются все результаты.

Параметрический запрос объединения с использованием операции NOTBETWEEN

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJNumPages NOTBETWEEN 5 100) )"
```

Параметрический запрос объединения с использованием операции IN

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJArticleTitle IN ('Java', 'Multi-Disk B-trees.',
'On Beyond Data.', 'IBM')) )"

```

Параметрический запрос объединения с использованием операции NOTIN

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJArticleTitle NOTIN ('Java', 'Multi-Disk B-trees.',
'On Beyond Data.', 'IBM')) )"

```

Параметрический запрос объединения с использованием операции ==

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJEditorName == 'Harth') )"

```

Параметрический запрос объединения с использованием операции <>

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJSectionTitle <> 'не доступен') )"

```

Параметрический запрос объединения с использованием операций AND и OR

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = ((fJTitle LIKE '%Java%') OR ((fJEditorName<NULL) AND
(fJArticleTitle LIKE 'Computer%')))) ); OPTION = (CONTENT = YES)"

```

Параметрический запрос объединения с использованием операции

CONTAINS_TEXT_IN_CONTENT

Этот пример находит текст в содержимом. Содержимое может быть словом или предложением. Он допустим, только если объект объединения, допускающий текстовый поиск (FedTextResource) отображается на тип элементов Content Manager Версии 8, допускающий текстовый поиск, или на объект Extended Search, допускающий текстовый поиск.

```
"PARAMETRIC_SEARCH = ( ENTITY = FedTextResource, MAX_RESULTS = 6,
COND = ( CONTAINS_TEXT_IN_CONTENT 'XML' ) ); OPTION =
( CONTENT = YES )"

```

Параметрический запрос объединения с использованием операции

CONTAINS_TEXT

Ищет текст в значениях атрибутов. Этот текст может быть словом или предложением. Он допустим, только если атрибут объединения, допускающий текстовый поиск (fJTitle) отображается на атрибут Content Manager Версии 8, допускающий текстовый поиск, или на атрибут Extended Search, допускающий текстовый поиск.

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal, MAX_RESULTS = 0,
COND = (fJTitle CONTAINS_TEXT 'Java') )"

```

Текстовый запрос объединения

Ищет текст в содержимом. Этот текст может быть словом или предложением. Ключевое слово ASSOCIATED_ENTITY можно применять, только если объект объединения допускает текстовый поиск. Он допустим, только если объект объединения, допускающий текстовый

поиск (FedEntity) отображается на тип элементов Content Manager Версии 8, допускающий текстовый поиск, или на объект Extended Search, допускающий текстовый поиск.

```
"TEXT_SEARCH = ( COND = ('XML') ); OPTION = ( ASSOCIATED_ENTITY=FedEntity )"
```

Текстовый запрос объединения

Ищет текст в содержимом. Этот текст может быть словом или предложением. Текстовый индекс объединения, FedTMINDEX, отображается на поисковый индекс Text Miner продукта Content Manager Версии 7. Ключевое слово SEARCH_INDEX применимо только для этого типа отображения. Чтобы задать слово или фразу в условии, нужно при определении сервера Content Manager Версии 7, который поддерживает текстовый поиск, задать строку конфигурации GENFEDTEXTQRY=YES.

```
"TEXT_SEARCH = ( COND = ('операционная система') );  
OPTION = ( SEARCH_INDEX = FedTMINDEX)"
```

Текстовый запрос объединения

Находит текст в содержимом Content Manager Версии 7, Content Manager Версии 8 и Extended Search. Этот текст может быть словом или предложением. Текстовый индекс объединения, FedTMINDEX, отображается на поисковый индекс Text Miner продукта Content Manager Версии 7. Ключевое слово SEARCH_INDEX применимо только для этого типа отображения. Чтобы задать слово или фразу в условии, нужно при определении сервера Content Manager Версии 7, который поддерживает текстовый поиск, задать строку конфигурации GENFEDTEXTQRY=YES. Ключевое слово ASSOCIATED_ENTITY можно применять, только если объект объединения допускает текстовый поиск. Он допустим, только если объект объединения, допускающий текстовый поиск (FedTextResource) отображается на тип элементов Content Manager Версии 8, допускающий текстовый поиск, или на объект Extended Search, допускающий текстовый поиск.

```
"TEXT_SEARCH = ( COND = ( 'операционная система' ) ); OPTION =  
( SEARCH_INDEX = FedTMINDEX; ASSOCIATED_ENTITY = FedTextResource;  
MAX_RESULTS = 5 )"
```

Параметрический и текстовый запрос объединения с использованием операции OR

```
"PARAMETRIC_SEARCH = ( ENTITY = FedTextResource,  
AX_RESULTS = 0,COND = (FedTextResourceJTitle LIKE '%test%'  
) ) OR TEXT_SEARCH =( COND = ('UNIX') );  
OPTION = ( ASSOCIATED_ENTITY =  
FedTextResource; MAX_RESULTS = 4 )"
```

Параметрический и текстовый запрос объединения с использованием операции AND

```
"PARAMETRIC_SEARCH = ( ENTITY = FedTextResource, COND =  
(FedTextResourceJTitle LIKE '%test%') ) AND TEXT_SEARCH =  
( COND = ('UNIX') ); OPTION = ( ASSOCIATED_ENTITY =  
FedTextResource)"
```

Параметрический и текстовый запрос объединения по атрибутам с использованием операции OR

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal,  
COND = (fJTitle LIKE 'Java%' OR fJArticleTitle  
CONTAINS_TEXT 'База данных') );OPTION = ( CONTENT = ATTRONLY )"
```

Параметрический и текстовый запрос объединения по атрибутам с использованием операции OR

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDb_Journal,  
COND = (fJTitle LIKE 'Java%' OR fJArticleTitle  
CONTAINS_TEXT 'База данных') );OPTION = ( CONTENT = YES )"
```

Сохранение результатов поиска в папках объединения (только Java)

Теперь Enterprise Information Portal Версия 8 Выпуск 2 предоставляет специальные объекты объединения, в которых могут храниться *папки объединения*. В папках объединения могут храниться совмещенные результаты запроса объединения, например, документ из Content Manager и связанный с ним документ из OnDemand. Затем можно отправить результаты прямо в рабочий поток.

EIP хранит папки в виде DDO, которые можно добавить в собрание DKFolder. Кроме того, эти папки можно хранить в виде XDO в собрании DKParts.

Специальный объект объединения хранит папки в дополнительных таблицах. Все остальные функции (например, запросы, API и атрибуты) работают так же, как и в обычном объекте объединения. Специальный объект объединения хранит в папках только PID DDO.

Если вы не хотите отображать специальный объект объединения, запрос объединения будет выполнять поиск только в специальных объектах объединения. Если вы отобразите специальный объект объединения на другие собственные объекты, запрос объединения будет дополнительно выполнять поиск и в этих объектах.

Примеры кода можно найти в каталоге CMBROOT\dk\samples.

Работа с управлением системой

Enterprise Information Portal обеспечивает для доступа к функциям управления системой классы и вызовы API. Информацию о конкретных классах смотрите в *электронном справочнике по API*.

Настройка клиента администратора системы EIP

Клиент администратора системы EIP поддерживает расширение программ управления системой, включающее функции настройки:

- Диалоговые окна пользователя и группы в клиенте администратора системы EIP можно заменить своими собственными диалоговыми окнами.

- В иерархию клиента администратора системы EIP можно добавить новые узлы.
- В меню Инструменты клиента администратора системы можно добавить новые пункты.

Обработчики пользователей можно вызывать как до регистрации в клиенте администратора системы EIP, так и после нее.

Создание программ с использованием интерфейсов прикладного программирования (API)

Интерфейсы прикладного программирования (API) - это набор классов для доступа к локальным или удаленным данным и для управления этими данными. В этом разделе описываются API, реализация функций множественного поиска и связи с Интернетом.

Эти API поддерживают:

- Единую объектную модель для доступа к данным.
- Множественный поиск и внесение изменений для сочетания разнородных контент-серверов.
- Гибкий механизм использования комбинации механизмов поиска; например, механизма текстового поиска Content Manager.
- Возможность рабочего потока.
- Функции управления.
- Только для **Java**: Реализация конфигурации клиент-сервер в прикладной программе Java.

Поддержка многопоточности для API Java полностью доступна только для серверов Windows. Серверы и клиенты AIX и клиенты Windows не могут поддерживать многопоточность.

Различия между API Java и C++

Ниже перечислены различия между наборами API Java и C++ в IBM Enterprise Information Portal for Multiplatforms:

- Операции, определенные в API C++, не определены в API Java. Они поддерживаются как функции Java.
- Для представления общего объекта вместо класса C++ DKAny используется объект класса Java (java.lang.Object).
- Общие и глобальные константы определены в API Java в интерфейсе DKConstant; в C++ они находятся в DKConstant.h.
- API Java используют сборщик мусора языка Java.
- Функции Java DKDDO.toXML() и DKDDO.fromXML() не доступны в C++.

Архитектура клиент-сервер (только Java)

API Java предоставляют удобный интерфейс программирования для создателей прикладных программ. Эти API могут располагаться как на сервере EIP, так и на клиенте (в обоих случаях их интерфейс совпадает). Для обращения к данным через сеть API клиента связывается с сервером, используя RMI (Remote Method Invocation - удаленный вызов метода) Java. Связь между клиентом и сервером обеспечивается классами; никакие дополнительные программы не требуются.

Классы API состоят из следующих пакетов: `server`, `client`, `cs` и `common`. Классы клиента и сервера обеспечивают одни и те же API, но реализованы по-разному.

- Пакет сервера называется `com.ibm.mm.sdk.server`. Классы в пакете сервера связываются напрямую с контент-сервером объединения или конечным контент-сервером.
- Пакет клиента называется `com.ibm.mm.sdk.client`. Классы в пакете клиента связываются классами в пакете сервера через RMI.
- Классы пакета `common` совместно используются клиентом и сервером. В некоторых случаях прикладная программа не знает, где находятся данные. Например, данные прикладной программы могут в один момент времени располагаться на клиенте, а в другой - на сервере. Пакет `cs` динамически соединяется с клиентом и сервером.

Прикладная программа клиента должна импортировать пакет `client`, динамическая прикладная программа должна импортировать пакет `cs`, а прикладная программа сервера должна импортировать пакет `server`.

Хотя и для клиента, и для сервера поддерживаются одни и те же API, у пакета `client` есть дополнительные API, поскольку он связывается с пакетом `server`. Но имейте в виду, что интерфейс клиент-сервер поддерживается не для всех соединений. Например, он не поддерживается для CM V8.

Пакеты для среды Java

API Enterprise Information Portal находятся в четырех пакетах, входящих в `com.ibm.mm.sdk`: `common`, `server`, `client` и `cs`.

server (com.ibm.mm.sdk.server)

Обращается к контент-серверу и работает с его информацией

client (com.ibm.mm.sdk.client)

Связывается с пакетом `server`, используя вызов удаленного метода (RMI)

common (com.ibm.mm.sdk.common)

Общие классы для пакетов `server`, `client` и `cs`

cs (com.ibm.mm.sdk.cs)

Динамически связывается с клиентом или с сервером

Прикладная программа должна использовать пакет `common`, а также пакет `server` для локальных прикладных программ, или пакет `client` для прикладных программ, обращающихся к удаленному серверу, или пакет `cs`.

Советы по программированию

Не импортируйте в одну программу пакеты `client` и `server`. Если вы разрабатываете прикладную программу клиента, импортируйте пакет `client`. В противном случае импортируйте пакет `server`. Если местонахождение данных неизвестно, используйте пакет `cs` (с пакетами `server` или `client`). При импорте нескольких пакетов могут возникнуть ошибки компиляции.

В реализации некоторых соединителей есть вызовы кода C. Для этих соединителей в прикладных программах Web, которым требуется чистый интерфейс Java, используйте пакет `client`. При создании пакета `client` использованы только программы Java; пакет `server` может содержать вызовы JNI.

Поскольку клиенту требуется исключительная ситуация `java.rmi.RemoteException`, всегда подключайте в программу эту исключительную ситуацию, независимо от того, где выполняется эта программа - на клиенте или на сервере.

Настройка среды Java (только Java)

При настройке среды Windows, AIX или Solaris нужно импортировать следующие пакеты:

пакет `server`

Импортируется, когда контент-сервер и прикладная программа находятся на сервере.

- `com.ibm.mm.sdk.common`
- `com.ibm.mm.sdk.server`

пакет `client`

Импортируется, когда контент-сервер и прикладная программа находятся на клиенте.

- `com.ibm.mm.sdk.common`
- `com.ibm.mm.sdk.client`

пакет `cs`

Импортируется, когда положение контент-сервера не совпадает с положением прикладной программы.

- `com.ibm.mm.sdk.common`
- `com.ibm.mm.sdk.cs`

Задание переменных среды Java для Windows

Командную строку Windows со средой, настроенной для разработки программ Enterprise Information Portal, можно открыть, выбрав **Пуск → Программы → IBM Enterprise Information Portal for Multiplatforms 8.2 → Окно Разработки**. Другой способ - задать среду, запустив в командной строке Windows файл `cmbenv81.bat`.

Если хотите, можете изменить переменные среды:

PATH Переменная PATH должна содержать `X:\CMBROOT\DLL`, где X - диск, где установлен Enterprise Information Portal.

CLASSPATH

Переменная CLASSPATH должна содержать `X:\CMBROOT\LIB\xxx`, где X - диск, где установлен Enterprise Information Portal, а xxx - файлы .jar (например, `cmbfed81.jar`).

Задание переменных среды Java для AIX

В среде AIX можно использовать сценарий оболочки `cmbenv81.sh`, для настройки среду разработки прикладных программ EIP.

Если вы не пользуетесь сценарием, то надо задать следующие переменные среды:

PATH Переменная PATH должна содержать `/usr/lpp/cmb/lib`

LIBPATH

Переменная LIBPATH должна содержать `/usr/lpp/cmb/lib`

LD_LIBRARY_PATH

Переменная LD_LIBRARY_PATH должна содержать `/usr/lpp/cmb/lib`

CLASSPATH

Переменная CLASSPATH должна содержать `/usr/lpp/cmb/lib/xxx`, где xxx - файлы .jar (например, `cmbfed81.jar`)

Для правильного выравнивания объектов используйте опцию компилятора `-qalign=packed`.

Задание переменных среды Java для Solaris

В среде Solaris можно использовать пакетный файл, `cmbenv81.sh`, чтобы задать среду для разработки прикладных программ EIP.

Если вы не пользуетесь сценарием, то надо задать следующие переменные среды:

PATH Переменная PATH должна содержать `/opt/cmb/lib`

LIBPATH

Переменная LIBPATH должна содержать `/opt/cmb/lib`

LD_LIBRARY_PATH

Переменная LD_LIBRARY_PATH должна содержать /opt/cmb/lib

CLASSPATH

Переменная CLASSPATH должна содержать /opt/cmb/lib/xxx, где xxx - файлы .jar (например, cmbfed81.jar)

Для правильного выравнивания объектов используйте опцию компилятора -qalign=packed.

Использование вызова удаленного метода (RMI) с контент-серверами

Поскольку для обращения к данным через сеть классы клиента в API Java должны связываться с классами сервера, клиент и сервер должны быть подготовлены для работы в режиме клиент/сервер. На компьютере сервера должен быть запущен сервер RMI, получающий запросы от клиента через порт с заданным номером. Программа клиента должна знать имя сервера и номер порта. Для связи между клиентом и сервером клиент должен знать номер порта, используемый сервером для соединения.

Сервер RMI может соединяться с неограниченным (точнее, ограниченным только системными ресурсами) числом контент-серверов, при этом каждый сервер RMI должен быть соединен хотя бы с одним контент-сервером. Главный сервер RMI может обращаться к другим серверам RMI в пуле серверов. При первом поиске контент-сервера клиент RMI начинает поиск с главного сервера RMI. Если этот контент-сервер не найден на главном сервере RMI, поиск производится в пуле серверов RMI.

Если тот же клиент RMI вновь обращается к этому контент-серверу, он выполняет поиск на том сервере RMI, где этот контент-сервер был найден в первый раз.

Для запуска сервера RMI используйте cmbregist81.bat в Windows, cmbregist81.sh в AIX или Solaris. Перед запуском сервера RMI задайте правильные номер порта и тип сервера. Информацию о конфигурировании серверов RMI и управлении ими смотрите в руководствах *Планирование и установка Enterprise Information Portal* и *Управление Enterprise Information Portal*.

Настройка среды C++ (только C++)

При настройке среды Windows или AIX надо задать параметры, как описано в этом разделе. В Табл. 4 на стр. 36 перечислены требования к библиотекам, совместно используемым объектам и DLL AIX .

Требование: Чтобы пользоваться C++, надо установить поддержку клиента DB2 и ассистента конфигурирования клиента на всех удаленных серверах, обращающихся к базе данных Enterprise Information Portal. Используемые для соединения с базой данных ID пользователя и пароль должны совпадать с ID

пользователя и паролем базы данных Enterprise Information Portal. Подробности приведены в руководстве *Управление Enterprise Information Portal*.

Внимание: Для построения производственной системы используйте библиотеку `cmbcm81x.lib`, а для отладочного построения - библиотеку `cmbcm81xd.lib`, где *x* указывает Версию 6 или 7 (.Net) компилятора Microsoft Visual C++.

Таблица 4. Информация среды совместно используемых объектах и DLL

| Соединитель | Библиотека | Модули DLL Windows | Совместно используемые объекты для AIX |
|--|---|--|--|
| Общий (это не соединитель, а общий набор API, используемых всеми соединителями). | <code>cmbcm81x.lib</code> <code>cmbcm81xd.lib</code> | <code>cmbcm81x.dll</code> <code>cmbcm81xd.dll</code> | <code>libcmbcm815.a</code> |
| Content Manager Версии 8 | <code>cmbicm81x.lib</code> <code>cmbicm81xd.lib</code> | <code>cmbicm81x.dll</code> <code>cmbicm81xd.dll</code> <code>cmbicmfac81x.dll</code> <code>cmbicmfac81xd.dll</code> | <code>libcmbicm815.a</code> <code>libcmbicmfac815.so</code> |
| Ранние версии Content Manager | <code>cmbdl81x.lib</code> <code>cmbdl81xd.lib</code> | <code>cmbdl81x.dll</code> <code>cmbdl81xd.dll</code> <code>cmbdlfac81x.dll</code> <code>cmbdlfac81xd.dll</code> <code>de_db2.dll</code> <code>de_db2_d.dll</code> <code>de_ora.dll</code> <code>de_ora_d.dll</code> | <code>libcmbdl815.a</code> <code>libcmbdlfac815.so</code> |
| Соединитель объединения | <code>cmbfed81x.lib</code> <code>cmbfed81xd.lib</code> | <code>cmbfed81x.dll</code> <code>cmbfed81xd.dll</code> <code>cmbfedfac81x.dll</code> <code>cmbfedfac81xd.dll</code> | <code>libcmbfed815.a</code> <code>libcmbfedfac815.so</code> |
| DB2 Universal Database Версии 8.1 | <code>cmbdb281x.lib</code> <code>cmbdb281xd.lib</code> | <code>cmbdb281x.dll</code> <code>cmbdb281xd.dll</code> <code>cmbdb2fac81x.dll</code> <code>cmbdb2fac81xd.dll</code> | <code>libcmbdb2815.a</code> <code>libcmbdb2fac815.so</code> |
| ODBC | <code>cmbodbc81x.lib</code> <code>cmbodbc81xd.lib</code> | <code>cmbodbc81x.dll</code> <code>cmbodbc81xd.dll</code> <code>cmbodbcfac81x.dll</code> <code>cmbodbcfac81xd.dll</code> | Не поддерживаются |
| OnDemand | <code>cmbod81x.lib</code> <code>cmbod81xd.lib</code> | <code>cmbod81x.dll</code> <code>cmbod816xd.dll</code> <code>cmbodfac81x.dll</code> <code>cmbodfac81xd.dll</code> | <code>libcmbod815.a</code> <code>libcmbodfac815.so</code> |

Таблица 4. Информация среды совместно используемых объектах и DLL (продолжение)

| Соединитель | Библиотека | Модули DLL Windows | Совместно используемые объекты для AIX |
|------------------------|---------------------------------|--|--|
| ImagePlus for OS/390 | cmbip81x.lib cmbip81xd.lib | cmbip81x.dll cmbip81xd.dll cmbipfac81x.dll cmbipfac81xd.dll | Не поддерживаются |
| VisualInfo | cmbv481x.lib cmbv481xd.lib | cmbv481x.dll cmbv481xd.dll cmbv4fac81x.dll cmbv4fac81xd.dll | Не поддерживаются |
| Domino.Doc | cmbdd81x.lib cmbdd81xd.lib | cmbdd81x.dll cmbdd81xd.dll cmbddf81x.dll cmbddf81xd.dll | Не поддерживаются |
| Domino Extended Search | cmbdes81x.lib cmbdes81xd.lib | cmbdes81x.dll cmbdes81xd.dll cmbdesfac81x.dll cmbdesfac81xd.dll | libcmbdes815.a libcmbodfac815.so |

Задание переменных среды C++ для Windows

Командную строку со средой, сконфигурированной для разработки программ Enterprise Information Portal, можно открыть, выбрав **Пуск → Программы → IBM Enterprise Information Portal for Multiplatforms 8.1 → Окно Разработка**. Другой вариант - задать среду, запустив в командной строке DOS CMBenv81.bat.

Если хотите, можете изменить переменные среды:

PATH

```
set PATH=x:\CMBROOT\DLL
```

где *x* - диск, на котором установлен EIP.

INCLUDE

```
set INCLUDE=x:\CMBROOT\INCLUDE
```

где *x* - диск, на котором установлен EIP.

Задание переменных среды C++ для AIX

Дополнительную информацию смотрите в примерах MAK-файлов в каталоге samples.

Задайте следующие переменные среды:

В среде AIX можно использовать один из трех пакетных файлов, задающих среду разработки.

1. В оболочке Bourne используйте `cmbenv81.sh`
2. В оболочке C воспользуйтесь `cmbenv81.csh`
3. В оболочке Korn воспользуйтесь `cmbenv81.ksh`

Задайте следующие переменные среды:

путь NLS

```
export NLSPATH=${NLSPATH}:/usr/lpp/cmb/msg/En_US/%N
```

PATH

```
export PATH=/usr/lpp/cmb/lib
```

LIBPATH

```
export LIBPATH=/usr/lpp/cmb/lib
```

INCLUDE

```
export INCLUDE=/usr/lpp/cmb/INCLUDE
```

Построение программ C++

Для создания MAK-файлов и построения своих прикладных программ руководствуйтесь процедурами своего компилятора и среды разработки.

Чтобы использовать интерфейсы API C++ для отладки, при построении программы скомпонуйте программу с отладочной версией библиотек API - ***d.lib**. При построении окончательного программного продукта скомпонуйте программу с не-отладочными библиотеками (***.lib**).

Работа с Microsoft Visual Studio .NET

API Enterprise Information Portal и Content Manager версий 8.2 и новее поддерживают Microsoft Visual Studio .NET. Однако когда для построения прикладных программ используется Microsoft Visual Studio .NET, во время компиляции необходимо использовать соответствующую библиотеку (определяемую соединителем и используемой версией Visual Studio C++).

Примеры MAK-файлов (поддерживающих Visual Studio Версии 6 и Visual Studio .NET) поставляются с примерами API ICM.

Чтобы задать, какая библиотека требуется во время компиляции, используйте формат *имя соединителя816.lib* для Microsoft Visual Studio C++ 6 и *имя соединителя817.lib* для Microsoft Visual Studio .NET, как показано в следующих таблицах.

Таблица 5. Пример имен библиотек Microsoft Visual Studio C++ 6

| Соединитель | Библиотека |
|-----------------------------|---------------|
| Общий | cmbcm816.lib |
| Content Manager 8.1 и новее | cmbicm816.lib |
| Соединитель объединения | cmbfed816.lib |

Таблица 6. Имена библиотек Microsoft Visual Studio .NET

| Соединитель | Библиотека |
|-----------------------------|---------------|
| Общий | cmbcm817.lib |
| Content Manager 8.1 и новее | cmbicm817.lib |
| Соединитель объединения | cmbfed817.lib |

Задание подсистемы консоли для преобразования кодовых страниц в Windows

C++

```
#include <DKConstant.h>
#include <DKEnvironment.hpp>

void main(int argc, char *argv[])
{
    // в начале программы задайте в качестве подсистемы консоль, тогда
    // кодовая страница, в которой возвращаются сообщения об ошибках
    // DKExceptions, будет преобразована из Windows Graphical
    // User Interface (формат ANSI) в Console (формат OEM)
    // Если не задать подсистему, по умолчанию используется DK_SS_WINDOWS
    DKEnvironment::setSubSystem(DK_SS_CONSOLE);
    ...
}
```

Опции множественного поиска

Поиск можно выполнять практически по любым частям элемента или компонента, по тексту в компоненте или элементе или по тексту внутри ресурсного содержимого.

Content Manager Версии 8 содержит встроенную возможность текстового поиска и для него уже не требуется отдельного средства текстового поиска (как для более ранних версий Content Manager). Смотрите “Понятие текстового поиска” на стр. 231.

Опции множественного поиска используются для поиска на определенном контекст-сервере (для этого применяется один из перечисленных ниже поддерживаемых типов запросов или их сочетание) или для поиска в результатах предыдущего поиска. Каждый тип поиска поддерживается одним или несколькими механизмами поиска. Не все контент-серверы поддерживают опции множественного поиска.

Параметрический поиск

Поиск по значениям свойств элемента или компонента, атрибутов, связей или содержимого папок. Например, с помощью параметрического запроса можно искать документы по имени заказчика. Такой запрос требует точного совпадения условия, заданного в предикате запроса, и значений данных с контент-сервера.

Текстовый поиск

Поиск в тексте, для которого разрешен текстовый поиск (`textSearchable(true)`), выполняемый с помощью механизма поиска, например, DB2 Net Search Engine (NSE). Такой запрос выполняет поиск по содержимому текстовых полей в соответствии с заданным выражением текстового поиска, требующим не точного совпадения, а, например, наличия или, наоборот, отсутствия определенных словосочетаний или основ слов.

Примечание: DB2 Net Search Engine (NSE) в ранних версиях DB2 назывался DB2 Text Information Extender (TIE).

Поиск изображений

Поиск по характеристикам изображений. Такой запрос выполняет поиск по содержимому изображений в соответствии с заданным выражением поиска изображений, требующим не точного совпадения, а, например, наличия в изображениях определенного цвета.

Комбинированный поиск

Поиск, включающий как параметрический, так и текстовый поиск.

Только Content Manager: В CM есть один механизм поиска и три метода выполнения поиска (и возвращения результатов): `execute()` (выполнение), `evaluate()` (оценка) и `executeWithCallback()` (выполнение с обратным вызовом). Дополнительную информацию смотрите в примере `SSearchICM`.

Трассировка

Для исправления ошибок в ваших прикладных программах с API можно использовать трассировку и обработку исключительных ситуаций.

Трассировка текстовых запросов, использующих механизм текстового поиска

Механизм текстового поиска (TSE) и все его функции можно использовать только с ранними версиями сервера Content Manager. Content Manager Версии 8

содержит встроенную возможность текстового поиска, не требующую отдельного средства текстового поиска. Смотрите “Понятие текстового поиска” на стр. 231.

Если задана следующая переменная среды, в указанный файл будет записываться трассировка запросов механизма текстового поиска (в двоичном формате):

`СМВТМDSTREAMTRACE=имяФайла`

(например, `.\tm.out` для Windows или `./tm.out` для AIX)

Если задана следующая переменная среды, в указанный файл будет записываться трассировка вызовов API механизма текстового поиска, использованных для текстовых запросов:

`СМВТМTRACE=имяФайла`

Если задана следующая переменная среды, в указанный файл будут записываться условия текстового поиска: `СМВТМTERM=имяФайла`
(например, `.\tmterm.out`)

Примечание: Content Manager Версии 8 использует встроенную возможность текстового поиска. Если вы все еще пользуетесь ранними версиями Content Manager, можно использовать механизм текстового поиска Text Search Engine (TSE).

Трассировка параметрических запросов

Если используется более ранняя версия Content Manager с механизмом текстового поиска, задайте следующую переменную среды, чтобы обеспечить передачу параметрического запроса менеджеру папок:

`СМБДLQRYTRACE=имяФайла`

(например, `<.\dlqry.out>` для Windows или `<./dlqry.out>` для AIX)

Обработка исключительных ситуаций

Если API обнаруживает ошибку, генерируется исключительная ситуация. При этом создается объект исключительной ситуации класса `DKEException` или одного из его подклассов.

При создании `DKEException` слой соединителя записывает диагностическую информацию в файл журнала, при этом предполагается, что используется конфигурация журнала по умолчанию. Подробнее файлы журнала и конфигурации, использующиеся API EIP, описаны в документе *Сообщения и коды*.

Перехват `DKEException` позволяет увидеть сообщения о возникших при выполнении ошибках, коды и состояния этих ошибок. При перехвате ошибки

выдается сообщение об ошибке и указывается место возникновения исключительной ситуации. Приводится также дополнительная информация (такая, как ID ошибки). Ниже показан пример программного кода, с процессом генерации и перехвата исключительной ситуации для EIP и CM:

Java

```
try{
    ... Операции API EIP ...
}
catch (DKException exc) {
    // ПРИМ.: Функция print приведена в примере API SConnectDisconnectICM.
    System.out.println("");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("X    !!! Exception !!!    X");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("    Name: " + exc.name());
    System.out.println("    Message: " + exc.getMessage());
    System.out.println(" Message ID: " + exc.getErrorId());
    System.out.println("Error State: " + exc.errorState());
    System.out.println(" Error Code: " + exc.errorCode());
    exc.printStackTrace();
    System.out.println("-----");
} catch (Exception exc) {
    // ПРИМ.: Функция print приведена в примере API SConnectDisconnectICM.
    System.out.println("");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("X    !!! Exception !!!    X");
    System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXX");
    System.out.println("    Name: " + exc.getClass().getName());
    System.out.println(" Message: " + exc.getMessage());
    exc.printStackTrace();
    System.out.println("-----");
}
```

C++

```
try{
    ... Операции API EIP ...
}
catch(DKException &exc) {
    // ПРИМ.: Функция print приведена в примере API SConnectDisconnectICM.
    cout << endl;
    cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    cout << "X    !!! Exception !!!    X" << endl;
    cout << "XXXXXXXXXXXXXXXXXXXXXXXXXXXX" << endl;
    cout << "      Name: " << exc.name() << endl;
    cout << " Message ID: " << exc.errorId() << endl;
    cout << "Error State: " << exc.errorState() << endl;
    cout << " Error Code: " << exc.errorCode() << endl;
    // Выводим положение API, где обнаружена ошибка
    for(unsigned int j=0; j < exc.locationCount(); j++){ //Выводим все положения
        const DKExceptionLocation* p = exc.locationAtIndex(j);
        cout << " Location " << j << ": " << p->fileName() << " : " << " : "
            << p->functionName() << '[' << p->lineNumber() << ']' << endl;
    }
    if(exc.textCount()<=0) // Если нет положений
        cout << " Locations: <none> " << endl;
    // Сообщения об ошибках
    for(unsigned int i=0; i < exc.textCount(); i++) // Печать всех сообщений
        cout << " Message " << i << ": " << exc.text(i) << endl;
    if(exc.textCount()<=0) // Если нет сообщений.
        cout << " Messages: <none> " << endl;
    cout << "-----" << endl;
}
```

Дополнительную информацию об обнаружении и обработке ошибок смотрите в учебном примере API SConnectDisconnectICM в каталоге CMBROOT\Samples\java\icm или CMBROOT\Samples\cpp\icm.

Константы

Константы задаются в виде DK_CM_ (общие константы) или DK_XX_ (где XX указывает различные контент-серверы). Список расширений (добавляемых к каждому DKDatastore) смотрите в Табл. 7 на стр. 44.

Для задания констант DDO используйте DK_CM_DATAITEM_TYPE_ ... с типом свойства (например, DK_CM_DATAITEM_TYPE_STRING). Для типов атрибутов используйте константы DK_CM_...тип (например, DK_CM_INTEGER).

Java

Общие константы определены в `DKConstant.java`. Текстовую версию этих констант можно также посмотреть в файле `DKConstant.txt`. Константы для конкретного типа контент-серверов определены в `DKConstantXX.java`; например, константы для Content Manager - в `DKConstantICM.java`.

C++

Общие константы определены в `DKConstant.h`. Список констант и соответствующих значений можно увидеть в `DKConstant2.h` (но не включайте этот файл в свою программу). Константы для конкретного типа контент-серверов определены в файлах заголовков с именами вида `DKConstantXX.h`; например, константы для Content Manager - в `DKConstantICM.h`.

Соединение с контент-серверами

Объект класса `DKDatastorexx` (где `xx` указывает определенный контент-сервер) представляет соединение с контент-сервером и обеспечивает возможность управления соединением, поддержку транзакций и запуск команд сервера. Список расширений смотрите в Табл. 7.

Таблица 7. Тип сервера и терминология имен классов

| Контент-сервер | Имя класса |
|--|--|
| Content Manager Версия 8.2 | <code>DKDatastoreICM</code> |
| Ранние версии Content Manager | <code>DKDatastoreDL</code> |
| Content Manager OnDemand | <code>DKDatastoreOD</code> |
| Content Manager for AS/400 (VisualInfo for AS/400) | <code>DKDatastoreV4</code> |
| Content Manager ImagePlus for OS/390 | <code>DKDatastoreIP</code> |
| Domino.Doc | <code>DKDatastoreDD</code> |
| Extended Search | <code>DKDatastoreDES</code> |
| Panagon Image Services (FileNET) | <code>DKDatastoreFN</code> |
| Реляционные базы данных | <code>DKDatastoreDB2</code> , <code>DKDatastoreJDBC</code> (для Java) <code>DKDatastoreODBC</code> (для C++) |

Установление соединения

Каждый класс `DKDatastorexx` содержит методы для соединения и разъединения с ним. В следующем примере используются библиотечный сервер с именем

ICMNLSDb, ID пользователя ICMADMIN и пароль PASSWORD. Информацию о Content Manager смотрите в разделе Соединение с системой Content Manager; по прочим контент-серверам - в разделе Работа с другими контент-серверами. Программа в этом примере создает объект DKDatastoreICM для контент-сервера CM, соединяется с этим объектом, работает с ним и затем отсоединяется от него.

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); //Создаем объект - склад данных
dsICM.connect("ICMNLSDb","ICMADMIN","PASSWORD",""); //Соединяемся со складом

System.out.println("Connected to datastore dbase: '"+dsICM.datastoreName()+
    "', UserName '"+dsICM.userName()+"'");

dsICM.disconnect(); // Отсоединяемся от склада данных
dsICM.destroy(); // Вызов метода destroy
```

Полный пример программы смотрите в файле SConnectDisconnectICM.java в каталоге CMBROOT\Samples\java\icm.

C++

```
DKDatastoreICM* dsICM = new DKDatastoreICM(); //Создаем объект - склад данных
dsICM->connect("ICMNLSDb","ICMADMIN","PASSWORD",""); //Соединяемся со складом

cout << "Connected to datastore dbase: '" << dsICM->datastoreName() <<
    "', UserName '" << dsICM->userName() << "')." << endl;

dsICM->disconnect(); //Отсоединяемся от склада данных
delete(dsICM); //Удаляем ссылку на объект
```

Полный пример программы смотрите в файле SConnectDisconnectICM.cpp в каталоге CMBROOT\Samples\cpp\icm.

При соединении с контент-сервером необходимо учитывать требования конкретного сервера; например, пароль для ImagePlus for OS/390 не может быть длиннее восьми символов.

Соединение клиента с контент-сервером и отсоединение от него

Для обращения из программы клиента к контент-серверу используется тот же программный код, что показан в разделе “Установка соединения” на стр. 44. Надо только заменить `import com.ibm.mm.sdk.server.*`; на `import com.ibm.mm.sdk.client.*`; Все возникшие ошибки связи должна обрабатывать программа клиента.

Задание и получение опций контент-сервера

Опции контент-сервера можно узнать или задать с помощью методов в DKDatastorexx. В следующем примере показано, как задать и получить опцию установления сеанса управления на библиотечном сервере Content Manager. Список опций и их описания смотрите в *электронном справочнике API*.

В этом примере задания и получения опций контент-сервера в Content Manager кэширование отключено. Для контент-серверов, поддерживающих эту опцию, рекомендуется использовать значение по умолчанию (ON). **Требование:** Должен быть уже создан правильный объект DKDatastoreICM в переменной с именем dsICM.

Java

```
dsICM.setOption(DKConstant.DK_CM_OPT_CACHE,  
    new Integer(DKConstant.DK_CM_FALSE));  
Object val = dsICM.getOption(DKConstant.DK_CM_OPT_CACHE);
```

C++

```
DKAny inVal = DK_CM_FALSE  
DKAny outVal;  
dsICM->setOption(DK_CM_OPT_CACHE,inVal);  
dsICM->getOption(DK_CM_OPT_CACHE,outVal);
```

При получении опции контент-сервера выходная_опция обычно имеет тип integer, но ее можно преобразовать в объект.

Получение списка контент-серверов

DKDatastorexx содержит метод для получения списка серверов, с которыми он может соединяться. Список серверов возвращается в собрании DKSequentialCollection объектов DKServerDefxx (где xx указывает определенный контент-сервер).

Ограничение: У контент-сервера Domino.Doc нет метода вывода списка серверов.

Получив объект DKServerDefxx, можно узнать имя и тип сервера и использовать эти данные для установления соединения с ним.

Следующий пример выводит список серверов, для которых сконфигурировано соединение.

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM();// Создаем объект - склад данных
dkCollection coll = dsICM.listDataSources(); // Получаем список источников
dkIterator iter = coll.createIterator(); // Создаем итератор
while(iter.more()){ // Пока есть еще элементы
    DKServerDefICM srvrDef = (DKServerDefICM) iter.next();
    System.out.println("Found server '"+srvrDef.getName()+"'");
}
```

C++

```
DKDatastoreICM* dsICM = new DKDatastoreICM();// Создаем объект: склад данных
dkCollection* coll = (dkCollection*)dsICM->listDataSources();// Получаем список
dkIterator* iter = coll->createIterator(); // Создаем итератор
while(iter->more()){ // Пока есть еще элементы
    DKServerDefICM* srvrDef = (DKServerDefICM*) iter->next()->value();
    cout << "Found server '" << srvrDef->getName() << "' << endl;
    delete(srvrDef); // Освобождаем память
}
delete(iter); // Освобождаем память
delete(coll);
delete(dsICM);
```

Вывод списка объектов и атрибутов для контент-сервера

DKDatastorexx содержит методы получения списка объектов и их атрибутов для контент-сервера. Каждое имя атрибута - часть имени пространства. Если имя пространства не задано, имя пространства по умолчанию применяется ко всем атрибутам.

Список объектов возвращается в объекте DKSequentialCollection - собрании объектов dkEntityDef. Атрибуты объекта возвращается в объекте DKSequentialCollection - собрании объектов dkAttrDef. Получив объект dkAttrDef, вы можете получить информацию об атрибуте, включая его имя и тип, и потом использовать эту информацию для создания запроса.

Более подробную информацию об этих двух методах смотрите в *электронном справочнике API*.

В следующем примере показано, как получить список типов элементов и список атрибутов с сервера Content Manager:

Java

```
. . .
try {
    DKSequentialCollection pCol = null;
    dkIterator pIter = null;
    DKSequentialCollection pCol2 = null;
    dkIterator pIter2 = null;
    DKServerDefICM pSV = null;
    String strServerName = null;
    String strItemType = null;
    DKComponentTypeDefICM itemTypeDef = null;
    DKAttrDefICM attrDef = null;
    DKDatastoreDefICM dsDefICM = null;
    int i = 0;
    int j = 0;
    // ----- Создаем склад данных и соединяемся с ним
    //          (в предположении, что параметры соединения заданы раньше)
    DKDatastoreICM dsICM = new DKDatastoreICM();
    dsICM.connect(db,userid,pw,"");
    // ----- Получаем список типов элементов
    pCol = (DKSequentialCollection) dsICM.listEntities();
    pIter = pCol.createIterator();
    i = 0;
    while (pIter.more() == true)
    {
        i++;
        itemTypeDef = (DKComponentTypeDefICM)pIter.next();
        strItemType = itemTypeDef.getName();
        System.out.println("item type name [" + i + "] - " + strItemType);
        System.out.println("    type " + itemTypeDef.getType());
        System.out.println("    itemTypeId " + itemTypeDef.getId());
        System.out.println("    compID " + itemTypeDef.getComponentTypeId());
        //продолжение следует . . .
    }
}
```

Java (продолжение)

```
// ----- Список атрибутов
pCol2 = (DKSequentialCollection) dsICM.listEntityAttrs(strItemType);
pIter2 = pCol2.createIterator();
j = 0;
while (pIter2.more() == true)
{
    j++;
    attrDef = (DKAttrDefICM)pIter2.next();
    System.out.println("Attr name [" + j + "] - " + attrDef.getName());
    System.out.println("    datastoreType " + attrDef.datastoreType());
    System.out.println("    attributeOf " + attrDef.getEntityName());
    System.out.println("    тип " + attrDef.getType());
    System.out.println("    размер " + attrDef.getSize());
    System.out.println("    id " + attrDef.getId());
    System.out.println("    nullable " + attrDef.isNullable());
    System.out.println("    точность " + attrDef.getPrecision());
    System.out.println("    масштаб " + attrDef.getScale());
    System.out.println("    stringType " + attrDef.getStringType());
    System.out.println("    sequenceNo " + attrDef.getSequenceNo());
    System.out.println("    userFlag " + attrDef.getUserFlag());
}
dsICM.disconnect();
}
catch(DKException exc)
{
}
// ----- Обработка исключительных ситуаций
```

В полном примере `SItemTypeRetrievalICM` показано, как получить список определений типов элементов. В другом полном примере `SAttributeDefinitionRetrievalICM` показано, как получить список определений атрибутов. Оба примера находятся в каталоге `CMBROOT\Samples\java\icm`.

В следующем примере C++ показано, как получить список индексных классов и атрибутов с сервера Content Manager:

C++

```
// Получаем собрание, содержащее все определения типов элементов.
DKSequentialCollection * itemTypeColl = (DKSequentialCollection *)
    dsICM->listEntities();
// Для каждого типа элементов получаем и выводим имя и описание.
cout << "\nItem Type Names in System:
    (" << itemTypeColl->cardinality() << ')' << endl;
// Создаем итератор для перемещения по собранию
dkIterator* iter = itemTypeColl->createIterator();
//пока в списке есть еще элементы, продолжаем
while(iter->more()) {
    DKItemTypeDefICM* itemType = (DKItemTypeDefICM*) iter->next()->value();
    cout << " - " << itemType->getName() << ": " <<
        itemType->getDescription() << endl;
    delete(itemType); // Освобождаем память

cout << endl;
delete(iter);
delete(itemTypeColl);
```

Дополнительную информацию смотрите в примере SItemTypeRetrievalICM, где показано, как получить список определений типов элементов. В примере SAttributeDefinitionRetrievalICM показано, как получить список определений атрибутов. Эти примеры находятся в каталоге CMBROOT\Samples\cpp\icm.

Совет: Можно отложить удаление pEnt и использовать функцию apply, чтобы удалить определение атрибута в собрании. Дополнительную информацию смотрите в разделе “Управление памятью в собраниях (только C++)” на стр. 121.

C++

```
...
pCol2->apply(deleteDKAttrDefICM);
delete pCol2;
...
```

Работа с динамическими объектами данных (DDO)

В этом разделе описано, как использовать объект DDO, и приведены примеры, объясняющие, как:

1. Связать DKDDO с контент-сервером.
2. Создать DKDDO.
3. Создать постоянные идентификаторы (PID) для атрибутов DKDDO.

4. Добавить атрибуты и определить свойства атрибутов.
5. Определить DKDDO как папку или как документ.
6. Добавить и посмотреть значения свойств атрибутов.
7. Проверить свойства DKDDO.
8. Проверить свойства атрибутов.
9. Вывести содержимое DKDDO.
10. Удалить DKDDO.

Класс DKDDO используется в программах IBM Enterprise Information Portal for Multiplatforms для динамических объектов данных (DDO). Объект DKDDO представляет элемент, который может быть, например, документом Content Manager, папкой или пользовательским объектом. Объект DKDDO содержит атрибуты. У каждого атрибута есть имя, значение и свойства. Каждый атрибут определяется своим ID данных. Атрибуты нумеруются последовательно, начиная с 1; номер атрибута и есть ID данных.

Поскольку имя, значение и свойство атрибута могут быть различными, в DKDDO есть гибкие механизмы представления данных, происходящих из разных контент-серверов и имеющих разные форматы. Например, это могут быть элементы различных типов элементов в Content Manager или строки разных таблиц в реляционной базе данных. DKDDO может также иметь свойства, относящиеся не к отдельному атрибуту, а ко всему DKDDO.

Прежде чем вызывать методы `add`, `retrieve`, `update` и `delete`, помещать атрибуты DKDDO на контент-сервер и получать их, нужно связать DKDDO с контент-сервером. Контент-сервер задается или как параметр при создании объекта DKDDO, или вызовом метода `setDatastore`.

У каждого DKDDO есть постоянный идентификатор объекта (PID), содержащий информацию о размещении атрибутов на контент-сервере.

Создание DKDDO

У DKDDO есть несколько конструкторов. Можно создать DKDDO, вызвав его конструктор без параметров.

Java

```
DKDDO ddo = new DKDDO();
```

C++

```
DKDDO ddo;
```

Этот DDO ddo должен динамически расширяться при добавлении новых атрибутов. Для более эффективного создания передайте методу точное число атрибутов (например, 10):

Java

```
DKDDO ddo = new DKDDO(10);
```

C++

```
DKDDO ddo = new DKDDO((short)10);
```

Внимание: В таких API, как DKDatastoreICM как DKDatastoreOD, для создания DKDDO используются методы createDDO() класса DKDatastoreXX. В CM V8 для создания DDO нужно использовать эти методы. В следующем примере при создании DKDDO передаются контент-сервер и тип объекта для Content Manager Версии 8:

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); //создаем склад данных CM
DKDDO ddo=dsICM.createDDO("ICMSAMPLE", //создаем DDO для типа объекта
DKConstant.DK_CM_DOCUMENT);
```

Дополнительную информацию о создании DDO смотрите в примере SitemCreationICM в каталоге CMBROOT\Samples\java\icm.

C++

```
// создаем склад данных Content Manager
DKDatastoreICM* dsICM = new DKDatastoreICM();
// создаем DDO для типа объекта
DKDDO* ddo = dsICM->createDDO("ICMSAMPLE",
DK_CM_DOCUMENT);
```

Дополнительную информацию о создании DDO смотрите в примере SitemCreationICM в каталоге CMBROOT\Samples\cpp\icm.

Для других соединителей (например, для предыдущих версий Content Manager) можно создать DKDDO, задав контент-сервер и тип объекта, как в следующем примере:

Java

```
DKDatastoreDL dsDL=new DKDatastoreDL(); //создаем склад данных CM
DKDDO ddo=new DKDDO(dsDL, "DLSAMPLE"); //создаем DDO для типа объекта
//DLSAMPLE в dsDL
```

C++

```
// создаем склад данных ранней версии Content Manager
DKDatastoreDL dsDL;
// создаем DDO для типа объекта DLSAMPLE в dsDL
DKDDO* cddo = new DKDDO(&dsDL, "DLSAMPLE");
```

Выбор конкретного конструктора зависит от вашей прикладной программы, сведения о конструкторах смотрите в *электронном справочнике API*.

Добавление свойств в DDO

При создании объекта DKDDO, представляющего DDO, нужно задать его свойство типа элементов (документ, папка или элемент).

Значение этого свойства можно задать в одном из параметров метода DKDatastoreXX.createDDO(itemTypeName,itemPropertyType/SemanticType). DKDatastoreICM.createDDO(itemTypeName,itemPropertyType/SemanticType)

Или, если DKDDO уже создан без задания его свойства типа элементов, можно задать это свойство, как в следующем примере (где задается тип DDO "документ").

Java

```
//----- Добавляем свойство, указывающее, что это документ
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, new Short(DK_CM_DOCUMENT));
```

C++

```
any = DK_CM_DOCUMENT; // это документ
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);
```

Создание постоянного идентификатора (PID)

У каждого DDO должен быть постоянный идентификатор (PID). PID содержит информацию об имени, типе и ID контент-сервера и типе объекта. PID идентифицирует постоянное положение данных DDO. Например, на

контент-сервере предыдущих версий Content Manager PID - это ID элемента. ID элемента - один из важнейших параметров для функций `retrieve`, `update` и `delete`. В Content Manager V8 PID содержит пять частей (дополнительную информацию смотрите в разделе *Работа с Content Manager 8.2*).

При вызове функции `add` контент-сервер создает и возвращает ID элемента. Например, соединители, у которых есть метод `DKDatastoreXX.createDDO()`, автоматически создают объект PID в операции `DKDDO.add()`.

В следующем примере создается DDO для получения известного элемента:

Java

```
// Допустим, у нас есть соединенный объект DKDatastoreICM с именем "dsICM"
// Создаем новый DDO
DKDDO ddo = dsICM.createDDO("book",DKConstant.DK_CM_DOCUMENT);
// Эта функция (в предыдущей строке) автоматически создает PID.
ddo.add(); // Добавляем новый элемент на склад данных.
// PID задается системой
DKPidICM pid = (DKPidICM)ddo.getPidObject();
```

C++

```
// Допустим, у нас есть соединенный объект DKDatastoreICM с именем "dsICM"
// Создаем новый DDO
DKDDO* ddo = dsICM->createDDO("book",DK_CM_DOCUMENT);
// Эта функция (в предыдущей строке) автоматически создает PID.
ddo->add(); // Добавляем новый элемент на склад данных.
// PID задается системой
DKPidICM* pid = (DKPidICM*) ddo->getPidObject();
```

Соединитесь с контент-сервером и вызовите функцию `retrieve`, чтобы получить DDO, созданный в этом примере.

У соединителя Content Manager 8.2 есть класс PID - подкласс `DKPid`. Он называется `DKPidICM` и представляет собой `pid`, используемый `DKDDO` и `dkResources`.

Работа с элементами данных и свойствами

`DKDDO` поддерживает методы для добавления атрибутов и свойств атрибутов в объект `DKDDO`.

Предположим, что у типа элементов `NameOfItemType` есть атрибуты `Name` типа `integer` и в репозитории CM V8 определено, что они могут содержать пустые значения. Вы создаете объект `DKDDO` для работы с элементом этого объекта и хотите добавить в него два элемента данных.

В следующем примере показано, как создать элемент, задать свойства атрибутов и сохранить элемент на постоянное хранение на контент-сервере Content Manager. **Требование:** Считается, что установлено соединение со складом данных (DKDatstoreICM в переменной dsICM). Пользовательский тип данных S_withChild должен быть определен с типами атрибутов varchar S_varchar, long integer S_integer, short integer S_short и time S_time, как показано в учебном примере API SitemTypeCreationICM.

Java

```
// Создаем новый элемент в памяти
DKDDO ddo = dsICM.createdDO("S_withChild", DKConstant.DK_CM_DOCUMENT);

// Задаем атрибуты (Доп. атрибуты заданы в примере SitemCreationICM)
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar"),
    "abcdefg");
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_integer"),
    new Integer("123"));
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_short"),
    new Short("5"));
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_time"),
    java.sql.Time.valueOf("10:00:00"));

// Добавляем на склад данных
ddo.add();
```

Полная программа примера, из которой взят приведенный фрагмент (SitemCreationICM), находится в каталоге CMBROOT\Samples\java\icm.

C++

```
// Создаем новый элемент в памяти
DKDDO* ddo = dsICM->createDDO("S_withChild", DK_CM_DOCUMENT);

// Задаем атрибуты (Доп. атрибуты заданы в примере SItemCreationICM)
// ПРИМ.: Показанные ниже значения автопреобразуются в тип DKAny
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_varchar"),
    DKString("this is a string value"));
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_integer"),
    (long) 123);
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_short"),
    (short) 5);
ddo->setData(ddo->dataId(DK_CM_NAMESPACE_ATTR,"S_time"),
    DKTime("10.00.00"));

// Добавляем на склад данных
ddo->add();
```

Полная программа примера, из которой взят приведенный фрагмент (SItemCreationICM), находится в каталоге CMBROOT\Samples\cpp\icm.

Для каждого атрибута должно быть задано свойство типа; другие свойства задавать не обязательно.

В следующем примере показано, как обращаться к значениям атрибутов DDO.

Java

```
// Возвращается подкласс типа Object, поэтому требуются операции
// преобразования типов.
// ПРИМ.: Обращение к другим атрибутам показано в примере SItemRetrievalICM.
```

```
String attrVal1 = (String) ddo.getData(ddo.dataId(
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar"));
Integer attrVal2 = (Integer) ddo.getData(ddo.dataId(
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_integer"));
Short attrVal3 = (Short) ddo.getData(ddo.dataId(
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_short"));
Time attrVal4 = (Time) ddo.getData(ddo.dataId(
    DKConstant.DK_CM_NAMESPACE_ATTR,"S_time"));

System.out.println("Attr 'S_varchar' value: "+attrVal1);
System.out.println("Attr 'S_integer' value: "+attrVal2);
System.out.println("Attr 'S_short' value: "+attrVal3);
System.out.println("Attr 'S_time' value: "+attrVal4);
```

Полная программа примера, из которой взят приведенный фрагмент (SItemRetrievalICM), находится в каталоге CMBROOT\Samples\java\icm.

C++

```
// Операции присваивания и преобразования типов позволяют преобразовать
// значения из типа DKAny в нужный тип.
// ПРИМ.: Обращение к другим атрибутам показано в примере SItemRetrievalICM.
```

```
DKString attrVal1 = ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("S_varchar"))).toString();
long attrVal2 = (long) ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("S_integer")));
short attrVal3 = (short) ddo->getData(ddo->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("S_short")));
DKTimestamp attrVal4 = (DKTimestamp) ddo->getData(ddo->dataId(
    DK_CM_NAMESPACE_ATTR, DKString("S_time")));

cout << "Attr 'S_varchar' value: " << attrVal1 << endl;
cout << "Attr 'S_integer' value: " << attrVal2 << endl;
cout << "Attr 'S_short' value: " << attrVal3 << endl;
cout << "Attr 'S_time' value: " << attrVal4 << endl;
```

Полная программа примера, из которой взят приведенный фрагмент (SItemRetrievalICM), находится в каталоге CMBROOT\Samples\cpp\icm.

Получение свойств DKDDO и атрибутов

При обработке DKDDO нужно знать его тип: документ, папка или элемент. В следующем примере программного кода показано, как определить тип DDO:

Java

```
short prop_id = ddo.propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    short type = ((Short) ddo.getProperty(prop_id)).shortValue();
    switch(type) {
        case DK_CM_DOCUMENT:
            // --- обрабатываем документ
            ....
            break;
        case DK_CM_FOLDER:
            // --- обрабатываем папку
        case DK_CM_ITEM:
            // --- Обрабатываем элемент в Content Manager
            ....
            break;
    }
}
```

Дополнительную информацию об обращении к свойствам "тип элементов" смотрите в учебном примере API SItemRetrieval ICM, который находится в каталоге CMBROOT\Samples\java\icm.

C++

```
unsigned short prop_id =
    ddo->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    unsigned short type = (unsigned short) ddo->getProperty(prop_id);
    switch(type) {
        case DK_CM_DOCUMENT:
            // обработать документ
            ...
            break;
        case DK_CM_FOLDER:
            // обработать папку
            ...
            break;
    }
}
```

Дополнительную информацию об обращении к свойствам "тип элементов" смотрите в учебном примере API SItemRetrieval ICM в каталоге CMBROOT\Samples\cpp\icm.

Чтобы получить свойства атрибута, нужно узнать data_id этого атрибута; затем можно получить его свойства.

Начальные значения переменных data_id и property_id - 1. Если задать 0, возникнет исключительная ситуация.

Java

```
data_id = ddo.dataId("Title"); // получаем data_id атрибута Title
// ----- Получаем число свойств этого атрибута
short number_of_data_prop = ddo.dataPropertyCount(data_id);
// ----- Выводим все свойства данных для этого атрибута
//           в цикле; счетчик цикла начинается с 1
for(short i = 1; i <= number_of_data_prop; i++) {
    System.out.println(i + " Property Name = " +
        ddo.getDataPropertyName(data_id,i)
        + " value = " + ddo.getDataProperty(data_id,i));
}
```

Полный пример программы смотрите в файле SItemRetrievalICM в каталоге CMBROOT\Samples\java\icm.

C++

```
// получить ID данных для Title
data_id = ddo->dataId("Title");
// сколько свойств у этого атрибута?
unsigned short number_of_data_prop = ddo->dataPropertyCount(data_id);
// вывод всех свойств данных, принадлежащих этому атрибуту
// обратите внимание на то, что 1 <= i <= number_of_data_prop
// начальное значение переменной цикла 1
for (unsigned short i = 1; i <= number_of_data_prop; i++) {
    cout << i << " Property Name = " << ddo->
        getDataPropertyName(data_id, i) << " value = " << ddo->
        getDataProperty(data_id, i) << endl;
}
```

Полный пример программы смотрите в файле SItemRetrievalICM в каталоге CMBROOT\Samples\cpp\icm.

Как вывести весь DDO

Для отладки при разработке программ может понадобиться вывести содержимое DKDDO.

Java

```
short number_of_attribute = ddo.dataCount();
short number_of_prop      = ddo.propertyCount();
short number_of_data_prop;
// вывести список свойств DDO
for (short k = 1; k <= number_of_prop; k++) {
    System.out.println( k + " Property Name = " + ddo.getPropertyName(k) +
                        ",\t value = " + ddo.getProperty(k));
}
// вывести список элементов данных и их свойств
for (short i = 1; i <= number_of_attribute; i++) {
    System.out.println( i + " Attr. Name = " + ddo.getDataName(i) +
                        ",\t value = " + ddo.getData(i));
    number_of_data_prop = ddo.dataPropertyCount(i);
    for (short j = 1; j <= number_of_data_prop; j++) {
        System.out.println( "\t" + j + " Data Prop. Name = " +
                            ddo.getDataPropertyName(i,j) +
                            ",\t value = " +
                            ddo.getDataProperty(i,j));
    }
}
```

Полный пример обращения к DDO и вывода его (и всех подкомпонентов) смотрите в примере `SItemRetrievalICM` (и там, где в нем используется статическая функция `printDDO()`) в каталоге `CMBROOT\Samples\java\icm`.

C++

```
unsigned short number_of_attribute = ddo->dataCount();
unsigned short number_of_prop;
unsigned short number_of_data_prop;
// вывести список свойств DDO
for (short k = 1; k <= number_of_prop; k++) {
    cout << k << " Property Name = " << ddo->getPropertyName(k) <<
        ",\t value = " << ddo->getProperty(k) << endl;
}
// вывести список элементов данных и их свойств
for (unsigned short i = 1; i <= number_of_attribute; i++) {
    cout << i << " Attr. Name = " << ddo->getDataName(i) <<
        << ",\t value = " << ddo->getData(i) << endl;
    number_of_data_prop = ddo->dataPropertyCount(i);
    for (unsigned short j = 1; j <= number_of_data_prop; j++) {
        cout << "\t" << j << " Data Prop. Name = "
            << ddo->getDataPropertyName(i, j)
            << ",\t value = " << ddo->getDataProperty(i, j)
            << endl;
    }
}
```

Полный пример обращения к DDO и вывода его (и всех подкомпонентов) смотрите в примере `ItemRetrievalICM` (и там, где в нем используется статическая функция `printDDO()`) в каталоге `CMBROOT\Samples\cpp\icm`.

Удаление DDO (только C++)

DKDDO существует в двух местах: в памяти и в виде постоянной копии. Чтобы удалить DKDDO из памяти, вызовите его деструктор. Учтите, что постоянная копия останется при этом на контент-сервере.

Для удаления постоянной копии с контент-сервера используйте функцию `dkddo:del()`. Это не повлияет на представление DKDDO в памяти (значения атрибутов в объекте DKAny). Деструктор удаляет объектные ссылки на `dkCollection` и `dkDataObjectBase`, в том числе ссылки на `DKParts`, `DKFolder`, `DKDDO` и `DKBlob`.

Работа с расширенными объектами данных (XDO)

XDO представляет компонент, который может хранить ресурсное содержимое (например, ресурсный элемент или часть документа).

Ресурсные элементы (XDO) - это расширение нересурсных элементов (DDO). Ресурсные элементы создаются почти так же, как обычные элементы. В зависимости от типа ресурсного элемента XDO может быть расширен и далее.

Java

| Иерархия классов | | | | |
|------------------|-------|-------------|----------------|--|
| Тип | DDO | XDO | Расширение | |
| ----- | ----- | ----- | ----- | |
| большой объект | DKDDO | -> DKLobICM | | |
| текст | DKDDO | -> DKLobICM | -> DKTextICM | |
| изображение | DKDDO | -> DKLobICM | -> DKImageICM | |
| поточковый | DKDDO | -> DKLobICM | -> DKStreamICM | |

Только для Content Manager: Поскольку для CM 8 требуется правильность подкласса XDO, DKDDO всегда нужно создавать при помощи метода `DKDatastoreICM.createDDO()`. Это позволит системе автоматически задать важную информацию в структуре DKDDO и позволит использовать такие возможности, как ресурсы, документная модель CM и папки. Элементы типа "ресурс" (возвращенные методом `DKDatastoreICM.createDDO()`) можно преобразовать в правильный XDO или подкласс в зависимости от классификации XDO. Дополнительную информацию о создании элементов и функции `DKDatastoreICM.createDDO()` смотрите в примере `SItemCreationICM`.

Для создания XDO для двоичных больших объектов используйте `DKBlobxx`, где `xx` - указывает определенный сервер. Например, используйте `DKBlobICM` для Content Manager, `DKBlobOD` для OnDemand или `DKBlobIP` для ImagePlus for OS/390. При создании объекта `DKBlobxx` ему нужно передать объект контент-сервера `DKDatastorexx`. Для создания XDO в Content Manager используйте `DKLobICM`.

Использование постоянного идентификатора (PID) XDO

Для постоянного хранения данных объекту XDO требуется PID. Чтобы использовать XDO для поиска и сохранения данных, нужно задать PID для объекта `DKBlobxx` с помощью `DKPidXDOxx`. Реляционным базам данных для поиска постоянных данных на контент-сервере требуются таблица, столбец и строка предиката данных. Для реляционных баз данных (RDB) для `DKPidXDOxx` требуются имя таблицы, имя столбца и предикат данных.

В соединителе ICM используйте для представления `pid` объекта `dkResource`, который является XDO, класс `DKPidICM`.

Свойства XDO

Для задания доступных свойств объекта XDO используйте методы `DKBlobxx`; не для всех контент-серверов доступны все свойства. Если при загрузке не задано конкретное значение свойства, для него задается значение по умолчанию. Например, для предыдущих версий Content Manager используются следующие значения по умолчанию:

RepType (тип представления)

Значение по умолчанию - FRN\$NULL. Для VisualInfo for AS/400, нужно использовать " " - восемь пробелов в кавычках.

ContentClass

Значение по умолчанию - DK_CM_CC_UNKNOWN. Допустимые значения смотрите в файле DKConstant2DL.h в каталоге \cmbroot\include для Enterprise Information Portal.

AffiliatedType

Значение по умолчанию - DK_DL_BASE.

AffiliatedData

Значение по умолчанию - NULL.

Для правильной индексации содержимого объекта в ранней версии Content Manager, нужно задать SearchEngine, SearchIndex и SearchInfo в объекте расширения DKSearchEngineInfoDL.

Дополнительную информацию о работе с XDO в Content Manager смотрите в разделе “Работа с элементами” на стр. 179.

Совет для C++: Посмотрите допустимые значения ContentClass в файле INCLUDE/DKConstant2DL.h в Content Manager.

Строки конфигурации DB2, ODBC и DataJoiner (только C++)

В этом разделе описаны строки конфигурации C++ DB2, ODBC и DataJoiner.

CC2MIMEFILE=(имяфайла)

Если хотите, задайте файл cmbcc2mime.ini.

DSNAME=(имя контент-сервера)

Если хотите, задайте имя контент-сервера. Если контент-сервер используется системой объединения, эта опция задается автоматически.

AUTOCOMMIT=ON | OFF

Если хотите, разрешите (ON) или запретите (OFF) автоматическое принятие. По умолчанию автоматическое принятие запрещено. Если контент-сервер используется системой объединения, автоматическое принятие всегда разрешено. Это значение задается автоматически.

В этом разделе описаны строки соединения C++ DB2, ODBC и DataJoiner.

NATIVECONNECTSTRING=(собственная строка соединения)

Если хотите, задайте собственную строку соединения, передаваемую при собственном вызове соединения.

SCHEMA=имя

Если хотите, задайте схему, используемую для функций listEntities, listEntityAttrs, listPrimaryKeyNames и listForeignKeyNames.

Советы по программированию на Java

В Content Manager V8 и новее XDO - это объект dkResource. PID объекта ресурсов представляется с помощью DKPidICM.

В ранних версиях Content Manager, Content Manager for AS/400 и IP 390 для идентификации XDO используется сочетание ID элемента, ID части и RepType (типа представления). В реляционных базах данных для идентификации XDO используется сочетание таблицы, столбца и строки предиката данных. Для работы с отдельным XDO используются ID элемента и ID части. RepType необязателен, так как система использует для него значение по умолчанию.

Для добавления текущих данных на контент-сервер используйте метод add объекта DKBlobxx. Если в дальнейшем надо выполнить еще какую-то операцию с этим объектом, можно получить значение ID части, присвоенное методом add .

Чтобы получить объект DKPid, примените метод getPidObject() к dkXDO.

Чтобы получить назначенный системой ID части, выполните после add следующий оператор:

Java

```
int partID = ((DKPidXDOICM)(axdo.getPidObject())).getPartId();
```

Внимание: В ранних версиях Content Manager при добавлении части нужно задать правильный ID части, чтобы эту часть мог использовать менеджер поиска (нельзя задавать ID части, равный 0).

В этом выпуске изменены два метода dkXDO: DKPid dkXDO.getPid() устарел и заменен getPidObject. DKPid dkXDO.getPidObject() Эти методы возвращали DKPidXDO; теперь они возвращают объект DKPid.

Советы по программированию на C++

В среде Content Manager, VI400 и IP390 для идентификации XDO используется сочетание ID элемента, ID части и RepType (тип представления). В среде реляционных баз данных ключом идентификации объекта XDO служит сочетание имени таблицы, имени столбца и datapredicate (предикат данных). Для отдельного XDO надо задать ID элемента и ID части. RepType задавать не обязательно, поскольку в системе есть значение по умолчанию (FRN\$NULL).

Для функции add нужно задать ID части. Если в дальнейшем надо выполнить еще какую-то операцию с этим объектом, можно получить значение ID части, присвоенное методом add .

Внимание: При добавлении части для индексации менеджером поиска на контент-сервере Content Manager надо указать для ID части допустимое ненулевое значение.

XDO как часть DDO

Если DDO представляет документ, то есть собрание объектов ресурсного содержимого, XDO представляет объект одной части. С объектом XDO можно работать как с компонентом объекта DDO или как с отдельным объектом. При обращении к XDO как к части DDO ID элемента для XDO получается из DDO. При работе с XDO как с отдельным объектом используется существующий ID элемента для этого XDO.

В следующем примере показано, как создать документ и добавить части документа в Content Manager. **Требование:** Пользовательский тип данных `S_docModel` должен быть определен в системе, классифицирован как документный и поддерживать типы частей `ICMBASE` и `ICMBASETEXT`, как показано в учебном примере `API SItemTypeCreationICM`.

Java

```
// Создаем документ
DKDDO ddoDocument = dsICM.createDDO("S_docModel", DKConstant.DK_CM_DOCUMENT);

// Создаем части
DKLobICM base = (DKLobICM) dsICM.createDDO("ICMBASE",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM baseText1 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_Basetext);
DKTextICM baseText2 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_Basetext);

// Задаем типы MIME частей (пример SResourceItemMimeTypesICM.txt)
base.setMimeType("application/msword");
baseText1.setMimeType("text/plain");
baseText2.setMimeType("text/plain");

// Загружаем содержимое в части (пример SResourceItemCreationICM)
base.setContentFromClientFile("SResourceItemICM_Document1.doc");
// Загружаем файл
baseText1.setContentFromClientFile("SResourceItemICM_Text1.txt");
// в память
baseText2.setContentFromClientFile("SResourceItemICM_Text2.txt");

// Обращаемся к атрибуту DKParts
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS));

// Добавляем части к документу
dkParts.addElement(base);
dkParts.addElement(baseText1);
dkParts.addElement(baseText2);

// Добавляем новый документ на постоянное хранение на склад данных
ddoDocument.add();
```

Полный пример программы смотрите в файле SDocModelItemICM.java в каталоге CMBROOT\Samples\java\icm. В SResourceItemCreationICM есть и другие примеры использования XDO.

C++

```
// Создаем документ
DKDDO* ddoDocument = dsICM->createDDO("S_docModel", DK_CM_DOCUMENT);
// Создаем части
DKLobICM* base = (DKLobICM*) dsICM->createDDO("ICMBASE",
                                                DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM* baseText1 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
                                                       DK_ICM_SEMANTIC_TYPE_BASETEXT);
DKTextICM* baseText2 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
                                                       DK_ICM_SEMANTIC_TYPE_BASETEXT);

// Задаем типы MIME частей (пример SResourceItemMimeTypesICM.txt)
base->setMimeType("application/msword");
baseText1->setMimeType("text/plain");
baseText2->setMimeType("text/plain");
// Загружаем содержимое в части (пример SResourceItemCreationICM)
// Загружаем файл в память
base->setContentFromClientFile("SResourceItemICM_Document1.doc");
baseText1->setContentFromClientFile("SResourceItemICM_Text1.txt");
baseText2->setContentFromClientFile("SResourceItemICM_Text2.txt");
// Обращаемся к атрибуту DKParts
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
    ddoDocument->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS));

// Добавляем части к документу
dkParts->addElement((dkDataObjectBase*)(DKDDO*)base);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText1);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText2);
// Добавляем новый документ на постоянное хранение на склад данных
ddoDocument->add();
```

Полный пример программы смотрите в файле SDocModelItemICM в каталоге CMBROOT\Samples\cpp\icm. В SResourceItemCreationICM есть и другие примеры использования XDO.

Отдельный XDO

Все примеры ниже относятся только к Content Manager 8.2. Примеры для предыдущих версий Content Manager и прочих контент-серверов можно найти в разделах “Представление элементов с помощью объектов DDO” на стр. 176 и “Работа с другими контент-серверами” на стр. 299; смотрите также примеры программ в каталоге CMBROOT\Samples.

Добавление XDO из буфера

В этом примере показано, как добавить XDO из буфера в Content Manager. В нем создается XDO, загружается содержимое в память и сохраняется содержимое из памяти на постоянное хранение. **Требование:** Пользовательский тип данных S_lob с классификацией ресурсный должен быть определен в системе, как показано в учебном примере API SItemCreationICM. Кроме того, менеджер

ресурсов и собрание SMS должны быть определены и заданы как менеджер и собрание по умолчанию для данного типа элементов или для данного пользователя, как это сделано в примерах SResourceMgrDefCreationICM, SSMSCollectionDefCreationICM, SResourceMgrDefSetDefaultICM и SSMSCollectionDefSetDefaultICM.

Java

```
// Создаем пустой ресурсный объект
DKLobICM lob = (DKLobICM) dsICM.createDDO("S_lob", DKConstant.DK_CM_DOCUMENT);

// Задаем тип MIME (пример SResourceItemMimeTypesICM.txt)
lob.setMimeType("application/msword");

// Загружаем содержимое в локальную память элемента
lob.setContentFromClientFile("SResourceItemICM_Document1.doc");

// Добавляем содержимое из памяти на склад данных
lob.add();
```

Полный пример программы смотрите в файле SResourceItemCreationICM в каталоге CMBROOT\Samples\java\icm.

C++

```
// Создаем пустой ресурсный объект
DKLobICM* lob = (DKLobICM*) dsICM->createDDO("S_lob", DK_CM_DOCUMENT);

// Задаем тип MIME (пример SResourceItemMimeTypesICM.txt)
lob->setMimeType("application/msword");

// Загружаем содержимое в локальную память элемента
lob->setContentFromClientFile("SResourceItemICM_Document1.doc");

// Добавляем содержимое из памяти на склад данных
lob->add();
```

Полный пример программы смотрите в файле SResourceItemCreationICM в каталоге CMBROOT\Samples\cpp\icm.

Добавление XDO из файла

В следующем примере показано, как добавить XDO на контент-сервер Content Manager (содержимое сохраняется напрямую из файла). **Требование:**

Пользовательский тип данных S_lob с классификацией ресурсный должен быть определен в системе, как показано в учебном примере API SItemCreationICM. Кроме того, менеджер ресурсов и собрание SMS должны быть определены и заданы как менеджер и собрание по умолчанию для данного

типа элементов или для данного пользователя, как это сделано в примерах SResourceMgrDefCreationICM, SSMSCollectionDefCreationICM, SResourceMgrDefSetDefaultICM и SSMSCollectionDefSetDefaultICM.

Java

```
// Создаем пустой ресурсный объект
DKTextICM text = (DKTextICM) dsICM.createDDO("S_text", DKConstant.DK_CM_ITEM);

// Задаем тип MIME (пример SResourceItemMimeTypesICM.txt)
text.setMimeType("text/plain");

// Сохраняем содержимое напрямую из файла
text.add("SResourceItemICM_Text1.txt");
```

Полный пример программы смотрите в файле SResourceItemCreationICM в каталоге CMBROOT\Samples\java\icm.

C++

```
// Создаем пустой ресурсный объект
DKTextICM* text = (DKTextICM*) dsICM->createDDO("S_text", DK_CM_ITEM);

// Задаем тип MIME (пример SResourceItemMimeTypesICM.txt)
text->setMimeType("text/plain");

// Сохраняем содержимое напрямую из файла
text->add("SResourceItemICM_Text1.txt");
```

Полный пример программы смотрите в файле SResourceItemCreationICM в каталоге CMBROOT\Samples\cpp\icm.

Добавление объекта комментария в XDO

В следующем примере показано, как добавить часть типа "комментарий" в документ в Content Manager. **Требование:** Пользовательский тип данных S_docModel должен быть определен в системе, классифицирован как документный и поддерживать тип частей ICMANNOTATION, как показано в примере SItemTypeCreationICM. Предполагается, что уже есть сохраненный на постоянном хранении экземпляр документа (переменная ddoDocument). Кроме того, предполагается, что есть объект DKDatastoreICM с установленным соединением (переменная dsICM).

Java

```
// Резервируем (то есть блокируем) элемент для изменения
dsICM.checkOut(ddoDocument);

// Создаем часть комментария
DKLobICM annot = (DKLobICM) dsICM.createDDO("ICMANNOATATION",
                                             DKConstantICM.DK_ICM_SEMANTIC_TYPE_ANNOTATION);

// Задаем тип MIME комментария (пример SResourceItemMimeTypesICM.txt)
annot.setMimeType("image/bmp");

// Загружаем содержимое в части (пример SResourceItemCreationICM)
annot.setContentFromClientFile("myAnnotation.bmp");

// Обращаемся к атрибуту DKParts
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
    DKConstant.DK_CM_NAMESPACE_ATTR, DKConstant.DK_CM_DKPARTS));

// Добавляем части к документу
dkParts.addElement(annot);

// Сохраняем изменения в постоянном хранении на складе данных
ddoDocument.update();

// Активируем (то есть освобождаем) элемент после изменения
dsICM.checkIn(ddoDocument);
```

Полный пример программы смотрите в файле `SDocModelItemICM` в каталоге `CMBROOT\Samples\java\icm`.

C++

```
// Резервируем (то есть блокируем) элемент для изменения
dsICM->checkOut(ddoDocument);

// Создаем часть комментария
DKLobICM* annot = (DKLobICM*) dsICM->createDDO("ICMANNOTATION",
                                                DK_ICM_SEMANTIC_TYPE_ANNOTATION);

// Задаем тип MIME комментария (пример SResourceItemMimeTypesICM.txt)
annot->setMimeType("image/bmp");

// Загружаем содержимое в части (пример SResourceItemCreationICM)
annot->setContentFromFile("myAnnotation.bmp");

// Обращаемся к атрибуту DKParts
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
    ddoDocument->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS));

// Добавляем части к документу
dkParts->addElement((dkDataObjectBase*)(DKDDO*)annot);

// Сохраняем изменения в постоянном хранении на складе данных
ddoDocument->update();

// Активируем (то есть освобождаем) элемент после изменения
dsICM->checkIn(ddoDocument);
```

Полный пример программы смотрите в файле SDocModelItemICM в каталоге CMBROOT\Samples\cpp\icm.

Примеры работы с XDO

Следующие примеры показывают использование отдельного XDO.

Получение, изменение и удаление XDO

Чтобы получить, изменить или удалить объект на контент-сервере, нужно указать правильные ID элемента, ID части и RepType для XDO, представляющего этот объект.

В следующем примере показано, как получить, изменить и удалить XDO в Content Manager. **Требование:** Пользовательский тип данных S_text с классификацией ресурсный должен быть определен в системе, как показано в учебном примере API SItemTypeCreationICM. Кроме того, менеджер ресурсов и собрание SMS должны быть определены и заданы как менеджер и собрание по умолчанию для данного типа элементов или для данного пользователя, как это сделано в примерах SResourceMgrDefCreationICM, SSMSCollectionDefCreationICM, SResourceMgrDefSetDefaultICM и

SSMSCollectionDefSetDefaultICM. Предполагается также, что переменная pidString содержит строку PID для ресурсного элемента, который уже существует на контент-сервере.

Java

```
// Задано: String pidString

// Вновь создаем пустые DDO для существующего элемента
// (пример SItemRetrievalICM)

DKLobICM lob = (DKLobICM) dsICM.createDDO(pidString);

// Получаем элемент с ресурсным содержанием
lob.retrieve(DKConstant.DK_CM_CONTENT_YES);

// Резервируем (то есть блокируем) элемент для изменения
// (пример SItemRetrievalICM)
dsICM.checkOut(lob);

// Задаем новый тип MIME (пример SResourceItemMimeTypesICM.txt)
lob.setMimeType("application/msword");

// Сохраняем новое содержимое на складе данных
lob.update("SResourceItemICM_Document2.doc");

// Активируем (то есть освобождаем) элемент после изменения
dsICM.checkIn(lob);

// Удаляем элемент
lob.del();
```

Этот фрагмент программного кода взят из примеров SResourceItemRetrievalICM, SResourceItemUpdateICM и SResourceItemDeletionICM в каталоге CMBROOT\Samples\java\icm.

C++

```
// Задано: DKString pidString

// Вновь создаем пустые DDO для существующего элемента
// (пример SItemRetrievalICM)
DKLobICM* lob = (DKLobICM*) dsICM->createDDO(pidString);

// Получаем элемент с ресурсным содержанием
lob->retrieve(DK_CM_CONTENT_YES);

// Резервируем (то есть блокируем) элемент для изменения
// (пример SItemUpdateICM)
dsICM->checkOut(lob);

// Задаем новый тип MIME (пример SResourceItemMimeTypesICM.txt)
lob->setMimeType("application/msword");

// Сохраняем новое содержимое на складе данных
lob->update("SResourceItemICM_Document2.doc");

// Активируем (то есть освобождаем) элемент после изменения
dsICM->checkIn(lob);

// Удаляем элемент
lob->del();

// Освобождаем память
delete(lob);
```

Этот фрагмент программного кода взят из примеров
SResourceItemRetrievalICM, SResourceItemUpdateICM и
SResourceItemDeletionICM в каталоге CMBROOT\Samples\cpp\icm.

Вызов функции XDO

В этом примере показано, как протестировать класс DKBlob с помощью ранней версии сервера Content Manager. В этом примере требуется знать ID элемента и ID части объекта XDO.

Java

```
public class txdomiscDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int    partId = 5;
        String itemId = "GAWCVGGVFUG428UJ";
        String repType = "";
        // Проверяем число аргументов метода main и решаем, что делать
        if (args.length == 3)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            itemId = args[2];
            System.out.println("You enter: java txdomiscDL " +
                + partId + " " + repType + " " + itemId);
        }
        if (args.length == 2)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            System.out.println("You enter: java txdomiscDL " +
                + partId + " " + repType);
        }
        if (args.length == 1)
        {
            partId =(short)Integer.parseInt(args[0], 10);
            System.out.println("You enter: java txdomiscDL " + partId );
            System.out.println("The supplied default repType = " + repType);
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 0)
        {
            System.out.println("invoke: java txdomiscDL  ");
            System.out.println("No parameter, following defaults provided:");
            System.out.println("    default partId = " + partId);
            System.out.println("    default repType = " + repType);
            System.out.println("    default itemId = " + itemId);
        }

        try
        {
            DKDatastoreDL dsDL = new DKDatastoreDL();
            System.out.println("connecting to datastore");
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            System.out.println("datastore connected");

            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
        }
    }
}

// продолжение следует...
```


Java (продолжение)

```
        apid.setRepType(repType);
        axdo.setPidObject(apid);
        System.out.println("repType=" + apid.getRepType());
        System.out.println("itemid=" + apid.getItemId());
        System.out.println("partId=" + apid.getPartId());
        // ----- Перед получением
        System.out.println("before retrieve:");
        System.out.println("  contentclass=" + axdo.getContentClass());
        System.out.print("  content length=" + axdo.length());
        System.out.println(" (the length of this object instance - in memory)");
        System.out.print("  getSize=" + axdo.getSize());
        System.out.println(" (get the object size without retrieving object)");
        System.out.println("  createdTimestamp=" + axdo.getCreatedTimestamp());
        System.out.println("  updatedTimestamp=" + axdo.getUpdatedTimestamp());
        axdo.retrieve();

        // ----- После получения
        System.out.println("after retrieve:");
        System.out.println("  contentclass=" + axdo.getContentClass());
        System.out.print("  content length=" + axdo.length());
        System.out.println(" (the length of this object instance - in memory)");
        System.out.print("  getSize=" + axdo.getSize());
        System.out.println(" (get the object size without retrieving object)");
        System.out.println("  createdTimestamp=" + axdo.getCreatedTimestamp());
        System.out.println("  updatedTimestamp=" + axdo.getUpdatedTimestamp());
        System.out.println("  affiliatedTyp=" + axdo.getAffiliatedType());
        if (axdo.getAffiliatedType() == DK_DL_ANNOTATION)
        {
            DKAnnotationDL ann =
            (DKAnnotationDL) (axdo.getExtension("DKAnnotationDL"));
            System.out.println("affil pageNumber=" + ann.getPageNumber());
            System.out.println("affil X=" + ann.getX());
            System.out.println("affil Y=" + ann.getY());
        }
        System.out.println("about to do open()...");
        axdo.setInstanceOpenHandler("notepad", true);
        int cc = axdo.getContentClass();
        if (cc == DK_DL_CC_GIF)
            axdo.setInstanceOpenHandler("lviewpro", true);
        else if (cc == DK_DL_CC_ASCII)
            axdo.setInstanceOpenHandler("notepad", true);
        else if (cc == DK_DL_CC_AVI)
            axdo.setInstanceOpenHandler("mplay32 ", true);
        axdo.open();
        dsDL.disconnect();
        dsDL.destroy();
    }
    catch(DKException exc)
    {
        // ----- Обработка исключительных ситуаций
    }
}
```


C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    long hsession;
    DKString itemId, repType;
    int partId;
    itemId = "GAWCVGGVFUG428UJ";
    repType = "FRN$NULL";
    partId = 2;

    cout <<"argc is "<<argc<<endl;
    if (argc == 1)
    {
        cout<<"invoke: txdomisc <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: txdomisc "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        cout<<"you enter: txdomisc "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        partId = atoi(argv[1]);
        repType = DKString(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"you enter: txdomisc "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }

    cout << connecting Datastore" << endl;
    try
    {
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;
        hsession = (long) (dsDL.connection()->handle());
        cout << "datastore handle" << hsession <<endl;
    }
    // продолжение следует...
```

C++ (продолжение)

```
DKBlobDL* axdo = new DKBlobDL(&dsDL);
DKPidXDODL* apid = new DKPidXDODL;
apid ->setPartId(partId);
apid ->setPrimaryId(itemId);
apid ->setRepType(repType);
axdo ->setPidObject(apid);
cout<<"itemId= "<<axdo->getItemId()<<endl;
cout<<"partId= "<<((DKPidXDODL*) (axdo->getPidObject()))
->getPartId()<<endl;
cout<<"repType= "<<axdo->getRepType()<<endl;

//== до получения
cout<<"before retrieve:"<<endl;
cout<<" content class="<<axdo->getContentClass()<<endl;
cout<<" content length="<<axdo->length();
cout<<" (the length of this object instance - in memory)"<<endl;
cout<<" getSize="<<axdo->getSize();
cout<<" (get the object size without retrieving object)"<<endl;
cout<<" createdTimestamp="<<axdo->getCreatedTimestamp()<<endl;
cout<<" updatedTimestamp="<<axdo->getUpdatedTimestamp()<<endl;
axdo->retrieve();

//== после получения
cout<<"after retrieve:"<<endl;
cout<<" content class="<<axdo->getContentClass()<<endl;
cout<<" content length="<<axdo->length();
cout<<" (the length of this object instance - in memory)"<<endl;
cout<<" getSize="<<axdo->getSize();
cout<<" (get the object size without retrieving object)"<<endl;
cout<<" createdTimestamp="<<axdo->getCreatedTimestamp()<<endl;
cout<<" updatedTimestamp="<<axdo->getUpdatedTimestamp()<<endl;
cout<<" mimeType="<<axdo->getMimeType()<<endl;
int atype = axdo->getAffiliatedType();
cout<<" affiliatedType= "<<axdo->getAffiliatedType()<<endl;
if (atype == DK_DL_ANNOTATION)
{
    DKAnnotationDL* ann=(DKAnnotationDL*)axdo
->getExtension("DKAnnotationDL");
    cout <<" pageNumber= "<<ann->getPageNumber()<<endl;
    cout <<" partId= "<<ann->getPart()<<endl;
    cout <<" X="<<ann->getX()<<endl;
    cout <<" Y="<<ann->getY()<<endl;
}
//== открыть содержимое
int concls = axdo->getContentClass();
if (concls == DK_DL_CC_ASCII)
    axdo->setInstanceOpenHandler("notepad", TRUE);
else if (concls == DK_DL_CC_GIF)
    axdo->setInstanceOpenHandler("lviewpro", TRUE);
// продолжение следует...
```

С++ (продолжение)

```
        else if (concls == DK_DL_CC_AVI)
            axdo->setInstanceOpenHandler("mplay32", TRUE);
        axdo->open();
        delete apid;
        delete axdo;
        dsDL.disconnect();
        cout<<"datastore disconnected"<<endl;
    }
    catch(DKException &exc)
    {
        cout << "Error id" << exc.errorId() << endl;
        cout << "Exception id " << exc.exceptionId() << endl;
        for(unsigned long i=0;i< exc.textCount();i++)
        {
            cout << "Error text:" << exc.text(i) << endl;
        }
        for (unsigned long g=0;g< exc.locationCount();g++)
        {
            const DKExceptionLocation* p = exc.locationAtIndex(g);
            cout << "Filename: " << p->fileName() << endl;
            cout << "Function: " << p->functionName() << endl;
            cout << "LineNumber: " << p->lineNumber() << endl;
        }
        cout << "Exception Class Name: " << exc.name() << endl;
    }
    cout << "done ..." << endl;
}
```

Добавление объекта мультимедиа XDO в ранних версиях Content Manager

Для каждого добавляемого объекта мультимедиа создается запись в таблице FRN\$MEDIA. Эта запись содержит информацию о пользовательских данных мультимедиа. Физический объект мультимедиа хранится на контент-сервере VideoCharger, заданном в сетевой таблице. В следующем примере требуется знать ID элемента объекта XDO.

Java

```
public class txdoAddVSDL implements DKConstantDL
{
// ----- Метод main
public static void main(String[] args)
{
    String fileName = "/icing1.mpg1";           //объект мультимедиа
    String itemId = "K1A04EWBVHJAV1D7";        //известный ID элемента
    int partId = 45;
    // ----- Проверка аргументов метода main
    if (args.length == 3)
    {
        fileName = args[0];
        partId = (int)Integer.parseInt(args[1], 10);
        itemId = args[2];
        System.out.println("You enter: java txdoAddVSDL " +
            fileName + " " + partId + " " + itemId);
    }
    if (args.length == 2)
    {
        fileName = args[0];
        partId = (int)Integer.parseInt(args[1], 10);
        System.out.println("You enter: java txdoAddVSDL " +
            fileName + " " + partId );
        System.out.println("The supplied default itemId = " + itemId);
    }
    if (args.length == 1)
    {
        fileName = args[0];
        System.out.println("You enter: java txdoAddVSDL " + fileName);
        System.out.println("The supplied default partId = " + partId);
        System.out.println("The supplied default itemId = " + itemId);
    }
    if (args.length == 0)
    {
        System.out.println("invoke: java txdoAddVSDL <filename> <part ID> <item ID>");
        System.out.println("No parameter, following defaults will be provided:");
        System.out.println("    default fileName = " + fileName);
        System.out.println("    default partId = " + partId);
        System.out.println("    default itemId = " + itemId);
    }
    // ----- Обработка
    try
    {
        // ----- соединяемся со складом данных
        DKDatastoreDL dsDL = new DKDatastoreDL();
        // подставьте в следующую строку библиотечный сервер,
        // ID пользователя, пароль
        System.out.println("connecting to datastore...");
        dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD", "");
        System.out.println("datastore connected");
    }
    // продолжение следует...
}
```

Java (продолжение)

```
// ----- создаем XDO и PID
DKBlobDL axdo = new DKBlobDL(dsDL);
DKPidXDODL apid = new DKPidXDODL();
apid.setPartId(partId);
apid.setPrimaryId(itemId);
axdo.setPidObject(apid);
// для объекта мультимедиа нужно использовать класс содержимого
// DK_DL_CC_IBMVSS
axdo.setContentClass(DK_DL_CC_IBMVSS);
System.out.println("contentClass=" + axdo.getContentClass());
System.out.println("partId = " + axdo.getPartId());
// ----- задаем DKMediaStreamInfoDL
DKMediaStreamInfoDL aVS = new DKMediaStreamInfoDL();
aVS.setMediaFullFileName(fileName);
// если fileName содержит список сегментов мультимедиа, используйте
// следующий оператор
//      aVS.setMediaObjectOption(DK_VS_LIST_OF_OBJECT_SEGMENTS);
aVS.setMediaObjectOption(DK_DL_VS_SINGLE_OBJECT);
aVS.setMediaHostName("<insert hostname here>");
aVS.setMediaUserId("<insert user ID here>");
aVS.setMediaPassword("<insert password here>");
// следующие строки не обязательны; если не задать, будут
// использованы значения по умолчанию
aVS.setMediaNumberOfUsers(2);
aVS.setMediaAssetGroup("AG");
// ----- то же, что определено на сервере VideoCharger
aVS.setMediaType("MPEG1");
aVS.setMediaResolution("SIF");
aVS.setMediaStandard("NTSC");
aVS.setMediaFormat("SYSTEM");
axdo.setExtension("DKMediaStreamInfoDL", (dkExtension)aVS);
System.out.println("about to call add()");
axdo.add();
System.out.println("add successfully.....");
System.out.println("after added check for status:");
boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
if (flag2)
{
    DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
        axdo.getExtension("DKMediaStreamInfoDL");
    System.out.println(" mediaformat=" + media.getMediaFormat());
    System.out.println(" mediaBitRate=" + media.getMediaBitRate());
    System.out.println(" mediastate(dynamic)=" +
        axdo.retrieveObjectState(DK_MEDIA_OBJECT));
}
dsDL.disconnect();
dsDL.destroy();
}
// продолжение следует...
```

Java (продолжение)

```
        catch (DKException exc) {
            try {
                dsDL.destroy();
            }
            catch (Exception e)
            {
                e.printStackTrace();
                System.out.println("Exception name " + exc.name());
                System.out.println("Exception message " + exc.getMessage());
                exc.printStackTrace();
            }
        }
        catch (Exception exc){
            try {
                dsDL.destroy();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            System.out.println("Exception message " + exc.getMessage());
            exc.printStackTrace();
        }
    }
}
```

C++

```
void main(int argc, char *argv[])
{
    DKString itemId, fileName;
    int partId;
    itemId = "K1A04EWBVHJAV1D7";
    partId = 22;
    fileName = "/icing1.mpg1";
    if (argc == 1)
    {
        cout<<"invoke: txdoAddVSDL <fileName> <partId> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default fileName = "<<fileName<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        fileName = DKString(argv[1]);
        cout<<"you enter: txdoAddVSDL "<<argv[1]<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        fileName = DKString(argv[1]);
        partId = atoi(argv[2]);
        cout<<"you enter: txdoAddVSDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        fileName = DKString(argv[1]);
        partId = atoi(argv[2]);
        itemId = DKString(argv[3]);
        cout<<"enter: txdoAddVSDL "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }
    try
    {
        //соединиться со складом данных
        cout << "Connecting datastore ..." << endl;
        DKDatastoreDL dsDL;
        // подставьте в строку ниже библиотечный сервер,
        // ID пользователя, пароль
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;
        // *** создать xdo и pid
        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setPrimaryId(itemId);
    }
    // продолжение следует...
```


С++ (продолжение)

```
axdo ->setPidObject(apid);
// для объекта мультимедиа нужно использовать класс содержимого
// DK_DL_CC_IBMVSS
axdo ->setContentClass(DK_DL_CC_IBMVSS);
cout <<"itemId= "<<axdo->getItemId()<<endl;
cout <<"partId= "<<axdo->getPartId()<<endl;
cout <<"repType= "<<axdo->getRepType()<<endl;
cout <<"content class="<< axdo->getContentClass()<<endl;
// *** настройка DKMediaStreamInfoDL
DKMediaStreamInfoDL aVS;
aVS.setMediaFullFileName(fileName);
aVS.setMediaObjectOption(DK_DL_VS_SINGLE_OBJECT);
aVS.setMediaHostName("<insert hostname here>");
aVS.setMediaUserId("<insert user ID here>");
aVS.setMediaPassword("<insert password here>");

//следующие строки не обязательны; если не задать,
//будут использованы значения по умолчанию
aVS.setMediaNumberOfUsers(1);
aVS.setMediaAssetGroup("AG");
// *** то же, что определено на сервере VideoCharger
aVS.setMediaType("MPEG1");
aVS.setMediaResolution("SIF");
aVS.setMediaStandard("NTSC");
aVS.setMediaFormat("SYSTEM");

axdo ->setExtension("DKMediaStreamInfoDL", (dkExtension*)&aVS);
cout <<"about to do add()"<<endl;
axdo ->add();
cout<<"Object added successfully "<<endl;

cout<<"after added check for status:"<<endl;
DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
if (flag2)
{
    DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
        axdo->getExtension("DKMediaStreamInfoDL");
    cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
    cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
    cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
    cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
    cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
    cout<<" MediaState(dynamic)="<<axdo->retrieveObjectState(DK_MEDIA_OBJECT)<<endl;
}
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
}
// продолжение следует...
```

C++ (продолжение)

```
cout << "Exception id " << exc.exceptionId() << endl;
for(unsigned long i=0;i< exc.textCount();i++)
{
    cout << "Error text:" << exc.text(i) << endl;
}
for (unsigned long g=0;g< exc.locationCount();g++)
{
    const DKExceptionLocation* p = exc.locationAtIndex(g);
    cout << "Filename: " << p->fileName() << endl;
    cout << "Function: " << p->functionName() << endl;
    cout << "LineNumber: " << p->lineNumber() << endl;
}
cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

Удаление объекта мультимедиа XDO

В следующем примере показано, как удалить объект мультимедиа XDO. В этом примере требуется знать ID элемента, ID части и RepType (тип представления) объекта XDO.

Java

```
public class txdoDelVSDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int    partId = 45;
        String repType = "";
        String itemId = "K1A04EWBVHJAV1D7";
        if (args.length == 3)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            itemId = args[2];
            System.out.println("You enter: java txdoDelVSDL " +
                + partId + " " + repType + " " + itemId);
        }
        // ----- Проверка аргументов метода main
        if (args.length == 2)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            repType = args[1];
            System.out.println("You enter: java txdoDelVSDL " +
                + partId + " " + repType);
        }

        if (args.length == 1)
        {
            partId = (short)Integer.parseInt(args[0], 10);
            System.out.println("You enter: java txdoDelVSDL " + partId );
            System.out.println("The supplied default repType = " + repType);
            System.out.println("The supplied default itemId = " + itemId);
        }
        if (args.length == 0)
        {
            System.out.println("invoke: java txdoDelVSDL <part ID> <RepType> <item ID>");
            System.out.println("No parameter, following defaults will be provided:");
            System.out.println("    default partId = " + partId);
            System.out.println("    default repType = " + repType);
            System.out.println("    default itemId = " + itemId);
        }

        // ----- Обработка
        try
        {
            DKDatastoreDL dsDL = new DKDatastoreDL();
            System.out.println("connecting to datastore...");
            // подставьте в следующую строку библиотечный сервер,
            // ID пользователя, пароль
            dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD", "");
            System.out.println("datastore connected");
        }

        // продолжение следует...
```

Java (продолжение)

```
DKBlobDL axdo = new DKBlobDL(dsDL);
DKPidXDODL apid = new DKPidXDODL();
apid.setPartId(partId);
apid.setPrimaryId(itemId);
apid.setRepType(repType);
axdo.setPidObject(apid);
boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
System.out.println("isMediaObject?" + flag2);
if (flag2)
{
    DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
        axdo.getExtension("DKMediaStreamInfoDL");
    System.out.println(" mediaformat=" + media.getMediaFormat());
    System.out.println(" mediaBitRate=" + media.getMediaBitRate());
    System.out.println(" mediastate(dynamic)=" +
        axdo.retrieveObjectState(DK_MEDIA_OBJECT));
    // ----- задаем опцию удаления для объекта мультимедиа
    axdo.setOption(DK_DL_OPT_DELETE_OPTION,
        (Object)new Integer(DK_DL_DELETE_NO_DROPITEM_MEDIA_AVAIL));
    System.out.println("The delete option =" +
        (Integer)(axdo.getOption(DK_OPT_DL_DELETE_OPTION)));
}

System.out.println("about to call del().. ");
axdo.del();
System.out.println("del successfully.....");
flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
System.out.println("after delete isMediaObject? = " + flag2);
System.out.println("about to call dsDL.disconnect()");
dsDL.disconnect();
dsDL.destroy();
}
// ----- Обработка исключительных ситуаций
catch (DKException exc) {
    try {
        dsDL.destroy();
    }
    catch (Exception e)
    {
        e.printStackTrace();
        System.out.println("Exception name " + exc.name());
        System.out.println("Exception message " + exc.getMessage());
        exc.printStackTrace();
    }
}
// продолжение следует...
```

Java (продолжение)

```
        catch (Exception exc){
            try {
                dsDL.destroy();
            }
            catch (Exception e)
            {
                e.printStackTrace();
                System.out.println("Exception message " + exc.getMessage());
                exc.printStackTrace();
            }
        }
    }
```

C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "Y68M1I@VYDG8SPQ4";
    partId = 1;
    repType = "FRN$NULL";
    if (argc == 1)
    {
        cout<<"invoke: txdoDelVSDL <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: txdoDelVSDL "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoDelVSDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        itemId = DKString(argv[3]);
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoDelVSDL "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }

    try
    {
        cout << "Connecting datastore ..." << endl;
        // подставьте в строку ниже библиотечный сервер,
        // ID пользователя, пароль
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
        cout << "datastore connected" << endl;

        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
    }
    // продолжение следует...
```

C++ (продолжение)

```
    apid ->setPartId(partId);
    apid ->setPrimaryId(itemId);
    apid ->setRepType(repType);
    axdo ->setPidObject(apid);
    cout <<"itemId= "<<axdo->getItemId()<<endl;
    cout <<"partId= "<<((DKPidXDODL*)(axdo->getPidObject()))
        ->getPartId()<<endl;
    DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
    cout <<"isMediaObject? = "<<flag2<<endl;
    if (flag2)
    {
        DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
            axdo->getExtension("DKMediaStreamInfoDL");
        cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
        cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
        cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
        cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
        cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
        cout<<" MediaState(dynamic)=
            "<<axdo->retrieveObjectState(DK_MEDIA_OBJECT)<<endl;

        cout<<"about to set the delete option for media object..."<<endl;
        DKAny delOpt = DK_DL_DELETE_NO_DROPITEM_MEDIA_AVAIL;
        axdo->setOption(DK_DL_OPT_DELETE_OPTION, delOpt);
        DKAny opt;
        axdo->getOption(DK_DL_OPT_DELETE_OPTION, opt);
        long lopt = opt;
        cout<<"The setted delete option = "<<lopt<<endl;

    }
    cout<<"about to do del()"<<endl;
    axdo->del();
    cout<<"del successfully..."<<endl;
    flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
    cout<<"after delete isMediaObject? = "<<flag2<<endl;
    delete axdo;
    delete apid;
    dsDL.disconnect();
    cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
}
// продолжение следует...
```

C++ (продолжение)

```
for (unsigned long g=0;g< exc.locationCount();g++)
{
    const DKExceptionLocation* p = exc.locationAtIndex(g);
    cout << "Filename: " << p->fileName() << endl;
    cout << "Function: " << p->functionName() << endl;
    cout << "LineNumber: " << p->lineNumber() << endl;
}
cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

Получение объекта мультимедиа XDO

В следующем примере показано, как получить объект мультимедиа XDO.

Полученный объект содержит только метаданные мультимедиа, а не сам объект мультимедиа. В этом примере требуется знать ID элемента и ID части объекта XDO.

Java

```
public class txdoretxsDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int    partId = 45;
        String itemId = "K1A04EWBVHJAV1D7";
        String repType = "";
        System.out.println("Processing using the following values: ");
        System.out.println("    Part Id = " + partId);
        System.out.println("    RepType = " + repType);
        System.out.println("    Item Id = " + itemId);
        try
        {
            DKDatastoreDL dsDL = new DKDatastoreDL();
            System.out.println("connecting to datastore...");
            // ----- подставьте в строку ниже библиотечный сервер,
            // ----- ID пользователя, пароль
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            System.out.println("datastore connected");
            DKBlobDL axdo = new DKBlobDL(dsDL);
            DKPidXDODL apid = new DKPidXDODL();
            apid.setPartId(partId);
            apid.setPrimaryId(itemId);
            apid.setRepType(repType);
            axdo.setPidObject(apid);
            System.out.println("repType=" + apid.getRepType());
            System.out.println("objectType=" + axdo.getObjectType());
            System.out.println("itemid=" + apid.getItemId());
            System.out.println("partId=" + apid.getPartId());

            boolean flag = axdo.isCategoryOf(DK_DL_INDEXED_OBJECT);
            boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
            System.out.println("isIndexedObject?=" + flag);
            System.out.println("isMediaObject?=" + flag2);
            if (flag)
            {
                DKSearchEngineInfoDL srch = (DKSearchEngineInfoDL)
                    axdo.getExtension("DKSearchEngineInfoDL");
                System.out.println("    serverName=" + srch.getServerName());
                System.out.println("    textIndex=" + srch.getTextIndex());
                System.out.println("    timeStamp=" + srch.getSearchTimestamp());
                System.out.println("    searchIndex=" + srch.getSearchIndex());
                System.out.println("    indexedState=" +
                    axdo.retrieveObjectState(DK_INDEXED_OBJECT));
            }

            if (flag2)
            {
                DKMediaStreamInfoDL media = (DKMediaStreamInfoDL)
                    axdo.getExtension("DKMediaStreamInfoDL");
            }
        }
    }
}

// продолжение следует...
```

Java (продолжение)

```
        System.out.println(" mediaformat=" + media.getMediaFormat());
        System.out.println(" mediaBitRate=" + media.getMediaBitRate());
        System.out.println(" mediastate(dynamic)=" +
            axdo.retrieveObjectState(DK_MEDIA_OBJECT));
    }
    System.out.println("before retrieve.....");
    System.out.println(" lob length=" + axdo.length());
    System.out.println(" size=" + axdo.getSize());
    System.out.println(" createdTimestamp=" + axdo.getCreatedTimestamp());
    System.out.println(" updatedTimestamp=" + axdo.getUpdatedTimestamp());
    // ----- Выполнить вызов retrieve
    axdo.retrieve();

    System.out.println("after retrieve.....");
    System.out.println(" lob length=" + axdo.length());
    System.out.println(" size=" + axdo.getSize());
    System.out.println(" mimeType=" + axdo.getMimeType());
    System.out.println(" createdTimestamp=" + axdo.getCreatedTimestamp());
    System.out.println(" updatedTimestamp=" + axdo.getUpdatedTimestamp());
    System.out.println("affiliatedTyp=" + axdo.getAffiliatedType());
    if (axdo.getAffiliatedType() == DK_DL_ANNOTATION)
    {
        DKAnnotationDL ann =
            (DKAnnotationDL)(axdo.getExtension("DKAnnotationDL"));
        System.out.println("affil pageNumber=" + ann.getPageNumber());
        System.out.println("affil X=" + ann.getX());
        System.out.println("affil Y=" + ann.getY());
    }
    System.out.println("about to do open()...");
    axdo.setInstanceOpenHandler("notepad", true); //default for Windows
    int cc = axdo.getContentClass();
    if ( cc == DK_DL_CC_GIF)
        axdo.setInstanceOpenHandler("lviewpro ", true); // использовать lviewpro
    else if (cc == DK_DL_CC_AVI)
        axdo.setInstanceOpenHandler("mplay32 ", true); // использовать mplay32
    else if (cc == DK_DL_CC_IBMVSS)
        axdo.setInstanceOpenHandler("iscoview ", true); // использовать iscoview
    axdo.open();

    dsDL.disconnect();
    dsDL.destroy();
}
catch(DKException exc)
{
    ... // обработка исключительных ситуаций и
        // разрушение объекта склада данных+
}
}
```

C++

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, repType;
    int partId;
    itemId = "K1A04EWBVHJAV1D7";
    partId = 1;
    repType = "FRN$NULL";
    if (argc == 1)
    {
        cout<<"invoke: txdoRetxsDL <partId> <repType> <itemId>"<<endl;
        cout<<" no parameter, following default will be provided:"<<endl;
        cout<<"The supplied default partId = "<<partId<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 2)
    {
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]<<endl;
        cout<<"The supplied default repType = "<<repType<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 3)
    {
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]<<" "<<argv[2]<<endl;
        cout<<"The supplied default itemId = "<<itemId<<endl;
    }
    else if (argc == 4)
    {
        itemId = DKString(argv[3]);
        repType = DKString(argv[2]);
        partId = atoi(argv[1]);
        cout<<"you enter: txdoRetxsDL "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
    }
    try
    {
        cout << "Connecting datastore ..." << endl;
        // подставьте в следующую строку библиотечный сервер,
        // ID пользователя, пароль
        dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
        cout << "datastore connected" << endl;

        DKBlobDL* axdo = new DKBlobDL(&dsDL);
        DKPidXDODL* apid = new DKPidXDODL;
        apid ->setPartId(partId);
        apid ->setPrimaryId(itemId);
        apid ->setRepType(repType);
    }
    // продолжение следует...
```

C++ (продолжение)

```
axdo ->setPidObject(apid);
cout <<"itemId= "<<axdo->getItemId()<<endl;
cout <<"partId= "
    <<((DKPidXDODL*)(axdo->getPidObject()))->getPartId()<<endl;
DKBoolean flag = axdo->isCategoryOf(DK_DL_INDEXED_OBJECT);
DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
cout <<"isIndexed? = "<<flag<<endl;
cout <<"isMediaObject? = "<<flag2<<endl;
if (flag)
{
    DKSearchEngineInfoDL* srchInfo = (DKSearchEngineInfoDL*)
        axdo->getExtension("DKSearchEngineInfoDL");
    cout<<" ServerName="<<srchInfo->getServerName()<<endl;
    cout<<" TextIndex="<<srchInfo->getTextIndex()<<endl;
    cout<<" srchEngine="<<srchInfo->getSearchEngine()<<endl;
    cout<<" srchIndex="<<srchInfo->getSearchIndex()<<endl;
    cout<<" indexedState="<<axdo
        ->retrieveObjectState(DK_DL_INDEXED_OBJECT)<<endl;
}

if (flag2)
{
    DKMediaStreamInfoDL* mediaInfo = (DKMediaStreamInfoDL*)
        axdo->getExtension("DKMediaStreamInfoDL");
    cout<<" copyRate="<<mediaInfo->getMediaCopyRate()<<endl;
    cout<<" mediaType="<<mediaInfo->getMediaType()<<endl;
    cout<<" mediaFrameRate="<<mediaInfo->getMediaFrameRate()<<endl;
    cout<<" mediaState="<<mediaInfo->getMediaState()<<endl;
    cout<<" mediaTimestamp="<<mediaInfo->getMediaTimestamp()<<endl;
    cout<<" MediaState(dynamic)= "
        <<axdo->retrieveObjectState(DK_DL_MEDIA_OBJECT)<<endl;
}

cout<<"before retrieve..."<<endl;
cout <<" length of lobdata = "<<axdo->length()<<endl;
cout<<" size of lobdata = "<<axdo->getSize()<<endl;
cout<<" created Timestamp = "<<axdo->getCreatedTimestamp()<<endl;
cout<<" updated Timestamp = "<<axdo->getUpdatedTimestamp()<<endl;
axdo->retrieve();
cout<<"after retrieve..."<<endl;
cout <<" length of lobdata = "<<axdo->length()<<endl;
cout <<" mimeType = "<<axdo->getMimeType()<<endl;
cout <<" size of lobdata = "<<axdo->getSize()<<endl;
cout<<" created Timestamp = "<<axdo->getCreatedTimestamp()<<endl;
cout<<" updated Timestamp = "<<axdo->getUpdatedTimestamp()<
    <endl;
// продолжение следует...
```

C++ (продолжение)

```
int atype = axdo->getAffiliatedType();
cout <<"affiliatedType= "<<axdo->getAffiliatedType()<<endl;
if (atype == DK_ANNOTATION)
{
    DKAnnotationDL* ann = (DKAnnotationDL*)axdo
        ->getExtension("DKAnnotationDL");
    cout<<"  pageNumber= "<<ann->getPageNumber()<<endl;
    cout<<"  partId= "<<ann->getPart()<<endl;
    cout<<"  X= "<<ann->getX()<<endl;
    cout<<"  Y= "<<ann->getY()<<endl;
}
cout<<"about to do open()..."<<endl;
axdo->setInstanceOpenHandler("notepad", TRUE);
//по умолчанию использовать Notepad в Windows
int concls = axdo->getContentClass();
if (concls == DK_DL_CC_GIF)
    axdo->setInstanceOpenHandler("lviewpro", TRUE);
// использовать lviewpro в Windows
else if (concls == DK_DL_CC_AVI)
    axdo->setInstanceOpenHandler("mplay32", TRUE);
// использовать mplay32 в Windows
else if (concls == DK_DL_CC_IBMVSS)
    axdo->setInstanceOpenHandler("iscoview", TRUE);
// использовать iscoview в Windows
axdo->open();

delete axdo;
delete apid;
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    for (unsigned long g=0;g< exc.locationCount();g++)
    {
        const DKExceptionLocation* p = exc.locationAtIndex(g);
        cout << "Filename: " << p->fileName() << endl;
        cout << "Function: " << p->functionName() << endl;
        cout << "LineNumber: " << p->lineNumber() << endl;
    }
    cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

Добавление XDO в собрание хранения

Чтобы добавить объект XDO, связанный с пользовательскими именами собрания хранения, используйте объект расширения DKStorageManageInfoxx, где xx - суффикс, представляющий определенный сервер.

В примере ниже показана работа DKStorageManageInfoDL с ранней версией сервера Content Manager; работе с Content Manager Версии 8 и новее посвящен раздел "Работа с Content Manager Версии 8.2" на стр. 159.

Java

```
String fileName = "e:\\test\\notepart.txt";    //добавляемый файл
int    partId = 0;                            //позволяем системе
                                              //выбрать ID части

String itemId = "V5SPB$WBLOHIQ4YI";          //существующий ID элемента
DKDatastoreDL dsDL = new DKDatastoreDL();    //необходимый склад данных
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD",""); //соединяемся со складом
DKBlobDL axdo = new DKBlobDL(dsDL);          //создаем XDO
DKPidXDODL apid = new DKPidXDODL();          //создаем PID
apid.setPartId(partId);                      //задаем ID части
apid.setPrimaryId(itemId);                   //задаем ID элемента
axdo.setPidObject(apid);                     //задаем объект PID
axdo.setContentClass(DK_DL_CC_ASCII);        //задаем класс содержимого

// ----- Создаем DKStorageManageInfoDL
StorageManageInfoDL aSMS = new DKStorageManageInfoDL();
aSMS.setRetention(888);                      //необязательно
aSMS.setCollectionName("TESTCOLLECT1");      //уже определено в DL SMS
aSMS.setManagementClass("TESTMGT1");        //необязательно
aSMS.setStorageClass("FIXED");               //необязательно
axdo.setExtension("DKStorageManageInfoDL", (dkExtension)aSMS);
axdo.add(fileName);                          //добавляем из файла
System.out.println("after add partId = " + axdo.getPartId());
                                              //выводим ID части после добавления
dsDL.disconnect(); //отсоединяемся от склада данных и
                  //разрушаем его объект
dsDL.destroy();
// ----- Обработка исключительных ситуаций
```

C++

```
DKString fileName = "e:\\test\\notepart.txt"; //добавляемый файл
int    partId = 0;                          //позволяем системе выбрать
                                           //ID части
DKString itemId = "V5SPB$WBLOHIQ4YI";      //существующий ID элемента
DKString rtype = "FRN$NULL";               //необязательно
DKDatastoreDL dsDL;                        //необходимый склад данных
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD"); //соединяемся со складом
DKBlobDL* axdo = new DKBlobDL(&dsDL);      //создать XDO
DKPidXDODL* apid = new DKPidXDODL;         //создать PID
apid->setPartId(partId);                    //задаем ID части
apid->setPrimaryId(itemId);                 //задаем ID элемента
apid->setRepType(rtype);                   //задаем тип представления
axdo->setPidObject(apid);                  //задаем объект PID
axdo->setContentClass(DK_DL_CC_ASCII);      //задаем класс содержимого

//---здать DKStorageManageInfoDL-----
DKStorageManageInfoDL aSMS = new DKStorageManageInfoDL();
aSMS.setRetention(888);                   //необязательно
aSMS.setCollectionName("TESTCOLLECT1");   //уже определено в DL SMS
aSMS.setManagementClass("TESTMGT1");      //необязательно
aSMS.setStorageClass("FIXED");            //необязательно
axdo->setExtension("DKStorageManageInfoDL", (dkExtension)aSMS);
axdo->add(fileName);                      //добавить из файла
System.out.println("after add partId = " + axdo->getPartId());
                                           //выводим ID части после добавления
dsDL.disconnect();                       //отсоединяемся от склада данных
System.out.println("datastore disconnected");
```

Примеры добавления в Content Manager объектов, индексированных для поиска, и объектов мультимедиа смотрите в следующих примерах кода в каталоге CMBROOT\Samples.

- TxdoAddBsmsDL
- TxdosAddBsmsDL
- TxdoAddFsmsDL
- TxdosAddFsmsDL
- TxdomAddsmsDL

Изменение собрания хранения объекта XDO

Вы можете изменить собрание хранения существующего объекта XDO. После задания объекта расширения DKStorageManageInfoICM вызовите метод changeStorage.

Java

```
System.out.println("about to call changeStorage().....");  
axdo.changeStorage();  
System.out.println("changeStorage() success.....");
```

C++

```
System.out.println("about to call changeStorage().....");  
axdo->changeStorage();  
System.out.println("changeStorage() success.....");
```

Полная программа примера, из которой взят приведенный фрагмент (TxdoChgSmsICM), находится в каталоге CMBROOT\Samples\java\d1 или CMBROOT\Samples\cpp\d1.

Работа с XML (только Java)

Enterprise Information Portal поддерживает импорт и экспорт содержимого из документов XML на сервер Content Manager и обратно в форме DDO и XDO на базе API Java. Эта возможность позволяет импортировать, сохранять и получать в Content Manager разнообразные объекты (данные или мультимедийное содержимое) из разнородных информационных систем без создания отдельных интерфейсов для каждой системы. Например, если некий объект хранится в одной системе данных, его можно преобразовать в файл XML и затем импортировать в Content Manager с помощью API Java Enterprise Information Portal. После импорта в Content Manager с этим объектом можно делать все то же, что и с любым другим объектом Content Manager.

Расширение возможностей импорта и экспорта XML

В настоящее время функции импорта и экспорта XML `DKDDO.toXML()` и `DKDDO.fromXML()` не поддерживают элементы, связанные ссылочными атрибутами, связями или через вхождение в папки. Операция экспорта выполняется, но она использует PID в качестве ссылки на элемент на конкретном контент-сервере. Если затем выполнить импорт на другой контент-сервер, такие элементы не будут существовать в системе назначения, поскольку эта информация PID применима только для того контент-сервера, с которого она получена. Эти две функции могут только создавать новые элементы и не поддерживают несколько версий одного элемента.

Для решения этой проблемы нужно построить отдельный инструмент, обрабатывающий элементы, на которые есть ссылки, обнаруживающий и перезаписывающий существующие элементы и выполняющий все другие действия, которые не поддерживаются этими функциями.

EIP Версии 8.1 Fixpack 1 и Версии 8.2 содержит новые примеры инструментов (TImportICM и TExportICM) и пример API инструментов (TExportPackageICM), которые выполняют все функции импорта и экспорта. Эти новые функции значительно расширяют возможности функций DKDDO.toXML() и DKDDO.fromXML(), показанные в SitemXMLImportExportICM.

Эти новые инструменты поставляются с Версией 8.1 Fixpack 1 и Версией 8.2 в виде примеров программ Java. Дополнительную информацию смотрите в документе Sample Import / Export Tools and API и в файле TExportPackageICM.doc в каталоге CMBROOT\Samples\java\icm.

Импорт документов XML

Данные XML можно импортировать из различных источников, включая стандартное устройство ввода, файлы, буферы и адреса Web (URL). Можно также импортировать файл XML целиком. Эти конструкторы выделяют содержимое документа XML, создают соответствующий DKDDO и все связанные с ним dkXDO. Затем можно вызвать для DDO метод add, чтобы добавить объект в Content Manager. Новый DDO относится к типу элемента Content Manager Версии 8 или к индексному классу Content Manager ранней версии, и его можно сохранить только в Content Manager. Импорт автореферентного файла XML позволяет сохранить исходный файл XML как XDO; то есть XML не теряется в процессе импорта и впоследствии можно использовать сам XML.

При импорте содержимого из XML воспользуйтесь следующими методами DKDDO:

```
toXML(DKNVPair xmlDestination, java.lang.String path, int options)
fromXML(DKNVPair xmlSource, int options)
```

Использование конструкторов DKDDO для импорта XML с ранней версией Content Manager более не поддерживается.

При импорте содержимого XML имейте в виду следующее:

1. Не забывайте, что импортировать можно только в Content Manager или раннюю версию Content Manager.
2. Файлы XML, содержимое которых нужно импортировать, должны удовлетворять показанному ниже определению типа документа XML.
3. Для импорта и экспорта XML можно использовать только API Java.

В следующих разделах описываются предварительные требования и методы импорта содержимого XML:

- Определение типа документа (DTD) XML
- Сохранение содержимого в документах XML
- Выделение содержимого из различных источников XML

- Импорт содержимого XML в Content Manager.

Определение типа документа (DTD) XML

Чтобы импортировать содержимое в Content Manager для хранения XML, нужно сохранить содержимое в виде документов XML, удовлетворяющих определению типа документа ddo.dtd (его полный путь: CMROOT\samples\java\dl\ddo.dtd).

```
<!ELEMENT ddo (pid?, pidIdStrings*, propertyCount?, property*, dataCount?, dataItem*, xdoValue?)>
<!ATTLIST ddo
    entityName CDATA #REQUIRED
    xmlns      CDATA #FIXED "http://www.omg.org/pub/docs/formal/97-12-12.pdf#ddo/EIP-7.1"
    xdoType    CDATA #IMPLIED
    dsType     CDATA #IMPLIED>

<!ELEMENT pid EMPTY>
<!ATTLIST pid
    dsType      CDATA #IMPLIED
    dsName      CDATA #IMPLIED
    objectType  CDATA #IMPLIED
    pidString   CDATA #IMPLIED>

<!ELEMENT pidIdStrings EMPTY>
<!ATTLIST pidIdStrings
    idPosition CDATA #REQUIRED
    idValue    CDATA #REQUIRED>

<!ELEMENT propertyCount (#PCDATA)>
<!ELEMENT property EMPTY>
<!ATTLIST property
    propertyId   CDATA #IMPLIED
    propertyName CDATA #IMPLIED
    propertyValue CDATA #IMPLIED
    propertyType CDATA #IMPLIED>

<!ELEMENT dataCount (#PCDATA)>
<!ELEMENT dataItem (dataPropertyCount?, dataProperty+, (dataValue | dataValues))>
<!ATTLIST dataItem
    dataId      CDATA #IMPLIED
    dataName    CDATA #REQUIRED
    dataNameSpace CDATA #IMPLIED>

<!ELEMENT dataPropertyCount (#PCDATA)>
<!ELEMENT dataProperty EMPTY>
<!ATTLIST dataProperty
    propertyId   CDATA #IMPLIED
    propertyName CDATA #IMPLIED
    propertyValue CDATA #IMPLIED
    propertyType CDATA #IMPLIED>

<!ELEMENT dataValues (dataValueCount?, dataValue*)>
<!ATTLIST dataValues
    collectionName CDATA #IMPLIED>
<!ELEMENT dataValueCount (#PCDATA)>
<!ELEMENT dataValue (#PCDATA | ddo | xdoRef | linkRef)*>
<!ELEMENT linkRef (linkSource, linkTarget, linkItem?)>
<!ATTLIST linkRef
    linkTypeName CDATA #REQUIRED>

<!ELEMENT linkSource (ddo)>
<!ATTLIST linkSource
    sameAsParentDDO (yes | no) "no">
<!ELEMENT linkTarget (ddo)>
<!ATTLIST linkTarget
    sameAsParentDDO (yes | no) "no">
<!ELEMENT linkItem (ddo)>
<!ELEMENT xdoRef (xdoPid, xdoIdStrings*, xdoValue)>
<!-- partId устарел, будет заменен xdoIdString в xdoRef -->
<!-- repType устарел, будет заменен xdoIdString в xdoRef -->
<!ELEMENT xdoPid EMPTY>
<!ATTLIST xdoPid
    dsType      CDATA #REQUIRED
    dsName      CDATA #IMPLIED
    xdoType     CDATA #REQUIRED
    objectType  CDATA #IMPLIED
    partId      CDATA #IMPLIED
    repType     CDATA #IMPLIED
    pidString   CDATA #IMPLIED>

<!ELEMENT xdoIdStrings EMPTY>
<!ATTLIST xdoIdStrings
    idPosition CDATA #REQUIRED
    idValue    CDATA #REQUIRED>

<!ELEMENT xdoValue (contentType?, specificInfo*, searchEngineInfo?, smsInfo?, xdoContent?)>
<!ATTLIST xdoValue
    refType     CDATA #REQUIRED
    refEncoding CDATA #IMPLIED
    mimeType    CDATA #REQUIRED
    XML-LINK    CDATA #IMPLIED
    HREF        CDATA #IMPLIED>

<!ELEMENT contentType (#PCDATA)>
<!ELEMENT specificInfo EMPTY>
<!ATTLIST specificInfo
    infoGroup    CDATA #REQUIRED
    infoName     CDATA #REQUIRED
    infoValue    CDATA #REQUIRED>

<!-- searchEngineInfo устарела, будет заменена specificInfo -->
<!ELEMENT searchEngineInfo EMPTY>
<!ATTLIST searchEngineInfo
    searchEngine CDATA #REQUIRED
    searchIndex  CDATA #REQUIRED
    searchInfo   CDATA #REQUIRED>

<!-- smsInfo устарела, будет заменена specificInfo -->
<!ELEMENT smsInfo EMPTY>
```

| | | |
|---------------------------------|-----------------|-----------------|
| <!ATTLIST smsInfo | smsRetention | CDATA #IMPLIED |
| | smsCollection | CDATA #IMPLIED |
| | smsMgmtClass | CDATA #IMPLIED |
| | smsStorageClass | CDATA #IMPLIED |
| | smsObjServer | CDATA #IMPLIED> |
| <!ELEMENT xdoContent (#PCDATA)> | | |

Сохранение содержимого в документах XML

Файлы XML могут различным способом представлять документы или папки для импорта в Content Manager. Эти документы и папки могут также содержать части. В примере ниже сначала показан типичный элемент данных XML, dataItem dataId="1", со значением Basuki. Но в DataItem 13 используется dataName DKParts, относящийся к автореферентному XDO.

Пример для сервера Content Manager

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ddo SYSTEM "ddo.dtd">
<ddo entityName="MagXML3" dsType="ICM" xdoType="DKLobICM">
  <pid dsType="ICM" dsName="ICMNLSDb"/>
  <property propertyId="1" propertyName="item-type" propertyValue="document"/>
  <dataItem dataName="Classification3">
    <dataProperty propertyName="type" propertyValue="string" />
    <dataValue>B</dataValue>
  </dataItem>
  <dataItem dataName="PublisherName3">
    <dataProperty propertyName="type" propertyValue="string"/>
    <dataValue>PublisherName3</dataValue>
  </dataItem>
  <dataItem dataName="MagEdrXML3" dataNameSpace="CHILD">
    <dataProperty propertyName="type" propertyValue="collection+ddo"/>
  <dataValues collectionName="DKChildCollection">
    <dataValue>
      <ddo entityName="MagEdrXML3" dsType="ICM" xdoType="DKLobICM">
        <pid dsType="ICM" dsName="ICMNLSDb"/>
        <dataItem dataName="Sophias3.USPostal3">
          <dataProperty propertyName="type" propertyValue="string"/>
          <dataValue>Title of Book</dataValue>
        </dataItem>
        <dataItem dataName="Sophias3.Association3">
          <dataProperty propertyName="type" propertyValue="string"/>
          <dataValue>Sophias3.Association3</dataValue>
        </dataItem>
      </ddo>
    </dataValue>
  </dataValues>
  </dataItem>
  <xdoValue mimeType="text/plain" refType="file"
    XML-LINK="SIMPLE" HREF="TSophiaBefore.txt" >
  </xdoValue>
</ddo>
```

Некоторые примеры сохранения документов XML на сервере Content Manager смотрите в следующих примерах в каталоге DK\Samples\java\icm\:

Пример для ранней версии Content Manager

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ddo SYSTEM "ddo.dtd">
<ddo entityName="DLSAMPLE">
  <pid dsType="DL" dsName="LIBSRVRN"/>
  <property propertyId="1" propertyName="item-type" propertyValue="document"/>
  <dataItem dataId="1" dataName="DLSEARCH_Author">
    <dataProperty propertyId="1" propertyName="type" propertyValue="string"/>
    <dataValue>Basuki</dataValue>
  </dataItem>
  . . .
  <dataItem dataId="13" dataName="DKParts">
    <dataProperty propertyId="1" propertyName="type" propertyValue="collection+xdo"/>
    <dataProperty propertyId="2" propertyName="nullable" propertyValue="false"/>
    <dataValues>
      <dataValue>
        <xdoRef>
          <xdoPid dsType="DL" xdoType="DKBlobDL"/>
          <xdoValue refType="self" mimeType="text/xml">
            <contentType>XML</contentType>
          </xdoValue>
        </xdoRef>
      </dataValue>
    </dataValues>
  </dataItem>
</ddo>
```

Некоторые примеры сохранения документов XML на сервере Content Manager смотрите в следующих примерах в каталоге примеров:

- dlsamp01.xml
- dlsamp02.xml
- dlsamp03.xml
- dlsamp04.xml
- dlsamp05.xml
- dltypes01.xml

Выделение содержимого из различных источников XML

Методы DKDDO могут извлекать контент из разнообразных источников XML, включая стандартный ввод, файлы, буферы и адреса Web (URL). С помощью методов DKDDO можно выделять содержимое из источника XML и запускать процесс импорта.

Примеры для каждого типа источников XML:

XML из файла:

Java

```
xmlSource = new DKNVPair("FILE", "dlsamp01.xml");
```

XML из буфера:

Java

```
File file = new File("dlsamp01.xml");
int fileSize = (int) file.length();
byte[] data = new byte[fileSize];
DataInputStream dis = new DataInputStream(new FileInputStream(file));
dis.readFully(data);
String strBuffer = new String(data);
DKNVPair xmlSource = new DKNVPair("BUFFER", strBuffer);
int importOptions=DK_CM_XML_VALIDATION;
```

XML из адреса Web (URL):

Java

```
xmlSource = new DKNVPair("URL", "file:///d://myxml//dlsamp01.xml");
// Для URL замените file:///d:// на http://www.web-адрес.com/
int importOptions=0;
```

Импорт содержимого XML в Content Manager

В следующем примере показаны основные шаги импорта:

1. Создание DKDDO.
2. Создание контент-сервера (в данном случае Content Manager или Content Manager ранней версии) и соединение с ним.
3. Добавление этого нового DKDDO на контент-сервер (в данном случае опять Content Manager).
4. Импорт источника XML с помощью либо `dkDataObjectBase fromXML(DKNVPair xmlSource)`, либо `dkDataObjectBase fromXML(DKNVPair xmlSource, int options)`.

Полученный DKDDO удовлетворяет спецификациям `ddo.dtd` и входит в индексный класс Content Manager или Content Manager ранней версии.

Java

```
// ----- Создаем DDO, импортируя документ XML
xmlSource = new DKNVPair("FILE", "icmsamp01.xml");
int importOptions = DK_CM_XML_VALIDATION;
DKDDO ddo = new DKDDO();
ds = new DKDatastoreICM();
// ..... соединяемся со складом данных
ddo.setDatastore (ds);
// ----- Добавляем DDO на склад данных
ddo.add()
// ----- Импортируем XML
ddo.fromXML(xmlSource, importOptions);
```

Полный пример программы смотрите в файле
SItemXMLImportExportICM.java в каталоге CMBROOT\Samples\java\icm.

Экспорт XML

Данные DDO или XDO можно экспортировать как документ XML в Content Manager или Content Manager ранней версии. Чтобы экспортировать, примените метод toXML(DKNVPair xmlDestination, String path, int options) из DKDDO.

Ограничение: Показанные в этом примере функции DKDDO.toXML() и DKDDO.fromXML() нельзя использовать для экспорта элемента с одного контент-сервера и импорта его на другой контент-сервер, если этот элемент связан с другими элементами при помощи связей, ссылочных атрибутов или через вхождение в папки. Операция импорта создаст новые элементы и не будет изменять или перезаписывать существующие элементы, а также не поддерживает несколько версий элемента. Информацию об инструментах, расширяющих возможности этих функций, смотрите в разделе “Расширение возможностей импорта и экспорта XML” на стр. 100.

В следующем примере показано, как экспортировать XML:

Java

```
DKNVPair xmlDestination = null;
//----- Эта строка показывает экспорт в STDOUT
//xmlDestination = new DKNVPair("STDOUT", null);

//----- Эта строка показывает экспорт в буфер
//xmlDestination = new DKNVPair("BUFFER", new Object());

//----- Эта строка показывает экспорт в файл
String xmlFile = "export.xml";
if(!fileName.equals("")){
    xmlFile = fileName;
}

String strOS = System.getProperty("os.name");
if (strOS.indexOf("Win") != -1) { // система Windows
    xmlFile = outputPath + "\\\" + xmlFile;
} else { // система, отличная от Win
    xmlFile = outputPath + "/" + xmlFile;
}

xmlDestination = new DKNVPair("FILE", xmlFile);

ddo.toXML(xmlDestination, outputPath, DKConstant.DK_CM_XDO_REFERENCE);
```

Полный пример программы смотрите в файле
SitemXMLImportExportICM.java в каталоге CMBROOT\Samples\java\icm.

Создание документов и использование атрибута DKPARTS

Атрибут DKPARTS в DDO представляет собрание частей в документе. Значение этого атрибута - объект DKParts, представляющий собой собрание объектов DKPart. Объекты DKPart - это элементы типа элементов с классификацией *часть документа*; они содержат ресурсное содержимое.

Только для Content Manager: Документ - это элемент, который хранится, получается и передается из системы Content Manager в другую систему или пользователю как отдельная единица. Если для элемента задан семантический тип *документ*, считается, что он содержит информацию, составляющую документ, но он не обязан представлять реализацию определенной модели документа. Элемент, созданный в типе элементов с классификацией "документ" (ее также называют "документной моделью"), будет содержать части документа - это особая реализация модели документа, поддерживаемая Content Manager. Элементам, созданным в типе элементов с классификацией "документ", можно задать семантический тип "документ" или "папка". Части документа могут включать разные типы содержимого (например, текст, изображения и электронные таблицы).

В следующем примере показано, как создать документ и добавить части документа в Content Manager. **Требование:** Пользовательский тип данных `S_docModel` должен быть определен в системе и классифицирован как документный. Он должен также поддерживать типы частей `ICMBASE` и `ICMBASETEXT`, как показано в учебном примере `API SItemCreationICM`.

Java

```
// Создаем документ
DKDDO ddoDocument = dsICM.createDDO("S_docModel", DKConstant.DK_CM_DOCUMENT);

// Создаем части
DKLobICM base = (DKLobICM) dsICM.createDDO("ICMBASE",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM baseText1 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASSETEXT);
DKTextICM baseText2 = (DKTextICM) dsICM.createDDO("ICMBASETEXT",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASSETEXT);

// Задаем типы MIME частей (пример SResourceItemMimeTypesICM.txt)
base.setMimeType("application/msword");
baseText1.setMimeType("text/plain");
baseText2.setMimeType("text/plain");

// Загружаем содержимое в части (пример SResourceItemCreationICM)
base.setContentFromClientFile("SResourceItemICM_Document1.doc");
// Загружаем файл
baseText1.setContentFromClientFile("SResourceItemICM_Text1.txt");
// в память
baseText2.setContentFromClientFile("SResourceItemICM_Text2.txt");

// Обращаемся к атрибуту DKParts
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
DKConstant.DK_CM_NAMESPACE_ATTR, DKConstant.DK_CM_DKPARTS));

// Добавляем части к документу
dkParts.addElement(base);
dkParts.addElement(baseText1);
dkParts.addElement(baseText2);

// Добавляем новый документ на постоянное хранение на склад данных
ddoDocument.add();
```

Информацию о создании документов с частями смотрите в учебном примере `API SDocModelItemICM` в каталоге `CMBROOT\Samples\java\icm`.

C++

```
// Создаем документ
DKDDO* ddoDocument = dsICM->createDDO("S_docModel", DK_CM_DOCUMENT);

// Создаем части
DKLobICM* base = (DKLobICM*) dsICM->createDDO("ICMBASE",
DK_ICM_SEMANTIC_TYPE_BASE);
DKTextICM* baseText1 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
DK_ICM_SEMANTIC_TYPE_BASETEXT);
DKTextICM* baseText2 = (DKTextICM*) dsICM->createDDO("ICMBASETEXT",
DK_ICM_SEMANTIC_TYPE_BASETEXT);

// Задаем типы MIME частей (пример SResourceItemMimeTypesICM.txt)
base->setMimeType("application/msword");
baseText1->setMimeType("text/plain");
baseText2->setMimeType("text/plain");

// Загружаем содержимое в части (пример SResourceItemCreationICM)
base->setContentFromClientFile("SResourceItemICM_Document1.doc");
// Загружаем файл в память
baseText1->setContentFromClientFile("SResourceItemICM_Text1.txt");
baseText2->setContentFromClientFile("SResourceItemICM_Text2.txt");

// Обращаемся к атрибуту DKParts
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
ddoDocument->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS));

// Добавляем части к документу
dkParts->addElement((dkDataObjectBase*)(DKDDO*)base);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText1);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText2);

// Добавляем новый документ на постоянное хранение на склад данных
ddoDocument->add();
```

Информацию о создании документов с частями смотрите в учебном примере API SDocModelItemICM в каталоге CMBROOT\Samples\cpp\icm.

Объект DDO владеет всеми частями в собрании частей. Для изменения или удаления частей используйте DDO документа.

В следующем примере показано, как получать части и обращаться к ним с помощью DDO.

Java

```
// ПРИМ.: Функция print приведена в примере SDocModelItemICM

// Получаем объект DKParts.
short dataid = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
if(dataid==0)
    throw new Exception("No DKParts Attribute Found! Either item type does not
        support parts or the document has not been explicitly retrieved.");
DKParts dkParts = (DKParts) ddoDocument.getData(dataid);

// Обрабатываем список частей
dkIterator iter = dkParts.createIterator(); // Создаем итератор
while(iter.more()){                       // Пока есть еще элементы
    DKDDO part = (DKDDO) iter.next();      // Перемещаем указатель и
                                           // получаем следующую часть
System.out.println("Item Id: "+((DKPidICM)part.getPidObject()).getItemId());
}
```

Полный пример программы смотрите в файле SDocModelItemICM в каталоге CMBROOT\Samples\java\icm.

C++

```
// ПРИМ.: Функция print приведена в примере SDocModelItemICM

// Получаем объект DKParts.
short dataid = ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
if(dataid==0)
    throw DKException("No DKParts Attribute Found! Either item type does not
        support parts or the document has not been explicitly retrieved.");
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(dataid);

// Обрабатываем список частей
dkIterator* iter = dkParts->createIterator(); // Создаем итератор
while(iter->more()){                         // Пока есть еще элементы
    DKDDO* part = (DKDDO*) iter->next()->value(); // Перемещаем указатель и
                                           // получаем следующую часть
    cout << "Item Id:" << ((DKPidICM*)part->getPidObject())->getItemId()<< endl;
}
delete(iter);                               // Освобождаем память
```

Информацию о создании документов с частями смотрите в учебном примере API SDocModelItemICM в каталоге CMBROOT\Samples\cpp\icm.

Создание папок и использование атрибута DKFOLDER

В DDO-папке атрибут DKFOLDER указывает на собрание документов и других папок, входящих в эту папку. Значение этого атрибута - объект DKFolder, являющийся собранием объектов DDO. Как показано ниже, атрибут DKFolder задается так же, как атрибут DKParts.

Только для Content Manager: Папка - это элемент любого типа элементов, независимо от классификации, с семантическим типом *папка*. Любой элемент с семантическим типом "папка" обладает обеспечиваемыми Content Manager функциями папки, а также всеми возможностями нересурсных элементов и любыми дополнительными функциями, связанными с классификацией типа элементов (например, как документного или ресурсного). Папки могут содержать любое число элементов любого типа, включая документы и подпапки. Папка индексируется по атрибутам.

В следующем примере показано, как создать папку и добавить содержимое в Content Manager. **Требование:** Пользовательский тип данных S_simple должен быть определен в системе, как показано в учебном примере API SItemTypeCreationICM.

Java

```
// Создаем новую папку в памяти
DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);

// Создаем и сохраняем содержимое, которое будет помещено в папку
DKDDO ddoDocument = dsICM.createDDO("S_simple", DKConstant.DK_CM_DOCUMENT);
DKDDO ddoFolder2 = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
DKDDO ddoItem = dsICM.createDDO("S_simple", DKConstant.DK_CM_ITEM);
ddoDocument.add();
ddoItem.add();
ddoFolder2.add();

// Обращаемся к атрибуту DKFolder
DKFolder dkFolder = (DKFolder)
ddoFolder.getData(ddoFolder.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
                                   DKConstant.DK_CM_DKFOLDER));

// Добавляем содержимое в папку
dkFolder.addElement(ddoDocument);
dkFolder.addElement(ddoItem);
dkFolder.addElement(ddoFolder2); // Обратите внимание: папки могут
// содержать подпапки.

// Сохраняем папку на постоянном складе данных
ddoFolder.add();
```

Дополнительную информацию о создании папок смотрите в учебном примере API SFolderICM в каталоге CMBROOT\Samples\java\icm.

C++

```
// Создаем новую папку в памяти
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);

// Создаем и сохраняем содержимое, которое будет помещено в папку
DKDDO* ddoDocument = dsICM->createDDO("S_simple", DK_CM_DOCUMENT);
DKDDO* ddoFolder2 = dsICM->createDDO("S_simple", DK_CM_FOLDER);
DKDDO* ddoItem = dsICM->createDDO("S_simple", DK_CM_ITEM);
ddoDocument->add();
ddoItem->add();
ddoFolder2->add();

// Обращаемся к атрибуту DKFolder
DKFolder* dkFolder = (DKFolder*)(dkCollection*) ddoFolder->getData(
    ddoFolder->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKFOLDER));

// Добавляем содержимое в папку
dkFolder->addElement(ddoDocument);
dkFolder->addElement(ddoItem);
dkFolder->addElement(ddoFolder2); // Обратите внимание: папки могут
                                // содержать подпапки.

// Сохраняем папку на постоянном складе данных
ddoFolder->add();
```

Дополнительную информацию о создании папок смотрите в учебном примере API SFolderICM в каталоге CMBROOT\Samples\cpp\icm.

В Content Manager Версии 8 элемент-папка не владеет содержимым папки. Один элемент можно добавить в несколько папок. При удалении элемента из папки удаляется только управляемое системой отношение "входит в папку". Элементы в папке можно изменять и удалять независимо. В предыдущих версиях Content Manager объект DDO владеет содержимым собрания объектов.

В следующем примере показано, как получать содержимое папки и обращаться к нему при помощи DDO.

Java

```
// ПРИМ.: Функция print приведена в учебном примере API SFolderICM

// Получаем объект DKFolder
short dataid = folder.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
                             DKConstant.DK_CM_DKFOLDER);

if(dataid==0)
    throw new Exception("No DKFolder Attribute Found! DDO is either not a
        Folder or Folder Contents have not been explicitly retrieved.");
DKFolder dkFolder = (DKFolder) folder.getData(dataid);

// Обращаемся к содержимому
dkIterator iter = dkFolder.createIterator(); // Создаем итератор
while(iter.more()){                         // Пока есть еще элементы
    DKDDO ddo = (DKDDO) iter.next();        // Перемещаем указатель и получаем
                                           // следующий элемент
    System.out.println("Item Id: "+((DKPidICM)ddo.getPidObject()).getItemId());
}
```

Полный пример программы смотрите в учебном примере SFolderICM в каталоге CMBROOT\Samples\java\icm.

C++

```
// ПРИМ.: Функция print приведена в учебном примере API SFolderICM

// Получаем объект DKFolder
short dataid = folder->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKFOLDER);
if(dataid==0)
    throw DKException("No DKFolder Attribute Found! DDO is either not a Folder
        or Folder Contents have not been explicitly retrieved.");
DKFolder* dkFolder = (DKFolder*)(dkCollection*) folder->getData(dataid);

// Обращаемся к содержимому
dkIterator* iter = dkFolder->createIterator(); //Создаем итератор
while(iter->more()){                          //Пока есть еще элементы
    DKDDO* ddo = (DKDDO*) iter->next()->value(); //Перемещаем указатель и
                                           // получаем следующий элемент
    cout << "Item Id: " << ((DKPidICM*)ddo->getPidObject())->getItemId()<< endl;
}
```

Полный пример программы смотрите в учебном примере SFolderICM в каталоге CMBROOT\Samples\cpp\icm.

Использование объектов DKAny (только C++)

DKAny содержит все объекты, тип которых может меняться во время выполнения. Объект DKAny может быть любого из следующих типов:

- null
- (unsigned) short
- (unsigned) long
- double
- char
- TypeCode
- DKBoolean
- DKString
- DKDate
- DKTime
- DKTimestamp

Если объекту DKAny не было присвоено значение (DKAny any), он может быть только пустым (NULL). Когда значение присвоено, функция `DKAny::isNull()` будет возвращать значение FALSE.

Кроме приведенных выше типов, объект DKAny может содержать следующие типы ссылок на объекты:

- `dkDataObjectBase*`
- `dkCollection*`
- `void*`

Использование кода типа

Узнать текущий тип объекта DKAny можно при помощи функции `typeCode`, которая вернет объект `TypeCode`, то есть `tc_null` для пустого, `tc_short` для короткого типа и так далее. Полный список кодов типов можно найти в *электронном справочнике API*.

Управление памятью в DKAny

Объект DKAny управляет хранением в памяти содержащегося в нем объекта, если этот объект - не указатель. Операции копирования, работающие со объектами-указателями, создают только копию указателя. При копировании и удалении объектов-указателей необходимо следить за объектами.

Использование конструкторов

В DKAny есть конструкторы для всех поддерживаемых им типов. В следующем примере показано, как создать объект DKAny, который содержит некоторые из описанных в предыдущем разделе типов.

C++

```
DKAny any1((unsigned short) 10);           // содержит unsigned short 10
DKAny any2((long) 200);                     // содержит long 200
DKAny any3(DKString("any string"));         // содержит DKString
DKAny any4(DKTime(10,20,30));               // содержит DKTime
DKAny any5((dkDataObjectBase*) new DKDDO); // содержит DKDDO
DKAny any6(new MyObject(5,"abc"));          // содержит MyObject
DKAny any7(new DKDDO);                     // краткая версия any5
```

Получение кода типа

Используйте функцию `typeCode`, чтобы узнать код типа для объекта в `DKAny`.

C++

```
DKAny::TypeCode type_code;
type_code = any1.typeCode(); // type_code равен tc_ushort
type_code = any4.typeCode(); // type_code равен tc_time
type_code = any5.typeCode(); // type_code равен tc_dobase
                           // (указатель на объект)
type_code = any6.typeCode(); // type_code равен tc_voidptr, так как
                           // MyObject не опознается DKAny
```

Присваивание нового значения объекту DKAny

Чтобы присвоить новое значение существующему объекту `DKAny`, используйте операцию присваивания - знак равенства (=). `DKAny` поддерживает присваивание для всех кодов типа.

C++

```
DKAny any;                               // any содержит null
long vlong = 300;
DKTimestamp vts(1997,8,28,10,11,12,999);
dkDataObjectBase* dobase =
(dkDataObjectBase*) new DKDDO;
any = vlong;                             // any содержит long 300
any = vts;                               // any содержит timestamp
any = dobase;                             // any содержит ddo
any = new DKDDO;                          // any содержит ddo
```

Присваивание значения из DKAny

Чтобы вновь присвоить `DKAny` обычный тип, требуется операция преобразования типов. Например:

C++

```
vlong      = (long) any2;           // задает vlong равным 200
DKTime at   = (DKTime) any4;        // задает at равным (10,20,30)
DKDDO* ddo  = (DKDDO*) ((dkDataObjectBase*) any5); // извлечь DDO
dkDataObjectBase* dobase = any7;    // извлечь DDO
```

При несоответствии типов возникнет исключительная ситуация недопустимого преобразования типов. Поэтому перед тем, как преобразовать DKAny в обычный тип, надо проверить код типа:

C++

```
if (any5.typeCode() == DKAny::tc_dobase)
    dobase = (dkDataObjectBase*) any5;
```

Для проверки типа DKAny можно использовать оператор CASE:

C++

```
switch(any.typeCode()) {
    case DKAny::tc_short:
        // операция для short
        ...
        break;
    case DKAny::tc_ushort:
        // операция для unsigned short
        ...
        break;
    ... и так далее
}
```

Если объект DKAny содержит ссылку на объект, можно получить содержимое DKAny как указатель void, и затем преобразовать его в нужный тип. Но эту операцию следует использовать, только если вы знаете код типа, использованный в DKAny:

C++

```
// точно знаем, что any5 содержит DKDDO
ddo = (DKDDO*) any5.value();
```

Вывод DKAny

cout позволяет вывести содержимое объекта DKAny:

C++

```
cout << any3 << endl;           // выводит "any string"
cout << any4 << endl;           // выводит "10:20:30"
cout << any5 << endl;           // выводит "(dkDataObjectBase*) <адрес>",
                                // где адрес - положение DDO в памяти
```

Уничтожение DKAny

Поскольку DKAny может содержать ссылку на объект, но не может управлять памятью для объектов-указателей, памятью для этих типов надо управлять. В следующем примере показано управление памятью для объекта DKAny:

C++

```
DKDDO* ddo = new DKDDO;           // создает DKDDO в куче
DKAny anyA((dkDataObjectBase*)ddo);
DKAny* anyB = new DKAny(anyA);     // создает anyB в куче
                                   // anyA и anyB содержат ссылку
                                   // на один и тот же DDO

...
delete anyB;                       // удалить anyB, не удалять DDO
if (anyA.typeCode() == DKAny::tc_dobase)
    delete ((dkDataObjectBase*) anyA.value()); // удаляет DDO
```

Последний оператор delete должен быть выполнен до выхода из сферы, иначе anyA будет удален, оставив DDO как потерю памяти.

Советы по программированию

Рекомендация: При преобразовании целого литерала (integer literal) в DKAny желательно объявлять тип явно, чтобы избежать ненужного преобразования типа. Примеры:

C++

```
any = 10;                          // неоднозначно
any = (unsigned long) 10;          // однозначно
any = (short) 4;                   // однозначно
```

Использование собраний и итераторов

dkCollection - это абстрактный класс, содержащий методы для работы с собраниями. DKSequentialCollection - это конкретная реализация dkCollection. Другие собрания реализованы как подклассы DKSequentialCollection. Эти собрания содержат в качестве элементов объекты данных.

Обычно собрание содержит объекты одного типа, однако оно может содержать элементы разных типов.

Только для C++: После добавления нового элемента собрание владеет им. Выполнив получение элемента, вы получаете указатель на объект DKAny в собрании. Этот объект принадлежит собранию, то есть собрание управляет памятью для элементов DKAny. Объект DKAny может содержать ссылку на объект, но не может управлять памятью для объектов-указателей; памятью для них должны управлять вы.

Использование методов последовательных собраний

DKSequentialCollection содержит методы для добавления, получения, удаления и замены своих элементов. Кроме того, в нем есть метод сортировки. В следующем примере показано, как добавить новый элемент в собрание (объект передается методу addElement в виде параметра).

Java

```
DKSequentialCollection sq = new DKSequentialCollection();
String str = " first member ";
sq.addElement(str);           // добавляем новый элемент в конец
```

C++

```
DKSequentialCollection sq;
DKAny any = DKString(" first member ");
sq.addElement(any);           // добавляем новый элемент в конец
                               // any будет скопирован в собрание
                               // вы владеете исходным any, собрание
                               // владеет копией
```

Использование последовательного итератора

Для перебора элементов собрания используются итераторы. В API есть два типа итераторов: dkIterator и DKSequentialIterator.

Java

Базовый класс `dkIterator` поддерживает методы `next`, `more` и `reset`. Подкласс `DKSequentialIterator` поддерживает дополнительные методы. Для создания итератора нужно вызвать метод `createIterator` собрания. Создав итератор, с помощью метода `setToFirst()` установите указатель на первый элемент. В следующем примере показано использование итератора:

```
dkIterator iter = sq.createIterator(); // создаем итератор для sq
Object member;
iter.setToFirst();
member = iter.at();
while(iter.more()) {                // Пока есть еще элементы, переходим
    member = iter.next();           // к следующему элементу и получаем его

    System.out.println(member);
    ....
}
```

C++

Итераторы служат для перебора элементов собрания. Есть два типа итераторов: базовый итератор `dkIterator`, который поддерживает функции `next`, `more` и `reset`, и его подкласс `DKSequentialIterator`, содержащий другие функции. Итератор создается вызовом функции `createIterator` для собрания. Эта функция создает новый итератор и возвращает его вашей программе. Следующая программа выполняет перебор для собрания:

```
dkIterator* iter = sq.createIterator(); // создать итератор для sq
DKAny* member;

                                // пока есть еще элементы
                                // получить текущий элемент и
                                // передвинуть iter на следующий
while(iter->more()) {           // элемент
    member = iter->next();

    cout << *member << endl; // если хотите, выведите элемент
    ...                      // другая обработка
}
delete iter;                  // не забудьте удалить iter
```

`DKSequentialIterator` содержит дополнительные методы для перемещения итератора в других направлениях. Предыдущий пример можно переписать так:

Java

```
// ----- Создаем последовательный итератор для sq
DKSequentialIterator iter =
    (DKSequentialIterator) sq.createIterator();
Object member;
iter.setToFirst();
while(iter.more()) {
    member = iter.at();                // получаем текущий элемент
    ....                             // другая обработка
    iter.setToNext();                 // вперед к следующей позиции
}
```

C++

```
DKSequentialIterator* iter =                // создать итератор для sq
(DKSequentialIterator*) sq.createIterator();
DKAny* member;
while(iter->more()) {
    member = iter->at();                    // получить текущий элемент
    ...                                    // другая обработка
    iter->setToNext();                      // вперед к следующей позиции
}
delete iter;
```

Эта программа позволяет выполнить некоторые операции с текущим элементом перед перемещением к следующему элементу. Эта операция может заменить или удалить элемент.

Java

```
String st1 = "the new first member";
sq.replaceElementAt(st1, iter); // заменяем текущий элемент на новый
....                          // или
sq.removeElementAt(iter);      // удаляем текущий элемент
....
```

C++

```
any = DKString("the new first member");

sq.replaceElementAt(any, *iter); // заменяем текущий элемент
...                             // или
sq.removeElementAt();           // удаляем текущий элемент
...
```

Совет: При удалении текущего элемента итератор перемещается к следующему элементу. При удалении элемента в цикле выполняйте проверку, как показано в следующем примере. Проверка условия удаления необходима для предотвращения пропуска следующего элемента после удаления текущего элемента.

Java

```
....
if (removeCondition == true)
    sq.removeElementAt(iter); // удален текущий элемент, не перемещать
                              // итератор, поскольку он будет
                              // перемещен после операции удаления
else
    iter.setToNext();         // удаления нет, переместить итератор
....                         // на следующую позицию
```

C++

```
...
if (removeCondition == TRUE)
    sq.removeElementAt(*iter); // удалить текущий элемент, не перемещать
                              // итератор, поскольку он будет перемещен
                              // после операции удаления
else
    iter->setToNext();         // удаления нет, переместить итератор
...                           // на следующую позицию
```

Управление памятью в собраниях (только C++)

Собрание управляет памятью своих элементов - объектов DKAny. Правила, управляющие объектами DKAny, действуют и здесь; если тип объекта в DKAny - указатель, вы несете ответственность за управление памятью при следующих операциях:

- Уничтожение собрания.
- Замена элемента.
- Удаление элемента.

В этом примере показано, как управлять памятью в таких ситуациях:

C++

```
// получить элемент и остаться на нем
member = iter->at();

// программа обработки этого элемента для предотвращения потерь памяти
if (member->typeCode() == DKAny::tc_dobase) {
    // удалить элемент, если больше не нужен
    delete ((dkDataObjectBase*) member->value());
}

sq.removeElementAt(*iter);           // удалить элемент из собрания
```

Вместо удаления элемента можно добавить его к другому собранию. Аналогичные шаги надо выполнить перед использованием функций `replaceElementAt` и `removeAllElement`.

Перед уничтожением собрания удалите его элементы. Можно написать функцию, который будет выполнять эту задачу, и передать эту функцию в функцию `apply` для собрания. Допустим, есть собрание объектов `DKAny`, содержащее объекты `DKAttributeDef`. В следующем примере удаляется собрание:

C++

```
DKDatastoreICM dsICM;
...
DKAny any = dsICM.listSchemaAttributes("GRANDPA");
dkCollection* acoll = (dkCollection*) any;
...
acoll->apply(deleteDKAttributeDef);    // используйте атрибуты
delete acoll;                        // удаляет все элементы
```

В этом примере `deleteDKAttributeDef` - функция, принимающая объект `DKAny` как параметр. Он определяется так:

C++

```
void deleteDKAttributeDef(DKAny& any) {
    delete ((DKAttributeDef*) any.value());
    any.setNull();           // рекомендуется
}
```

Можно написать свою функцию для удаления вашего собрания или удаления некоторых элементов перед удалением собрания.

Деструкторы для некоторых известных собраний, например, для DKParts, DKFolder и DKResults, выполняют эти необходимые шаги очистки. Однако они не управляют хранением при выполнении функции `replaceElementAt`, `removeElementAt` и `removeAllElement`.

Сортировка собрания

Функция `sort` служит для сортировки элементов собрания в возрастающем или убывающем порядке на основе заданного ключа. Надо задать объект сортировки и нужный порядок. Интерфейс для объектов сортировки определен в файле `dkSort.java` (Java) или `dkSort.hpp` (C++). Вы можете написать свою функцию для сортировки своих собраний. В следующем примере показано, как отсортировать собрание объектов DDO:

Java

```
DKResults rs;
....           // Выполнить запрос, заполняющий DKResults объектами DDO
....
DKSortDDOId sortId; // Объявление объекта функции сортировки; по умолчанию
rs.sort(sortId);    // сортирует по ID элемента в восходящем порядке
....
```

C++

```
DKResults* rs;
....
// Выполнить запрос, заполняющий DKResults объектами DDO
....
DKSortDDOId* sortId; // Объявление объекта функции сортировки; по умолчанию
rs->sort(sortId);    // сортирует по ID элемента в восходящем порядке
....
```

Совет: Объект сортировки создается в стеке, поэтому его не требуется удалять явно. Функция является повторно-входной, что означает, что отдельную копию можно использовать совместно, использовать повторно или передавать другой функции.

Собрание и итератор объединения

Собрание объединения используется в программах для обработки объектов данных, возвращенных запросом в виде собрания. Собрание объединения сохраняет существующие отношения подгрупп между объектами данных.

Собрание объединения - это собрание объектов DKResults. Оно создается для результатов DKFederatedQuery, которые могут поступать от нескольких разнородных контент-серверов. Каждый объект DKResults содержит результаты

поиска от одного контент-сервера. Собрание объединения может содержать неограниченное число вложенных собраний.

Для перемещения по собранию объединения создайте и используйте `dkIterator` или `DKSequentialIterator`. Затем для перемещения по объекту `DKResults` создайте другой `dkIterator`, чтобы перебрать и обработать элементы с учетом контент-сервера, откуда поступил объект результатов.

Можно также создать итератор объединения `dkFederatedIterator` и использовать его для перебора всех элементов собрания, независимо от того, с какого контент-сервера пришел результат.

Ограничение: Собрание объединения нельзя запросить.

На рис. 8 показана структура и поведение `DKFederatedCollection`.

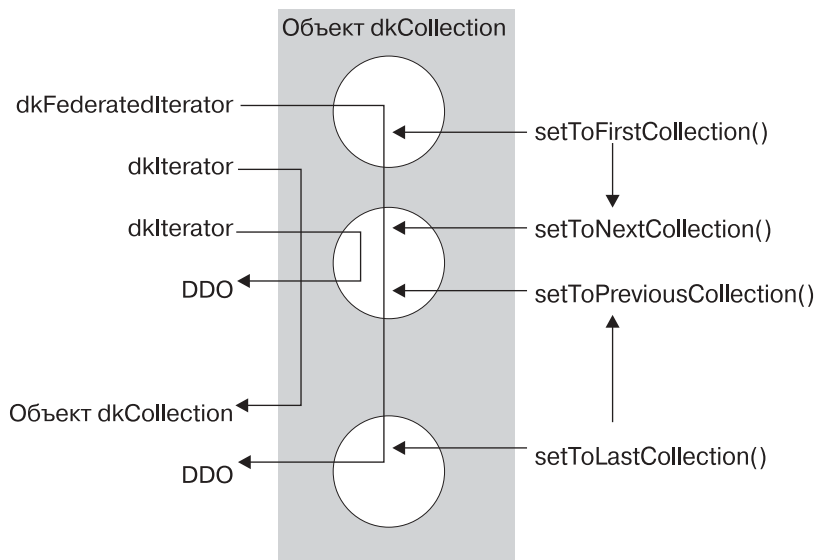


Рисунок 8. Структура и поведение `DKFederatedCollection`

На рис. 8 фигура, представляющая `DKFederatedCollection`, содержит несколько меньших кругов - объектов `DKResults`. `dkFederatedIterator` проходит через границы собраний и возвращает по одному `DDO` за один раз.

Первый `dkIterator` - это итератор для `DKFederatedCollection`, он возвращает по одному объекту `DKResults` за один раз. Второй `dkIterator` - это итератор для второго объекта `DKResults`; он возвращает объект `DDO` для каждого элемента этого собрания `DKResults`.

Функция `setToFirstCollection` в `dkFederatedIterator` задает позицию равной первому DDO в `DKFederatedCollection`. В данном случае это первый элемент первого объекта собрания `DKResults`. Если в этот момент вызвать функцию `setToNextCollection`, она задаст позицию итератора равной первому DDO второго собрания `DKResults`.

Функция `setToLastCollection` в `dkFederatedIterator` задает позицию итератора равной последнему DDO в `DKFederatedCollection`. В данном случае это последний элемент последнего объекта собрания `DKResults`. Если вызвать функцию `setToPreviousCollection`, она задаст позицию итератора равной последнему DDO предыдущего собрания `DKResults`.

Запрос контент-сервера

Можно провести поиск на контент-сервере и получить результаты в объекте `dkResultSetCursor` или `DKResults`. Для некоторых серверов можно создать для этого запроса объект запроса и вызвать функцию `execute` или `evaluate` этого объекта. При помощи своих контент-серверов объект запроса выполняет задачи обработки запроса, такие как подготовка и выполнение запроса, слежение за состоянием выполнения запроса и сохранение результатов. Некоторые контент-серверы поддерживают и объект запроса как альтернативу; например, Content Manager ранних версий и контент-сервер объединения.

Для контент-серверов, поддерживающих объекты запроса, существуют четыре типа объектов запроса: параметрический, текстовый, поиска изображений и комбинированный. Комбинированный запрос содержит и текстовый запрос, и параметрический. Не все контент-серверы могут обрабатывать комбинированные запросы. Content Manager ранней версии поддерживает запрос по изображениям.

В Content Manager параметрический и текстовый запросы интегрированы. Объектами запроса пользоваться не надо; запросы в Content Manager описаны в разделе “Запрос сервера Content Manager” на стр. 227. Информацию о выполнении запросов на сервере Content Manager предыдущих версий смотрите в разделе *Работа с другими контент-серверами*.

Контент-сервер использует два метода для выполнения запроса: `execute` и `evaluate`. Функция `execute` возвращает объект `dkResultSetCursor`, метод `evaluate` возвращает объект `DKResults`. Объект `dkResultSetCursor` используется для работы с большими наборами результатов и для выполнения функций `delete` и `update` в текущей позиции указателя набора результатов. Можно использовать функцию `fetchNextN`, чтобы получить группу объектов в собрании.

Объект `dkResultSetCursor` можно также использовать для повторного выполнения запроса при помощи методов `close` и `open`. Это описано в разделе “Использование указателя набора результатов” на стр. 151.

`DKResults` содержит все результаты запроса. По элементам этого собрания можно перемещаться вперед или назад, можно выполнить запрос по содержимому собрания или использовать его в качестве области видимости для другого запроса.

Дополнительную информацию смотрите в разделе “Открытие и закрытие указателя набора результатов для повторного выполнения запроса” на стр. 151.

Ограничение: При запросе к контент-серверу Domino.Doc возвращается объект `DKResults`. Однако нельзя выполнять запросы по содержимому этого объекта или использовать его в качестве области видимости для другого запроса.

Различие между `dkResultSetCursor` и `DKResults`

`dkResultSetCursor` и собрание `DKResults` имеют следующие различия:

- `dkResultSetCursor` работает как указатель контент-сервера; он может использоваться для больших наборов результатов, поскольку содержащиеся в нем объекты `DKDDO` выбираются по одному. Его также можно использовать для повторного запуска запроса, чтобы получить самые свежие результаты.

Ограничение: Для контент-сервера Domino.Doc нельзя повторно выполнить запрос, даже если используется `dkResultSetCursor`.

- Объект `DKResults` содержит весь набор результатов и поддерживает двунаправленный итератор.
- Если держать `dkResultSetCursor` открытым в течение длительного времени, то это может ухудшить производительность для одновременно работающих пользователей на некоторых контент-серверах.

Использование параметрических запросов

Параметрический запрос - это запрос, в котором требуется точное совпадение условия в предикате запроса, и значений данных с контент-сервера.

Примечание: Примеры запросов в следующих разделах относятся к ранним версиям Content Manager. Информацию о запросах для Content Manager V8 смотрите в разделе “Язык запросов” на стр. 227.

Формулирование строки параметрического запроса

Чтобы создать запрос, нужно сначала сформулировать строку запроса. В следующем примере определяется строка запроса, представляющая запрос для индексного класса с именем GP2DLS2 для ранней версии Content Manager. Примеры строки запроса в Content Manager смотрите в разделе “Примеры запросов” на стр. 236. Условие этого запроса: искать все документы или папки, для которых значение атрибута `DLSEARCH_DocType` не равно `null`.

Максимальное число возвращаемых результатов ограничено пятью, а опция CONTENT имеет значение YES, поэтому возвращается также содержимое документов или папок.

Java

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS2," +  
             "MAX_RESULTS=5," +  
             "COND=(DLSEARCH_DocType > null));" +  
             "OPTION=(CONTENT=YES;" +  
             "TYPE_QUERY=DYNAMIC;" +  
             "TYPE_FILTER=FOLDERDOC)";
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,";  
cmd += "MAX_RESULTS=5,";  
cmd += "COND=(DLSEARCH_DocType <> NULL));";  
cmd += "OPTION=(CONTENT=YES;";  
cmd += "TYPE_QUERY=DYNAMIC;";  
cmd += "TYPE_FILTER=FOLDERDOC)";
```

В этом примере задано, что сервер ранней версии Content Manager использует для запроса динамический SQL и что поиск должен производиться во всех документах и папках. Различные контент-серверы используют разный синтаксис строки запроса; у контент-сервера объединения свой синтаксис строки запроса. Обратитесь к информации о том контент-сервере, где надо вести поиск, или смотрите в *электронном справочнике API*. Если имя атрибута состоит из нескольких слов или использует язык с набором двухбайтных символов, надо заключить его в апострофы ('). Если значение атрибута состоит из символов DBCS, его нужно заключить в двойные кавычки (").

Формулирование параметрического запроса с несколькими критериями

В одном параметрическом запросе можно задать несколько критериев поиска. В следующем примере показано, как задать запрос для двух индексных классов в ранней версии Content Manager:

Java

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS1,MAX_RESULTS=3," +  
             "COND=(DLSEARCH_DocType <> null);" +  
             "INDEX_CLASS=GP2DLS1,MAX_RESULTS=8," +  
             "COND=('First name'==\"Robert\"));" +  
             "OPTION=(CONTENT=YES;" +  
             "TYPE_QUERY=DYNAMIC;" +  
             "TYPE_FILTER=FOLDERDOC)";
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,MAX_RESULTS=3,";  
cmd += "COND=(DLSEARCH_DocType <> NULL);";  
cmd += "INDEX_CLASS=DLSAMPLE,MAX_RESULTS=8,";  
cmd += "COND=('First name' == \"Robert\");";  
cmd += "OPTION=(CONTENT=YES;";  
cmd += "TYPE_QUERY=DYNAMIC;";  
cmd += "TYPE_FILTER=FOLDERDOC)";
```

Выполнение параметрического запроса

Задав строку запроса, создайте объект запроса. У объекта `DKDatastorexx`, представляющий контент-сервер, есть метод для создания объекта запроса. Объект запроса используется для выполнения запроса и получения результатов. В следующем примере показано, как создать объект параметрического запроса и выполнить этот запрос на сервере Content Manager ранней версии; объектом запроса не следует пользоваться с Content Manager Версии 8 или новее. После выполнения запроса результаты возвращаются в собрании `DKResults`.

Внимание: Когда вы удаляете объект `DKResults`, все его элементы также удаляются. Убедитесь, что вы не удаляете элемент дважды.

Java

```
// ----- Создаем склад данных, объект запроса и набор результатов
DKDatastoreDL dsDL = new DKDatastoreDL();
dkQuery pQry = null;
DKResults pResults = null;
DKNVPair parms[] = null;
// ----- Соединяемся со складом данных
dsDL.connect(libSrv,userid,pw,"");
// ----- Формулируем строку запроса
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
             "MAX_RESULTS=5," +
             "COND=(DLSEARCH_DocType <> NULL));" +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=STATIC;" +
             "TYPE_FILTER=FOLDERDOC)";
// ----- Создаем запрос, используя строку запроса
pQry = dsDL.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Выполняем запрос
pQry.execute(parms);
// ----- Обработка результатов
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
// ----- Завершив работу, отсоединяемся
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TSamplePQryDL.java), находится в каталоге CMBROOT\Samples\java\d1.

C++

```
DKDatastoreDL dsDL;
dkQuery* pQry;
DKAny any;
DKResults* pResults;

cout << "connecting to datastore" << endl;
dsDL.connect(libsrv,userid,pw);
cout << "datastore connected libsrv: " <<libsrv<< " userid: "<<userid<<endl;

DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE, ";
cmd += "MAX_RESULTS=5, ";
cmd += "COND=(DLSEARCH_DocType <> NULL)); ";
cmd += "OPTION=(CONTENT=YES, ";
cmd += "TYPE_QUERY=STATIC;TYPE_FILTER=FOLDERDOC)";
cout << "query string " << cmd << endl;
cout << "create query" << endl;
pQry = dsDL.createQuery(cmd);
cout << "executing query" << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "get query results" << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsDL.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TSamplePQryDL.cpp), находится в каталоге Cmbroot/Samples/cpp/dl.

Выполнение параметрического запроса с контент-сервера

У объекта DKDatastorexx, представляющего контент-сервер, есть метод для выполнения запроса. В следующем примере показано, как выполнить параметрический запрос на контент-сервере Content Manager ранней версии. После выполнения запроса результаты возвращаются в объекте dkResultSetCursor.

Java

```
// ----- Создаем склад данных и указатель
DKDatastoreDL dsDL = new DKDatastoreDL();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Соединяемся с контент-сервером
dsDL.connect(libSrv,userid,pw,"");
// ----- Формулируем строку запроса
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
             "MAX_RESULTS=5," +
             "COND=((DLSEARCH_DocType <> NULL)" +
             "AND (DLSEARCH_Date >= 1995)))" +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC)";

...
// ----- Выполняем запрос, используя строку запроса
pCur = dsDL.execute(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Обработка результатов запроса
...
// ----- Завершив работу с указателем, удаляем его и отсоединяемся
pCur.destroy();
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TExecutedL.java), находится в каталоге CMBROOT\Samples\java\d1.

C++

```
...
DKDatastoreDL dsDL;
dkResultSetCursor* pCur = 0;
cout << "Datastore DL created" << endl;
cout << "connecting to datastore" << endl;
dsDL.connect(libsrv,userid,pw);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;
// DKString cmd = "SEARCH=(COND=('DLSEARCH_DocType' == \"html\"));";
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,";
cmd += "MAX_RESULTS=5,";
cmd += "COND=(DLSEARCH_DocType <> NULL));";
cmd += "OPTION=(CONTENT=YES);";
cmd += "TYPE_QUERY=STATIC;TYPE_FILTER=FOLDERDOC";
cout << "query string " << cmd << endl;
cout << "executing query" << endl;
pCur = dsDL.execute(cmd);
cout << "query executed" << endl;
...
...
if (pCur != 0)
delete pCur;
dsDL.disconnect();
...
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteDL.cpp), находится в каталоге Cmbroot/Samples/cpp/dl.

Оценка (evaluate) параметрического запроса с контент-сервера

У объекта DKDatastorexx, представляющего контент-сервер, есть метод для оценки запроса. Результаты возвращаются в собрании DKResults. В следующем примере показано, как оценить параметрический запрос на контент-сервере Content Manager ранней версии.

Java

```
// ----- Создаем строку запроса
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
             "COND=((DLSEARCH_Date >= \"1995\") AND \" +
             "(DLSEARCH_Date <= \"1996\")))\";" +
             "OPTION=(CONTENT=NO;" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC)";

DKNVPair parms[] = null;
DKDDO item = null;
// ----- Создаем склад данных и соединяемся с ним
// подставьте в строку ниже библиотечный сервер, ID пользователя,
// пароль
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Вызываем метод evaluate, получаем результаты и создаем итератор для
// их обработки
DKResults pResults =
    (DKResults)dsDL.evaluate(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
    item = (DKDDO)pIter.next();
    ... // ----- Обработка DKDDO
}
dsDL.disconnect();
dsDL.destroy();
```

C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(INDEX_CLASS=GP2DLS5,";
cmd += "COND=((DLSEARCH_Date >= \"1995\") AND ";
cmd += "(DLSEARCH_Date <= \"1996\")))";
cmd += "OPTION=(CONTENT=NO;";
cmd += "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)";

...
DKAny any = dsDL.evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
    element = pIter->next();
    item = (DKDDO*)element->value();
    // Обработка DKDDO
}
delete pIter;
delete pResults;
dsDL.disconnect();
```

Использование текстового запроса

В Content Manager версии 8 и позднее текстовые и параметрические запросы интегрированы, смотрите раздел “Создание комбинированного параметрического и текстового поиска” на стр. 234.

На контент-сервере Content Manager ранних версий можно выполнять операции текстового и параметрического поиска. При операциях текстового поиска для нахождения текстовых документов запрашиваются текстовые индексы, созданные механизмом текстового поиска.

Формулирование строки текстового запроса

Для текстового поиска нужно сначала сформулировать строку запроса. В следующем примере создается строка запроса, представляющая запрос для текстового индекса TMINDEX. Строка запроса содержит критерий для поиска всех текстовых документов со словом UNIX или member. Число возвращаемых результатов ограничено пятью.

Java

```
String cmd = "SEARCH=(COND=(UNIX OR member));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";
```

C++

```
DKString cmd = "SEARCH=(COND=(UNIX OR member));";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";
```

Формулирование текстового запроса с несколькими индексами

Текстовый запрос можно использовать для поиска по нескольким индексам. В следующем примере показано, как задать запрос с двумя индексами.

Внимание: Если в запросе используются несколько индексов текстового поиска, все эти индексы должны быть одного типа. Например, можно задать в запросе два точных индекса, но нельзя задать в запросе точный индекс и лингвистический индекс.

Java

```
DKString cmd = "SEARCH=(COND=(UNIX OR member));";  
cmd += "OPTION=(SEARCH_INDEX=(TMINDEX,TMINDEX2); MAX_RESULTS=5)";
```

C++

```
String cmd = "SEARCH=(COND=(UNIX OR member));" +  
            "OPTION=(SEARCH_INDEX=(TMINDEX, INDEX2); MAX_RESULTS=5)";
```

Выполнение текстового запроса

Задав строку тестового запроса, создайте объект запроса. У объекта `DKDatastorexx`, представляющий контент-сервер, есть метод для создания объекта запроса. Результаты возвращаются в собрании `DKResults`. Объект запроса используется для выполнения запроса и получения результатов. В следующем примере показано, как создать объект текстового запроса и выполнить запрос.

Java

```
// ----- Создаем склад данных; объявляем запрос и набор результатов
DKDatastoreTS dsTS = new DKDatastoreTS();
dkQuery pQry = null;
DKResults pResults = null;
DKNVPair parms[] = null;
// ----- Соединяемся со складом данных
//         например, dsTS.connect("zebra","7502",DK_СТЫП_TCPIP);
dsTS.connect(srchSrv,"","","");
// ----- Формулируем строку запроса
String cmd = "SEARCH=(COND=(member AND UNIX));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
// ----- Создаем и выполняем запрос
pQry = dsTS.createQuery(cmd, DK_CM_TEXT_QL_TYPE, parms);
pQry.execute(parms);
// ----- Обработка результатов
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
// ----- Завершив работу, отсоединяемся
dsTS.disconnect();
dsTS.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TSampleTQryTS.java), находится в каталоге CMBROOT\Samples\java\d1.

C++

```
DKDatastoreTS dsTS;
dkQuery* pQry;
DKAny any;
DKResults* pResults;

cout << "connecting to datastore" << endl;
//dsTS.connect("zebra","7502",DK_CTYP_TCPIP);
dsTS.connect(srchSrv,"","");
cout << "connected to datastore srchSrv: " << srchSrv << endl;

DKString cmd = "SEARCH=";
cmd += "(COND=(UNIX OR member));";
cmd += "OPTION=(SEARCH_INDEX=";
cmd += srchIndex;
cmd += ")";
cout << "query string " << cmd << endl;
cout << "create query" << endl;
pQry = dsTS.createQuery(cmd);
cout << "executing query" << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "get query results" << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TSampleTQryTS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

Выполнение текстового запроса с контент-сервера

У объекта DKDatastorexx, представляющего контент-сервер, есть метод для выполнения запроса. Результаты возвращаются в объекте dkResultSetCursor. В следующем примере показано, как выполнить текстовый запрос для сервера Content Manager ранней версии:

Java

```
// ----- Создаем склад данных; объявляем запрос и набор результатов
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNameValuePair parms[] = null;
// ----- Соединяемся со складом данных
//          например, dsTS.connect("zebra", "7502", DK_TS_CTYP_TCPIP);
dsTS.connect(srchSrv, "", "", "");

// ----- Формулируем строку запроса
String cmd = "SEARCH=(COND=(internet OR UNIX));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;" +
             "MAX_RESULTS=5)";

...
// ----- Выполняем запрос и обрабатываем результаты
pCur = dsTS.execute(cmd, DK_CM_TEXT_QL_TYPE, parms);
...
// ----- Завершив работу, удаляем указатель и отсоединяемся
pCur.destroy();
dsTS.disconnect();
dsTS.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteTS.java), находится в каталоге CMBROOT\Samples\java\d1.

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
...

dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
while (pCur->isValid()) {
    item = pCur->fetchNext();
    if (item != 0) {
        // Обработка DKDDO
        ...
        delete item;
    }
}
delete pCur;
dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteTS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

Оценка (evaluate) текстового запроса с контент-сервера

У объекта `DKDatastorexx`, представляющего контент-сервер, есть метод для оценки (evaluate) запроса, выполняющий запрос и возвращающий собрание `DKResults`. В следующем примере показано, как выполнить текстовый запрос на контент-сервере Content Manager ранней версии.

Java

```
// ----- Создаем склад данных и строку запроса
DKDatastoreTS dsTS = new DKDatastoreTS();
String cmd = "SEARCH=(COND=($MC=*$ UN*));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreTS dsTS;
// ----- Соединяемся со складом данных
dsTS.connect("TM", "", ' ');
...
// ----- Вызываем метод evaluate, получаем и обрабатываем результаты
DKResults pResults = (DKResults)dsTS.evaluate(cmd,DK_CM_TEXT_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
    item = (DKDDO)pIter.next();
    // ----- Обработка отдельных объектов DKDDO
}
// ----- Отсоединение
dsTS.disconnect();
dsTs.destroy();
```

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');
DKAny *element;
DKDDO *item;
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";

...
DKAny any = dsTS.evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more()) {
    element = pIter->next();
    item = (DKDDO*) element->value();
    // Обработка DKDDO
    ...
}
delete pIter;
delete pResults;
dsTS.disconnect();
```

Получение информации выделения соответствий

Информация соответствия содержит текст документа и информацию выделения для каждого соответствия этого запроса.

При формулировании строки запроса задайте опции MATCH_INFO и MATCH_DICT. Задайте для MATCH_INFO значение YES, чтобы возвращалась информация выделения соответствий. Опция MATCH_DICT задает, будет ли использоваться словарь для получения информации выделения соответствий. Информация о соответствиях возвращается в атрибуте DKMATCHESINFO объекта DKDDO, возвращаемого текстовым запросом. Значение атрибута DKMATCHESINFO - объект DKMatchesInfoTS.

Получение информации выделения соответствий может занять много времени, поскольку документ получается от контент-сервера и проходит лингвистический анализ для обнаружения возможных соответствий. Выполнение этой операции влияет на производительность текстового запроса.

Получение информации выделения соответствий для каждого элемента

результатов запроса: В следующем примере во время текстового запроса получается информация выделения соответствий для каждого элемента результатов запроса. Поскольку для опции MATCH_DICT задано значение NO, словарь не используется.

Java

```
// ----- Создаем склад данных
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Соединяемся с контент-сервером
// подставьте в строку ниже библиотечный сервер,
// ID пользователя, пароль
dsTS.connect("TM","", "", "LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
// ----- Формулируем строку запроса
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TMINDEX; MAX_RESULTS=5; +
             "MATCH_INFO=YES; MATCH_DICT=NO)";

...

pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);
DKDDO item = null;
DKMatchesInfoTS pMInfo = null;
DKMatchesDocSectionTS pMSect = null;
DKMatchesParagraphTS pMPara = null;
DKMatchesTextItemTS pMText = null;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
int lCCSID = 0;
int lLang = 0;
int lOffset = 0;
int lLen = 0;
int numberSections = 0;
int numberParagraphs = 0;
int numberTextItems = 0;
int numberNewLines = 0;
String strDoc = "";
String strSection = "";
String strText = "";
Object anyObj = null;
while (pCur.isValid()) {
    // ----- Получаем следующий DKDDO
    item = pCur.fetchNext();
    if (item != null)
    {
        // продолжение следует...
```

Java (продолжение)

```
// ----- Обработка этого DKDDO
for (i = 1; i <= item.dataCount(); i++)
{
    anyObj = item.getData(i);
    if (anyObj instanceof String)
    {
        ...
    }
    else if (anyObj instanceof Integer)
    {
        ...
    }
    else if (anyObj instanceof Short)
    {
        ...
    }
    else if (anyObj instanceof DKMatchesInfoTS)
    {
        pMInfo = (DKMatchesInfoTS)anyObj;
        // ----- обработка информации выделения соответствий
        if (pMInfo != null)
        {
            strDoc = pMInfo.getDocumentName();
            numberSections = pMInfo.numberOfSections();
            // ----- цикл по разделам документа
            for (j = 1; j <= numberSections; j++)
            {
                pMSect = pMInfo.getSection(j);
                strSection = pMSect.getSectionName();
                numberParagraphs = pMSect.numberOfParagraphs();
                // ----- цикл по абзацам раздела
                for (k = 1; k <= numberParagraphs; k++)
                {
                    pMPara = pMSect.getParagraph(k);
                    lCCSID = pMPara.getCCSID();
                    lLang = pMPara.getLanguageId();
                    numberTextItems = pMPara.numberOfTextItems();
                    // ----- цикл по текстовым элементам абзаца
                    for (m = 1; m <= numberTextItems; m++)
                    {
                        pMText = pMPara.getTextItem(m);
                        strText = pMText.getText();
                        // ----- если в текстовом элементе найдено
                        // соответствие, получить смещение и длину
                    }
                }
            }
        }
    }
}

// продолжение следует...
```

Java (продолжение)

```
// соответствия в текстовом элементе
if (pMText.isMatch() == true)
{
    lOffset = pMText.getOffset();
    lLen = pMText.getLength();
}
numberNewLines = pMText.numberOfLines();
}
}
}
}
}
}
}
}
dsTS.disconnect();
```

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));"
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5;MATCH_INFO=YES;MATCH_DICT=NO)";
...
dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
DKAny anyObj;
dkDataObjectBase *pDObjBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
while (pCur->isValid())
{
    item = pCur->fetchNext();
    if (item != 0)
    {
        // Обработка DKDDO
        for (i = 1; i <= item->dataCount(); i++)
        {
            anyObj = item->getData(i);
            switch (anyObj.typeCode())
            {
                case DKAny::tc_string :
                {
                    ...
                    break;
                }
                case DKAny::tc_long :
                {
                    ...
                    break;
                }
            }
        }
    }
}
// продолжение следует...
```

С++ (продолжение)

```
        case DKAny::tc_short :
        {
            ...
            break;
        }
        case DKAny::tc_dobase :
        {
            // обработать элемент выделения соответствий
            pDOBase = a;
            pMInfo = (DKMatchesInfoTS*)pDOBase;
            if (pMInfo != 0)
            {
                strDoc = pMInfo->getDocumentName();
                numberSections = pMInfo->numberOfSections();
                // цикл по разделам документа
                for (j = 1; j <= numberSections; j++)
                {
                    pMSect = pMInfo->getSection(j);
                    strSection = pMSect->getSectionName();
                    numberParagraphs = pMSect->numberOfParagraphs();
                    // цикл по абзацам раздела
                    for (k = 1; k <= numberParagraphs; k++)
                    {
                        pMPara = pMSect->getParagraph(k);
                        lCCSID = pMPara->getCCSID();
                        lLang = pMPara->getLanguageId();
                        numberTextItems = pMPara->numberOfTextItems();
                        // цикл по текстовым элементам абзаца
                        for (m = 1; m <= numberTextItems; m++)
                        {
                            pMText = pMPara->getTextItem(m);
                            strText = pMText->getText();
                            // если в текстовом элементе найдено соответствие, получить
                            // смещение и длину соответствия в текстовом элементе
                            if (pMText->isMatch() == TRUE)
                            {
                                lOffset = pMText->getOffset();
                                lLen = pMText->getLength();
                            }
                            numberNewLines = pMText->numberOfNewLines();
                        }
                    }
                }
            }
            break;
        }
    }
    // продолжение следует...
```

C++ (продолжение)

```
        default :  
        {  
            break;  
        }  
    }  
    ...  
    delete item;  
}  
}  
delete pCur;  
dsTS.disconnect();
```

Получение информации выделения соответствий для одного элемента результатов запроса: В следующем примере получается информация выделения соответствий для отдельного элемента, возвращенного текстовым запросом. Указатель `dkResultSetCursor`, передаваемый этой подпрограмме, должен находиться в открытом состоянии.

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;

dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";

...
pCur = dsTS.execute(cmd);
DKDDO item = null;
Object anyObj = null;
DKMatchesInfoTS pMInfo = null;
DKMatchesDocSectionTS pMSect = null;
DKMatchesParagraphTS pMPara = null;
DKMatchesTextItemTS pMText = null;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
int lCCSID = 0;
int lLang = 0;
int lOffset = 0;
int lLen = 0;
int numberSections = 0;
int numberParagraphs = 0;
int numberTextItems = 0;
int numberNewLines = 0;
String strDoc;
String strSection;
String strText;
String strDID = "";
String strXNAME = "";
String strDataName = "";
DKPid pid = null;
while (pCur.isValid()) {
    item = pCur.fetchNext();
    if (item != null)
    {
        pid = item.getPid();
        // Обработка DKDDO
        for (i = 1; i <= item.dataCount(); i++)
        {
            anyObj = item.getData(i);
            strDataName = item.getDataName(i);
            if (strDID.equals(""))
            {
                strDID = pid.getId();
            }
            if (strXNAME.equals(""))
            {
                // продолжение следует...
```

Java (продолжение)

```
        strXNAME = p.getObjectType();
    }
    ...
}
pMInfo = dsTS.getMatches(pCur, strDID, strXNAME, false); // Получаем
strDID = ""; // информацию выделения соответствий
strXNAME = "";
if (pMInfo != null)
{
    strDoc = pMInfo.getDocumentName();
    numberSections = pMInfo.numberOfSections();
    // цикл по разделам документа
    for (j = 1; j <= numberSections; j++)
    {
        pMSect = pMInfo.getSection(j);
        strSection = pMSect.getSectionName();
        numberParagraphs = pMSect.numberOfParagraphs();
        // цикл по абзацам раздела
        for (k = 1; k <= numberParagraphs; k++)
        {
            pMPara = pMSect.getParagraph(k);
            lCCSID = pMPara.getCCSID();
            lLang = pMPara.getLanguageId();
            numberTextItems = pMPara.numberOfTextItems();
            // цикл по текстовым элементам абзаца
            for (m = 1; m <= numberTextItems; m++)
            {
                pMText = pMPara.getTextItem(m);
                strText = pMText.getText();
                // если в текстовом элементе найдено соответствие, получить
                // смещение и длину соответствия в текстовом элементе
                if (pMText.isMatch() == true)
                {
                    lOffset = pMText.getOffset();
                    lLen = pMText.getLength();
                }
                numberNewLines = pMText.numberOfNewLines();
            }
        }
    }
}
dsTS.disconnect();
dsTS.destroy();
```


C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
DKString cmd = "SEARCH=(COND=('UNIX operating' AND system));"
cmd += "OPTION=(SEARCH_INDEX=TMINDEX;MAX_RESULTS=5)";
...
dkResultSetCursor* pCur = dsTS.execute(cmd);
DKDDO *item = 0;
DKAny anyObj;
dkDataObjectBase *pDOBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
DKString strDID;
DKString strXNAME;
DKString strDataName;
DKPid pid;
while (pCur->isValid())
{
    item = pCur->fetchNext();
    if (item != 0)
    {
        pid = item->getPid();
        // Обработка DKDDO
        for (i = 1; i <= item->dataCount(); i++)
        {
            anyObj = item->getData(i);
            strDataName = item->getDataName(i);
            if (strDataName == "")
            {
                strDID = pid.getId();
            }
            if (strXNAME == "")
            {
                // продолжение следует...
```

C++ (продолжение)

```
        strXNAME = p->getObjectType();
    }
    switch (anyObj.typeCode())
    {
        ...
    }
}
// Получаем информацию выделения соответствий
pMInfo = dsTS.getMatches(pCur, strDID, strXNAME, FALSE);
strDID = "";
strXNAME = "";
if (pMInfo != 0)
{
    strDoc = pMInfo->getDocumentName();
    numberSections = pMInfo->numberOfSections();
    // цикл по разделам документа
    for (j = 1; j <= numberSections; j++)
    {
        pMSect = pMInfo->getSection(j);
        strSection = pMSect->getSectionName();
        numberParagraphs = pMSect->numberOfParagraphs();
        // цикл по абзацам раздела
        for (k = 1; k <= numberParagraphs; k++)
        {
            pMPara = pMSect->getParagraph(k);
            lCCSID = pMPara->getCCSID();
            lLang = pMPara->getLanguageId();
            numberTextItems = pMPara->numberOfTextItems();
            // цикл по текстовым элементам абзаца
            for (m = 1; m <= numberTextItems; m++)
            {
                pMText = pMPara->getTextItem(m);
                strText = pMText->getText();
                if (pMText->isMatch() == TRUE) // если в текстовом элементе
                {                               // найдено соответствие, получить
                    lOffset = pMText->getOffset(); // смещение и длину соответствия
                    lLen = pMText->getLength();    // в текстовом элементе
                }
                numberNewLines = pMText->numberOfNewLines();
            }
        }
    }
    delete pMInfo;
}
...
delete item;
}
delete pCur;
dsTS.disconnect();
```

Использование указателя набора результатов

dkResultSetCursor - это указатель контент-сервера, который управляет виртуальным собранием объектов DDO. Это означает, что это собрание не материализуется, пока вы не выберете из него элемент. Это собрание представляет собой набор результатов, состоящий из элементов, отвечающих критерию запроса. Завершив работу с указателем, вызовите метод `destroy`, чтобы освободить занятую им память.

Внимание: Информация в этом разделе не относится к Content Manager 8.2. Подробности смотрите в разделе “Язык запросов” на стр. 227.

Открытие и закрытие указателя набора результатов для повторного выполнения запроса

При создании указателя набора результатов он оказывается открытым состоянием. Чтобы повторить выполнение запроса, закройте и вновь откройте этот указатель.

Java

```
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);" +
             "OPTION=(CONTENT=YES);" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC);";

DKNVPair parms[] = null;
...

dkResultSetCursor pCur = dsDL.execute(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);

pCur.close();
pCur.open();           //еще раз выполнить запрос
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);";
cmd += "OPTION=(CONTENT=YES);";
cmd += "TYPE_QUERY=DYNAMIC;";
cmd += "TYPE_FILTER=FOLDERDOC);";
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
// повторить запрос
pCur->close();
pCur->open();
```

Задание и получение позиций в указателе набора результатов

Используя указатель набора результатов, можно получить и задать его текущую позицию. В следующем примере создается и выполняется запрос. В цикле `while`

позиция указателя задается равной первой (или следующей) допустимой позиции. Затем выполняется выборка объекта DDO из этой позиции. Метод `fetchObject` возвращает пустое значение, если указатель находится после последнего элемента.

Java

```
// ----- Формулируем строку запроса
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);" +
             "OPTION=(CONTENT=YES);" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC";

DKNVPair parms[] = null;
DKDDO item = null;
int i = 0;
...
// ----- Выполняем запрос; возвращается указатель набора результатов
dkResultSetCursor pCur = dsDL.execute(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Используем цикл while для перебора элементов собрания
while (pCur.isValid()) {
    pCur.setToNext();
    item = pCur.fetchObject();
    if (item != null)
    {
        i = pCur.getPosition();
    }
}
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE)";
cmd += "OPTION=(CONTENT=YES)";
cmd += "TYPE_QUERY=DYNAMIC";
cmd += "TYPE_FILTER=FOLDERDOC";
pCur = 0;
DKDDO *item = 0;
long i = 0;
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
while (pCur->isValid()) {
    pCur->setToNext();
    item = pCur->fetchObject();
    if (item != 0) {
        i = pCur->getPosition();
        delete item;
    }
}
delete pCur;
```

Другой способ:

Java

```
Object a = null;
pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
while (pCur.isValid()) {
    pCur.setPosition(DK_CM_NEXT,a);
    item = pCur.fetchObject();
    if (item != null) {
        i = pCur.getPosition();
    }
}
```

C++

```
DKAny a;
pCur = dsDL.execute(cmd);
while (pCur->isValid()) {
    pCur->setPosition(DK_CM_NEXT,a);
    item = pCur->fetchObject();
    if (item != 0) {
        i = pCur->getPosition();
        delete item;
    }
}
delete pCur;
```

Для перемещения по элементам можно использовать относительные позиции. В следующем примере пропускается каждый второй элемент в наборе результатов.

Java

```
Object a = null;
pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
a = new Integer(2);
while (pCur.isValid()) {
    pCur.setPosition(DK_CM_RELATIVE,a); // переместить указатель на 2
    item = pCur.fetchObject();          // позиции вперед от текущей
    if (item != null) {                  // позиции (относительно)
        i = pCur.getPosition();
    }
}
```

C++

```
DKAny a;  
long increment = 2;  
pCur = dsDL.execute(cmd);  
a = increment;  
while (pCur.isValid()) {  
    pCur->setPosition(DK_CM_RELATIVE,a);  
    item = pCur->fetchObject();  
    if (item != 0) {  
        i = pCur->getPosition();  
        delete item;  
    }  
}  
delete pCur;
```

Создание собрания по указателю набора результатов

При помощи указателя набора результатов можно задать собрание с заданным числом элементов по указателю набора результатов. Первый параметр метода `fetchNextN` задает, сколько элементов поместить в собрание. Ноль в качестве первого параметра означает, что в собрание будут помещены все элементы.

В следующем примере все элементы из указателя набора результатов отбираются в последовательное собрание. Если `fItems` равен `TRUE`, возвращен хотя бы один элемент.

Java

```
DKSequentialCollection seqColl = new DKSequentialCollection();  
boolean fItems = false;  
int how_many = 0;  
fItems = pCur.fetchNextN(how_many,seqColl);
```

C++

```
DKSequentialCollection seqColl;  
DKBoolean fItems = FALSE;  
long how_many = 0;  
fItems = pCur->fetchNextN(how_many,seqColl);
```

Запросы собраний

Запрашиваемое собрание - это собрание, которое допускает дальнейшие запросы, что позволяет ограничить сферу поиска или получить более точные результаты. Пример реализации запрашиваемого собрания - объект `DKResults`, возвращаемый в качестве результата оценки запроса. `DKResults` - это собрание объектов `DDO`, подкласс собрания `dkQueryableCollection`.

Получение результатов запроса

В следующем примере показано, как передать параметрический запрос и получить результаты. Результаты находятся в `rs` - объекте `DKResults`. Для обработки собрания и получения `DDO` можно использовать предыдущие примеры программ.

Java

```
// ----- Создаем склад данных и устанавливаем соединение
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Создаем и выполняем объект запроса
String query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> null));";
DKParametricQuery pq =
(DKParametricQuery) dsDL.createQuery(query1,DK_CM_PARAMETRIC_QL_TYPE, null);
pq.execute();
// ----- Получаем результаты
DKResult rs = (DKResults) pq.result();
```

C++

```
// установить соединение
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
// создать объект запроса
DKString query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> NULL));";
DKParametricQuery* pq =(DKParametricQuery*)
dsDL.createQuery(query1,DK_PARAMETRIC_QL_TYPE, NULL);
pq->execute();
DKAny any = pq->result();
DKResult* rs = (DKResults*) any.value();
```

Оценка (evaluate) нового запроса

Для повышения избирательности результат запроса можно использовать в следующем запросе. В следующем фрагменте кода, основанном на предыдущем примере, показано повторная оценка запроса:

Java

```
String query2 = "SEARCH=(INDEX_CLASS=GRANDPA,  
    COND=(Subject == 'Детектив'))";  
Object obj = rs.evaluate(query2,DK_CM_PARAMETRIC_QL_TYPE, null);  
....
```

C++

```
DKString query2 = "SEARCH=(INDEX_CLASS=GRANDPA,  
    COND=(Subject == 'Детектив'))";  
    any = rs->evaluate(query2,DK_PARAMETRIC_QL_TYPE, NULL);  
    ...
```

Второй запрос возвращает obj - объект DKResults, содержащий уточненные результаты. Результаты последовательного выполнения этих двух запросов эквивалентны результатам запроса:

```
"SEARCH=(INDEX_CLASS=GRANDPA, COND=(Title <> NULL AND Subject == 'Детектив'))";
```

Запрос можно повторять до получения удовлетворительных результатов. Последующие запросы должны быть запросами того же типа, что и первый. Если это не так, вы можете получить пустой результат.

В следующем примере показаны последовательные текстовые запросы:

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();  
dsTS.connect("TM","","","");  
  
// ----- Первый запрос  
String tquery1 = "SEARCH=(COND=(IBM)); OPTION=(SEARCH_INDEX=TMINDEX)";  
DKTextQuery tq =  
    (DKTextQuery) dsTS.createQuery(tquery1, DK_CM_TEXT_QL_TYPE, null);  
tq.execute();  
DKResults trs = (DKResults) tq.result();  
// ----- Второй запрос  
String tquery2 = "SEARCH=(COND=(Tivoli)); OPTION=(SEARCH_INDEX=TMINDEX)";  
Object obj = trs.evaluate(tquery2, DK_CM_TEXT_QL_TYPE, null);
```


C++

```
DKDatastoreTS dsTS;  
dsTS.connect("TM","","","");  
  
DKString tquery1 = "SEARCH=(COND=(IBM)); OPTION=(SEARCH_INDEX=TMINDEX)";  
DKTextQuery* tq =  
    (DKTextQuery*) dsTS.createQuery(tquery1,DK_TEXT_QL_TYPE, NULL);  
tq->execute();  
any = tq->result();  
DKResults* trs = (DKResults*) any.value();  
  
DKString tquery2 = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";  
any = trs->evaluate(tquery2,DK_TEXT_QL_TYPE, NULL);
```

Второй запрос возвращает obj - объект DKResults, содержащий уточненные результаты. Результаты последовательного выполнения этих двух запросов эквивалентны результатам запроса:

```
"SEARCH=(COND=(IBM AND Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
```

Использование запрашиваемого собрания вместо комбинированного запроса

Комбинированный запрос обеспечивает гибкость: можно передавать сочетание параметрических и текстовых запросов, с заданными сферами или без них. Однако все эти запросы должны передаваться одновременно, а не один за другим, как при оценке запрашиваемого собрания.

Комбинированный запрос возвращает объект DKResults; однако для него нельзя выполнять оценку другого параметрического запроса. Комбинированные запросы можно использовать не на всех контент-серверах.

Оценка запрашиваемого собрания при помощи дальнейших запросов обеспечивает гибкость путем последовательного повышения избирательности результатов до получения удовлетворительных окончательных результатов. Последовательные запросы полезны при динамическом просмотре содержимого контент-сервера и формулировании очередного запроса с учетом предыдущих результатов. Но если весь запрос известен заранее, эффективнее сразу передавать полный запрос или использовать комбинированный запрос.

Работа с Content Manager Версии 8.2

В этом разделе описываются интерфейсы прикладного программирования (API) соединителя Content Manager Версия 8 Выпуск 2 (соединителя ICM). Соединитель ICM - это расширение структуры данных Enterprise Information Portal (EIP), поэтому прежде чем знакомиться с дальнейшей информацией, надо усвоить основные понятия структуры данных EIP, которым посвящена “Основные понятия прикладного программирования в Enterprise Information Portal” на стр. 13.

API соединителя ICM можно использовать для создания и внедрения пользовательских прикладных программ, которые обращаются к контент-серверу Content Manager. Эти API можно также использовать для интеграции ваших существующих программ с контент-сервером Content Manager.

В этом разделе содержится следующая информация:

- Что такое система Content Manager
- Основные понятия системы Content Manager
- Планирование прикладной программы Content Manager
- Создание прикладной программы Content Manager
- Управление доступом к информации
- Обработка транзакций
- Поиск элементов

Что такое система Content Manager

Основные компоненты системы Content Manager - это библиотечный сервер, один или несколько менеджеров ресурсов и набор объектно-ориентированных интерфейсов прикладного программирования (API). Для управления системой Content Manager можно также использовать клиент системного администратора на основе Java.

Библиотечный сервер обеспечивает возможность гибкого моделирования данных, защищенный доступ к системе, эффективное управление содержимым и другие возможности. Библиотечный сервер управляет отношениями между компонентами системы и доступом ко всей информации системы, включая информацию, хранящуюся на менеджерах ресурсов, сконфигурированных в системе.

Менеджер ресурсов - это компонент, который хранит реальное содержимое любого двоичного объекта, например отсканированные изображения, документы или видеофильмы. В систему Content Manager можно интегрировать другие менеджеры ресурсов, такие как Content Manager VideoCharger, и продукты других производителей (не IBM). Менеджер ресурсов позволяет выполнять следующие задачи:

- Автоматически переносить содержимое с быстрых носителей на медленные и менее дорогие при помощи SMS (System Managed Storage - хранение, управляемое системой).
- Обращаться к менеджеру ресурсов непосредственно из браузера.
- Получать весь объект или его часть.
- Синхронизовать ваши данные с библиотечным сервером.

API обеспечивают доступ к системе Content Manager для прикладных программ. Эти API доступны для Java и C++. С помощью этих API прикладная программа может использовать все функциональные возможности Content Manager, такие как моделирование данных, интегрированный параметрический и текстовый поиск, обращение к данным других систем и получение этих данных и т.п.

Приведенная на рис. 9 диаграмма показывает, как компоненты системы связаны между собой. Имейте в виду, что это лишь одна из возможных реализаций системы Content Manager. В другой конфигурации системы, например, может использоваться четыре менеджера ресурсов.

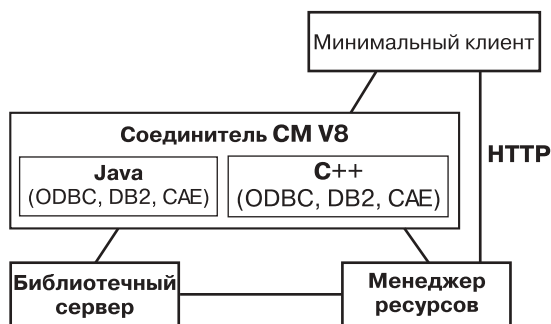


Рисунок 9. Конфигурация системы

Основные понятия системы Content Manager

В этом разделе описываются важные понятия системы Content Manager. Прежде чем приступить к заданиям по программированию, необходимо усвоить основные понятия Content Manager. В разделе описываются:

- Элементы.
- Атрибуты.
- Типы элементов.

- Корневые и дочерние компоненты.
- Объекты.
- Связи и ссылки.
- Документы.
- Папки.
- Управление версиями.
- Управление доступом.
- Модель данных управления документами.

Элементы

Элемент данных или просто элемент - это базовый объект, управляемый библиотечным сервером. Примеры элементов это - страховой полис, страховой иск, телефонный номер и т.п. *Элемент* - это общее обозначение для экземпляра типа элемента. Если объект - это отдельная часть цифрового содержимого, то элемент - это представление этого объекта. Элемент - это не сам объект, но он полностью идентифицирует объект и определяет, как его найти. В системе элементы представляют объекты, в том числе документы и папки. Чтобы определять бизнес-объекты, например, документы, вы работаете с определениями элементов.

Когда прикладная программа создает элемент, Content Manager присваивает ему несколько системных атрибутов и позволяет вам задать несколько своих собственных атрибутов.

К системным атрибутам относятся отметка времени создания и идентификатор элемента (ID элемента). ID элемента уникален для каждого элемента. ID элемента хранится в Content Manager и используется для поиска элемента на библиотечном сервере. В своей программе вы используете ID элемента для доступа ко всем данным, связанным с элементом.

Атрибуты

Атрибут - это единица данных, описывающая определенную характеристику или свойство (например, имя, адрес, возраст и т.п.) элемента; по ней можно выполнять поиск элемента.

Группируя атрибуты, можно составлять группы атрибутов. Например, атрибут адрес может быть образован из группы атрибутов, в которую входят улица, город, штат и почтовый индекс.

Можно также определить атрибуты с несколькими значениями. Такие атрибуты называются многозначными; они реализуются как дочерние компоненты. Например, у владельца полиса может быть несколько адресов, домашний адрес, рабочий адрес и так далее.

Дополнительную информацию смотрите в примере `SAttributeDefinitionCreationICM`.

Типы элементов

Тип элемента (в ранних версиях Content Manager он назывался индексным классом) - это, по существу, шаблон для определения и последующего поиска подобных документов. Тип элемента состоит из корневого компонента, дочерних компонентов (необязательно) и классификации. Тип элемента - это общая структура, содержащая все компоненты и связанные с ними данные. Например, в сценарии со страховой компанией, тип элемента страховой полис содержит элементы с такими атрибутами, как номер полиса, имя, страховой иск и т.п.

В Content Manager типы элементов делятся на четыре рода: нересурсные, ресурсные, документные и части документа. Нересурсный тип элементов представляет объекты, не хранящиеся в менеджере ресурсов. Ресурсный тип элементов представляет объекты, хранящиеся в менеджере ресурсов, такие как файлы в файловой системе, видеозаписи на сервере видео, большие объекты (LOB) в таблицах базы данных и тому подобное. Документный тип элементов представляет объекты, которые содержат части документов; эти части содержат ресурсное содержимое, подобно отдельным элементам ресурсного типа. Тип элементов часть документа представляет объекты, которые хранятся на менеджере ресурсов и представляют части документов, входящие в элемент документного типа и принадлежащие ему. Система Content Manager предоставляет базовый набор ресурсных типов элементов: большие объекты, текстовые объекты, изображения, объекты видеопотоков и видеообъекты.

Дополнительную информацию смотрите в примере `SItemTypeCreationICM`.

Корневые и дочерние компоненты

Тип элементов состоит из корневого компонента и любого числа необязательных дочерних компонентов.

Корневой компонент - это первый или единственный уровень иерархического типа элементов. Тип элементов состоит из системных и пользовательских атрибутов. При внутреннем представлении большинство элементов основного типа состоит только из одного компонента.

Дочерний компонент - это необязательный второй или последующий уровень иерархического типа элементов. Каждый дочерний компонент непосредственно связан с вышестоящим уровнем. На рис. 10 на стр. 163 изображена схема метамодели Content Manager. Показаны корневой и дочерние компоненты и их отношения, образующие иерархию элементов. На этой схеме показаны также связи, ссылки, ресурсные элементы и ресурсные объекты.

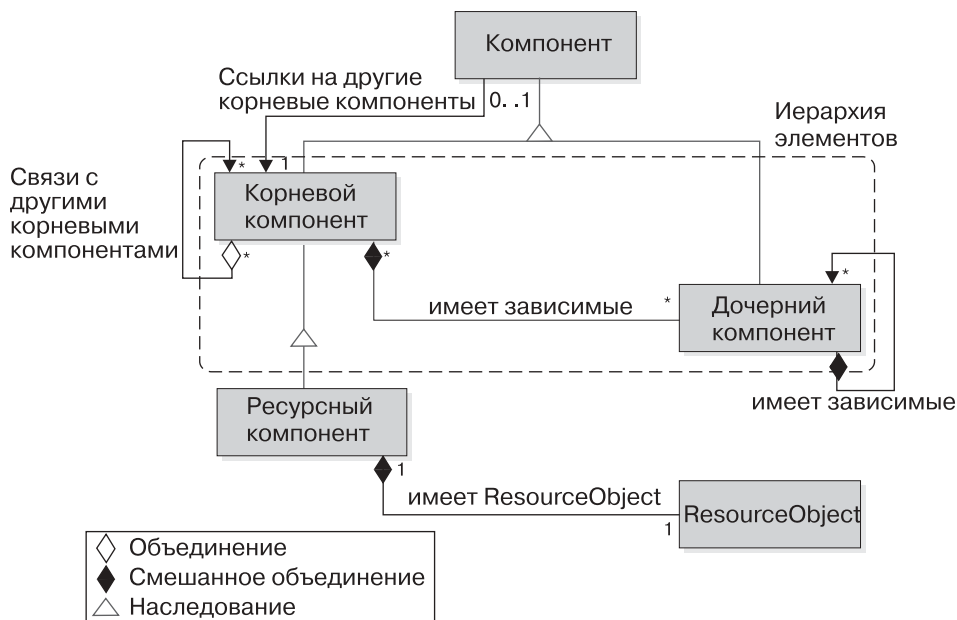


Рисунок 10. Мета модель Content Manager: логическая структура.

На рис. 11 показан пример модели данных для страховой компании, где корневой компонент Полис связан с дочерними компонентами Страховой иск, Полицейский рапорт и Оценка ущерба.

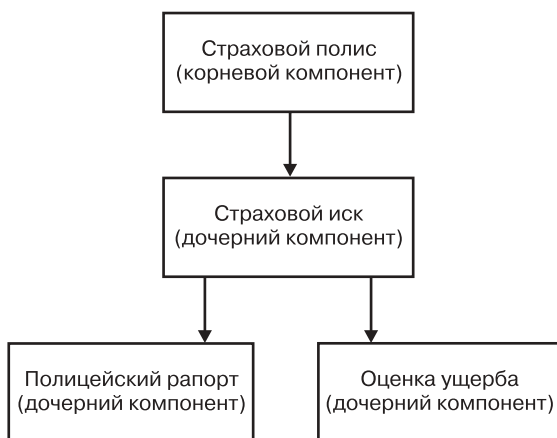


Рисунок 11. Иерархия компонентов

Дополнительную информацию о понятиях моделей данных Content Manager смотрите в примере SItemCreationICM.

Объекты

Объект (ресурсное содержимое) - это любое цифровое содержимое, которое пользователь может сохранять, получать и использовать как единое целое (например, изображения JPEG, аудиофайлы MP3, видеофайлы AVI и фрагменты текста из книг). Объект всегда хранится в менеджере ресурсов. Доступом к объекту управляет библиотечный сервер.

Дополнительную информацию смотрите в примере `SResourceItemCreationICM`.

Связи и ссылки

Связь можно использовать для моделирования одной или нескольких связей между элементами. Как показано на рис. 10, связи можно использовать только для установления отношений между корневыми компонентами элементов. Связи не учитывают конкретной версии элемента. Число связей не ограничено. Связи создают гибкие зависимости между источником и назначением. Связь только соединяет два элемента, позволяя вызывать элементы - участники связи или элементы, связанные с этими двумя элементами. Использование связей определяется во время выполнения пользовательской программы. В программе можно использовать любое число связей.

Связи служат открытыми, наращиваемыми и гибкими конструктивными элементами при прикладном программировании. Вы сами можете вводить ограничения на связи в вашей прикладной программе.

Например, вы можете использовать связь для представления отношения вхождения или содержащее - содержимое. Если вы решите реализовать такое отношение с помощью связей, помните, что контейнер не *обладает* своим содержимым, то есть элементы из контейнера не будут удалены, если удалить сам контейнер. Дополнительную информацию о связях смотрите в примере `SLinksICM`.

Ссылка ведет от одного компонента (корневого или дочернего) к другому корневому компоненту. Ссылка представляется как ссылочный атрибут в компоненте, определяемый во время разработки. В определении компонента может быть сколько угодно ссылочных атрибутов, заданных при определении типа компонента, которые ссылаются на другие корневые компоненты. Обычно ссылка не означает принадлежности, но если есть такая потребность, свойство принадлежности можно реализовать.

При добавлении ссылки в тип компонентов элементы этого типа компонентов получают возможность ссылаться на другие элементы. С точки зрения динамических объектов данных (DDO) у объекта DDO есть атрибут, который идентифицируется именем ссылки. Значение атрибута может указывать на другой DDO. Значение атрибута для DDO - это DDO, на который указывает ссылка.

Ссылки можно определять и для корневых, и для дочерних типов компонентов, ссылающихся на другой корневой тип компонентов. Смотрите рис. 10 на стр. 163. Кроме того, ссылки могут указывать на конкретную версию элемента, в то время как связи указывают на все версии. Дополнительную информацию о ссылочных атрибутах смотрите в примере `SReferenceAttrDefCreationICM`.

Документы

В вашей системе могут быть документы двух типов. Первый тип документа - элемент с семантическим типом "документ"; предполагается, что он содержит информацию, составляющую документ. Этот тип может использоваться самостоятельно или содержать части, если вы реализовали эту модель документа в вашей модели данных. Дополнительную информацию об этом типе документа смотрите в учебном примере `API SItemCreationICM`.

Другой тип документа - элемент, созданный с "документным" типом элементов (так называемая "документная модель"). Этот тип документа содержит части, реализующие документную модель `Content Manager`. Части документа могут иметь разные типы содержимого, включая, например, текст, изображения и электронные таблицы. Дополнительную информацию о документной модели смотрите в примере `SDocModelItemICM`.

Папки

Папка - это элемент, который может содержать другие элементы любого типа. Эти элементы могут сами быть папками. В `Content Manager` Версии 8 папки реализованы с помощью отношения связи между элементами. Элемент может содержать другие элементы, образуя иерархия вложенности, которая называется иерархией папок. Например, элемент страховой полис принадлежит типу элементов страховой полис и может содержать много страховых исков, поэтому страховой полис является папкой, в которой хранятся другие элементы, например, фотографии, номер социального страхования и тому подобное. Папки - очень гибкая структура, так как любой элемент может быть папкой и может содержать сколько угодно других элементов. Дополнительную информацию смотрите в примере `SFolderICM`.

Поддержка версий

Поддержка версий - это возможность хранить и поддерживать несколько версий элемента, включая версии его дочерних компонентов. Правила поддержки версий определяются при создании типа элементов. Если в типе элементов разрешена поддержка версий, все элементы этого типа будут иметь версии.

Есть два типа поддержки версий - постоянная поддержка и поддержка прикладной программой. Если для какого-нибудь элемента разрешена постоянная поддержка версий, новая версия элемента создается автоматически при каждом обновлении или сохранении этого элемента на контент-сервере. Если для какого-нибудь элемента разрешена поддержка версий прикладной программой, система создает новую версию только тогда, когда это задано в пользовательской прикладной программе.

Поддержку версий осуществляет библиотечный сервер Content Manager. У каждой версии элемента, содержимое которого хранится в менеджере ресурсов, будет собственная копия содержимого в менеджере ресурсов. К важным факторам поддержки версий относятся следующие характеристики:

- Поддержка версий распространяется на корневой компонент и всю его иерархию.
- Для типов элементов может использоваться одно из трех правил поддержки версий: "поддержка версий - всегда", "поддержка версий - никогда" (по умолчанию) или "поддержка версий под управлением программой".
- Все версии элемента в системе СМ можно искать и получать.
- Любая версия элемента может быть изменена или удалена.
- Для типов элементов с версиями, управляемыми прикладной программой при изменении элемента у пользователя есть выбор применить эти изменения к существующей версии или создать на их основе новую версию.
- У каждой версии элемента есть свой собственный постоянный идентификатор (PID). PID состоит из нескольких частей, две из которых важны в текущем контексте. Первая важная часть - это ID элемента, он одинаков для всех версий элемента. Вторая часть - номер версии. У каждой версии элемента есть свой собственный номер версии, который можно получить и задать для него строчное значение. Вот пример, показывающий, как работать с номерами версий:

```
DKPidICM pid = (DKPidICM)ddo.getPidObject();
String version = pid.getVersionNumber();
....
pid.setVersionNumber(version);
```

- Тип элементов можно сконфигурировать так, чтобы сохранялось только ограниченное число версий каждого элемента. Если при обновлении элемента превышает максимально разрешенное число версий, система отбрасывает самую старую из сохраненных версий и создает новую версию.
- При переиндексации элемента, для которого включена поддержка версий, все его предыдущие версии автоматически удаляются.
- Дочерние компоненты элемента наследуют версии своего родительского компонента.
- Версию дочернего типа компонентов нельзя изменить, поскольку она соответствует версии своего родительского типа.
- Правила поддержки версий на уровне частей можно получить из объекта отношений типов элементов, который используется для представления типов.

Подробная информация о поддержке версий приведена в примерах `SItemUpdateICM` и `SItemTypeCreationICM`.

Управление доступом

В модели управления доступом Content Manager используются следующие основные понятия:

- Привилегии и наборы привилегий.
- Управляемые объекты.
- Пользователи и группы пользователей.
- Списки управления доступом.

Различные элементы управления доступом работают следующим образом. Каждому пользователю Content Manager предоставляется набор привилегий пользователя. Эти привилегии определяют операции, которые разрешено выполнять пользователю. Фактические права доступа пользователя никогда не могут превысить определенные для него привилегии.

Модель управления доступом Content Manager применяется для управляемых объектов. *Управляемый объект* - это единица защищенных пользовательских данных. В Content Manager для управляемых объектов предусмотрено управление на уровне элементов, на уровне типа элементов и на уровне целой библиотеки. Например, чтобы реализовать управление доступом на уровне типа элементов, с этим типом можно связать список управления доступом. Операции над контролируруемыми объектами регулируются одним или несколькими правилами управления, которые называются списками управления доступом (ACL). В системе Content Manager каждый управляемый объект должен быть связан с каким-то ACL.

Когда пользователь инициирует операцию на элементе, система проверяет его привилегии и список управления доступом, связанный с данным элементом, чтобы определить, имеет ли пользователь право на выполнение этой операции. Логически для права доступа к элементу требуется также и право доступа к типу элементов, в котором он определен. На рис. 12 показан пример определения системой прав доступа пользователя к элементу на основе привилегий и списков управления доступом.



Рисунок 12. Диаграмма управления доступом

Привилегии и наборы привилегий

Привилегии позволяют пользователю выполнять определенные операции над элементом системы, например, создавать и удалять его. Каждому пользователю Content Manager предоставляется набор привилегий пользователя. Привилегии определяют максимум операций, которые пользователь может выполнять, работая с информацией в системе Content Manager. Права доступа пользователя не превышают определенные для этого пользователя привилегии.

В Content Manager задан ряд предопределенных привилегий, называемых привилегиями, определенными системой; изменить их нельзя. Можно также определить свои собственные привилегии, называемые пользовательскими привилегиями. Пользовательские привилегии можно реализовать в прикладной программе с помощью подпрограмм обработчика пользователя.

У каждой привилегии есть сгенерированный системой уникальный код, называемый кодом определения привилегии. Коды определений привилегий от 0 до 999 зарезервированы для привилегий, определенных системой. Для пользовательских привилегий можно использовать коды от 1000 и выше.

Пользовательские привилегии подразделяются на две категории: привилегии управления системой и привилегии доступа к данным. Привилегии управления системой можно использовать для моделирования пользовательских данных и управления, а также для поддержки системы Content Manager. Привилегии управления системой нужны для выполнения таких задач, как конфигурирование системы, управление конфигурацией библиотечного сервера и управление типами элементов. Привилегии доступа к данным можно использовать для обращения к данным системы, например, к элементам и типам элементов, и их изменения.

Выделенная пользователю группа привилегий называется *набором привилегий*. Например, один набор привилегий может содержать привилегии создавать, изменять и удалять. Наборы привилегий облегчают управление системой. Для возможности использования привилегий их нужно сгруппировать в наборы. Число привилегий, которые может содержать набор привилегий, не ограничивается.

Предопределенные наборы привилегий Content Manager: привилегии System Admin

AllPrivSet; PrivSetCode: 1

Пользователь с этим набором привилегий может выполнять все функции со всеми объектами Content Manager. В состав этого набора входят все привилегии, определенные системой, и все пользовательские привилегии.

NoPrivSet; PrivSetCode: 2

Пользователи с этим набором привилегий не могут выполнять никакие функции ни с какими объектами Content Manager. В состав этого набора не входят никакие привилегии.

SystemAdminPrivSet; PrivSetCode: 3

Пользователи с этим набором привилегий могут выполнять все функции администратора и моделирования данных системы Content Manager. В состав этого набора привилегий входят:

ItemAdminPrivSet; PrivSetCode: 4

Пользователи с этим набором привилегий могут выполнять все функции моделирования данных и доступа к элементам Content Manager. В состав этого набора привилегий входят:

- Привилегия для системного типа элементов
- Привилегия SQL select для элемента
- Привилегия запросов для типа элементов
- Привилегия запросов для элемента
- Привилегия добавления для элемента
- Привилегия пользовательского атрибута для набора элементов
- Привилегия системного атрибута для набора элементов
- Привилегия удаления для элемента
- Привилегия перемещения для элемента
- Привилегия задания связи с элементом
- Привилегия связывания элемента
- Привилегия делать элемент владельцем
- Привилегия владеть элементом
- Привилегия добавления связи для элемента
- Привилегия изменения связи для элемента
- Привилегия удаления связи для элемента
- Привилегия блокировать элемент

Таблица 8. Коды привилегий

| Имя привилегии | Код |
|----------------------|-----|
| ICMLogon | 1 |
| SystemAdmin | 40 |
| SystemDefineItemType | 45 |
| ItemSQLSelect | 121 |
| ItemTypeQuery | 122 |
| ItemQuery | 123 |

Таблица 8. Коды привилегий (продолжение)

| Имя привилегии | Код |
|-----------------|-----|
| ItemAdd | 124 |
| ItemSetUserAttr | 125 |
| ItemSetSysAttr | 126 |

Пользователи и группы пользователей

Скорее всего, у вас есть группа пользователей, для которых требуется одинаковый тип доступа к системе. Например, всем страховым агентам в страховой компании требуются привилегии поиска, получения и изменения для типа элементов страховой иск. Страховых агентов и других пользователей с одинаковыми потребностями доступа можно объединить в группу пользователей. Однако группу пользователей нельзя включить в другую группу пользователей.

Группа пользователей - это только удобный способ группировать отдельных пользователей, выполняющих похожие задачи. В группе пользователей может быть от нуля и более пользователей. Группе пользователей не назначается набор привилегий. У каждого пользователя в группе пользователей есть набор привилегий. Группы пользователей упрощают создание списков управления доступом к объектам вашей системы. Группа пользователей не может принадлежать другим группам.

Списки управления доступом (ACL)

Когда пользователь создает элемент в системе CM, он должен определить, какими правами доступа к этому элементу будут обладать другие пользователи и какие операции они смогут производить с этим элементом. Список пользователей, имеющих доступ к данному элементу, и операций, которые они могут выполнять с этим элементом, называется *списком управления доступом* (ACL). Список управления доступом (ACL) может содержать один или несколько ID пользователей или групп пользователей и назначенные им привилегии. Со списком ACL можно связать элементы, типы элементов и рабочие списки. Список привилегий определяет для пользователя максимальные возможности использования системы, а список управления доступом дополнительно ограничивает доступ отдельного пользователя к элементу. Например, если список ACL позволяет Джону удалить некоторую фотографию, но в наборе привилегий Джона нет привилегии удаления, он не сможет удалить эту фотографию.

Управляемый объект связывается с конкретным списком управления доступом при помощи кода ACL. При связи с управляемыми объектами списки управления доступом определяют авторизацию связанных объектов, но не в обход привилегий пользователей. Применяется ACL и проверяются привилегии пользователя.

В правилах управления доступом можно указывать отдельных пользователей, группы пользователей или общедоступность использования. Интерпретация задается в поле UserKind (Тип пользователей) данного правила. Эти типы правил можно для наглядности назвать соответственно правило ACL для пользователя, правило ACL для группы и правило ACL для public (то есть для всех пользователей). Если задано Правило ACL для общедоступного использования, оно авторизует всех пользователей на выполнение операций, заданных в привилегиях ACL на связанном объекте, обеспечивая прохождение пользователями проверки их привилегий. Привилегии ACL для связанного объекта можно задать для всех пользователей (public) на системном уровне. Возможность всех пользователей открывать связанный объект можно сконфигурировать в масштабах всей системы. Этот параметр конфигурации называется PubAccessEnabled (он определен в таблице ICMSTSysControl). Если он отключен, все правила ACL для общедоступного использования игнорируются процессом управления доступом.

В одном списке управления доступом пользователь может быть задан в нескольких типах правил. Тогда приоритет в этих трех правилах отдается сначала правилу ACL для общедоступного использования, затем правилу ACL для пользователей и, наконец, правилу ACL для групп. При применении списка управления доступом ACL проверяет, какое правило с наиболее высоким приоритетом проходит, после чего разрешается авторизация, и процесс проверки прекращается. В случае неудачного завершения проверки по правилу ACL для общедоступного использования проверка будет продолжена по правилам с более низкими приоритетами.

Если проверка правила ACL для пользователя не дала положительного результата, проверка прекращается. Правила ACL для группы не проверяются. Нет смысла проверять правило для группы, поскольку если для пользователя есть отдельное правило в таблице списка доступа, алгоритм управления доступом исключает этого пользователя из группового доступа. Проверка управления доступом по типу отдельных пользователей и проверка по типу групп - не последовательные процессы. Это ситуация или-или, даже если нет ничего страшного в выполнении последовательной проверки.

Если пользователь неудачно проходит проверку по типу отдельных пользователей (или у данного пользователя нет правила в таблице списка доступа), процесс проверки будет продолжен для типа групп. Если пользователь принадлежит одной из групп, и пройдена проверка привилегий, авторизация разрешается, и процесс проверки прекращается. В противном случае доступ отклоняется, но проверка все равно прекращается. Если пользователь задан в нескольких правилах ACL для групп, он авторизуется по совокупности всех привилегий ACL этих правил. Пользователь никогда не задается более, чем в одном правиле ACL для пользователей.

Система CM содержит следующие заранее сконфигурированные ACL: SuperUserACL, NoAccessACL и PublicReadACL.

SuperUserACL

В этот ACL входит одно правило, которое авторизует пользователя ICMADMIN, заранее сконфигурированного в системе CM, выполнять все функции CM (AllPrivSet) на связанных объектах.

NoAccessACL

В этот ACL входит одно правило, которое задает, что для всех пользователей CM (public) не разрешаются никакие операции.

PublicReadACL

В этот ACL входит одно правило, которое задает, что для всех пользователей CM (ICMPUBLIC) разрешается операция чтения (ItemReadPrivSet). Это значение назначается по умолчанию для пользовательского кода DfltACLCode.

Планирование прикладной программы Content Manager

Этот раздел поможет вам определить требования при создании прикладной программы Content Manager; он содержит информацию о том, как работает Content Manager. Ключевая часть планирования прикладной программы - создание модели данных, соответствующей вашим потребностям. Дополнительная информация о модели данных Content Manager есть в *Планирование и установка вашей системы Content Management* и в примере tSItemCreationICM в каталоге samples.

В этом разделе обсуждаются следующие темы:

- Определение свойств прикладной программы.
- Обработка ошибок.

Определение свойств прикладной программы

Подход при разработке прикладной программы зависит от потребностей вашей организации. Чтобы созданная прикладная программа была эффективной, все заинтересованные подразделения вашей организации должны принять участие в ее планировании и проектировании. Дополнительную информацию по планированию смотрите в разделе *Планирование и установка вашей системы Content Management*.

Прежде чем создавать прикладную программу, надо ответить на все или на большую часть следующих вопросов:

- Какие типы документов использует ваша организация?
- Какого типа содержимое уже существующих документов?
- Как вы обрабатываете документы?
- Можно ли автоматизировать обработку документов?

- Как вы получаете, отображаете, храните и распределяете документы?
- Как часто вы извлекаете документы после их сохранения?
- Каким объемом документов управляет ваша организация?
- Какие типы среды вы хотите использовать для хранения ваших больших объектов?
- Пользуется ли ваша организация какими-нибудь другими программами?
- Сколько будет пользователей и какое вы планируете управление доступом?

Ответы на приведенные выше вопросы помогут вам определить, какие возможности надо реализовать в прикладной программе.

Обработка ошибок

При обработке ошибок самая важная исключительная ситуация, которую нужно перехватывать - это класс `DKException`. Не используйте исключительные ситуации для реализации программных алгоритмов и не применяйте них, чтобы узнать, существует ли некий объект на контент-сервере, или для обнаружения любых других ситуаций, кроме действительно исключительных. При использовании исключительных ситуаций в логике программы уменьшается производительность, а информация трассировки и журналов может оказаться бесполезной для отладки и обслуживания.

Внимательно просмотрите всю информацию об исключительных ситуациях. Есть много подклассов `DKException` и, в зависимости от программы, возможно, лучше использовать для каждого подкласса отдельный обработчик. Информация о `DKException` приведена в Табл. 9.

Таблица 9. Информация о `DKException`

| <code>DKException</code> | Описание |
|---------------------------------|--|
| Имя | Имя класса исключительных ситуаций. Содержит имя подкласса. |
| Сообщение | Ошибку объясняет специальное сообщение. Это сообщение может содержать существенную информацию, в которую иногда включены важные состояния переменных на момент обнаружения ошибки. |
| ID сообщения | Уникальный ID сообщения идентифицирует тип ошибки и ставит его в соответствие с сообщением ядра, используемым выше. |

Таблица 9. Информация о `DKException` (продолжение)

| DKException | Описание |
|--------------------|---|
| Состояние ошибки | <p>Может содержать дополнительную информацию об ошибке - о состоянии интерфейса API ОО API или об ошибке библиотечного сервера. Если библиотечный сервер обнаруживает ошибку, сюда включаются следующие четыре сегмента информации:</p> <ul style="list-style-type: none"> Код возврата Код причины Внешний код возврата / код возврата SQL Внешний код причины / код причины SQL |
| Код ошибки | Может содержать код возврата библиотечного сервера. |
| Трассировка стека | Важная информация, указывающая на точку ошибки в программе пользователя и точное место, где ошибка была обнаружена или обрабатывалась ООAPI. |

При работе в Java требуется обработка `java.lang.Exception`. Перехват и печать ошибок показан в примере `SConnectDisconnectICM` в каталоге `samples`. Информацию о регистрации и трассировке смотрите в книге *Сообщения и коды*.

Работа с примерами Content Manager

В Content Manager обеспечен набор разнообразных кодов примеров, которые помогут вам в выполнении ключевых задач Content Manager. Примеры существенно помогают в изучении API; они содержат справочную информацию, указания для программиста, примеры использования API и инструменты.

Просмотреть примеры можно в *электронном справочнике по прикладному программированию* в Информационном центре продукта. Сами примеры находятся в каталогах `CMBROOT\Samples\java\icm` и `CMBROOT\Samples\cpp\icm`. Имейте в виду, однако, что записываются туда, только если во время установки EIP вы выбрали компонент Примеры и инструменты.

Чтобы получить максимум пользы от примеров, обязательно прочитайте файл `Readme` для примеров. Там приведен полный справочный указатель для быстрого поиска примера, содержащего нужно понятие или тему. Каждый пример тщательно отредактирован и снабжен углубленной информацией о понятиях и объяснениями для каждого шага задачи. В каждом примере содержится следующая дополнительная информация:

- Подробная информация заголовка, объясняющая показанные в примере понятия.
- Описание файла примера, включающее предварительные требования и использование командной строки.

- Полностью комментированный код, который можно легко скопировать, настроить и использовать в ваших прикладных программах.
- Функция, которую можно использовать при разработке прикладных программ.

Раздел *Начинаем работу* файла Readme для примеров поможет вам быстро узнать, как выполнять следующие общие задачи:

- Моделирование данных.
- Соединение с сервером и обработка ошибок.
- Определение атрибутов и групп атрибутов.
- Работа со ссылочными атрибутами.
- Определение вашей модели данных.
- Работа с элементами.
- Работа с ресурсными элементами.
- Работа с папками.
- Работа со связями.
- Определение менеджера ресурсов.
- Определение собрания SMS.
- Поиск элементов.

Пример сценария со страховой компанией

С Content Manager поставляются примеры кода возможной практической реализации для страховой компании. Информация, взятая для создания этого примера страховой компании, приводится только для объяснения ключевых возможностей Content Manager. Полный список примеров сценария со страховой компанией смотрите в файле Readme примеров.

Создание прикладной программы Content Manager

Интерфейсы API, реализующие функциональные возможности Content Manager Версии 8.1, сгруппированы в так называемые соединители ICM. У интерфейсов API соединителя ICM есть суффикс ICM, как например, DKDatastoreICM.

В этом разделе содержится следующая информация:

- Программные компоненты.
- Представление элементов с помощью объектов DDO.
- Соединение с системой Content Manager.
- Работа с элементами.

Программные компоненты

На уровне концепций можно подразделить объектно-ориентированные API на группы служб:

- Моделирование данных и документов.

- Поиск и получение.
- Импорт данных и доставка.
- Управление системой.
- Маршрутизация документов.

В модуле моделирования данных и документов содержатся API, которые позволяют отображать модель данных вашего бизнеса на базовую иерархическую модель данных Content Manager. Например, в модели данных страховой компании есть *страховые полисы*, которые в модели данных Content Manager, по существу, являются элементами. API из модуля моделирования данных и документов обеспечивают интерфейсы для определения элементов, представляющих *страховые полисы*.

Модуль поиска и получения обрабатывает запросы об управляемых элементах, таких как документы или папки. API модуля поиска позволяют выполнять комбинированный текстовый и параметрический поиск элементов, хранящихся в системе Content Manager. Результаты поиска возвращаются прикладной программе в виде наборов результатов поиска.

Модуль импорта и доставки данных поддерживает API, позволяющие импортировать данные в вашу систему и доставлять эти данные через различные среды, например локальную сеть или Web.

Модуль управления системой обеспечивает интерфейсы для конфигурирования и поддержки эффективной и защищенной системы Content Manager. Например, можно включить API управления системой в прикладную программу, что позволит настраивать параметры управления системой, управлять пользователями, назначать привилегии пользователям, разрешать доступ к системе и так далее.

API из модуля маршрутизации документов помогают маршрутизировать бизнес-объекты, например, документы, по процессам, как это требуется для вашего бизнеса.

Представление элементов с помощью объектов DDO

Для создания прикладной программы надо изучить понятия протокола DDO/XDO, которые объясняются в разделе “Основные понятия динамических объектов данных” на стр. 14. Информация в этом разделе относится только к Content Manager Версии 8.1.

По сути объект DDO является контейнером из атрибутов. У атрибута есть имя, значение и несколько свойств. Одним из самых важных свойств атрибута является его тип. У DDO есть постоянный идентификатор (PID), показывающий положение объекта данных в постоянном месте хранения. У DDO есть методы заполнения и соответствующие методы для получения информации элемента. В

число методов DDO входят методы add, retrieve, update и delete. Эти методы используются для перемещения данных элементов в постоянное место хранения (например, Content Manager) и обратно.

Элементы Content Manager представляются в памяти как объекты DDO. Атрибуты элементов представляются как атрибуты DDO с именем, типом и значением. Связи и ссылки представляются как специальные типы и атрибуты. Разница между атрибутом связи и атрибутом ссылки заключается в том, что атрибут ссылки ссылается на другой (отдельный) объект DDO или XDO, а атрибут связи ссылается на собрание (то есть на несколько) объектов DDO или XDO. Объекты XDO используются для представления больших объектов.

Ссылка на элемент - объект XDO или другой DDO - имеет имя, ее свойство типа имеет значение ссылки на объект, а ее значение указывает на экземпляр объекта, на который она ссылается. Дочерние компоненты и связи тоже представлены как атрибуты объекта DDO, у которых свойство типа имеет значение собрания объектов данных, а значением является собрание объектов DDO. В случае дочернего компонента имя атрибута - это имя дочернего компонента. Значение - это собрание дочерних компонентов, принадлежащих данному корневому компоненту. При удалении корневого элемента удаляются и все дочерние компоненты корневого элемента.

Соединение с системой Content Manager

Одно из первых действий, которое надо выполнить при построении прикладной программы Content Manager - это соединиться с сервером. Данный раздел поможет вам в различных задачах, возникающих при соединении с сервером Content Manager и отсоединении от него.

Для получения доступа к серверу Content Manager прикладная программа должна создать контент-сервер, который работает как обычный сервер. Чтобы создать контент-сервер и соединиться с ним:

1. Создайте объект контент-сервера.

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM();
```

C++

```
DKDatastoreICM *dsICM =new DKDatastoreICM();
```

2. Задайте параметры соединения.

Java

```
String database = "icmnlsdb";  
String userName = "icmadmin";  
String password = "password";
```

C++

```
char * database = "icmnlsdb";  
char * userName = "icmadmin";  
char * password = "password";
```

3. Вызовите для контент-сервера операцию соединения. `databaseNameStr` - имя базы данных, с которой вы хотите установить соединение.

Java

```
dsICM.connect(databaseNameStr,usridStr,pwStr,"");
```

C++

```
dsICM->connect(database, userName, password, "");
```

В зависимости от конфигурации вашей системы у вас может быть несколько библиотечных серверов и менеджеров ресурсов, с которыми вы можете устанавливать соединение. Чтобы посмотреть список имен библиотечных серверов, с которыми вы можете установить соединение, воспользуйтесь методом `listDataSourceNames()` и вызовите метод `DKDatastoreICM`, а затем метод `listDataSources()`. Метод `listDataSources()` выдает список библиотечных серверов, доступных в данный момент для соединения.

После соединения с библиотечным сервером воспользуйтесь методом `DKRMConfiguration` и вызовите метод `listResourceMgrs()` для получения списка менеджеров ресурсов, связанных с этим библиотечным сервером.

Чтобы отсоединиться от системы, вызовите операцию `disconnect` на контент-сервере.

Дополнительная информация приведена в `SConnectDisconnectICM` - полном примере, из которого взяты приведенные выше фрагменты кода.

Изменение пароля

Вы можете предоставить пользователям возможность менять свой пароль каждый раз, когда они начинают новый сеанс на библиотечном сервере. Чтобы реализовать опцию изменения пароля, воспользуйтесь API DKDatastoreICM и вызовите метод `changePassword()`.

Java

```
changePassword(String userID, String oldPwd, String newPwd)
```

C++

```
changePassword(const char* userID, const char* oldPwd, const char* newPwd)
```

Работа с элементами

В этом разделе описываются процессы, участвующие в создании, изменении и удалении элементов.

Дополнительную информацию о работе с элементами смотрите в примере `SItemCreationICM`.

Создание типа элементов

Прежде, чем создавать какие-либо элементы, сначала нужно создать типы этих элементов. Тип элементов нужно определить для каждого создаваемого элемента. Например, элемент страховой иск является дочерним компонентом типа элементов полис.

При создании типа элементов для него можно определить классификацию. *Классификация типа элементов* - это подразделение типа элементов, служащее в дальнейшем для его идентификации. Все элементы одного типа имеют одну и ту же классификацию типов элементов. Используемые в Content Manager типы элементов классифицируются на элементные, ресурсные, документные и части документов. Если при создании типа элементов не задать классификацию типов элементов, для элемента (DDO) используется классификация типов элементов по умолчанию. Предопределенные классификации типов элементов и соответствующие константы ID приведены в Табл. 10.

Таблица 10. Классификации типов элементов

| Классификация | Константа ID | Номер | Описание |
|---------------|--------------------------------|-------|----------------------------|
| Элемент | DK_ICM_ITEMTY PE_CLASS_ITEM | 0 | Стандартный элемент (DDO). |

Таблица 10. Классификации типов элементов (продолжение)

| Классификация | Константа ID | Номер | Описание |
|---|---|-------|---|
| Ресурс | DK_ICM_ITEMTY PE_CLASS_RES OURCE_ITEM | 1 | Ресурсный элемент - описывает и содержит данные, хранимые в менеджере ресурсов. Примеры ресурсных элементов - видео, изображения, документы и другие данные, архивируемые в менеджере ресурсов. |
| Модель документа | DK_ICM_ITEMT YPE_CLASS_DOC _MODEL | 2 | Элемент, моделирующий документы с частями. Документ может состоять из любого числа частей, которые содержатся в атрибуте DKConstant.DK_CM_DKPARTS. |
| часть | DK_ICM_ITEMTY PE_CLASS_DOC_ PART | 3 | Элементы (части) в классификации модели документа. |
| Замечание: Константы расположены в каталоге com\ibm\mm\sdm\common\DKConstantICM.java для Java и в dk\icm\DKConstantICM.h для C++. | | | |

Чтобы создать тип элемента (смотрите код примера ниже):

1. Создайте тип элементов и передайте ему ссылку на контент-сервер.
2. Дайте типу элементов имя.
3. Добавьте к типу элементов атрибуты, установив такие спецификаторы, как nullable (допустимость пустых значений), textsearchable (возможность текстового поиска) и unique (уникальность).
4. Добавьте тип элементов на постоянный контент-сервер.

Java

```
//Этот пример создает определение типа элементов и присваивает ему имя.  
//Имя типа элементов должно быть не длиннее 15 символов.  
DKItemTypeDefICM bookItemType = new DKItemTypeDefICM(dsICM);  
bookItemType.setName("book");  
bookItemType.setDescription("book - пример имени типа элементов.");  
//Ниже добавляется атрибут "название книги", допускающий текстовый поиск.  
//Атрибут определяется с условием уникальности имени и существования  
//значения. Значение не обязано быть уникальным.  
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_title");  
attr.setTextSearchable(true);  
attr.setUnique(true);  
attr.setNullable(false);  
bookItemType.add(attr);  
//Здесь к типу элементов book добавляется атрибут book_num_pages  
//со следующими характеристиками: допускающий текстовый поиск, уникальный,  
//значение не требуется.  
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_num_pages");  
attr.setTextSearchable(false);  
attr.setUnique(false);  
attr.setNullable(false);  
bookItemType.addAttr(attr);  
bookItemType.add();
```

C++

```
DKDatastoreICM* pDs;  
DKDatastoreICM* pDs;  
...  
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*)pDs->datastoreDef();  
// Создаем новый тип элементов  
DKItemTypeDefICM * bookItemType = new DKItemTypeDefICM(pDs);  
bookItemType->setName("книга");  
bookItemType->setDescription("Это пример имени типа элементов.");  
// Создаем новый атрибут; добавляем его на склад данных и к типу элементов  
DKAttrDefICM * attr = (DKAttrDefICM *)dsDefICM->createAttr();  
    attr->setName("book_title");  
    attr->setType(DK_CM_VARCHAR);  
    attr->setSize(80);  
    attr->setTextSearchable(TRUE);  
    attr->setUnique(TRUE);  
    attr->setNullable(FALSE);  
//Делаем атрибут постоянным на складе данных  
attr->add();  
// Добавляем этот созданный атрибут к типу элементов.  
bookItemType->addAttr(attr);  
// Создаем новый атрибут; добавляем его на склад данных и к типу элементов  
    attr = (DKAttrDefICM *)dsDefICM->createAttr();  
    attr->setName("book_num_pages");  
    attr->setType(DK_CM_INTEGER);  
    attr->setTextSearchable(FALSE);  
    attr->setUnique(FALSE);  
    attr->setNullable(FALSE);  
    attr->add();  
bookItemType->addAttr(attr);  
//Добавляем объект bookItemType на склад данных.  
bookItemType->add();
```

Дополнительная информация есть в примере SitemTypeCreationICM.

Перечисление типов элементов

Чтобы получить список доступных, определенных типов элементов:

1. Соединитесь с контент-сервером DKDatastoreICM.
2. Получите ссылку на определение контент-сервера.
3. Чтобы получить массив строк из имен типов элементов, вызовите метод `listEntityNames` для объекта определения на контент-сервере.
4. Выполните цикл перебора всех имен.

Java

```
String itemTypeNames[] = dsICM.listEntityNames();
DKSequentialCollection itemTypeColl =
    (DKSequentialCollection) dsICM.listEntities();
dkIterator iter = itemTypeColl.createIterator();
while(iter.more()){
    dkEntityDef itemType = (dkEntityDef) iter.next();
    System.out.println(" Item type name : " + itemType.getName()); }
```

C++, пример 1

```
long larraySize = 0;
DKString * itemTypeNames = dsICM->listEntityNames(larraySize);
for (int i = 0; i < larraySize; i++) {
    cout <<(char*)itemTypeNames[i] <<endl;
}
delete [] itemTypeNames;
```

C++, пример 2

```
DKSequentialCollection * itemTypeColl = (DKSequentialCollection *)
dsICM->listEntities();
dkIterator * iter = itemTypeColl->createIterator();
while (iter->more()) {
    dkEntityDef* itemType=(dkEntityDef*)((void*)(*iter->next()));
    cout <<(char*)itemType->getName() < delete(itemType);
}
delete(iter);
delete(itemTypeColl);
```

Дополнительную информацию смотрите в примере SItemTypeRetrievalICM.

Создание атрибутов

Чтобы создать атрибуты:

1. Создайте объект определения атрибутов.
2. Опишите создаваемый объект, задав его имя, описание, тип, размер и т.п.
Вот примеры типов атрибутов, которые вы можете создать:
 - DKConstant.DK_CM_BLOB.
 - DKConstant.DK_CM_CHAR.
 - DKConstant.DK_CM_CLOB.
 - DKConstant.DK_CM_DATE.
 - DKConstant.DK_CM_DECIMAL.

- DKConstant.DK_CM_INTEGER.
- DKConstant.DK_CM_LONG.
- DKConstant.DK_CM_SHORT.
- DKConstant.DK_CM_TIME.
- DKConstant.DK_CM_TIMESTAMP.
- DKConstant.DK_CM_VARCHAR.

3. Добавьте новое определение на постоянный контент-сервер.

Java

```
//В этом примере определяется атрибут для названия книги.
attr = new DKAttrDefICM(dsICM);
attr.setName("book_title");
attr.setDescription("Название книги.");
attr.setType(DKConstant.DK_CM_VARCHAR);
attr.setSize(100);
attr.add();
//В этом примере определяется атрибут для количества страниц в книге.
attr = new DKAttrDefICM(dsICM);
attr.setName("book_num_pages");
attr.setDescription("Количество страниц в книге.");
attr.setType(DKConstant.DK_CM_INTEGER);
attr.setMin((short) 0);
attr.setMax((short) 100000);
attr.add();
```

C++

```
//В этом примере определяется атрибут для названия книги.
DKDatastoreICM * dsICM; .....
DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
    attr->setName("book_title");
    attr->setDescription("Название книги.");
    attr->setType(DK_CM_VARCHAR);
    attr->setSize((long) 100);
    attr->add();
//В этом примере определяется атрибут для количества страниц в книге.
DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
    attr->setName("book_num_pages");
    attr->setDescription("Число страниц в книге.");
    attr->setType(DK_CM_INTEGER);
    attr->setMin((long) 0);
    attr->setMax((long) 100000);
    attr->add();
```

Дополнительная информация о создании атрибутов есть в примере SAttributeDefinitionCreationICM.

Создание, изменение и удаление групп атрибутов

Группы атрибутов облегчают процесс добавления наборов атрибутов к элементам и подкомпонентам. Атрибут может входить в любое число групп атрибутов (в частности, ни в одну). Атрибут можно добавить в несколько групп атрибутов. Элемент данных (DDO) может содержать несколько групп атрибутов. В DDO могут быть атрибуты с одинаковыми именами, но каждый атрибут будет самостоятелен в соответствии с его пространством имен. Когда атрибут добавлен в группу атрибутов, он влияет только на типы компонентов, созданные после добавления. Созданные ранее типы компонентов останутся неизменными. В следующем примере показано, как создать группу атрибутов:

Java

```
// Создаем объект определения для склада, с которым установлено соединение
DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM)dsICM.datastoreDef();
// Создаем новую группу атрибутов
DKAttrGroupDefICM attrGroup = new DKAttrGroupDefICM(dsICM);
// Задаем имя (максимум 15 символов)
attrGroup.setName("Book_Details");
// Задаем описание для этой новой группы атрибутов
attrGroup.setDescription("Подробная информация о книге");
// Получаем определение атрибута, который будет
// добавлен в эту группу атрибутов.
DKAttrDefICM title = (DKAttrDefICM) dsDefICM.retrieveAttr("book_title");
// Получаем определение атрибута, который будет
// добавлен в эту группу атрибутов.
DKAttrDefICM publisher =
    (DKAttrDefICM) dsDefICM.retrieveAttr("publisher");
// Добавляем определения атрибутов в группу атрибутов
attrGroup.addAttr(title);
attrGroup.addAttr(publisher);
// Добавляем группу на постоянное хранение на склад данных
attrGroup.add();
```

C++

```
// Создаем объект определения для склада, с которым установлено соединение
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Создаем новую группу атрибутов
DKAttrGroupDefICM* attrGroup = new DKAttrGroupDefICM(dsICM);
// Задаем имя (максимум 15 символов)
attrGroup->setName("Book_Details");
// Задаем описание для этой новой группы атрибутов
attrGroup->setDescription("Подробная информация о книге");
// Получаем определение атрибута, который будет
// добавлен в эту группу атрибутов.
DKAttrDefICM* title = (DKAttrDefICM *) dsDefICM->retrieveAttr("book_title");
// Получаем определение атрибута, который будет
// добавлен в эту группу атрибутов.
DKAttrDefICM* publisher = (DKAttrDefICM *) dsDefICM->retrieveAttr("publisher");
// Добавляем определения атрибутов в группу атрибутов
attrGroup->addAttr(title);
attrGroup->addAttr(publisher);
// Добавляем группу на постоянное хранение на склад данных
attrGroup->add();

delete(attrGroup);
```

Чтобы изменить имя и описание группы атрибутов, вызовите метод `update()` класса `DKAttrGroupDefICM` и задайте массив новых ID атрибутов. Если эта группа атрибутов уже была связана с каким-нибудь типом компонентов, изменить эту группу атрибутов нельзя.

Для удаления группы атрибутов также используется класс `DKAttrGroupDefICM`. При удалении группы атрибутов первичные атрибуты, составляющие эту группу атрибутов, остаются на библиотечном сервере. При удалении группы атрибутов действуют следующие исключения:

- Нельзя удалить группу атрибутов, если она связана с типом компонента и постоянно действует.
- Нельзя удалить атрибут из группы атрибутов, если группа атрибутов связана с типом компонентов и постоянно действует.
- Нельзя добавить атрибут в группу атрибутов, если группа атрибутов связана с типом компонентов и постоянно действует.

Дополнительная информация есть в примере `SAttributeGroupDefCreationICM`.

Получение списка атрибутов на контент-сервере

В следующем примере показано, как получить список атрибутов на контент-сервере.

Java

```
DKDatastoreICM dsICM;
DKDatastoreDefICM dsDefICM=DKDatastoreDefICM(dsICM.datastoreDef());
//Получаем собрание, содержащее все определения атрибутов.
DKSequentialCollection attrDefColl =
    (DKSequentialCollection)dsDefICM.listAttrs();
if ((attrDefColl!=null) &&(attrDefColl.cardinality()>0)){
    //Создаем итератор для перемещения по собранию
    dkIterator iter = attrDefColl.createIterator();
    while(iter.more()){
        //пока в списке есть еще элементы, продолжаем
        dkAttrDef attrDef = (dkAttrDef) iter.next();
        System.out.println("-"+attr.getName()+":"+attr.getDescription());
    }
}
```

C++

```
DKDatastoreICM * dsICM; .....
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
//Получаем собрание, содержащее все определения атрибутов.
DKSequentialCollection*attrDefColl =
    (DKSequentialCollection *)dsDefICM->listAttrs();
if (attrDefColl &&(attrDefColl->cardinality()>0)){
    //Создаем итератор для перемещения по собранию
    dkIterator*iter =attrDefColl->createIterator();
    while(iter->more()){
        //пока в списке есть еще элементы, продолжаем
        dkAttrDef*attrDef = (dkAttrDef *)iter->next()->value();
        cout <<"-"<<attrDef->getName()<<":"<<attrDef->getDescription()<<endl;
        delete(attrDef);
    }
    delete(iter);
    delete(attrDefColl);
}
delete(dsDefICM);
```

Получение списка имен атрибутов для типа элементов

В следующем примере показано, как получить список имен атрибутов для типа элементов. Дополнительная информация есть в учебном примере API SItemTypeRetrievalICM.

Java

```
//Получаем собрание, содержащее все определения атрибутов для типа элементов.
DKSequentialCollection attrColl = (DKSequentialCollection)
    dsICM.listEntityAttrs(itemTypeName);
//Для каждого атрибута получаем и печатаем имя и описание.
System.out.println("\nAttributes of Item Type '"+itemTypeName+":'
    (" +attrColl.cardinality()+')');
//Создаем итератор для перемещения по собранию
dkIterator iter = attrColl.createIterator();
while(iter.more()){
    //пока в списке есть еще элементы, продолжаем
    DKAttrDefICM attr = (DKAttrDefICM)iter.next();
    System.out.println("-"+attr.getName()+":"+attr.getDescription());
}
```

C++

```
//Получаем собрание, содержащее все определения атрибутов для типа элементов.
DKSequentialCollection*attrColl =(DKSequentialCollection*)
    dsICM->listEntityAttrs(itemTypeName);
//Для каждого атрибута получаем и печатаем имя и описание.
cout <<"\nАтрибуты типа элементов '"<<itemTypeName <<":'
    ("<<attrColl->cardinality()<<') '<<
//Создаем итератор для перемещения по собранию
dkIterator* iter =attrColl->createIterator();
while(iter->more()) {
    //пока в списке есть еще элементы, продолжаем
    DKAttrDefICM* attr =(DKAttrDefICM*)iter->next()->value();
    cout <<"-"<<attr->getName()<<":"<<attr->getDescription()<<endl;
    delete(attr);
}
delete(iter);
delete(attrColl);
```

Создание элемента

Элемент - это все дерево объектов-компонентов DDO, содержащее как минимум один корневой компонент. Этому корневому DDO должен быть присвоен тип свойств или семантический тип. При создании элемента ему нужно назначить свойство - тип элементов. Элементу можно назначить тип документ, папка или элемент. Свойство тип элементов надо указать в качестве второго параметра функции `createDDO` контент-сервера. Значение типа хранится в свойстве объекта DDO, называемом `DK_CM_PROPERTY_ITEM_TYPE`. Не путайте это свойство типа элементов с общим определением типа элемента, которое описывает структуру элемента. В Табл. 11 на стр. 189 приведен список доступных типов.

Таблица 11. Определения свойств типа элементов

| Тип | Константа | Определение |
|------------------------|----------------|--|
| Документ | DK_CM_DOCUMENT | Элемент представляет документ или данные, хранящиеся в системе. Информация, содержащаяся в этом элементе, может представлять документ. Этот элемент можно рассматривать как общий документ, поскольку он не означает строгую реализацию конкретной модели документа. |
| Папка | DK_CM_FOLDER | Элемент представляет собой объект, который содержит данные или объекты либо ссылки на них. Этот элемент можно рассматривать как общую папку, поскольку его смысл не в строгой реализации конкретной модели документа. |
| Элемент (по умолчанию) | DK_CM_ITEM | Элемент общего типа. Этот элемент не соответствует никакому системному или пользовательскому семантическому типу. |

Элементы создаются как объекты DKDDO. Для создания объектов DKDDO всегда используйте методы DKDatastoreICM createDDO(), так как система использует методы DKDatastoreICM's createDDO для автоматической настройки важной информации в структуре DKDDO.

При создании элемента вы определяете компоненты, составляющие этот элемент. Например, если вы решили создать иерархический элемент, надо создать дочерние компоненты.

1. С помощью методов createDDO и createChildDDO контент-сервера создайте объект DDO элемента и задайте все его атрибуты и другую необходимую информацию. В этом примере используется зарегистрированный и соединенный объект DKDatastoreICM с именем dsICM.
2. Создайте корневой компонент.

Java, Пример 1

```
DKDDO myDocumentDDO = dsICM.createDDO("EmployeeDoc",
    DKConstantICM.DK_CM_DOCUMENT);
```

Java, Пример 2

```
DKDDO myFolderDDO = dsICM.createDDO("DeptFolder",
    DKConstantICM.DK_CM_FOLDER);
```

С++, пример 1

```
DKDDO* myDocumentDDO = dsICM->createDDO("EmployeeDoc",DK_CM_DOCUMENT);
```

С++, пример 2

```
DKDDO* myFolderDDO = dsICM->createDDO("DeptFolder",DK_CM_FOLDER);
```

3. Создайте дочерний компонент под корневым компонентом "EmployeeDoc", например, "Dependent".

Java

```
DKDDO myDDO =dsICM.createChildDDO("EmployeeDoc","Dependent");
```

С++

```
DKDDO* myDDO = dsICM.createChildDDO("EmployeeDoc","Dependent");
```

4. Добавьте дочерний компонент к родительскому.

Java

```
short dataid = myDocumentDDO.dataId(DK_CM_NAMESPACE_CHILD,"Dependent");  
DKChildCollection children =  
    (DKChildCollection) myDocumentDDO.getData(dataid);  
children.addElement(myDDO);
```

С++

```
short dataid = myDocumentDDO->dataId(DK_CM_NAMESPACE_CHILD,"Dependent");  
DKChildCollection* children =  
    (DKChildCollection*)(dkCollection*)  
myDocumentDDO->getData(dataid);  
children->addElement(myDDO);
```

5. При помощи метода setData заполните DDO соответствующими значениями.

Java

```
myDDO.setData(.....);
```

C++

```
myDDO->setData(.....);
```

6. Сохраните элемент на постоянном складе данных.

Java

```
myDocumentDDO.add();
```

C++

```
myDocumentDDO->add();
```

В предыдущем примере на последнем шаге создается документ на контент-сервере. Когда DDO-документ добавляется на контент-сервер, добавляются все его атрибуты, включая все части в собрании DKParts. Когда DDO-документ добавляется на контент-сервер, добавляются все его атрибуты, включая все дочерние, и все части в собрании DKParts.

Семантический тип определяет использование элемента или правила для элемента. Content Manager поставляется с несколькими предопределенными семантическими типами, но можно также определить и свои собственные семантические типы.

Семантический тип можно задать в качестве второго параметра функции createDDO контент-сервера. Значение семантического типа хранится в свойстве DDO DKConstantICM.DK_ICM_PROPERTY_SEMANTIC_TYPE, в котором может содержаться то же значение, что и для типа свойств элемента. Соединитель ICM поддерживает семантические типы папка и документ. Можно также создать и свои собственные семантические типы. В Табл. 12 на стр. 192 перечислены доступные семантические типы.

Таблица 12. Предопределенные семантические типы

| Семантический тип | Константа | Определение |
|-------------------|---------------------------------|---|
| Документ | DK_ICM_SEMANTIC_TYPE_DOCUMENT | Указывает, что этот элемент - документ |
| Папка | DK_ICM_SEMANTIC_TYPE_FOLDER | Указывает, что этот элемент - папка |
| Комментарий | DK_ICM_SEMANTIC_TYPE_ANNOTATION | Указывает, что этот элемент или часть - комментарий к основной части |
| Контейнер | DK_ICM_SEMANTIC_TYPE_CONTAINER | Указывает, что этот элемент - контейнер, который может содержать другие документы. Отношение контейнер-содержимое представляется при помощи ссылок. |
| Хронология | DK_ICM_SEMANTIC_TYPE_HISTORY | Указывает, что этот элемент или часть - хронология для основной части |
| Примечание | DK_ICM_SEMANTIC_TYPE_NOTE | Указывает, что этот элемент или часть - примечание к основной части |
| Базовый | DK_ICM_SEMANTIC_TYPE_BASE | Указывает, что этот элемент или часть - основная часть, с которой могут быть связаны комментарий, примечание или хронология. |
| Пользовательский | Пользовательская | Пользовательский семантический тип, интерпретируемый прикладной программой. |

Примечания:

- В Java эти константы определены в DKConstantICM.java.
- В C++ эти константы определены в DKConstantICM.h.
- В Content Manager Версии 7 семантический тип назывался присоединенным типом (affiliated type).

При создании элемента в типе элементов, который был определен как ресурсный тип элементов, возвращается верный XDO. Дополнительная информация о ресурсах есть в примере SResourceItemCreationICM. Напоминаем также, что информация о создании элемента есть в примере SItemCreationICM.

Задание и получение значений атрибутов элементов

Java

Значения атрибутов хранятся и получаются как объекты `java.lang.Objects`. Чтобы задать или получить значения атрибутов, воспользуйтесь соответственно методами DDO `setData` и `getData()`. Задайте значение при помощи объекта, тип которого соответствует типу атрибута. **Пример:**

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
    nameOfAttrStr), valueObj);
Object obj = ddo.getData(ddo.dataId(DK_CM_NAMESPACE_ATTR,nameOfAttrStr));
```

Этот оператор задает значение атрибута для значения, переданного как `java.lang.Object valueObj`. `nameOfAttrStr` - это строка имени атрибута. Этот атрибут был определен и задан в типе элементов, когда определялся тип элементов этого DDO. `valueObj` - это значение, которое вы должны задать; оно должно иметь тип, соответствующий атрибуту.

C++

При задании значений для отдельных атрибутов используйте имя определения конкретного атрибута. Для доступа к атрибутам, входящим в группу атрибутов, используйте такой формат: <имя группы атрибутов>.<имя атрибута>. **Пример:**

```
//В следующем фрагменте кода показано, как символьному атрибуту
//"book_title" для элемента, представленного DDO ddoDocument, задается
//значение "Effective C++"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("book_title")),DKString("Effective C++"));
//В следующем фрагменте кода показано, как целочисленному атрибуту
//"book_num_pages" для элемента, представленного DDO ddoDocument,
//присваивается значение "250"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
    DKString("book_num_pages")), (long)250);
//В следующем фрагменте кода показано, как в строчную переменную "title"
//записывается значение атрибута "book_title" элемента,
//представленного DDO ddoDocument.
DKString title =(DKString) ddoDocument->getData(ddoDocument->
    dataId(DK_CM_NAMESPACE_ATTR,DKString("book_title")));
```

Изменение атрибутов элемента

Чтобы изменить атрибуты элемента, воспользуйтесь методом DDO `setData` и установите для них нужные значения, указав эти значения с помощью `java.lang.Object`, как в приведенном ниже примере. В приведенном ниже примере `nameOfAttrStr` - это имя строки атрибута, а `valueObj` - его значение.

Java

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,  
    nameOfAttrStr), valueObj);
```

C++

```
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,"book_title"),  
    DKString("More Effective C++"));
```

Изменение элемента

Чтобы изменить элемент:

1. Получите или создайте элемент и добавьте его на контент-сервер.
2. Измените элемент - его атрибуты, собрания связей и тому подобное.
3. Вызовите операцию update объекта DDO.

Java

```
ddo.update();
```

C++

```
ddo->update();
```

Если для элемента включена поддержка версий, вы можете создать новую версию элемента, а не изменять текущий, как показано в следующем примере:

Java

```
ddo.update(DKConstant.DK_CM_VERSION_NEW);
```

C++

```
ddo->update(DK_CM_VERSION_NEW);
```

Чтобы изменить последнюю версию элемента, воспользуйтесь форматом, показанным в примере ниже:

Java

```
ddo.update(DKConstant.DK_CM_VERSION_LATEST);
```

C++

```
ddo->update(DK_CM_VERSION_LATEST);
```

Можно также изменить объект DDO, вызвав метод `updateObject` для `DKDatastoreICM` с соответствующими опциями, как показано в примере:

Java

```
// Объект соединенного DKDatastoreICM с именем ds;  
// Объект DKDDO с именем ddo;  
int options = DK_CM_VERSION_NEW + DK_CM_CHECKIN;  
ds.updateObject(ddo, options);
```

C++

```
int options = DK_CM_VERSION_NEW + DK_CM_CHECKIN;  
ds->updateObject(ddo, options);
```

Внимание: Прежде чем вызывать метод для изменения, надо зарезервировать элемент, а после изменения - вновь активировать его. Смотрите “Активирование и резервирование элементов” на стр. 202. Дополнительная информация об изменении элементов есть в учебном примере `API SItemUpdateICM`.

Как определить тип ресурсного элемента

Ресурсный элемент - это элемент с дополнительными системными атрибутами, которые определяют положение, тип, размер и другие свойства объекта, представляемого этим элементом. Этот объект, иногда называемый ресурсом, может быть видеофайлом, изображением документом текстового процессора и т.п. Дополнительную информацию смотрите в примере `SItemTypeCreationICM` (в каталоге `samples`).

Ниже описаны шаги процесса определения типа ресурсного элемента.

1. Получите объект определений контент-сервера от соединенного контент-сервера.

Java

```
DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM)dsICM.datastoreDef();
```

C++

```
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
```

2. Создаем новое определение типа элементов.

Java

```
DKItemTypeDefICM itemType = new DKItemTypeDefICM(dsICM);  
itemType.setName("SampleResource");  
itemType.setDescription("Simple Resource Lob Item Type");
```

C++

```
DKItemTypeDefICM * itemType = new DKItemTypeDefICM(dsICM);  
itemType->setName("SampleResource");  
itemType->setDescription("Простой ресурсный тип элементов для больших  
объектов");
```

3. Добавляем атрибут.

Java

```
DKAttrDefICM attr = (DKAttrDefICM)dsDefICM.retrieveAttr("S_varchar");  
itemType.addAttr(attr);  
//Класс будет ресурсным, так как он будет содержать файл данных  
itemType.setClassification  
(DKConstantICM.DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
```

C++

```
DKAttrDefICM * attr = (DKAttrDefICM*)dsDefICM->retrieveAttr("S_varchar");  
itemType->addAttr(attr);  
// Класс будет ресурсным, так как он будет содержать  
// файл данных  
itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
```

4. Задаем класс XDO и тип ресурса для этого типа элементов.

Java

```
itemType.setXDOClassName(DKConstantICM.DK_ICM_XDO_LOB_CLASS_NAME);  
itemType.setXDOClassID(DKConstantICM.DK_ICM_XDO_LOB_CLASS_ID);  
itemType.add();
```

C++

```
itemType->setXDOClassName(DK_ICM_XDO_LOB_CLASS_NAME);  
itemType->setXDOClassID(DK_ICM_XDO_LOB_CLASS_ID);  
itemType->add();
```

Создание ресурсного элемента

Создание ресурсных элементов очень похоже на создание обычных элементов. XDO расширяют DDO, и в зависимости от типа ресурсного элемента XDO можно расширять далее. В Табл. 13 приводятся типы ресурсных элементов и иерархия классов, используемых для их создания. Дополнительную информацию смотрите в примере SResourceItemCreationICM (в каталоге samples).

Таблица 13. Иерархия классов для типа ресурсного элемента

| Тип | DDO | XDO | Расширение |
|----------------|----------|-------------|-------------|
| Большой объект | DKDDO -> | DKLobICM | |
| Текст | DKDDO -> | DKLobICM -> | DKTextICM |
| Изображение | DKDDO -> | DKLobICM -> | DKImageICM |
| Поточный | DKDDO -> | DKLobICM -> | DKStreamICM |

Ниже описаны шаги процесса создания ресурсного элемента. Есть несколько способов задавать и хранить содержимое ресурса. Следующий процесс - пример сохранения непосредственно из файла во время добавления элемента на контент-сервере.

1. Создайте ресурсный элемент. Обратите внимание на то, что он может иметь любой семантический тип, а для создания ресурсного элемента точно так же, как и для обычного элемента, используется вызов `DKDatastoreICM::createDDO`. Элемент ресурсного типа, возвращаемый `DKDatastoreICM.createDDO`, преобразуется в правильный подкласс (в данном случае `DKLobICM`) на основе классификации XDO, заданной при создании типа элементов, на котором основан этот ресурсный элемент.

Java

```
DKLobICM    lob = (DKLobICM)dsICM.createDDO  
            ("SampleResource",DKConstant.DK_CM_DOCUMENT);
```

C++

```
DKLobICM*    lob = (DKLobICM*)  
dsICM->createDDO("SampleResource",DK_CM_DOCUMENT);
```

2. Задайте содержимое или данные в объекте. После создания большого объекта можно задать тип MIME для ресурса. В данном случае ресурс - это документ MS Word. Тип MIME описывает тип сохраняемого содержимого.

Java

```
lob.setContentFromClientFile(fileName);  
lob.setMimeType("application/msword");
```

C++

```
lob->setContentFromClientFile(fileName);  
lob->setMimeType("application/msword");
```

Дополнительную информацию о типах MIME смотрите в примере `SResourceItemMimeTypesICM`.

3. Добавьте данные на контент-сервер. В этом случае данные - это документ Word примера. Обратите внимание на то, что содержимое ресурсного элемента сохраняется на менеджере ресурсов.

Java

```
lob.add("SResourceItemICM_Document1.doc");
```

C++

```
lob->add("SResourceItemICM_Document1.doc");
```

Внимание: В C++ надо очистить память.

```
delete(lob);
```

Дополнительную информацию смотрите в примере SItemUpdateICM.

Поиск элементов

Для поиска элементов, соответствующих набору критериев, выполните следующие действия:

1. Соединитесь с объектом DKDatastoreICM.
2. Выполните правильный запрос. Ваш запрос должен соответствовать языку запросов. Дополнительную информацию смотрите в разделе “Язык запросов” на стр. 227.
3. Сохраните результаты запроса в объекте DKResult.

Java

```
DKNVPair options[] = new DKNVPair[3];
options[0] =
    new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, "0"); // No Max (Default)
options[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,
    new Integer(DKConstant.DK_CM_CONTENT_ATTRONLY));
options[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
DKResults results = (DKResults)dsICM.evaluate(query,
    DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
```

C++

```
DKNVPair *options    = new DKNVPair[3];
// Будет возвращен только один результат
options[0].set(DK_CM_PARM_MAX_RESULTS, "1");// Получаем только один элемент
options[1].set(DK_CM_PARM_RETRIEVE , (long)DK_CM_CONTENT_ATTRONLY);
// Последняя опция должна иметь такое значение
options[2].set(DK_CM_PARM_END , NULL);
// Учтите, что если предполагается получить несколько результатов,
// надо использовать метод DKDatastoreICM::execute.
// Этот метод возвращает dkResultSetCursor.
DKResults* results = (DKResults*)(dkCollection*)dsICM->evaluate
    (query, DK_CM_XQPE_QL_TYPE, options);
```

Дополнительную информацию смотрите в примере SSearchICM.

Получение элемента

Чтобы явно получить элемент или обновить элемент, полученный путем запроса, можно вызвать объект PID или строку PID этого элемента. Если известен только ID элемента, можно выполнить запрос на получение полной информации PID. Как получить элемент явно, если известно имя типа элемента и ID элемента, показано в следующем сценарии.

Java

1. После соединения с объектом `datastoreICM` выполните поиск данного элемента при помощи соответствующего ему ID элемента.
Информацию о написании запросов смотрите в разделе “Запрос сервера Content Manager” на стр. 227.
2. Сохраните результаты запроса в объекте `DKResults`. В коде примера ниже `queryStr` - это строка поиска в правильном формате языка запросов. `DKResults results = (DKResults)dsICM.evaluate(queryStr, DKConstantICM.DK_CM_XQPE_QL_TYPE, null);`
3. Создайте итератор `DKResults` и используйте его для получения DDO - теперь их можно получать один за другим.
`ddo.retrieve();`
4. Предполагается, что у вас есть строка PID, которую вы получили из DDO, как показано в примере:
`DKPidICM pid = ddo.getPidObject();
String pidStr = pid.pidString();`

Используя эту строку PID, можно получить DDO, выполнив следующие действия.

1. Создайте DDO с помощью метода `createDDO` из `DKDatastoreICM`. Не используйте конструктор `DKDDO`. В приведенном ниже примере объект `dsICM` уже соединен со складом данных Content Manager. PID - это строка с именем `pidStr`.
`DKDDO ddo = dsICM.createDDO(pidStr);`
2. Вызовите операцию получения DDO.
`ddo.retrieve();`

C++

1. После соединения с объектом datastoreICM выполните поиск данного элемента при помощи соответствующего ему ID элемента.
Информацию о написании запросов смотрите в разделе Запрос сервера Content Manager.
2. Сохраните результаты запроса в объекте DKResult. В коде примера ниже queryStr - это строка поиска в правильном формате языка запросов.

```
DKResults* results = (DKResults*)(dkCollection*)  
dsICM->evaluate(queryStr, DK_CM_XQPE_QL_TYPE,NULL);
```
3. Создайте DDO с помощью метода createDDO объекта DKDatastoreICM. Не используйте конструктор DKDDO. В приведенном ниже примере объект dsICM уже соединен со складом данных Content Manager. PID задан как строка с именем pidStr.

```
DKDDO* ddo = dsICM->createDDO(pidStr);
```
4. Вызовите операцию получения DDO.

```
ddo->retrieve();
```

Если для элемента включена поддержка версий, вы можете получить заданную версию элемента при помощи метода получения класса DKDDO с соответствующими опциями. Чтобы получить последнюю версию элемента, используйте в качестве опции получения константу DK_CM_VERSION_LATEST. По умолчанию (если не задано никаких опций) ddo.retrieve() получает последнюю версию элемента.

Пример:

```
DKDDO ddo = ds.createDDO(...);  
.... ddo.retrieve(DK_CM_VERSION_LATEST);
```

Можно также получить DDO, вызвав метод retrieveObject в объекте DKDatastoreICM.

Пример:

```
DKDatastoreICM ds = new DKDatastoreICM();  
// соединяемся и т.п.  
DKDDO ddo =ds.createDDO(itemType,option);  
// задаем PID, как показано выше  
ds.retrieveObject(ddo);
```

Дополнительную информацию о получении элементов смотрите в учебных примерах API SItemRetrievalICM и SResourceItemRetrievalICM.

Если для элемента включена поддержка версий и вы хотите получить самую последнюю версию элемента, включая его атрибуты и дочерние компоненты, используйте следующий формат:

Java

```
int options =DK_CM_CONTENT_ATTRONLY + DK_CM_CONTENT_CHILDREN +  
                                                    DK_CM_VERSION_LATEST;  
ds.retrieveObject(ddo,options);
```

C++

```
DKDDO* ddo;  
long options =DK_CM_CONTENT_ATTRONLY +DK_CM_CONTENT_CHILDREN +  
                                                    DK_CM_VERSION_LATEST;  
dsICM->retrieveObject(ddo,options);
```

Дополнительную информацию о получении, включая опции получения, смотрите в примере SItemRetrievalICM.

Удаление элементов

Чтобы удалить элемент с контент-сервера, используйте метод delete объекта DDO.

Java

```
ddoDocument.del();
```

C++

```
ddoDocument->del();
```

Для DDO должны быть заданы ID элемента и тип объекта и установлено правильное соединение с контент-сервером.

Дополнительную информацию об удалении элементов смотрите в учебном примере API SItemDeletionICM.

Активирование и резервирование элементов

Чтобы предотвратить одновременное изменение одного и того же элемента двумя пользователями, нужно резервировать элемент перед его изменением. Резервирование элемента дает зарезервировавшему элемент пользователю монопольное право на его изменение. Резервируя элемент, вы получаете

постоянную блокировку записи для этого элемента. Весь элемент резервируется и активируется как единое целое, включая все его версии и дочерние компоненты. Но элементы, с которыми установлены связи или на которые сделаны ссылки, не резервируются вместе с элементом. Постоянная блокировка элемента снимается, когда вы снова активируете элемент.

Для активирования и резервирования элементов используются операции `checkIn` и `checkOut` в `DKDatastoreICM`, как показано в приведенном ниже примере.

1. Установив соединение с объектом `DKDatastoreICM` с именем `dsICM` и используя DDO под названием `myDataObject`, резервируем элемент.

Java

```
dsICM.checkOut(myDataObject);
```

C++

```
dsICM->checkOut(myDataObject);
```

2. Активируем элемент.

Java

```
dsICM.checkIn(myDataObject);
```

C++

```
dsICM->checkIn(myDataObject);
```

Дополнительную информацию о резервировании и активировании элементов смотрите в учебном примере `API SItemUpdateICM`.

Задание и получение свойств версий элемента

В следующем примере показано, как задать и как получить свойства версий элемента.

Java

```
DKPidICM pid = (DKPidICM)dco.getPidObject();
// Получаем информацию о версии
String version = pid.getVersionNumber();
...
// Задаем номер версии в DDO
pid.setVersionNumber(version);
```

C++

```
DKPidICM * pid = (DKPidICM *)dco->getPidObject();
// Получаем информацию о версии
DKString version = pid->getVersionNumber();
....
//Задаем номер версии для PID, связанного с элементом (DDO).
pid->setVersionNumber((char *)version);
```

Работа со свойствами поддержки версий

В этом разделе приведены примеры, которые помогут при работе со свойствами поддержки версий.

Получение правила управления версиями для типа элементов

У элемента есть правила управления версиями, содержащие свойство управления версиями для этого элемента. В следующем списке представлены три возможных свойства управления версиями и значения, используемые для представления каждого свойства в правилах управления версиями.

DK_ICM_VERSION_CONTROL_ALWAYS

Постоянная поддержка версий.

DK_ICM_VERSION_CONTROL_NEVER

Версии для этого типа элементов не поддерживаются (по умолчанию).

DK_ICM_VERSION_CONTROL_BY_APPLICATION

Схема управления версиями определяется прикладной программой.

Java

```
short versionControlPolicy;
DKItemTypeDefICM item = newDKItemTypeDefICM();
....
versionControlPolicy = item.getVersionControl();
```


C++

```
short versionControlPolicy = 0;
DKItemTypeDefICM * item = NULL;
...
versionControlPolicy = item->getVersionControl();
```

Задание управления версиями для типа элементов

Java

```
DKItemTypeDefICM item = new DKItemTypeDefICM();
short versionControl = DK_ICM_VERSION_CONTROL_ALWAYS;
....
item.setVersionControl(versionControl);
```

C+

```
DKItemTypeDefICM * item = NULL;
short versionControl = DK_ICM_VERSION_CONTROL_ALWAYS;
...
item->setVersionControl(versionControl);
```

Полный пример вызова информации управления версиями из определений типа элементов в системе есть в учебном примере API `SitemTypeRetrievalICM`.

Получение максимального разрешенного числа версий для типа элементов

Java

```
short versionMax;
DKItemTypeDefICM item = new DKItemTypeDefICM();
....
versionMax = item.getVersionMax();
```

C++

```
short versionMax = 0;
DKItemTypeDefICM * item = NULL;
....
versionMax = item->getVersionMax();
```

Задание максимального разрешенного числа версий для типа элементов

В этом примере системой поддерживаются только десять версий элемента, относящегося к данному типу элементов.

Java

```
short versionMax=10;
DKItemTypeDefICM item = new DKItemTypeDefICM();
...
item.setVersionMax(versionMax);
```

C++

```
short versionMax    = 10;
DKItemTypeDefICM * item = NULL;
....
item->setVersionMax(versionMax);
```

Задание значения типа поддержки версий для типа элементов

C++

```
DKItemTypeDefICM * item = NULL;
short versionType = DK_ICM_ITEM_VERSIONING_OPTIMIZED;
...
item->setVersioningType(versioningType);
```

Работа с папками

Папка - это полностью поддерживаемый объект DDO, представляющий элемент. В папке поддерживается полная иерархическая структура данных для типа элементов, который в ней создается. Папка - это объект DDO семантического типа папка, в котором используется атрибут, DKConstant.DK_CM_DKFOLDER, содержащий DKFolder, независимо от классификации типа элемента. Объект DKFolder (собрание DKSequentialCollection) может содержать любое число DDO - корневых компонентов, представляющих другие элементы.

Фрагменты кода, включенные в процедуру ниже, используются только для информации. Не копируйте и не вставляйте их сразу в свою программу, поскольку они не будут работать в приведенном виде. Полный пример папок, готовый для запуска, смотрите в примере SFolderICM в каталоге samples.

Создание папки

Для создания DDO во время выполнения используется метод DKDatastoreICM.createDDO(). Как показано в примере SItemCreationICM (в

каталоге samples), при создании элементов-папок свойство типа элементов или семантического типа надо задать как DKConstant.DK_CM_FOLDER.

При помощи метода DKDatastoreICM.createDDO создается элемент-папка с семантическим типом DK_CM_FOLDER. Создать папку можно, задав DKConstant.DK_CM_FOLDER как второй параметр метода createDDO.

Java

```
DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
```

C++

```
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);
```

При помощи метода setData заполните ddoFolder. Когда DDO папки создан, он сохраняется на постоянном контент-сервере.

Java

```
ddoFolder.add();
```

C++

```
ddoFolder->add();
```

Добавление содержимого в папку

Перед вызовом метода add() или update() для DDO папки все содержимое, которое надо поместить в DKFolder, уже должно быть постоянным на контент-сервере. Чтобы сделать содержимое папки постоянным, вызовите соответствующие методы DKDDO.add().

Чтобы добавить элементы в папку, воспользуйтесь функцией DKFolder addElement(). Когда вы добавите в эту папку достаточно элементов, вызовите метод add или update, чтобы сохранить отношения между папкой и содержимым на постоянном складе. Она группирует любое число изменений папки в одном вызове в библиотечный сервер. При добавлении элементов в папку не добавляйте в DKFolder несколько копий одного элемента. Поскольку все изменения в папках записываются, не делайте конфликтующих или дублирующих изменений. Например, не добавляйте в папку несколько копий одного элемента.

Чтобы добавить содержимое в папку, выполните следующие действия: Обратите внимание на то, что одним из элементов содержимого папки может быть другая папка (подпапка).

Java

1. Создайте три новых элемента. Эти элементы будут добавлены в папку в качестве ее содержимого. Содержимое папки может иметь любой семантический тип.

```
DKDDO ddoDocument=dsICM.createDDO("S_simple",DKConstant.DK_CM_DOCUMENT);  
DKDDO ddoFolder2=dsICM.createDDO("S_simple",DKConstant.DK_CM_FOLDER);  
DKDDO ddoItem=dsICM.createDDO("S_simple",DKConstant.DK_CM_ITEM);
```
2. При помощи метода setData заполните эти DDO. Элементы, добавляемые как содержимое папки, должны быть постоянными на складе данных.

```
ddoDocument.add();  
ddoItem.add();  
ddoFolder2.add();
```
3. Получите атрибут dkFolder ранее созданного постоянного DDO папки.

```
short dataid = ddoFolder.dataId  
            (DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKFOLDER);  
dkFolder = (DKFolder) ddoFolder.getData(dataid);
```
4. Перед изменением (то есть добавлением содержимого) папку на складе данных надо зарезервировать.

```
dsICM.checkOut(ddoFolder);
```
5. Добавьте ранее созданные элементы в папку.

```
dkFolder.addElement(ddoDocument);  
dkFolder.addElement(ddoItem);  
dkFolder.addElement(ddoFolder2);
```
6. Выполните принятие изменений в папке и явно активируйте ее.

```
ddoFolder.update();  
dsICM.checkIn(ddoFolder);
```

C++

1. Создайте элементы, которые будут добавлены в папку.

```
DKDDO * ddoDocument = dsICM->createDDO("book", DK_CM_DOCUMENT);  
DKDDO * ddoFolder2 = dsICM->createDDO("book", DK_CM_FOLDER);  
DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);
```
2. Сохраните созданные элементы на складе данных.

```
ddoDocument->add();  
ddoItem->add();  
ddoFolder2->add();
```
3. Получите атрибут DKFolder для этой папки.

```
DKFolder * dkFolder;  
short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR,  
                                DK_CM_DKFOLDER);  
if (dataid!=0) dkFolder = (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
```
4. Зарезервируйте папку (папку надо зарезервировать перед изменением).

```
dsICM->checkOut(ddoFolder);
```
5. Добавьте созданные элементы в папку.

```
dkFolder->addElement(ddoDocument);  
dkFolder->addElement(ddoItem);  
dkFolder->addElement(ddoFolder2); // Папки могут содержать подпапки.
```
6. Измените папку. При этом папка также будет неявно активирована (разблокирована).

```
ddoFolder->update();
```
7. Явно активируем папку.

```
dsICM->checkIn(ddoFolder);
```

Удаление содержимого из папки

Есть два способа удаления элементов из папки. Отложенное удаление (само удаление при этом выполняется при изменении DDO папки, когда несколько вызовов контент-сервера объединяются в один) реализуется при помощи методов `removeElementXX`. Немедленное удаление (менее экономный способ, так как при этом происходит несколько вызовов контент-сервера) можно выполнить при помощи метода `dkFolder.removeMember`. Дополнительные подробности по работе с папками смотрите в примере `SFolderICM` (в каталоге `samples`).

Для удаления содержимого из папки выполните следующие действия:

Java

1. Зарезервируйте DDO папки, из которой вы хотите удалить элемент.
`dsICM.checkOut(ddoFolder);`
2. Найдите удаляемый элемент. В данном случае надо найти созданный ранее DDO docItem. Создайте итератор для просмотра содержимого папки.

```
dkIterator iter = dkFolder.createIterator();
String itemPidString = ddoItem.getPidObject().pidString();
while(iter.more()){
    // Передвигает указатель на следующий элемент и возвращает этот
    // объект. Сравните строки PID DDO, возвращенного итератором,
    // с удаляемым DDO.
    DKDDO ddo = (DKDDO) iter.next();
    if(ddo.getPidObject().pidString().equals(itemPidString)) {
        // Удаляемый элемент найден.
        // Удаляем его (то есть текущий элемент итератора)
        dkFolder.removeElementAt(iter);
    }
}
```

3. Сделайте изменение постоянным и активируйте DDO папки.

```
ddoFolder.update();
dsICM.checkIn(ddoFolder);
```

C++

1. Создаем элемент и добавляем его в папку. Эта папка была создана ранее.

```
DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);
ddoItem->add();
DKFolder * dkFolder;
short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR, DK_CM_DKFOLDER);
if (dataid!=0) dkFolder =
    (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
dsICM->checkOut(ddoFolder);
dkFolder->addElement(ddoItem);
ddoFolder->update();
```

2. Явно активируем папку.

```
dsICM->checkIn(ddoFolder);
```

3. Создаем итератор для папки.

```
dkIterator* iter = dkFolder->createIterator();
```

4. С помощью итератора перемещаемся по содержимому папки, пока не найдем удаляемый элемент. Удаляем этот элемент.

```
while (iter->more())
{
    DKDDO* ddo = (DKDDO*) iter->next()->value();
    if ( ((DKPidICM*) ddo->getPidObject())->pidString() ==
        ((DKPidICM*) ddoItem->getPidObject())->pidString() )
    {
        //Элемент, который нужно удалить, найден. Удаляем его
        dkFolder->removeElementAt(*iter);
        // Теперь, когда ddoItem найден, удаляем его.
    }
}

delete(iter);
```

Получение содержимого папки

Чтобы получить содержимое папки, нужно задать опцию получения. По умолчанию опция получения не установлена. Если не установить опцию получения, объект не будет содержать атрибут DKFolder или DK_CM_DKFOLDER, пока не будет вызван метода retrieve и не будут установлены правильные опции. К опциям получения относятся:

- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND
- DKConstant.DK_CM_CONTENT_ITEMTREE

Для получения элемента папки с собранием атрибутов DKFolder и заданными исходящими связями используется следующий формат:

Java

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);
```

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_LINKS_OUTBOUND);
```

Для получения элемента папки, где дерево элементов содержит связи и содержимое папки, используется такой формат:

Java

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_ITEMTREE);
```

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_ITEMTREE);
```

Получение всех папок, содержащих конкретный DDO

Один и тот же элемент (DDO) может содержаться в нескольких папках. Это позволяет связывать элемент с разными прикладными программами или пользователями. Как определить, какие папки сейчас содержат конкретный DDO, показано в примере ниже.

Ниже подробно показано, как найти папки, содержащие ранее созданный объект `ddoDocument`. Подробнее о работе с папками можно узнать из примера `SFolderICM` (в каталоге `samples`).

Java

1. Получите объект расширения склада данных.

```
DKDatastoreExtICM dsExtICM =(DKDatastoreExtICM)
    dsICM.getExtension(DKConstant.DK_CM_DATASTORE_EXT);
```

2. Вызовите метод `getFoldersContainingDDO` для объекта расширения, чтобы найти папки, которые содержат объект `ddoDocument`. Он возвращает собрание DDO, причем каждый возвращенный DDO - это папка, содержащая объект `ddoDocument`.

```
DKSequentialCollection list =
    dsExtICM.getFoldersContainingDDO(ddoDocument);
```

3. Создайте итератор для просмотра возвращенного собрания папок.

```
iter =list.createIterator();
while(iter.more()){
    // Перемещаем итератор на следующий элемент и
    // возвращаем этот объект.
    DKDDO ddo = (DKDDO) iter.next();
    // Печатаем Id элемента для папки с DDO, чтобы идентифицировать
    // объект - возвращаемую папку.
    System.out.println(" - ID элемента: " +
        ((DKPidICM)ddo.getPidObject()).getItemId());
}
```

C++

```
DKDDO* ddoDocument = ....;
//Создаем объект склада данных, соединяемся со складом данных,
// создаем DDO и т.д.
....
//Создаем новый объект расширения склада данных из подсоединенного объекта
DKDatastoreExtICM* dsExtICM = (DKDatastoreExtICM*)
    dsICM->getExtension(DK_CM_DATASTORE_EXT);
//Получаем PID для созданного DDO
DKPidICM* pid = (DKPidICM*) ddoDocument->getPidObject();
//Получаем список папок, содержащих созданный ранее DDO
DKSequentialCollection *list =
    dsExtICM->getFoldersContainingDDO(ddoDocument);
//Создаем итератор для возвращенного собрания DDO
dkIterator* iter =list->createIterator();
while(iter->more()) {
    DKDDO* ddo =(DKDDO*) iter->next()->value();
    pid = (DKPidICM*) ddo->getPidObject();
    cout << "-Item ID:" << pid->getItemId() << endl;
    delete ddo;
}
delete iter;
```

Определение связей между элементами

Связи можно использовать для связывания исходного элемента и элемента назначения с необязательным элементом описания. Связи определяются в объектах DKLink, в которых вы можете задать следующие типы элементов связей:

Таблица 14. Структура данных связи

| DKLink | Определение |
|--------------|------------------------------------|
| LinkTypeName | Тип связи. |
| Source | Исходный элемент связи. |
| Target | Элемент назначения связи. |
| LinkItem | Элемент описания (необязательный). |

Поскольку связь представляет отношение один-ко-многим, она представлена в DDO элементом данных в пространстве имен LINK со значением DKLinkCollection. Дополнительную информацию о работе со связями смотрите в примере SLinksICM в каталоге samples.

Сама связь не принадлежит ни источнику, ни назначению. Она просто соединяет источник и назначение между собой. Например, если источник А связан с назначением В, А всегда будет источником, а В - назначением, независимо от того, на какой DDO (А или В) есть ссылка.

В памяти и DDO источника, и DDO назначения содержат копии одного и того же объекта DKLink. Поскольку объект DKLink содержит и ссылку на источник, и ссылку на назначение, DDO, содержащий связь, также содержит связь, которая ссылается на него и на другой DDO. Например, если источник А связать с назначением В, тогда и А, и В будут содержать одну и ту же связь, как показано в примере в Табл. 15.

Таблица 15. Пример определения DKLink

| Связь DKLink, определенная от А к В | DDO А | DDO В |
|-------------------------------------|----------------|----------------|
| Источник - А | Источник - А | Источник - А |
| Назначение - В | Назначение - В | Назначение - В |

При добавлении или удалении связей постоянного склада операция выполняется только для одного из двух элементов, для источника или для назначения. На другом элементе связь изменяется автоматически.

Дополнительную информацию о связях и полный пример со связями, готовый для запуска, смотрите в примере SLinksICM в каталоге samples.

Входящие и исходящие связи

При использовании связей для ссылок на конкретные DDO (источника или назначения) связи можно рассматривать как *входящие* или *исходящие*. С точки зрения объекта DDO - назначения связи, эта связь является входящей. В то же время для DDO на другом конце связи та же связь считается исходящей.

В примере в Табл. 15 на стр. 214 связь от DDO А к DDO В является исходящей, а та же связь для DDO В - исходящей.

Имена типов связей

У отношений связей есть имена. В объекте DDO связи группируются в собрания связей. Предусмотрены имена типов связей, определенные системой, но можно также определить и свои собственные. Можно использовать любое число пользовательских имен типов связей и имен типов связей, определенных системой. Системные имена типов связей:

Имя типа связей содержит

Константа Java: DKConstant.DK_ICM_LINKTYPENAME_CONTAINS

Константа C++: DK_ICM_LINKTYPENAME_CONTAINS

Имя типа связей: DKFolder

Константа Java : DKConstant.DK_ICM_LINKTYPENAME_DKFOLDER

Константа C++ : DK_ICM_LINKTYPENAME_DKFOLDER

Имя типа связи DKFolder обеспечивается и используется системой для управления папками. Объект DKFolder - это простой интерфейс для собрания исходящих связей, упрощающий работу с папками. Следовательно, не надо использовать имя типа связи DKFolder для определения или удаления связей. Кроме того, никогда не используйте имя типа связи DKFolder с DKLinkCollection. Однако для поиска папок с использованием связей надо указать это имя DKFolder.

Дополнительную информацию о связях смотрите в примере SLinksICM в каталоге smbroot\samples. Информацию по созданию пользовательских типов связей смотрите в примере SLinkTypeDefinitionCreationICM.

Получение связанных элементов

Для получения связей существуют опции получения только входящих, только исходящих и обоих типов связей. Чтобы получать входящие и исходящие связи, нужно задать опцию получения. Для получения связей можно задать следующие опции:

Java

- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND
- DKConstant.DK_CM_CONTENT_LINKS_INBOUND

- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND + DKConstant.DK_CM_CONTENT_LINKS_INBOUND
- DKConstant.DK_CM_CONTENT_ITEMTREE

C++

- DK_CM_CONTENT_LINKS_OUTBOUND
- DK_CM_CONTENT_LINKS_INBOUND
- DK_CM_CONTENT_LINKS_OUTBOUND + DK_CM_CONTENT_LINKS_INBOUND
- DK_CM_CONTENT_ITEMTREE

Помните, что перед вызовом методов DDO add() или update() уже должны существовать все элементы, относящиеся к связям.

Работа с управлением доступом

Если у вас есть доступ к клиенту системного администратора Content Manager или к вызовам API Content Manager, вы можете при создании своей собственной управляющей программы управлять доступом к информации в системе Content Manager с помощью функций управления доступом. Разнообразные API управления доступом позволяют управлять доступом ко всей системе, к набору близких операций над системой и к отдельным элементам.

Вот некоторые задачи, которые можно выполнять, используя вызовы API управления доступом:

- Создание привилегий.
- Связывание списка операций с информацией в системе.
- Предоставление пользователям разрешения выполнять действия с информацией на библиотечном сервере.

Создание привилегии

Привилегию представляет класс DKPrivilegeICM. Следующие шаги и примеры кода показывают, как создать новый объект привилегии, сделать его постоянно действующим и получить привилегию. Чтобы создать привилегию, выполните следующие действия:

Java

1. Соединитесь со складом данных

```
DKDatastoreICM ds = new DKDatastoreICM();  
ds.connect("ICMNLSDb","icmadmin",password,"");  
dkDatastoreDef dsDef =(dkDatastoreDef)ds.datastoreDef();  
DKDatastoreAdminICM dsAdmin =(DKDatastoreAdminICM)dsDef.datastoreAdmin();
```

2. Получите объект DKAuthorizationMgmtICM. Этот класс обрабатывает задачи управления полномочиями.

```
DKAuthorizationMgmtICM aclMgmt = (DKAuthorizationMgmtICM)  
dsAdmin.authorizationMgmt();
```

3. Создайте новый набор привилегий.

```
DKPrivilegeICM priv = new DKPrivilegeICM(ds);
```

4. Назначьте имя объекту привилегии.

```
priv.setName("UserPriv");
```

5. Назначьте описание объекту привилегии.

```
priv.setDescription("Это пользовательская привилегия");
```

6. Добавьте новую привилегию в объект менеджера авторизации.

```
aclMgmt.add(priv);
```

7. Получите вновь созданную привилегию из объекта менеджера авторизации.

```
dkPrivilege aPriv = aclMgmt.retrievePrivilege("UserPriv");
```

8. Выведите информацию об объекте привилегии.

```
System.out.println("имя привилегии = " + aPriv.getName());  
System.out.println("privilege description = " + aPriv.getDescription());
```

C++

1. Создайте объект склада данных, а также все связанные с ним объекты управления.

```
//Создаем объект склада данных
DKDatastoreICM * ds = new DKDatastoreICM();
//Соединяемся со складом данных
ds->connect("ICMNLSDDB", "icmdmin", "password", "");
//Получаем объект определения склада данных (используемый для
//доступа к метаданным CM и управления ими)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Создаем класс, используемый для представления и обработки функций
//управления склада данных.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
    dsDef->datastoreAdmin();
//Получаем класс, используемый для представления
//функций авторизации склада данных ICM и управления ими
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
    dsAdmin->authorizationMgmt();
```

2. Создайте новый объект привилегии и задайте его свойства

```
DKPrivilegeICM * priv = new DKPrivilegeICM(ds);
//Задаем имя объекта привилегии
priv->setName("UserPriv");
//Задаем описание привилегии
priv->setDescription("Это пользовательская привилегия");
```

3. Добавьте привилегию на склад данных, используя объект управления авторизацией.

```
aclMgmt->add(priv);
```

Создание набора привилегий

Набор привилегий представляется классом `DKPrivilegeSetICM`. Для создания наборов привилегий нужно установить соединение с контент-сервером. Чтобы создать набор привилегий, выполните следующие действия:

Java

1. Создайте новые (или получите существующие) привилегии, чтобы добавить в новый набор привилегий.

```
DKPrivilegeICM priv_1 = new DKPrivilegeICM(ds);
priv_1.setName("ItemCheckOut");
DKPrivilegeICM priv_2 = new DKPrivilegeICM(ds);
priv_2.setName("ItemQuery");
DKPrivilegeICM priv_3 = new DKPrivilegeICM(ds);
priv_3.setName("ItemAdd");
```

2. Создайте новый набор привилегий.

```
DKPrivilegeSetICM privSet1 = new DKPrivilegeSetICM(ds);
```

3. Назначьте имя набору привилегий.

```
privSet1.setName("UserPrivSet");
```

4. Назначьте описание набору привилегий.

```
privSet1.setDescription("Это пользовательский набор привилегий");
```

5. Добавьте привилегии в набор привилегий.

```
privSet1.addPrivilege(priv_1);
privSet1.addPrivilege(priv_2);
privSet1.addPrivilege(priv_3);
```

6. Добавьте вновь созданный набор привилегий в менеджер авторизации.

```
AclMgmt.add(privSet1);
```

7. Выведите информацию о вновь созданном наборе привилегий.

```
DKPrivilegeSetICM aPrivSet = (DKPrivilegeSetICM)
    aclMgmt.retrievePrivilegeSet("UserPrivSet");
System.out.println("privilege set name = " + aPrivSet.getName());
System.out.println("privilege set description = " + aPrivSet.getDescription());
dkCollection coll = aPrivSet.listPrivileges();
dkIterator iter = coll.createIterator();
while(iter.more()){
    DKPrivilegeICM _priv = (DKPrivilegeICM) iter.next();
    System.out.println(" имя привилегии = " + _priv.getName());
}
```

C++

1. Создайте объект склада данных, а также все связанные с ним объекты управления.

```
//Создаем объект склада данных
DKDatastoreICM * ds = new DKDatastoreICM();
//Соединяемся со складом данных
ds->connect("ICMNLSDDB", "icmdmin", "password", "");
//Получаем объект определения склада данных
//(используемый для доступа к метаданным СМ и управления ими)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Создаем класс, используемый для представления и обработки функций
//управления склада данных.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
    dsDef->datastoreAdmin();
//Получаем класс, используемый для представления
//функций авторизации склада данных ICM и управления ими
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
    dsAdmin->authorizationMgmt();
```

2. Создайте три привилегии и задайте их свойства.

```
DKPrivilegeICM * priv_1 = new DKPrivilegeICM(ds);
priv_1->setName("ItemCheckOut");
DKPrivilegeICM * priv_2 = new DKPrivilegeICM(ds);
priv_2->setName("ItemQuery");
DKPrivilegeICM * priv_3 = new DKPrivilegeICM(ds);
priv_3->setName("ItemAdd");
```

3. Создайте новый набор привилегий и задайте его свойства.

```
DKPrivilegeSetICM * privSet1 = new DKPrivilegeSetICM(ds);
privSet1->setName("UserPrivSet");
privSet1->setDescription("Это пользовательский набор привилегий");
```

4. Добавьте созданные привилегии в этот новый набор привилегий.

```
privSet1->addPrivilege(priv_1);
privSet1->addPrivilege(priv_2);
privSet1->addPrivilege(priv_3);
```

5. Добавьте набор привилегий на склад данных, используя объект управления авторизацией.

```
aclMgmt->add(privSet1);
```

Как вывести свойства набора привилегий

Пример вывода свойств набора привилегий.

C++

```
//Получаем набор привилегий по его имени
DKPrivilegeSetICM * sPrivSet = (DKPrivilegeSetICM *)
    aclMgmt->retrievePrivilegeSet("UserPrivSet");
//Выводим свойства набора привилегий
cout<<"имя набора привилегий = "<< (char *)sPrivSet->getName() << endl;
cout<<"priv set descrip="<< (char *)sPrivSet->getDescription() << endl;
//Получаем список привилегий, входящих в этот набор
dkCollection * coll = sPrivSet->listPrivileges();
dkIterator * iter = coll->createIterator();
while (iter->more())
{
    DKPrivilegeICM* _priv = (DKPrivilegeICM *) (void *) (*iter->next());
    cout<<"    имя привилегии = "<< (char *)_priv->getName() << endl;
}
delete(iter);
```

Определение списка управления доступом (ACL)

Модель управления доступом системы Content Manager применяется на уровне управляемого объекта. Управляемый объект - это единица защищаемых пользовательских данных. Такой объект может быть отдельным элементом, типом элементов или целой библиотекой. Операции над управляемыми объектами регулируются одним или несколькими правилами управления. Эти правила управления содержатся в списке управления доступом (access control list - ACL). ACL представляется классом DKAccessControlListICM. С каждым управляемым объектом в системе Content Manager должен быть связан ACL.

Ниже в примерах показано, как определить ACL.

Java

```
//Определяем новый объект DKACLData.
//Класс DKACLData используется для хранения данных, относящихся к ACL.
DKACLData aclData1 = new DKACLData();
//задаем имя группы пользователей для ACL
aclData1.setUserGroupName("ICMADMIN");
//задает тип заказчика ACL.
aclData1.setPatronType(DK_CM_USER_KIND_USER);
//Задаем набор привилегий, относящийся к этому ACL
aclData1.setPrivilegeSet(privSet_1);
//Создаем другой объект DKACLData.
DKACLData aclData2 = new DKACLData();
aclData2.setUserGroupName("ICMPUBLIC");
aclData2.setPatronType(DK_CM_USER_KIND_GROUP);
aclData2.setPrivilegeSet(privSet_2);
//Создаем новый ACL. DKAccessControlListICM представляет ACL CM v8.2.
DKAccessControlListICM acl1 = new DKAccessControlListICM(ds);
//Назначаем имя для вновь созданного ACL
acl1.setName("UserACL");
//Назначаем описание для вновь созданного ACL
acl1.setDescription("Это пользовательский ACL");
//Добавляем в ACL ранее созданные объекты данных ACL
acl1.addACLData(aclData1);
acl1.addACLData(aclData2);
//Добавляем вновь созданный ACL в менеджер авторизации
aclMgmt.add(acl1);
//Получаем и выводим информацию о вновь созданном ACL.
DKAccessControlListICM acl_1 = (DKAccessControlListICM)
aclMgmt.retrieveAccessControlList("UserACL");
System.out.println("имя ACL = " + acl_1.getName());
System.out.println("    описание = " + acl_1.getDescription());
dkCollection coll = acl_1.listACLData();
dkIterator iter = coll.createIterator();
while(iter.more()){
    DKACLData aclData = (DKACLData) iter.next();
    DKPrivilegeSetICM _privSet = (DKPrivilegeSetICM)
        aclData.getPrivilegeSet();
    System.out.println("    Имя набора привилегий = " + _privSet.getName());
    System.out.println("    Описание набора = " + _privSet.getDescription());
    String usrGrpName = aclData.getUserGroupName();
    System.out.println("    Имя группы пользователей = " + usrGrpName);
    short patronType = aclData.getPatronType();
    System.out.println("    Тип заказчика = " + patronType);
}
```

C++

1. Определите новые объекты DKACLData. Эти объекты DKACLData используются для данных ACL.

```
DKACLData * aclData1 = new DKACLData();  
// Задаем имя группы пользователей, связанной с этим ACL  
aclData1->setUserGroupName("ICMADMIN");  
// Задаем тип пользователей ACL. Допустимо любое из трех значений:  
// DK_CM_USER_KIND_USER, DK_CM_USER_KIND_GROUP или  
// DK_CM_USER_KIND_PUBLIC.  
aclData1->setPatronType(DK_CM_USER_KIND_USER);  
// Задаем набор привилегий для этого ACL.  
DKPrivilegeSetICM * privSet_1 = (DKPrivilegeSetICM *)  
    aclMgmt->retrievePrivilegeSet("UserPrivSet");  
aclData1->setPrivilegeSet(privSet_1);
```
2. Создайте еще один объект DKACLdata.

```
DKACLData * aclData2 = new DKACLData();  
aclData2->setUserGroupName("ICMPUBLIC");  
aclData2->setPatronType(DK_CM_USER_KIND_GROUP);  
DKPrivilegeSetICM * privSet_2 = (DKPrivilegeSetICM *)  
    aclMgmt->retrievePrivilegeSet("PublicPrivSet");  
aclData2->setPrivilegeSet(privSet_2);
```
3. Создайте новый ACL. Задайте значения свойств этого нового ACL.

```
DKAccessControlListICM * acl1 = new DKAccessControlListICM(ds);  
// Задаем новое имя для создаваемого ACL.  
acl1->setName("UserACL");  
// Задаем описание для создаваемого ACL.  
acl1->setDescription("Это пользовательский ACL");
```
4. Добавьте в этот ACL (созданный на шаге 3) объекты данных.

```
acl1->addACLData(aclData1);  
acl1->addACLData(aclData2);
```
5. Добавьте этот ACL (созданный на шаге 3) в менеджер авторизации.

```
aclMgmt->add(acl1);
```

Полная программа примера находится в каталоге samples.

Как получить и вывести информацию ACL

Чтобы получить и вывести информацию ACL, выполните следующие действия.

1. Получите ACL из объекта управления авторизацией, используя имя ACL.

C++

```
DKAccessControlListICM * acl_1 = (DKAccessControlListICM *) aclMgmt->
    retrieveAccessControlList("UserACL");
cout<<"Имя ACL = "<< (char *)acl_1->getName() << endl;
cout<<"описание = "<< (char *)acl_1->getDescription() << endl;
```

2. Получите данные для этого ACL.

C++

```
dkCollection * coll = acl_1->listACLData();
dkIterator * iter = coll->createIterator();
char szpatronType[12] = {0x00};
while (iter->more())
{
    DKACLData * aclData = (DKACLData *) (void *) (*iter->next());
    DKPrivilegeSetICM * _privSet = (DKPrivilegeSetICM *) aclData->
        getPrivilegeSet();
    cout<<"имя набора привилегий = "<< (char *) _privSet->getName()
        << endl;
    cout<<"описание набора привилегий = "<< (char *) _privSet->
        getDescription() << endl;
    DKString usrGrpName = aclData->getUserGroupName();
    cout<<" имя группы пользователей = "<< (char *) usrGrpName
        << endl;
    short patronType = aclData->getPatronType();
    sprintf(szpatronType, "%d", patronType);
    cout<<" тип пользователей = "<< szpatronType << endl;
}
delete(iter);
```

Назначение ACL для типа элементов

После создания ACL его можно связать с конкретным типом элементов. Вы можете назначить ACL для типа элементов, выполнив следующие шаги:

1. Задайте новый тип элементов.

Java

```
DKItemTypeDefICM it = new DKItemTypeDefICM(dsICM);
//Назначаем имя для этого типа элементов
it.setName("TextResource1");
//Назначаем описание для этого типа элементов
it.setDescription("Текстовый ресурсный тип элементов CMv8.2.");
//Назначаем код ACL для этого типа элементов
it.setItemTypeACLCode((int)DK_ICM_ITEMACL_BIND_AT_ITEM);
....
it.add(); // делаем тип элементов постоянно действующим
...
```

C++

```
DKItemTypeDefICM * itemType = new DKItemTypeDefICM(dsICM);
// Задаем имя для этого типа элементов.
itemType->setName("TextResource1");
// Задаем описание для этого типа элементов.
itemType->setDescription("Текстовый ресурсный тип элементов CMv8.2.");
```

2. Получите данные для этого ACL.

Java

```
int itemTypeACLCode = it.getItemTypeACLCode();
// Устанавливая для флага ACL типа элементов значение TRUE(1), мы
// подтверждаем, что связывание ACL происходит на уровне типа элементов.
// Установленное для флага ACL типа элементов значение FALSE(0)
// говорило бы, что связывание ACL происходит не на уровне типа
// элементов
it.setItemLevelACLFlag(1);// - true
//Определяем, будет ли связывание ACL происходить на уровне типа
// элементов
int itemTypeACLFlag = it.getItemLevelACLFlag();
```

C++

```
itemType->setItemTypeACLCode((long) 1);
// Задав для флага ACL типа элементов значение TRUE (1),
// мы указываем, что связывание ACL выполняется на уровне типа элементов.
// Значение FALSE (0) флага ACL типа элементов говорило бы,
// что связывание ACL выполняется не на уровне типа элементов.
itemType->setItemLevelACLFlag((short)1);
itemType->add();
// Делаем этот тип элементов постоянным.
```

Полная программа примера доступна в каталоге `samples`.

Назначение ACL для элемента

Чтобы разрешить управление доступом на уровне элементов, нужно связать ACL с элементом. Для этого надо создать элемент при помощи метода `add()` для DDO, как показано в фрагменте кода ниже. Во фрагменте кода ниже предполагается, что уже есть соединение с контент-сервером и создан ACL. Полная программа находится в каталоге `samples`.

Java

```
//Создаем новый DDO
DKDDO ddoItem = dsICM.createDDO("myItemType",DKConstantICM.DK_CM_ITEM);
//создаем новый ACL (список управления доступом)
DKAccessControlListICM ac11 = new DKAccessControlListICM(ds);
//Назначаем имя для ACL
ac11.setName("MyACL");
//Добавляем новое свойство в DDO. Это свойство будет
//называться DK_ICM_PROPERTY_ACL
int propId = ddoItem.addProperty(DK_ICM_PROPERTY_ACL);
//Задаем ранее созданный (или полученный) ACL в качестве значения этого
//свойства
ddoItem.setProperty(propId,"MyACL");
//Делаем DDO постоянным на складе данных
ddoItem.add();
```

C++

1. Создайте новый элемент (DDO) существующего типа элементов.
`DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);`
2. Создайте новый ACL (список управления доступом) и задайте его свойства.
`DKAccessControlListICM * ac11 = new DKAccessControlListICM(dsICM);`
3. Задайте имя этого ACL.
`ac11->setName("MyACL");`
4. Добавьте новое свойство в DDO. Это свойство будет называться `DK_ICM_PROPERTY_ACL`.
`ushort propId = ddoItem->addProperty(DK_ICM_PROPERTY_ACL);`
5. В качестве значения этого свойства задайте созданный выше ACL. Можно также получить существующий ACL и использовать его в качестве ACL для этого элемента.
`ddoItem->setProperty(propId, "MyACL");`
6. Сохраните этот DDO на складе данных.
`ddoItem->add();`

Язык запросов

В Content Manager 8.2 доступен мощный язык запросов на основе XML. Когда прикладная программа ищет элементы, хранящиеся в системе Content Manager, базовый механизм обработки выполняет поиск на основе запросов. Чтобы эффективно работать с иерархической моделью данных Content Manager, используется язык запросов Content Manager. Этот язык запросов обеспечивает следующие преимущества:

- Поддерживает полную модель данных.
- Поддерживает версии - возможен поиск конкретной версии и поиск последней версии.
- Позволяет выполнять поиск внутри иерархий представлений типов компонентов через связанные элементы и ссылки.
- Сочетает параметрический и текстовый поиск.
- Поддерживает возможность SORTBY.
- Поддерживает управление доступом к Content Manager.
- Соответствует XQuery Path Expressions (XQPE) - подмножеству рабочего варианта W3C XML Query.
- Обеспечивает высокую эффективность поиска.

Язык запросов Content Manager представляет собой язык на основе XML и в отличие от языков внутреннего пользования соответствует стандартам XQuery Path Expressions (XQPE) - подмножеству рабочего варианта W3C XML Query. Язык запросов позволяет проводить поиск в элементах иерархических типов, быстро и просто находить нужные элементы. Для составления запросов нужно разобраться в понятиях, синтаксисе и грамматике языка запросов.

Все запросы выполняются для представлений типов компонентов. Поэтому имена, используемые для корневых или дочерних компонентов в строке запроса, могут быть или именами базовых представлений типов компонентов, созданных системой (например, Journal, Journal_Article), или именами пользовательских представлений типов компонентов (например, My_Journal, My_Book_Section). В sys admin вызов производных таблиц типов компонентов происходит как вызов поднаборов типов компонентов.

Когда вы передаете запрос на документ, папку, объект и т.п., этот запрос сначала направляется механизму обработки запросов Content Manager, который обрабатывает запрос и переводит его в соответствующие операторы SQL. Затем библиотека добавляет проверку безопасности (по ACL) и выполняет запрос.

Запрос сервера Content Manager

Чтобы запросить библиотечный сервер Content Manager, нужно:

1. Создайте запрос к контент-серверу, создав строку запроса, представляющую ваши условия поиска.

2. Вызвать метод `evaluate`, `execute` или `executeWithCallback` со своей строкой и условиями запроса.
3. Получить результаты через объект `DKResults`, `dkResultSetCursor` или `dkCallback`.

Вызовы API запросов выполняют такие задачи обработки запросов, как подготовка и выполнение запроса, мониторинг состояния выполнения запроса и получение результатов.

Логически строка запроса может содержать один из трех типов запросов: параметрический, текстовый и комбинированный. В параметрическом запросе используются такие условия, как равенство и сравнение. В текстовом запросе используются функции текстового поиска, делающие поиск более результативным. В комбинированный запрос входят условия и текстового, и параметрического запросов.

Для выполнения запроса можно воспользоваться одним из следующих методов: `evaluate`, `execute` или `executeWithCallback`. Метод `evaluate` возвращает все результаты как собрание объектов `DDO`; он полезен для небольших наборов результатов. Метод `execute` возвращает объект `dkResultSetCursor` со следующими характеристиками:

- `dkResultSetCursor` ведет себя аналогично указателю контент-сервера.
- Его можно использовать для больших наборов результатов, поскольку возвращаемые им объекты `DKDDO` пользователь получает блоками, по мере их требования.
- Объект `dkResultSetCursor` можно использовать для повторного запроса с помощью вызова методов `close` и `open`.
- Объект `dkResultSetCursor` можно использовать для удаления и изменения текущей позиции указателя набора результатов.

Метод `executeWithCallback` выполняет запрос асинхронно и посылает результаты в заданный объект обратного вызова блоками. Таким образом метод `executeWithCallback` освобождает главный поток от задачи получения результатов запроса. Дополнительную информацию о методах запроса смотрите в примере `SSearchICM`. Методы `evaluate`, `execute` и `executeWithCallback` в классе `DKDatastoreICM` описаны в справочнике `Application Programming Reference`.

Применение языка запросов к модели данных Content Manager

Чтобы было проще понять язык запросов, можно мысленно представлять себе библиотечный сервер как документ XML. Аналогия с документом XML используется только с целью пояснить язык запросов. Поэтому важно не забывать, что XML-представление библиотечного сервера - это лишь виртуальное представление, элементы данных на библиотечном сервере - это не элементы XML и при выполнении запроса вы не получаете элементы XML. В

XML-представлении библиотечного сервера элементы модели данных Content Manager представляются следующим образом:

Элементы

В общем виде каждый элемент CM представляется вложенными элементами XML: элемент XML верхнего уровня представляет корневой компонент, а вложенные элементы XML представляют дочерние компоненты. Таким образом, вложение элементов XML представляет иерархию компонентов.

Корневые компоненты

Корневой компонент представляется первым уровнем элемента XML. У корневого компонента есть следующие атрибуты XML: ID ITEMID , STRING COMPONENTID, INTEGER VERSIONID, INTEGER SEMANTICTYPE, а также любые другие пользовательские атрибуты компонента. В библиотечном сервере ITEMID уникален.

Дочерние компоненты

Дочерний компонент представлен вложенным элементом XML, и у него есть следующие атрибуты: STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID, а также любые другие пользовательские атрибуты этого компонента.

Обратите внимание на то, что в компоненте Content Manager уникален только COMPONENTID. ITEMID и VERSIONID дочернего компонента совпадают с ITEMID и VERSIONID корневого компонента.

Пользовательские атрибуты

Каждый пользовательский атрибут представлен вложенным атрибутом XML в элементе XML, который представляет содержащий компонент.

Связи Хотя входящие и исходящие связи не являются частью самого элемента в модели данных CM (они хранятся отдельно в таблице связей) при создании запроса удобно представлять себе, что они являются частью элемента XML, представляющего элемент. Это освобождает программы от явной записи операторов JOIN в запросах. Связи представляют отношение один-с-многими между элементами. Обратите внимание на то, что такое отношение возможно только между корневыми компонентами (связь не может исходить от дочернего компонента или заканчиваться на нем).

Связи от элементов представляются элементами XML <OUTBOUNDLINK> со следующими атрибутами: IDREF LINKITEMREF , IDREF TARGETITEMREF и STRING LINKTYPE. LINKITEMREF - это ссылка на элемент, где содержатся метаданные для данной связи. TARGETITEMREF - это ссылка на элемент, на который указывает эта связь. LINKTYPE - это тип связи. Аналогично связи к элементам представляются элементами XML <INBOUNDLINK> с атрибутами IDREF LINKITEMREF, IDREF SOURCEITEMREF и STRING

LINKTYPE. SOURCEITEMREF - это ссылка на элемент, от которого исходит эта связь. Дополнительную информацию о семантике связей смотрите в примерах SLinksICM и SSearchICM.

Ссылки (атрибуты ссылок)

Атрибуты ссылок представляются атрибутами XML типа IDREF. Ссылка представляет отношение один-с-одним между элементом или компонентом и другим элементом. Поэтому ссылка может указывать только на корневой, но не на дочерний компонент. Однако атрибут ссылки может исходить либо от корневого, либо от дочернего компонентов.

Атрибуты ссылок могут быть определенными системой (SYSREFERENCEATTRS) или пользовательскими (PublicationRef в примерах запросов ниже). Переход по ссылкам возможен в обоих направлениях.

Обращение ссылок выполняется подобно описанному выше обращения связей. Элемент, на который указывает ссылка, можно считать элементом, у которого есть элемент XML REFERENCEDBY с атрибутом XML REFERENCER (типа IDREF), который указывает на компонент - источник ссылки. Это соответствует элементу INBOUNDLINK с атрибутом SOURCEITEMREF для обращения связей.

Они также поддерживают семантику удаления (с ограничением, каскадного, с присвоением пустого значения и без дополнительных действий). Дополнительную информацию о ссылочных атрибутах смотрите в примерах SReferenceAttrDefCreationICM и SSearchICM.

Документы

Функции папок в Content Manager исполняет DKFolder, который сохраняет связи набора с типом связей "DKFolder" (DK_ICM_LINKTYPE_NAME_DKFOLDER). Каждое отношение папка - содержимое моделируется как исходящая связь для папки и как входящая связь для содержимого, то есть папка является источником, а содержимое - целями всех связей. Следуйте семантике поиска связей и перехода по связям для поиска папок и перехода по папкам. Дополнительную информацию смотрите в примерах SFolderICM и SSearchICM.

Понятие параметрического поиска

Для получения элементов часто используется поиск по выбранным атрибутам. В одном запросе можно проверять определенные системой и пользовательские атрибуты элементов на контент-сервере. Условия одиночного запроса включают имя атрибута, операцию и значение, в целом составляющие условие. Для выполнения параметрического поиска система Content Manager предоставляет ряд операций сравнения. В их число входят:

"="

"< "

"<="

"> "

">="

"!="

"LIKE"

"NOT LIKE"

"BETWEEN"

"NOT BETWEEN"

"IS NULL"

"IS NOT NULL"

Вы можете задать сложные условия поиска, скомбинировав простые условия поиска в критерий при помощи логических операций AND, OR и NOT. Дополнительные подробности смотрите в примерах запросов.

Понятие текстового поиска

При использовании модуля DB2 Universal Database Net Search Extender (NSE) (прежнее название в CM 8.1 - Text Information Extender (TIE)) в Content Manager поддерживаются два типа текстового поиска: текстовый поиск атрибутов, которые содержат текст в компонентах, и текстовый поиск объектов. Основное различие между этими двумя типами текстового поиска - способ хранения содержимого. Когда вы определяете, что атрибут должен допускать текстовый поиск, в столбце этого атрибута вы указываете, что для него возможен текстовый поиск. Чтобы сделать текст атрибута (столбца) доступным для поиска, NSE создает текстовый индекс. В текстовом индексе хранится информация о тексте, который нужно найти. Эта информация используется для более эффективного выполнения текстового поиска. Например, системный администратор Фред создает тип элементов под названием Journal (журнал) с дочерним типом компонента Journal_Article (журнальная статья), который он намеревается сделать доступным для текстового поиска. Один из атрибутов типа Journal_Article - Title (заголовок); Фред делает этот атрибут доступным для текстового поиска. Когда Лили ищет заголовок, содержащий слово "Java", система проводит поиск по текстовому индексу Title любых вхождений "Java".

Информация о TIE в CM 8.1 есть в документации по модулю DB2 Universal Database Text Information Extender (TIE). Информация об NSE в CM 8.2 есть в документации по модулю DB2 Universal Database Net Search Extender (NSE). В отношении текстового поиска на языке запросов ICM NSE и TIE взаимозаменяемы. С точки зрения языка запросов ICM, нет никаких отличий между NSE и TIE в синтаксисе текстового поиска и в доступных функциях.

Поиск содержимого объектов

Поиск содержимого объектов работает несколько иначе. Вместо непосредственной индексации система использует ссылку на положение объекта в менеджере ресурсов. NSE использует эту ссылку для выборки из содержимого при создании текстового индекса. Конечный пользователь не замечает никаких отличий, выполняя поиск объектов, хранимых в менеджере ресурсов. Однако чтобы механизм поиска смог найти содержимое на менеджере ресурсов, системный администратор должен задать представление текстового ресурсного типа элементов. Текстовый поиск выполняется на атрибуте ресурсного типа элементов, "TIEREF", который при текстовом поиске обращается к содержимому, хранимому в менеджере ресурсов.

Поиск документов

Текстовый поиск можно выполнять на содержимом частей документов. В запросе поддерживается виртуальное представление типов компонентов "ICMPARTS" как дочерний компонент для каждого документа в системе. При текстовом поиске атрибут "TIEREF" в представлении типа компонента "ICMPARTS" обращается к содержимому всех частей документа, доступных для текстового поиска. Конкретное применение этой возможности смотрите в примерах запросов.

Как объявить пользовательские атрибуты допускающими текстовый поиск

С помощью вызовов API DKAttrDefICM и DKItemTypeDefICMC пользовательские атрибуты можно сделать допускающими текстовый поиск. С помощью класса DKTextIndexDefICM можно изменить свойства по умолчанию созданного текстового индекса. Дополнительную информацию по интерфейсам API смотрите в электронном справочнике API и в примере SItemCreationICM.

Понятие синтаксиса текстового поиска

Запросы текстового поиска можно выполнять при помощи синтаксиса простого или сложного текстового поиска.

Простой текстовый поиск

Поскольку при выполнении текстового поиска чаще всего просто используется перечисление нескольких слов (одно за другим), для этого наиболее распространенного случая был специально разработан синтаксис простого (или упрощенного) текстового поиска, который делает поиск удобным для пользователей. Этот синтаксис предусматривает также использование "+" , "-" и текста в кавычках. Упрощенный текстовый поиск выполняется с использованием функций "contains-text-basic" и "score-basic". Функция "contains-text-basic" используется для поиска в атрибутах или в содержимом ресурсов или документов. В функции "score-basic" используется тот же синтаксис, и она используется для сортировки результатов на основе ранга результатов текстового поиска. Не забывайте, что при проверке на истинность для функции

"contains-text-basic" задается значение, равное 1, а при проверке на ложь - значение 0. Использование этих функций смотрите в примерах запросов.

Дополнительные сведения о синтаксисе простого текстового поиска:

- Текстовый поиск выполняется с учетом регистра (как и при использовании сложного синтаксиса по умолчанию). Информацию об опциях регистрозависимого поиска смотрите в документации по NSE.
- Термы в кавычках рассматриваются как словосочетания.
- Использование плюса (+) и минуса (-)
 - + (плюс) = документ должен содержать это слово.
 - - (минус) = в документе не должно быть этого слова
 - Если плюс или минус не заданы, механизм запроса использует алгоритм совпадения заданных слов с текстом.
- Логические операции (AND, OR, NOT) недопустимы и игнорируются
- Скобки не поддерживаются в базовом синтаксисе.
- Допустимые символы подстановки
 - ? (вопросительный знак) = заменяет один символ
 - * (звездочка) = заменяет любое число символов

Дополнительную информацию о простом текстовом поиске смотрите в примере SSearchICM .

Сложный текстовый поиск

Сложный синтаксис текстового поиска позволяет пользователю задать более сложные условия поиска. При таком текстовом поиске используется синтаксис текстового поиска NSE; это позволяет использовать такие мощные возможности, как контекстный поиск и поиск неполных соответствий. В синтаксисе сложного текстового поиска используются функции "contains-text" и "score" аналогично способу использования функций "contains-text-basic" и "score-basic" для простого текстового поиска. Строки, передаваемые этим функциям расширенного поиска, должны иметь синтаксис NSE, но двойные кавычки надо заменить на одинарные и наоборот. Например, условие CONTAINS (описание, 'IBM')=1 модуля NSE на языке запросов CM будет выглядеть так: contains-text(@описание, " 'IBM' ")=1. Это нужно сделать для поддержки простоты составления запросов с минимальным использованием эскейп-символов. Не забывайте, что при проверке на истинность для функции "contains-text" задается значение, равное 1, а при проверке на ложь - значение 0. Подробности о сложном текстовом поиске смотрите в примерах запросов.

Дополнительную информацию о расширенном текстовом поиске смотрите в примере SSearchICM.

Создание комбинированного параметрического и текстового поиска

Поиск можно выполнять практически по любым частям элемента или компонента, по тексту в компоненте или элементе или по тексту внутри ресурсного содержимого. Возможные типы поиска:

Параметрический поиск

Поиск по свойствам элемента или компонента, атрибутам, связям, содержимому папок и т.д.

Текстовый поиск

Поиск в тексте.

Комбинированный поиск

Поиск, включающий как параметрический, так и текстовый поиск.

Выполните приведенные ниже шаги, чтобы создать комбинированный параметрический и текстовый поиск.

Java

1. Создайте склад данных ICM и установите с ним соединение.
2. Сгенерируйте строку комбинированного запроса.

```
String queryString =  
    "//Journal_Article [Journal_Author/@LastName = \"Richardt\" +  
    \" AND contains-text (@Text, \" 'Java' & 'XML' \")=1]\";
```
3. Если вы хотите использовать опции получения, отличающиеся от опций по умолчанию, скомпонуйте их при помощи массива DKNVPair.

```
DKNVPair parms[] = new DKNVPair[3];  
String strMax = "5";  
parms[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, strMax);  
parms[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,  
    DKConstant.DK_CM_CONTENT_ATTRONLY |  
    DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);  
parms[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
```
4. Выполните комбинированный запрос одним из трех способов: evaluate, execute или executeWithCallback.

```
DKResults resultsCollection =  
    (DKResults)dsICM.evaluate(queryString,  
    DKConstant.DK_CM_XQPE_QL_TYPE,parms);
```
5. Обработайте результаты. Процедура обработки результатов зависит от используемого вами метода выполнения.

Полный пример и дополнительную документацию смотрите в учебном примере API SSearchICM в CMBROOT\Samples\java\icm.

1. Задайте опции поиска. Учтите, что в массиве опций всегда должен быть конечный элемент. Если, например, нужно задать две опции, массив опций должен содержать три элемента.

```

DKNVPair * parms = new DKNVPair[3];
DKNVPair * pparm = NULL;
DKString strMax = "5";
DKAny * anyNull = new DKAny();
//Разрешаем вернуть при поиске максимум 5 элементов
pparm = new DKNVPair(DK_CM_PARM_MAX_RESULTS, strMax);
parms[0] = *pparm;
delete pparm;
//Задаем, что нужно получить содержимое
pparm = new DKNVPair((long)DK_CM_PARM_RETRIEVE,
    DK_CM_CONTENT_ATTRONLY | DK_CM_CONTENT_LINKS_OUTBOUND);
parms[1] = *pparm;
delete pparm;
pparm = new DKNVPair(DK_CM_PARM_END, *anyNull);
parms[2] = *pparm;
delete pparm;

```

2. Выполните поиск. Есть три метода для выполнения поиска:

evaluate

Возвращает все результаты в виде собрания; удобен для небольших наборов результатов.

execute

Возвращает указатель набора результатов, используемый вызывающей программой для перемещения по результатам.

executeWithCallback

Создает поток, перемещающийся по набору результатов и вызывающий объект callback для каждого блока результатов. Вызывающая программа использует объект callback для получения результатов.

В приведенном ниже примере нужны только пять результатов, поэтому используется метод `DKDatastoreICM.evaluate`.

```

DKResults * resultsCollection = (DKResults *) (dkCollection *)
    dsICM->evaluate(queryString, DK_CM_XQPE_QL_TYPE, parms);

```

3. Выведите результаты поиска.

```

//Создаем итератор для перемещения по собранию результатов.
dkIterator* iter = resultsCollection->createIterator();

```

```

cout << "Результаты:" << endl;
cout << "      - Всего: " << results->cardinality() << endl;

```

//продолжение следует . . .

C++ (продолжение)

```
while (iter->more())
{
    //В возвращаемом массиве каждый элемент представлен DDO
    DKDDO* ddo = (DKDDO*) iter->next()->value();
    cout << "      - ID элемента: " << ((DKPidICM*)ddo->getPidObject())
    ->getItemId() << " (" << ((DKPidICM*)ddo->getPidObject())
    ->getObjectType() << ")" << endl;
}
```

- Освободите память.

```
delete(iter);
delete[] parms;
....
```

Полный пример и дополнительную документацию смотрите в учебном примере API SSearchICM в CMBROOT\Samples\cpp\icm.

Примеры запросов

В этом разделе приводится следующая информация, которая поможет вам лучше понять язык запросов и начать составлять запросы:

- Пример модели данных.
- Представление этой модели данных в виде документа XML.
- Примеры запросов.
- Грамматика языка запросов.

Запросы примеров следующих разделов основаны на модели данных примеров запросов, описанной на рис. 13 на стр. 237. Посмотрите эту иллюстрацию модели данных при изучении запросов примеров.

Дополнительный синтаксис запросов и примеры на основе другой модели данных приведены в примере SSearchICM

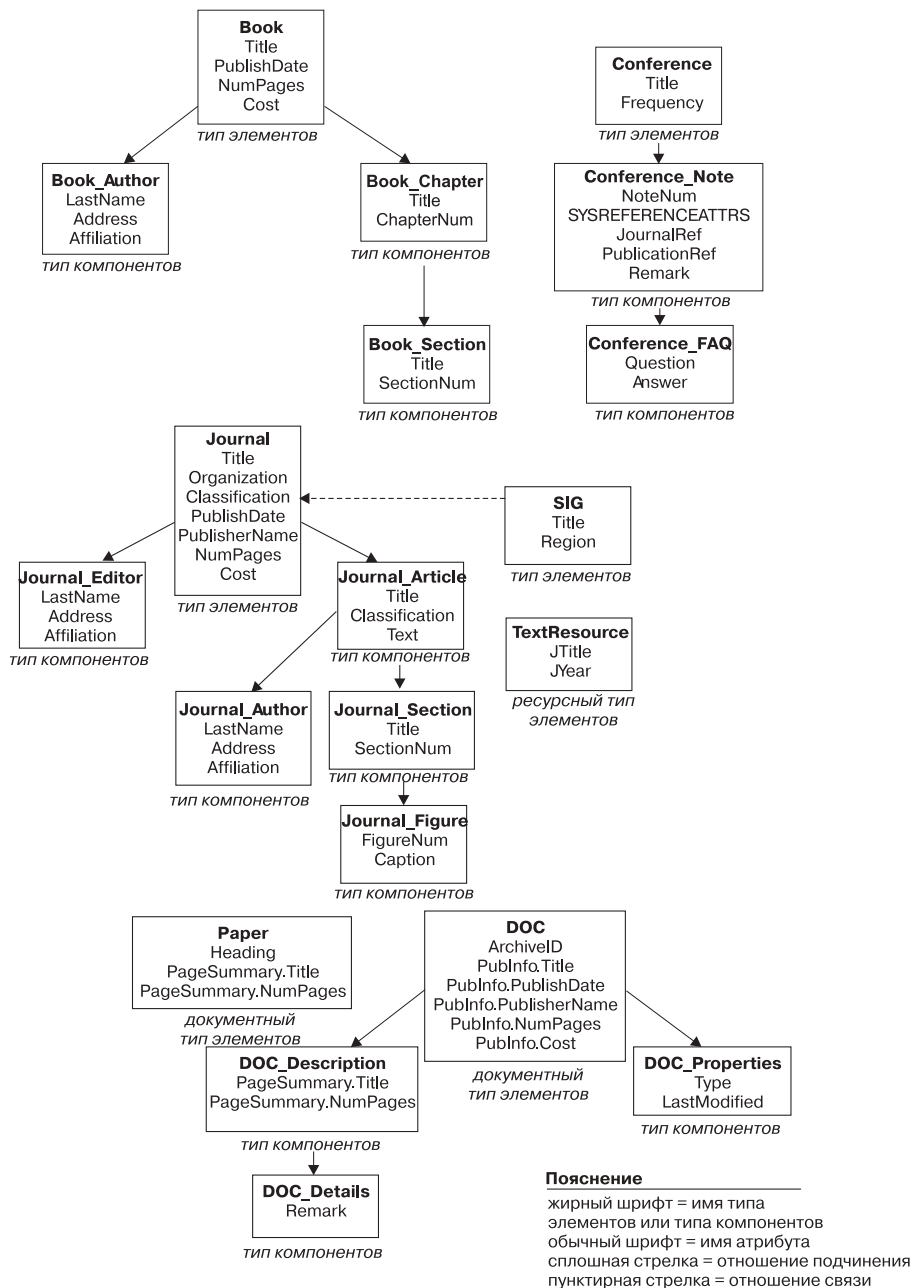


Рисунок 13. Модель данных примеров запросов

Приведенный ниже документ XML представляет модель данных на рисунке рис. 13.

Представление XML модели данных запросов примеров:

```
<Journal (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
          INTEGER SEMANTICTYPE, Title, Organization, Classification,
          PublishDate, PublisherName, NumPages, Cost)>
  <Journal_Editor (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                  LastName, Address, Affiliation)>
</Journal_Editor>
... (repeating <Journal_Editor>)

  <Journal_Article (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                  Title, Classification, Text)>
    <Journal_Section (STRING ITEMID, STRING COMPONENTID,
                     INTEGER VERSIONID, Title, SectionNum)>
      <Journal_Figure (STRING ITEMID, STRING COMPONENTID,
                      INTEGER VERSIONID, FigureNum, Caption)>
        </Journal_Figure>
      ... (repeating <Journal_Figure>)
    </Journal_Section>
    ... (repeating <Journal_Section>)

    <Journal_Author (STRING ITEMID, STRING COMPONENTID,
                    INTEGER VERSIONID, LastName, Address, Affiliation)>
      </Journal_Author>
    ... (repeating <Journal_Author>)
  </Journal_Article>
  ... (repeating <Journal_Article>)

  <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                STRING LINKTYPE) >
</OUTBOUNDLINK>
... (repeating <OUTBOUNDLINK>)

  <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                STRING LINKTYPE)>
</INBOUNDLINK>
... (repeating <INBOUNDLINK>)

  <REFERENCEDBY (IDREF REFERENCER)>
</REFERENCEDBY>
... (repeating <REFERENCEDBY>)
</Journal>
... (repeating <Journal>)

<Book (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID, INTEGER
       SEMANTICTYPE, Title, PublishDate, NumPages, Cost)>
  <Book_Author (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
               LastName, Address, Affiliation)>
</Book_Author>
... (repeating <Book_Author>)

  <Book_Chapter (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                Title, ChapterNum)>
    <Book_Section (STRING ITEMID, STRING COMPONENTID,
                  INTEGER VERSIONID, Title, SectionNum)>
    </Book_Section>
```

```

        ... (repeating <Book_Section>)
    </Book_Chapter>
    ... (repeating <Book_Chapter>)

    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
        STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
        STRING LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</Book>
... (repeating <Book>)

<SIG (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
    INTEGER SEMANTICTYPE, Title, Region)>
    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
        STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
        STRING LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</SIG>
... (repeating <SIG>)

<TextResource (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
    INTEGER SEMANTICTYPE, JTitle, JYear)>
    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
        STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
        STRING LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>

```

```

    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</TextResource>
... (repeating <TextResource>)

```

Примеры запросов

Приведенные в этом разделе запросы примеров основаны на модели данных, представленной на рис. 13 на стр. 237, и документе XML примера на странице 238. Вот несколько советов, помогающие понять примеры запросов:

- Передвижение по строке запроса аналогично передвижению по структуре каталогов
- Знак косой черты "/" означает прямое дочернее отношение
- Двойная косая черта "/" означает либо дочернее, либо родительское отношение
- "." (DOT) представляет текущий элемент в иерархии
- ".." (DOT-DOT) представляет родительский элемент текущего компонента
- "@" (знак "at") обозначает атрибут
- "[]" (квадратные скобки) означают оператор условий или список
- "=>" (операция DEREFERENCE) представляет операцию связывания или ссылки
- Результат запроса должен быть компонентом (атрибут, например, не может быть последним членом в пути)

Дополнительные примеры и информацию смотрите в примере SSearchICM.

Пример 1: обращение к компонентам

Этот пример находит все журналы.

```
/Journal
```

Объяснение: Символ "/" указывает на неявный корень документа XML, которым в этом случае является весь библиотечный сервер. Любой тип элемента является элементом под этим корнем. Если LS.xml - это документ XML, который содержит всю модель как описано выше, неявным корнем этого документа будет документ (LS.xml).

Пример 2: обращение к атрибутам

Этот запрос находит все журнальные статьи, в которых ровно 50 страниц.

```
/Journal[@NumPages=50]
```

Объяснение: Предикат @NumPages = 50 будет истинным для всех журналов, у которых атрибут CM "NumPages" имеет значение 50.

Пример 3: несколько типов элементов

Этот запрос находит все книги или журналы, где одним из авторов является “Williams”, и существует заголовок раздела, начинающийся с “XML”.

```
(/Book | /Journal)
[(./Journal_Author/@LastName = "Williams"
OR ./Book_Author/@LastName = "Williams")
AND (./Book_Section/@Title LIKE "XML%"
OR ./Journal_Section/@Title LIKE "XML%")]
```

ИЛИ

```
(/Book[./Book_Author/@LastName = "Williams"
AND ./Book_Section/@Title LIKE "XML%"])
| (/Journal[./Journal_Author/@LastName = "Williams"
AND ./Journal_Section/@Title LIKE "XML%"])
```

Объяснение: Приведенные два запроса возвращают одинаковый результат. “./Journal_Author” означает, что компонент нужно искать либо непосредственно под компонентом в пути (в первом случае это Book или Journal), либо где-то глубже в иерархии. Обратите внимание, что операция LIKE используется совместно с символом подстановки (в данном случае это “%”).

Пример 4: арифметические операции в условиях

Этот запрос находит все журналы, в которых от 45 до 200 страниц.

```
/Journal[@NumPages BETWEEN 45-4 AND 2*100]
```

Объяснение: Обратите внимание, что для вычисления значений результата, используемых с операцией BETWEEN, можно применять арифметические операции.

Пример 5: переход по связям в прямом направлении

Этот запрос находит все статьи, которые редактировал “Williams”, в журналах, содержащихся в SIG с заголовком “SIGMOD”.

```
/SIG[@Title = "SIGMOD"]/OUTBOUNDLINK
[@LINKTYPE = "contains"]/@TARGETITEMREF =>
Journal[Journal_Editor/@LastName = "Williams"]
/Journal_Article
```

Объяснение: Это пример перехода по связям в прямом направлении. Виртуальный элемент XML OUTBOUNDLINK и его атрибут TARGETITEMREF используются для перемещения по всем журналам, а на заключительном этапе - по их статьям (Journal_Articles). Последний компонент в пути - это компонент, который возвращается как результат запроса. Результат может быть ограничен переходами только по конкретным типам связей (в этом примере “contains”) для конкретных типов элементов (в этом примере Journal). Поскольку представление XML библиотечного сервера по сути выглядит как входящие и

исходящие связи, являющиеся частями элементов, чтобы освободить программы от записи операторов JOIN в явном виде, можно использовать операцию DEREFERENCE.

Пример 6: переход по связям в обратном направлении

Этот запрос находит все элементы любого типа, в которых есть журналы стоимостью меньше пяти долларов со статьями автора “Nelson”.

```
/Journal[@Cost < 5  
AND ../Journal_Author/@LastName = "Nelson"]  
/INBOUNDLINK[@LINKTYPE = "contains"]  
/@SOURCEITEMREF => *
```

Объяснение: Это пример перехода по связям в обратном направлении. Символ подстановки “*” после операции DEREFERENCE “=>” гарантирует возврат в качестве результата элементов ЛЮБОГО NBGF.

Пример 7: текстовый поиск (функции contains-text и score)

Этот запрос находит журнальные статьи автора “Richardt”, содержащие текст “Java” и текст “XML”. Результаты упорядочиваются функцией score текстового поиска.

```
//Journal_Article[Journal_Author/@LastName = "Ричард"  
AND contains-text(@Text, " 'Java' & 'XML' ")=1]  
SORTBY(score(@Text, " 'Java' & 'XML' "))
```

Объяснение: Это пример выполнения текстового поиска с функцией contains-text. Синтаксис этой функции описан в документации для модуля DB2 Net Search Engine (NSE). Обратите внимание, что для проверки на истинность значение функции contains-text должно быть равно 1, а для проверки на ложь - 0. Функция score использует информацию о показателях, возвращаемую NSE, которая используется в этом случае для сортировки журнальных статей оператором SORTBY.

Пример 7: текстовый поиск (функция contains-text и сортировка атрибутов)

Этот запрос находит все журналы, содержащие в заголовке либо слово “Design”, либо слово “Index”, и сортирует результаты по заголовкам в убывающем порядке.

```
/Journal  
[Journal_Article[contains-text(@Title, " 'Design' | 'Index' ")=1]]  
SORTBY (@Title DESCENDING)
```

Объяснение: Это другой пример выполнения текстового поиска при помощи функции contains-text. При сортировке в этом случае используется операция DESCENDING на атрибуте “Title”. Для SORTBY значение по умолчанию - ASCENDING.

Пример 9: текстовый поиск (функции contains-text-basic и score-basic)

Этот запрос находит все статьи журналов, содержащие текст “Java” и

текст “JDK 1.3”, но не содержащие текст “XML”; используется синтаксис упрощенного (простого) текстового поиска и сортировка при помощи функции score.

```
//Journal_Article  
[contains-text-basic(@Title, " +Java -XML +'JDK 1.3'")=1]  
SORTBY (score-basic(@Title, " +Java -XML +'JDK 1.3' ") )
```

Объяснение: Это пример выполнения текстового поиска с использованием синтаксиса упрощенного текстового поиска. “+” (плюс) используется для указания слов или фраз, которые должны быть представлены в атрибуте “Title”, и аналогично, “-” (минус) - чтобы исключить другие слова или фразы. Функция score-basic работает аналогично функции score в предыдущем примере, но для нее используется упрощенный синтаксис.

Пример 10: текстовый поиск ресурсных элементов

Этот запрос находит текстовые ресурсы в ресурсном типе элементов “TextResource”, которые содержат текст “Java” и текст “XML”.

```
/TextResource[contains-text(@TIEREF, " 'Java' & 'XML'  
")=1]
```

Объяснение: Это пример выполнения текстового поиска в ресурсах менеджера ресурсов. Обратите внимание, что атрибут “TIEREF” используется в качестве представления ресурсов, представленных типом элементов “TextResource”. В этом случае в функции contains-text обычно используется синтаксис TIE. Синтаксис этой функции описан в документации для модуля DB2 Net Search Engine (NSE).

Пример 11: переход по ссылкам в прямом направлении

Этот запрос находит все часто задаваемые вопросы для конференций, в материалах которых есть ссылки на книги с заголовками, где упоминается EIP.

```
/Conference/Conference_Note [@PublicationRef =>  
Book[@Title LIKE "%EIP%"]]  
/Conference_FAQ
```

Пример 12: переход по ссылкам в прямом направлении

Этот запрос находит все главы книг, на которые есть ссылки в материалах конференций в Интернете.

```
/Conference[@Title LIKE "%Internet%"]  
/Conference_Note/@PublicationRef =>  
*/Book_Chapter
```

Пример 13: переход по ссылкам в обратном направлении

Этот запрос находит все компоненты, где есть ссылки, указывающие на какие-либо книги.

```
/Book/REFERENCEDBY/@REFERENCER => *
```

Пример 14: переход по ссылкам в обратном направлении

Этот запрос находит все часто задаваемые вопросы в материалах конференций, где есть ссылки на книги о EIP.

```
/Book[@Title LIKE "XML"]/REFERENCEDBY/@REFERENCER =>  
Conference_Note/Conference_FAQ
```

Объяснение: Обратите внимание, что поскольку атрибуты ссылок исходят из компонента Conference_Note, этот компонент должен быть первым компонентом после операции DEREERENCE. Этот запрос возвратит пустой набор результатов, если, например, операция “=>” будет стоять после слова Conference.

Пример 15: переход по ссылкам в обратном направлении

Этот запрос находит все компоненты, замечания которых содержат XML, и где есть ссылки, указывающие на книги.

```
/Book/REFERENCEDBY/@REFERENCER =>  
*[@Remark LIKE "%XML%"]
```

Пример 16: функция latest version

Этот запрос выполняет поиск во всех журналах последней версии. По умолчанию возвращаются все версии представления типа компонентов, удовлетворяющие условиям запроса. VERSIONID - это атрибут, определенный системой, который содержится в каждом типе компонентов.

```
/Journal[@VERSIONID = latest-version(.)]
```

Пример 17: применение функции latest version на элементах назначения оператора DEREERENCE

Этот запрос находит все книги для последней версии, на которые есть ссылки в материалах каких-либо конференций.

```
/Conference/Conference_Note/@SYSREFERENCEATTRS =>  
Book[@VERSIONID = latest-version(.)]
```

Пример 18: использование функции latest version для компонентов с символами подстановки

Этот запрос находит все компоненты последней версии, где есть ссылки, указывающие на какие-либо книги.

```
/Book/REFERENCEDBY/@REFERENCER => *  
[@VERSIONID = latest-version(.)]
```

Пример 19: атрибуты, определенные системой

Этот запрос находит все корневые компоненты с заданным ID элементов.

```
/*[@ITEMID =  
"A1001001A01J09B00241C95000"]
```

Пример 20: текстовый поиск по модели документов

Этот запрос находит все документы, содержащие слово “XML” в любой из их частей.


```
/Doc[contains-text(../ICMPARTS/@TIEREF, " 'XML' ")=1]
```

Объяснение: В языке запросов предлагается виртуальный компонент “ICMPARTS”, который позволяет обращаться ко всем типам элементов частей ICM, содержащимся в конкретном типе элемента классификации документов.

Пример 21: модель документов (обращение к частям ICM)

Этот запрос находит все части документа с ID хранения 555.

```
/Doc[@ArchiveID = 555]/ICMPARTS/  
@SYSREFERENCEATTRS => *
```

Пример 22: модель документов (обращение к частям ICM)

Этот запрос находит все части во всех документах в системе.

```
//ICMPARTS/@SYSREFERENCEATTRS => *
```

Объяснение: Поскольку оба типа элементов (и Doc, и Paper) определены как документы в системе, в результате возвращаются части ICM от них обоих.

Пример 23: существование атрибутов

Этот пример находит все корневые компоненты, у которых есть заголовок.

```
/*[@Title]
```

Объяснение: Чтобы устранить ограничение, что следует возвращать только корневые компоненты, запрос можно переписать, начав его с двойной косой черты.

```
//*[@Title]
```

Пример 24: список и литералов и выражений

Этот запрос находит все журналы, заголовки которых совпадают с заголовками их статей или разделов, а также журналы “IBM Systems Journal”.

```
/Journal[@Title = [Journal_Article/@Title,  
../Journal_Section/@Title,"IBM Systems Journal"]]
```

Пример 25: список числовых литералов

Этот запрос находит все книги стоимостью \$10, \$20 и \$30.

```
/Book[@Cost = [10, 20, 30]]
```

Пример 26: список результатов запроса

Этот запрос находит все журналы и все книги с заголовком “Star Wars”.

```
[/Journal, /Book[@Title = "Star Wars"]]
```

Пример 27: группы атрибутов

Этот запрос находит всю информацию по документам, объем описания которых не менее 20 страниц.

```
/Doc[Doc_Description/@PageSummary.NumPages >=
20]//Doc_Details
```

Объяснение: Обратите внимание: если атрибут (например, “NumPages”) содержится в группе атрибутов (например, “PageSummary”), вы должны обращаться к нему как ИмяГруппы.ИмяАтрибута (например, PageSummary.NumPages). Атрибут “@NumPages” не будет найден под Doc_Description.

Промежуточные результаты, полученные INTERSECT/EXCEPT, нельзя сочетать с арифметическими (унарными/бинарными) операциями и операциями сравнения. Их можно совмещать с операциями set (UNION/INTERSECT/EXCEPT) или указывать отдельно.

Примеры правильного использования UNION/INTERSECT/EXCEPT:

1. (/Journal/Journal_Article[@Title = "Content Management"]
EXCEPT
//Journal_Article[@Classification =
"Security"])/Journal_Section

Такой запрос допустим, так как результат EXCEPT является результатом всего запроса - в сочетании с ним не используются никакие операции.

2. /Journal[(Journal_Editor/@LastName
UNION ../Journal_Author/@LastName) = "Davis"]

Этот запрос допустим, так как для оператора UNION отсутствует ограничение.

3. /Journal[Journal_Article[Journal_Section/@Title INTERSECT
../Journal_Figure/@Caption]/@Title = "Content Management"]

Такой запрос допустим, так как в сочетании с результатом INTERSECT не используются никакие операции.

4. /Journal[@Title = "VLDB"]
UNION /Journal[@Cost = 20]
INTERSECT /Journal[@Organization = "ACM"]

Такой запрос допустим, так как в сочетании с результатом операции INTERSECT используется операция set (UNION).

Примеры неправильного использования INTERSECT/EXCEPT:

1. /Journal[(Journal_Editor/@LastName
INTERSECT ../Journal_Author/@LastName) = "Davis"]

Такой запрос недопустим, так как в сочетании с результатом операции INTERSECT используется операция сравнения (=).

2. /Journal[(../Journal_Section/@SectionNum
EXCEPT ../Journal_Figure/@FigureNum) + 5 = 10]

Такой запрос недопустим, так как в сочетании с результатом EXCERPT используется арифметическая операция (+).

Язык запросов

Для поддержки расширенных возможностей языка запросов (например, символов подстановки "%" или "_" в текстовых строках) используются эскейп-последовательности, позволяющие различать случаи, когда символы подстановки рассматриваются как обычные символы, и случаи, когда им дается специальное значение. Пользователю важно знать, какие символы используются в качестве символов подстановки, поскольку при необходимости использовать такие символы как обычные им должен предшествовать эскейп-символ. Эскейп-последовательности применяются также для одинарных и двойных кавычек.

Эскейп-последовательности надо добавлять, когда строки, используемые в запросе, содержат специальные символы (двойные кавычки, апостроф), символы подстановки (знак процента, подчеркивание, звездочку, вопросительный знак) или эскейп-символ по умолчанию (обратная дробная черта). Обработываются такие строки просто в операциях сравнения и чуть сложнее в операции LIKE и функциях текстового поиска. Правильная обработка специальных символов обеспечивает успешное выполнение запросов и правильность результатов поиска.

Внимание: Не используйте специальные символы без необходимости, поскольку они могут привести к значительному росту объема результатов, снизить производительность и привести к появлению неожиданных результатов поиска.

Применение эскейп-последовательностей в операторах сравнения ("=", "!", ">", "<", "BETWEEN" и прочих)

Двойные кавычки "

Перед двойными кавычками ставьте еще одни двойные кавычки.

Пример:

```
//Journal_Article[@Title = "Анализ книги ""Машина времени""  
самим Гербертом Уэлсом"]
```

Поскольку заголовок этой статьи содержит название книги в двойных кавычках - "Машина времени", для этих внутренних двойных кавычек надо использовать эскейп-символы.

Одиночная кавычка (апостроф) '

В этом случае эскейп-символ не нужен.

Пример:

```
/Book[@Title != "Скарлетт О'Хара"]
```

Применение эскейп-последовательностей с оператором LIKE

Двойные кавычки "

Перед двойными кавычками ставьте еще одни двойные кавычки.

Пример:

```
//Journal_Article[@Title LIKE "Анализ книги ""Машина времени"" %"]
```

Поскольку заголовок этой статьи содержит название книги в двойных кавычках - "Машина времени", для этих внутренних двойных кавычек надо использовать эскейп-символы.

Одиночная кавычка (апостроф) '

В этом случае эскейп-символ не нужен.

Пример:

```
/Book[@Title LIKE "Скарлетт О'Хара"]
```

Символы подстановки ("% ", "_ ")

Знак процента "%" - символ подстановки, означающий любое число некоторых символов в строке, используемой в операции LIKE.

Подчеркивание "_" - символ подстановки, означающий один некоторый символ. Если вы хотите, чтобы эти символы рассматривались, как обычные символы, надо:

1. Поставить перед символом подстановки эскейп-символ
2. Добавить условие ESCAPE с эскейп-символом после операции LIKE.

Пример А:

```
/Book[@Title LIKE "Сонеты _ильям%Шекспир%"]
```

В этом примере символы подстановки "%" и "_" используются для поиска книги, точное название которой неизвестно.

Пример В:

```
//Journal_Article[@Title LIKE "Использование подчеркивания (!_) в запросах" ESCAPE "!"]
```

Поскольку строка поиска в этом примере содержит символ подчеркивания "_" как обычный символ (а не как символ подстановки), перед ним надо поставить эскейп-символ (в данном случае восклицательный знак "!"). В качестве эскейп-символа можно использовать любой отдельный символ.

Пример С:

```
//Journal_Article[@Title LIKE "_спользование подчеркивания \_ в%" ESCAPE "\"]
```

В этом запросе символ подчеркивания используется и как обычный символ ("_", перед которым стоит "\"), и как символ подстановки (в начале строки, чтобы находить слово "Использование" независимо от регистра первой буквы). Знак процента "%" используется как символ подстановки, так как конец строки неизвестен.

Пример D:

```
//Journal_Article[@Title LIKE "Использование подчеркивания !_ в Yahoo!!" ESCAPE "!"]
```

Эскейп-символ можно включать в строку поиска в качестве обычного символа. Для этого его надо повторить дважды, как в этом примере, где требуется найти "Yahoo!" .

Применение эскейп-последовательностей со сложным поиском (функции "contains" и "score")

Двойные кавычки "

Перед двойными кавычками ставьте еще одни двойные кавычки.

Пример:

```
//Journal_Article[contains-text (@Title, " 'Анализ книги  
"Машина времени" " %' ")=1]
```

Поскольку заголовок этой статьи содержит название книги в двойных кавычках - "Машина времени", для этих внутренних двойных кавычек надо использовать эскейп-символы.

Одиночная кавычка (апостроф) '

Перед апострофом надо ставить еще один апостроф. В сложном текстовом поиске одиночный апостроф не допускается, так как в апострофы заключается словосочетание. Если апостроф встречается в словосочетании, надо иметь возможность отличать его от апострофа, который завершает словосочетание.

Пример А:

```
/Book[contains-text (@Title, " 'Шпага Д''Артаньяна' ")=1] SORTBY  
(score (@Title, " 'Шпага Д''Артаньяна' "))
```

Обратите внимание на два апострофа в слове "Д''Артаньяна".

Пример В:

```
/Journal_Article[contains-text (@Title, " ('Дэвид' & 'О'Хара')  
& NOT 'Скарлетт' ")=1] SORTBY (score (@Title, " ('Дэвид' &  
'О'Хара') & NOT 'Скарлетт' "))
```

Обратите внимание на два апострофа в слове "О'Хара".

Символы подстановки ("% ", "_ ")

Как и в операции LIKE, в синтаксисе сложного поиска "%" и "_ " используются как символы подстановки. Знак процента "%" - символ подстановки, означающий любое число некоторых символов. Подчеркивание "_" - символ подстановки, означающий один некоторый символ. Если вы хотите, чтобы эти символы рассматривались, как обычные символы, надо:

1. Поставить перед символом подстановки эскейп-символ
2. Добавить условие ESCAPE после КАЖДОГО словосочетания, где используется символ подстановки

Пример А:

```
/Book[contains-text (@Title, " 'Использование подчеркивания (!_  
в запросах' ESCAPE '!' ")=1] SORTBY (score (@Title, "  
'Использование подчеркивания (!_) в запросах' ESCAPE '!' "))
```

В этом примере восклицательный знак "!" используется как эскейп-символ перед подчеркиванием.

Пример В:

```
/Book[contains-text (@Title, " 'Использование подчеркивания (!_  
в запросах' ESCAPE '!' | 'Yahoo! для чайников' | 'Использование  
подчеркивания (!_) в Yahoo!!' ESCAPE '!' | 'Война и мир' ")=1]
```

Обратите внимание на то, что условие ESCAPE надо добавлять к каждому словосочетанию, где используется эскейп-символ для символов подстановки, даже если эскейп-символы для всех этих словосочетаний одни и те же.

Применение эскейп-последовательностей с основным текстовым поиском (функции "contains-text-basic" и "score-basic")

Двойные кавычки "

Перед двойными кавычками ставьте еще одни двойные кавычки.

Пример:

```
//Journal_Article[contains-text-basic (@Title, "Анализ книги  
""Машина времени"" ")=1]
```

Поскольку заголовок этой статьи содержит название книги в двойных кавычках - "Машина времени", для этих внутренних двойных кавычек надо использовать эскейп-символы. Название книги заключено в апострофы, чтобы указать, что это словосочетание.

Одиночная кавычка (апостроф) ’

Перед апострофом надо ставить еще один апостроф. Синтаксис простого текстового поиска допускает заключать словосочетания в апострофы (словосочетание может содержать пробел). Поэтому необходимо отличать апостроф внутри словосочетания от апострофа, завершающего словосочетание.

Пример А:

```
/Book[contains-text-basic (@Title, "Шпага  
Д'Артаньяна")=1] SORTBY (score-basic (@Title, "Шпага  
Д'Артаньяна"))
```

Обратите внимание на два апострофа в слове "Д'Артаньяна".

Пример В:

```
/Journal_Article[contains-text-basic (@Title, " +актер + 'Дэвид  
О'Хара' -Скарлетт ")=1] SORTBY (score-basic (@Title, " +актер  
+ 'Дэвид О'Хара' -Скарлетт "))
```

Обратите внимание на два апострофа в слове О'Хара и на то, что 'Дэвид О'Хара' заключен в апострофы, чтобы показать, что это словосочетание.

Символы подстановки ("*", "?" и "\")

Ставьте перед символами "*", "?" и "\" обратную дробную черту "\", если эти символы не должны рассматриваться как символы подстановки. Звездочка "*" - символ подстановки, означающий любое число некоторых символов в простом текстовом поиске для функций, где используется contains-text-basic и score-basic. Вопросительный знак "?" - символ подстановки, означающий один некоторый символ. Язык запросов для простого текстового поиска предусматривает использование в качестве эскейп-символа обратную дробную черту "\", если строка поиска содержит один из этих символов, который надо рассматривать как обычный символ.

Пример А:

```
/Book[contains-text-basic (@Title, " +сонеты + '?ильям*Шекспир*'  
-пьесы ")=1] SORTBY (score-basic (@Title, " +сонеты  
+ '?ильям*Шекспир*' -пьесы "))
```

В этом примере показано, как искать в простом тестовом поиске словосочетание, точное написание которого неизвестно. В данном случае "*" и "?" означают символы подстановки, поэтому эскейп-символ перед ними не нужен.

Пример В:

```
/Book[contains-text-basic (@Title, "Что делать\?")=1] SORTBY  
(score-basic (@Title, "Что делать\?"))
```

В этом примере название книги содержит вопросительный знак "?" как обычный символ, поэтому перед ним ставится обратная дробная черта.

Пример С:

```
//Journal_Section[contains-text-basic (@Title,  
"C:\\OurWork\\IsNeverDone")=1] SORTBY (score-basic (@Title,  
"C:\\OurWork\\IsNeverDone"))
```

Каждую обратную дробную черту, которая должна входить в строку поиска "C:\\OurWork\\IsNeverDone", надо повторить дважды.

Применение эскейп-последовательностей в Java и C++

Перед спецсимволами (например, двойными кавычками или обратной косой чертой) ставьте обратную косую черту.

Пример:

Запрос:

```
/Book[contains-text-basic (@Title, "Что делать\?")=1]
```

Java

```
String query = "/Book[contains-text-basic (@Title, \"Что делать\\?\")=1]";
```

C++

```
DKString query ("/Book[contains-text-basic (@Title, \"Что делать\\?\")=1]");
```

Обратите внимание на то, что перед внутренними двойными кавычками и обратной дробной чертой перед вопросительным знаком стоит обратная дробная черта. Такая обработка присуща языкам программирования Java и C++. Дополнительную информацию смотрите в описании этих языков.

Грамматика языка запросов

Формальная грамматика языка запросов:

- (* ключевые слова *)
- AND = ("a" | "A"), ("n" | "N"), ("d" | "D") ;
- ASCENDING = ("a" | "A"), ("s" | "S"), ("c" | "C"), ("e" | "E"), ("n" | "N"), ("d" | "D"), ("i" | "I"), ("n" | "N"), ("g" | "G") ;
- BETWEEN = ("b" | "B"), ("e" | "E"), ("t" | "T"), ("w" | "W"), ("e" | "E"), ("e" | "E"), ("n" | "N") ;
- DESCENDING = ("d" | "D"), ("e" | "E"), ("s" | "S"), ("c" | "C"), ("e" | "E"), ("n" | "N"), ("d" | "D"), ("i" | "I"), ("n" | "N"), ("g" | "G") ;
- DIV = ("d" | "D"), ("i" | "I"), ("v" | "V") ;
- EXCEPT = ("e" | "E"), ("x" | "X"), ("c" | "C"), ("e" | "E"), ("p" | "P"), ("t" | "T") ;
- INTERSECT = ("i" | "I"), ("n" | "N"), ("t" | "T"), ("e" | "E"), ("r" | "R"), ("s" | "S"), ("e" | "E"), ("c" | "C"), ("t" | "T") ;
- LIKE = ("l" | "L"), ("i" | "I"), ("k" | "K"), ("e" | "E") ;
- MOD = ("m" | "M"), ("o" | "O"), ("d" | "D") ;
- NOT = ("n" | "N"), ("o" | "O"), ("t" | "T") ;
- OR = ("o" | "O"), ("r" | "R") ;
- SORTBY = ("s" | "S"), ("o" | "O"), ("r" | "R"), ("t" | "T"), ("b" | "B"), ("y" | "Y") ;
- UNION = ("u" | "U"), ("n" | "N"), ("i" | "I"), ("o" | "O"), ("n" | "N") ;
- IS = ("i" | "I"), ("s" | "S") ;
- NULL = ("n" | "N"), ("u" | "U"), ("l" | "L"), ("l" | "L") ;
- ESCAPE_KEYWORD = ("e" | "E"), ("s" | "S"), ("c" | "C"), ("a" | "A"), ("p" | "P"), ("e" | "E") ;
- КЛЮЧЕВОЕ СЛОВО = (AND | ASCENDING | BETWEEN | DESCENDING | DIV | EXCEPT | INTERSECT | LIKE | MOD | NOT | OR | SORTBY | UNION | IS | NULL) ;
- (* литералы *)
- DIGIT = ("0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") ;
- NONZERO_DIGIT = ("1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9") ;
- Exponent = (e | E), ["+" | "-"], DIGIT, { DIGIT } ;
- INTEGER_LITERAL = "0" | NONZERO_DIGIT, { DIGIT } ;
- FLOAT_LITERAL = DIGIT, { DIGIT }, ".", { DIGIT }, [Exponent] | ["."], DIGIT, { DIGIT }, [Exponent] ;
- (* UNICODE_CHARACTER - это набор всех символов Unicode и эскейп-последовательностей. Его определение не включено в этот документ *)
(* Строковые литералы заключаются в двойные кавычки и могут содержать любые символы кроме знака двойной кавычки. Чтобы включить в строковый литерал знак двойной кавычки, нужно задать два последовательных знака

двойных кавычек, то есть знак двойной кавычки предваряется другим знаком двойной кавычки. Они будут рассматриваться как один символ двойной кавычки *)

- `STRING_LITERAL = ''' , { (UNICODE_CHARACTER - ''') | (' ' , ' ') } , ' ' ;`
- (* Эскейп-последовательность - это отдельный символ, заключенный в кавычки. Если надо задать в качестве эскейп-символа двойные кавычки, задайте подряд два символа двойных кавычек - первый символ кавычек будет эскейп-символом для второго. Они будут рассматриваться как один символ двойной кавычки. Полное перечисление допустимых значений `ESCAPE_CHARACTER` приведено в справочнике DB2 SQL Reference в описании предиката `LIKE`. *)
- `ESCAPE_LITERAL = ' ' , ((ESCAPE_CHARACTER - ' ') | (' ' , ' ')), ' ' ;`
- `LETTER = ("a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | " _ " | "$") ;`
- (* `IDENTIFIER` начинается либо с латинской буквы (a-z, A-Z), либо с символа подчеркивания, либо с символа доллара, за которыми могут следовать от 0 и более букв, символов подчеркивания, символов доллара или цифр (0-9). Чтобы использовать задать идентификатор, ключевое слово, совпадающий с одним из ключевых слов, надо заключить его в одинарные кавычки *)
- `IDENTIFIER = (LETTER , { LETTER | DIGIT }) - KEYWORD | ' ' , LETTER , { LETTER | DIGIT } , ' ' ;`
- `ExpressionWithOptionalSortBy = LogicalOrSetExpression , SORTBY , "(" , SortSpecList , ")" | Expression ;`
- `Expression = LogicalOrSetExpression ;`
- `SortSpecList = SortSpec , { " , " , SortSpec } ;`
- `SortSpec = Expression , [ASCENDING | DESCENDING] ;`
- `LogicalOrSetExpression = LogicalOrSetTerm | LogicalOrSetExpression , (OR | UNION | " | " | EXCEPT) , LogicalOrSetTerm ;`
- `LogicalOrSetTerm = LogicalOrSetPrimitive | LogicalOrSetTerm , (AND | INTERSECT) , LogicalOrSetPrimitive ;`
- `LogicalOrSetPrimitive = [NOT] , SequencedValue ;`
- `SequencedValue = ValueExpression ;`
- `ValueExpression = Comparison ;`
- `Comparison = ArithmeticExpression | Comparison , CompareOperator , ArithmeticExpression , ESCAPE_KEYWORD , ESCAPE_LITERAL | Comparison , CompareOperator , ArithmeticExpression | Comparison , [NOT] , BETWEEN , ArithmeticExpression , AND , ArithmeticExpression | Comparison , IS , [NOT] , NULL ;`
- `ArithmeticExpression = ArithmeticTerm | ArithmeticExpression , (" + " | " - ") , ArithmeticTerm ;`

- ArithmeticTerm = ArithmeticFactor | ArithmeticTerm, ("*" | DIV | MOD), ArithmeticFactor ;
- ArithmeticFactor = ArithmeticPrimitive | ("+" | "-"), ArithmeticFactor ;
- ArithmeticPrimitive = BasicExpression, OptionalPredicateList | PathExpression ;
- PathExpression = Path | ("/" | "///"), Path | BasicExpression, OptionalPredicateList, ("/" | "///"), Path ;
- Path = Step | Path, ("/" | "///"), Step ;
- Step = NodeGenerator, OptionalPredicateList ;
- NodeGenerator = NameTest | "@", NameTest | "@", NameTest, "=>", NameTest | ".." ;
- OptionalPredicateList = {Predicate} ;
- Predicate ::= [", Expression, "]" ;
- BasicExpression = Literal | FunctionName, "(", OptionalExpressionList, ")" | "(" Expression ")" | ListConstructor | "." ;
- FunctionName = QName ;
- Literal = STRING_LITERAL | INTEGER_LITERAL | FLOAT_LITERAL ;
- OptionalExpressionList = [ExpressionList] ;
- ExpressionList = Expression, {"", Expression} ;
- ListConstructor = "[", [ListContent], "]" ;
- ListContent = Expression, {"", Expression} ;
- NameTest = QName | "*" ;
- QName = LocalPart ;
- LocalPart = IDENTIFIER ;
- CompareOperator = "=" | "<" | "<=" | ">" | ">=" | "!=" | [NOT] LIKE ;

Работа с менеджером ресурсов

Менеджер ресурсов Content Manager управляет собранием управляемых ресурсов (объектов). Он управляет также необходимым местом хранения и инфраструктурой HSM (Hierarchical Storage Management - иерархическое управление хранением), но для поддержки HSM менеджер ресурсов сначала нужно сконфигурировать. У менеджера ресурсов есть средства поддержки служб для ряда конкретных типов различных объектов, такие как поддержка потоков, архивирование и распаковка архивов, шифрование, кодирование, декодирование, поиск и исследование информации.

Отдельный менеджер ресурсов используется только одним библиотечным сервером. Каждый менеджер ресурсов из системы Content Manager обеспечивает общий поднабор интерфейсов API для доступа к своим данным, через которые к

нему обращается управляющий библиотечный сервер, другие компоненты Content Manager и прикладные программы, как локально (с того же узла сети), так и удаленно.

Другие API доступа к данным, обеспечивающие удаленный доступ к менеджеру ресурсов, работают либо через собственный клиент менеджера ресурсов, либо через стандартные сетевые протоколы доступа, например CIFS, NFS и FTP. Для реализации удаленного доступа воспользуйтесь соединением клиент-сервер. Клиенты соединяются с менеджерами ресурсов Content Manager через стандартный Web-сервер при помощи протокола HTTP. Доставка данных осуществляется по протоколам передачи данных HTTP, FTP и FS. При помощи HTTP любая прикладная программа или компонент Content Manager, которым нужно управляемое Content Manager содержимое, может динамически образовать треугольник с библиотечным сервером и менеджером ресурсов. Этот треугольник представляет собой прямой путь доступа к данным между программой и каждым из менеджеров ресурсов и путь для управления между библиотечным сервером и менеджером ресурсов. Вы можете воплотить этот воображаемый треугольник в любой конфигурации сети, начиная от конфигурации с одним узлом, вплоть до географически распределенной сети.

Такая архитектура приспособлена и для работы с менеджерами ресурсов, к которым прикладная программа не может обращаться напрямую, например для подсистемы на хосте, однопользовательской системы, которая не работает с правами доступа, или для системы, содержащей очень важную информацию, прямой доступ к которой прикладных программ нежелателен по соображениям бизнеса. Доступ к такому менеджеру ресурсов будет косвенным. Система Content Manager обеспечивает обе парадигмы передачи данных - pull и push, а также синхронные и асинхронные вызовы.

Информацию о конфигурировании менеджера ресурсов смотрите в книге *Планирование и установка вашей системы Content Management* и в примере SResourceMgrDefCreationICM каталога samples в cmbroot\samples\java\icm.

Работа с объектами менеджера ресурсов

В системе Content Manager каждый управляемый объект называется элементом данных или просто элементом. Элементы делятся на два типа: тип, представляющий чисто логические элементы - например, документы или папки, и тип элементов, представляющих физический объект данных, такой как текстовый документ, созданный в текстовом редакторе, отсканированное изображение страхового иска или видеозапись автомобильной аварии. У объектов есть специфические состояния и поведение, необходимые для обработки физических данных, связанных с логическим документом.

Ресурсные объекты также представляют такие сущности, как файлы в файловой системе, видеозаписи на видеосервере и двоичные большие объекты. Во время выполнения ресурсные объекты используются для доступа к физическим

данным, на которые они указывают. По этой причине ресурсным объектам в Content Manager придается тип. Это значит, что у них есть определенные состояния и поведение. Библиотечный сервер и менеджер ресурсов используют общую схему для хранения состояния объекта. Система Content Manager предоставляет следующие базовые типы объектов: общие объекты BLOB или CLOB, текстовые объекты, изображения и объекты видеосодержания. Можно также создать подклассы предопределенных типов. У ресурсного объекта могут быть пользовательские атрибуты, которые используются для его поиска и получения.

Для системы Content Manager каждый объект представляется уникальным логическим идентификатором - его универсальным идентификатором ресурса (URI). Библиотечный сервер управляет пространством имен URI. При запросе библиотечный сервер отображает идентификаторы URI на унифицированные указатели ресурсов (URL). Указатели URL используются, чтобы получить доступ к физическим данным. Указатели URL не указывают напрямую на управляемую менеджером ресурсов область хранения. Вместо этого менеджер ресурсов использует локальное пространство имен, чтобы преобразовать имена логических объектов в имена физических файлов. Идентификаторы URI объектов создаются определенным менеджером ресурсов. Библиотечный сервер или конечный пользователь могут предложить URI для объекта (его имя), но решение принимает менеджер ресурсов.

К объекту можно обращаться при помощи вызовов API менеджера ресурсов Content Manager (store, retrieve, update, delete и т.п.). В некоторых случаях можно использовать вызовы API (stream, multicast и stage), собственные для объекта или для файловой системы.

Информацию о работе с объектами менеджера ресурсов смотрите в примере SResourceItemCreationICM в каталоге samples, CMBROOT\Samples\java\icm или CMBROOT\Samples\cpp\icm.

Управление документами в Content Manager

Система Content Manager реализует гибкую модель данных управления документами, которую можно использовать для управления бизнес-объектами. Основные элементы этой модели - папки, документы и объекты.

Как уже упоминалось ранее, документы, папки и другие объекты в системе Content Manager представляются элементами. На уровне API различие между документом и папкой сводится к семантическому типу и соответствующим функциям DKFolder. Документ включает в себя атрибуты или метаданные, которые его описывают, включая однозначные атрибуты (имя документа, дата, тема), многозначные атрибуты (ключевые слова) и собрания многозначных атрибутов (адрес, состоящий из улицы, города, штата и индекса).

В модели данных управления документами для связи объектов (ресурсных элементов) с документом используются части документов. Такая модель поддерживает несколько частей для конструирования документа. Например, каждая страница может быть отдельной частью. Чтобы прикладная программа могла определить порядок частей в документе, в частях документов сохраняется номер части. Части документов содержат указатель на объект (ссылочный атрибут), включающий другую информацию о части, например, тип MIME, размер, ID менеджера ресурсов, в котором находится эта часть, имя собрания в этом менеджере ресурсов и так далее. Атрибуты разных объектов могут отличаться. Например, у комментария могут быть координаты X и Y, а у журнала примечаний - идентификатор набора кодовых символов CCSID для текста примечания.

Чтобы лучше понять модель данных управления документами, рассмотрим следующий сценарий, в котором пользователь импортирует документ с помощью клиентской программы:

- У пользователя открывается окно.
- Пользователь вводит (или выбирает) имя файла, который он хочет импортировать в систему. Например, полицейский протокол о ДТП.
- Пользователь выбирает тип документа (служебная записка, страховой иск, проект).
- Открывается новое окно, в котором пользователь вводит атрибуты, описывающие документ. Например, можно ввести дату, номер страхового иска и номер страхового полиса.
- Пользователь определяет части документа и вводит значения для этих атрибутов. Например, частью документа страхового иска будет полицейский рапорт.
- Пользователь заканчивает ввод описания документа и завершает задачу. Полицейский рапорт создан в системе.

После этого программа клиента соединяется с библиотечным сервером и менеджером ресурсов, используя для этого API или JavaBeans. Чтобы хранить документ, система создает два элемента - ресурсный и нересурсный. Два элемента создаются, поскольку в полис включается фотография, которая хранится на менеджере ресурсов. Документ создается при помощи одного вызова API. После этого объект сохраняется на менеджере ресурсов и менеджер ресурсов возвращает отметку времени и другие метаданные об объекте. Менеджер ресурсов создает ссылочный атрибут для объекта и вставляет этот ссылочный атрибут в дочерний компонент документа. Заключительный вызов библиотечного сервера выполняется, чтобы сохранить дочерний компонент и обновить атрибуты. Весь процесс осуществляется как транзакция, поэтому ошибки в работе API не приведут к созданию неполного документа.

После создания документа вы можете изменить его. Можно выполнять два типа изменений: изменение метаданных или изменение содержимого. Библиотечный сервер автоматически создает новую запись для элемента со следующим номером версии (если для элемента включена поддержка версий) и копирует все дочерние компоненты, связанные с этим элементом.

Создание модели данных управления документами

Этот раздел поможет вам выполнить основные задачи, связанные с моделью данных управления документами:

- Создание документного типа элементов.
- Создание документа.
- Изменение документа.
- Получение и удаление документа.
- Поддержка версий частей в модели данных управления документами.

Создание документного типа элементов

Java:

1. Создаем документный тип элемента (ItemType) с классификацией = `DK_ICM_ITEMTYPE_CLASS_DOC_MODEL`, задаем следующее:

```
docItemTypeDef.setVersionControl  
    ((short)DKConstantICM.DK_ICM_VERSION_CONTROL_ALWAYS);  
docItemTypeDef.setVersioningType(DKConstantICM.DK_ICM_ITEM_VERSIONING_FULL);  
docItemTypeDef.setDeleteRule(DKConstantICM.DK_ICM_DELETE_RULE_CASCADE);
```
2. Создаем отношение ItemType, чтобы добавить части документа. Для каждой части получаем EntityDef.

```
Parttype = (DKItemTypeDefICM) dsDef.retrieveEntity(PartName);
```
3. Для каждой части создает отношение ItemType и задаем значения.

```
DKItemTypeRelationDefICM itRel = new DKItemTypeRelationDefICM(ds);  
itRel.setTargetItemTypeID(PartName);  
itRel.setDefaultRMCode((short)1);  
itRel.setDefaultACLCode(DKConstantICM.DK_ICM_SUPER_USER_ACL);  
itRel.setDefaultCollCode((short)1);  
itRel.setDefaultPrefetchCollCode((short)1);  
itRel.setVersionControl(DK_ICM_VERSION_CONTROL_NERVER);
```
4. Добавляем отношение ItemType в документ (исходный).

```
docItemTypeDef.addItemTypeRelation(itRel);
```
5. Добавляет ItemType в постоянное место хранения.

```
docItemTypeDef.add();
```

C++:

1. Получите объект определения контент-сервера.

```
DKDatastoreDefICM * dsDefICM = (DKDatastoreDefICM *)dsICM->datastoreDef();
```

2. Получите объект управления контент-сервера.

```
DKDatastoreAdminICM * pdsAdmin = (DKDatastoreAdminICM *)dsDefICM->datastoreAdmin();
DKItemTypeDefICM *itemType = NULL;
DKItemTypeRelationDefICM *itemTypeRel = NULL;
DKAttrDefICM* attr = NULL;
```

3. Получите атрибут для этого документного типа элементов. Если атрибут не существует, создайте его.

```
dkAttrDef *pAttr = dsDefICM->retrieveAttr("docTitle1");
if(pAttr == NULL)
{
    attr = new DKAttrDefICM(dsICM);
    attr->setName("docTitle1"); //имя столбца имени атрибута
    attr->setType(DK_CM_CHAR);
    attr->setSize(100);
    attr->setNullable(false);
    attr->setUnique(false);
    attr->add();
    pAttr = attr;
}
itemType = new DKItemTypeDefICM(dsICM);
itemType->setName("DocModelTest");
itemType->setDescription("Это тестовый тип элементов");
itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);
itemType->setAutoLinkEnable(false);
itemType->setVersionControl((short)DK_ICM_VERSION_CONTROL_ALWAYS);
itemType->setVersioningType(DK_ICM_ITEM_VERSIONING_FULL);
itemType->addAttr(pAttr);
```

4. Создайте отношение между только что созданным документом и part1. Для этого сначала получите EntityDef для каждой части.

```
DKItemTypeDefICM *itemTypePart1 = (DKItemTypeDefICM *)
dsDefICM->retrieveEntity("ICMBASE");
//int part1ITypeId = itemTypePart1->getItemTypeId();
int part1ITypeId = itemTypePart1->getIntId();
```

5. Для каждой части создайте отношение типа элементов и задайте значения.

```
DKItemTypeRelationDefICM *itemTypeRelPart1= new DKItemTypeRelationDefICM(dsICM);
itemTypeRelPart1->setTargetItemTypeID(part1ITypeId);
//Задаем менеджер ресурсов по умолчанию
itemTypeRelPart1->setDefaultRMCode((short)1);
//Задаем код ACL по умолчанию
itemTypeRelPart1->setDefaultACLCode(1);
```

6. Задайте собрание по умолчанию, где будут храниться ресурсные элементы этого типа элементов.

```
itemTypeRelPart1->setDefaultCollCode((short)1);
```

7. Задайте собрание предварительной выборки по умолчанию, где будут храниться ресурсные элементы этого типа элементов.

```
itemTypeRelPart1->setDefaultPrefetchCollCode((short)1);
itemTypeRelPart1->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);
itemTypeRelPart1->setSourceItemTypeID(itemType->getIntId());
```

8. Добавьте отношение типа элементов в документ (исходный).

- ```
itemType->addItemTypeRelation(itemTypeRelPart1);
```
9. Создайте отношение между только что созданным документом и part2. Для этого сначала получите EntityDef для каждой части.
 

```
DKItemTypeDefICM *itemTypePart2 = (DKItemTypeDefICM *)
 dsDefICM->retrieveEntity("ICMANNOTATION");
int part2ITypeId = itemTypePart2->getIntId();
```
  10. Для каждой части создайте отношение типа элементов и задайте значения.
 

```
DKItemTypeRelationDefICM * itemTypeRelPart2= new
 DKItemTypeRelationDefICM(dsICM);
itemTypeRelPart2->setTargetItemTypeID(part2ITypeId);
itemTypeRelPart2->setDefaultRMCode((short)1);
itemTypeRelPart2->setDefaultACLCode(1);
itemTypeRelPart2->setDefaultCollCode((short)1);
itemTypeRelPart2->setDefaultPrefetchCollCode((short)1);

itemTypeRelPart2->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);

itemTypeRelPart2->setSourceItemTypeID(itemType->getIntId());
```
  11. Добавьте отношение типа элементов в документ (исходный).
 

```
itemType->addItemTypeRelation(itemTypeRelPart2);
```
  12. Обновите определение этого типа элементов в библиотечном сервере.
 

```
itemType->add();
```

Дополнительную информацию смотрите в примере SItemTypeCreationICM.

## Создание документа

Элемент с семантическим типом "Document" может содержать атрибуты (как и элементы других семантических типов) и несколько "частей" (в отличие от элементов других семантических типов). Ниже описаны шаги процесса создания элемента (основанного на предопределенном типе элементов документ), который содержит один атрибут и одну "часть". Ниже в примере предполагается, что тип элемента под названием "s\_simple" с одним атрибутом под названием "S\_varchar" и одной частью ("ICMBASE") уже определен.

## Java

1. Создайте DDO документа.

```
DKDDO ddoDocument = dsICM.createDDO("S_simple",
 DKConstant.DK_CM_DOCUMENT);
short dataId = 0;
String attrValue = "Test";
```

2. Задайте атрибут документа. В данном случае мы предполагаем, что у типа элементов только один атрибут.

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
DKParts parts = null;
// Части документа
```

3. Вызовите собрание частей документа.

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
 DKConstantICM.DK_CM_DKPARTS);
if (dataId == 0) {
 dataId = ddoDocument.addData(DKConstant.DK_CM_NAMESPACE_ATTR,
 DKConstantICM.DK_CM_DKPARTS);
 parts = new DKParts();
 ddoDocument.setData(dataId, parts);
}
else
{
 parts = (DKParts)ddoDocument.getData(dataId);
 if (parts == null)
 {
 parts = new DKParts();
 ddoDocument.setData(dataId, parts);
 }
}
```

4. Создайте часть предопределенного типа "ICMBASE". Эта часть будет добавлена к создаваемому документу. Предполагается, что документ, создаваемый ниже, основан на типе элементов с единственной частью.

```
DKLobICM pLobPart = (DKLobICM) dsICM.createDDO("ICMBASE",
 DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
pLobPart.setPartNumber(1);
// Задаёт тип mime для добавляемой части
pLobPart.setMimeType("text/plain");
String partValue = "Это основная часть";
pLobPart.setContent(partValue.getBytes());
```

5. Добавьте созданную часть в собрание частей. Обратите внимание на использование отложенного сохранения (изменения на складе данных не будут приняты, пока DDO документа не станет постоянным).

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. Сделайте документ постоянным на складе данных.

```
ddoDocument.add();
```

**C++**

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Создаем новый DDO типа DocModelTest с семантическим типом DK_CM_DOCUMENT
DKDDO* ddoDocument = dsICM->createDDO("DocModelTest",DK_CM_DOCUMENT);
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
 DKString("docTitle1")),DKString("это строчное значение"));
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
 pdsDef->retrieveEntity("DocModelTest");
// Получаем со склада данных собрание объектов DKItemTypeRelationDefICM
// для заданного исходного типа элементов
dkCollection* pRelationColl = itemType->retrieveItemTypeRelations();
dkIterator* pIter = pRelationColl->createIterator();
int noOfPartsToCreate = pRelationColl->cardinality();
DKParts* pPartColl= NULL;
// Создаем собрание частей для объекта, если оно не существует
short dataId = ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
if (dataId == 0) {
 dataId = ddoDocument->addData(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
 pPartColl = new DKParts();
 ddoDocument->setData(dataId,pPartColl);
}
else {
 pPartColl =(DKParts*) (ddoDocument->getData(dataId).value());
 if (pPartColl ==NULL) {
 pPartColl = new DKParts();
 ddoDocument->setData(dataId,pPartColl);
 }
}
int i=0;
DKItemTypeRelationDefICM* itemTypeRelPart =NULL;
DKItemTypeDefICM* pEnt =NULL;
// Двоичный большой объект CV v8
DKLobICM* pPart =NULL;
DKString str = "Проверяем модель документа с двумя частями";
while(pIter->more()) {
 i=i+1;
 itemTypeRelPart=(DKItemTypeRelationDefICM*) pIter->next()->value();
 pEnt = (DKItemTypeDefICM*)
 ((DKDatastoreDefICM*) pdsDef)->retrieveEntity(
 (long)itemTypeRelPart->getTargetItemTypeID());
 pPart =(DKLobICM*) dsICM->createDDO(pEnt->getName(), DK_CM_RESOURCE);
 pPart->setPartNumber(i);
 pPart->setContent(str);

 DKAny any = (dkDataObjectBase*) pPart;
 pPartColl->addElement(any);
}
//Добавляем этот DDO на склад данных
ddoDocument->add();
```

Дополнительную информацию смотрите в примере `SDocModelItemICM`.

## **Изменение документа**

Ниже описаны шаги процесса изменения элемента семантического типа "Document". В этом процессе добавляется новая часть и изменяется значение атрибута.

## Java

1. Измените значение атрибута для этого типа документа.

```
String attrValue = "Новое значение";
short dataId=ddoDocument.dataId
 (DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
```

2. Вызовите собрание частей документа.

```
DKParts parts = null;
// Части документа
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
 DKConstantICM.DK_CM_DKPARTS);
if (dataId == 0) {
 dataId = ddoDocument.addData(DKConstant.DK_CM_NAMESPACE_ATTR,
 DKConstantICM.DK_CM_DKPARTS);
 parts = new DKParts();
 ddoDocument.setData(dataId, parts);
}
else
{
 parts = (DKParts)ddoDocument.getData(dataId);
 if (parts == null)
 {
 parts = new DKParts();
 ddoDocument.setData(dataId, parts);
 }
}
```

3. Создайте данные для новой части.

```
String partValue = "Это комментарий";
```

4. Создайте часть предопределенного типа "ICMANNOTATION". Эта часть будет добавлена к создаваемому документу. Здесь предполагается, что созданный документ основан на типе элементов с единственной частью. После добавлений новой части документ будет содержать две части.

```
DKLobICM pLobPart = (DKLobICM)dsICM.createDDO("ICMANNOTATION",
 DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
pLobPart.setContent(partValue.getBytes());
pLobPart.setPartNumber(2);
```

5. Добавьте созданную часть в собрание частей. Обратите внимание на использование отложенного сохранения (изменения на складе данных не будут приняты, пока DDO документа не станет постоянным).

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. Сделайте измененный документ постоянным на складе данных.

```
ddoDocument.update();
```

**C++**

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Создаем DDO документа по строке PID
DKString pidString =;
//...
DKDDO* ddoDocument = dsICM->createDDO(pidString);
DKPidICM* pPID = (DKPidICM*) ddoDocument->getPidObject();
DKString* pstrItemType = &pPID->getObjectType();
// Получаем из постоянного склада данных определение для типа
// элементов "DocModelTest". Это определение возвращается в виде
// объекта *dkEntityDef, преобразованного в тип DKItemTypeDefICM
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
 pdsDef->retrieveEntity(*pstrItemType);
ddoDocument->setData(ddoDocument->dataId(
 DK_CM_NAMESPACE_ATTR,DKString("docTitle1")),
 DKString("это новое строчное значение"));

DKString updateString = "Это измененная часть";
DKSequentialIterator* pSeqIter = NULL;
DKLobICM* pPart = NULL;
short dataId = ddoDocument->dataId (DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS);
DKParts* pParts = (DKParts*) &ddoDocument->getData(dataId);
if (pParts != NULL) {
 pSeqIter = (DKSequentialIterator*) pParts->createIterator();
}
else return; // завершить работу
pPart = (DKLobICM *) pSeqIter->next();
// Изменяем содержимое существующей части
pPart->setContent(updateString);
// Добавляем новую часть к документу
DKLobICM* pPart1 = (DKLobICM*) dsICM->createDDO
 ("ICMNOTELOG", DK_CM_RESOURCE);
pPart1->setPartNumber(3);
DKString pTempData = "Проверяем модель документа с двумя частями";
pPart1->setContent(pTempData);
DKAny any = (dkDataObjectBase*)(DKDDO*) pPart1;
pParts->addElement(any);
// Изменяем информацию DDO на складе данных.
ddoDocument->update();
delete pSeqIter;
```

Дополнительную информацию смотрите в примере SDocModelItemICM.

## Получение и удаление документа

Чтобы получить документ, вызовите `ddo.retrieve(опция)`. Если задать опцию `DK_ICM_CONTENT_YES`, будут получены список содержания частей, а также сами части. В противном случае будет получен только список содержания частей.

Чтобы удалить документ, вызовите `ddo.del()`. Удаляются документ и присоединенные к нему части. Дополнительную информацию о получении и удалении документов, имеющих части, смотрите в примере API `SDocModelItemICM`.

## Поддержка версий частей в модели данных управления документами

Свойства поддержки версий частей документа определяются свойствами поддержки версий отношений типов элементов со всеми типами частей, выбранными в типе элементов документа. Характеристики поддержки версий частей включают в себя:

- Как и для обычных документов, для частей может использоваться одна из трех моделей поддержки версий: "поддержка версий - всегда", "поддержка версий - никогда" (по умолчанию) или "поддержка версий под управлением программой".
- Если для типа элементов используется правило "поддержка версий - никогда", для его частей также используется правило "поддержка версий - никогда".
- Если для типа элементов `T` задано правило "поддержка версий - всегда", и к нему относится элемент `I`, а вы изменяете его атрибуты или собрание частей (добавляете, удаляете или изменяете какую-то часть), создается новая версия элемента `I`.
- У частей документов (в отличие от самих документов) не может быть максимального числа версий.
- Правила поддержки версий на уровне частей можно получить из объекта отношений типов элементов для нужной части (базовой части, примечания, комментария и т.д.).

Следующие примеры показывают, как получить правила поддержки версий для базовой части типа элементов.

### Java

```
String itemTypeName="book"; //тип элементов примера
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM item =null;
DKDatastoreICM ds = new DKDatastoreICM();
...
item = (DKItemTypeDefICM)ds.datastoreDef.retrieveEntity(itemTypeName;
DKItemTypeRelationDefICM itemRelation =
(DKItemTypeRelationDefICM)item.retrieveItemTypeRelation(IDчасти);
versionControlPolicy = itemTypeRelation.getVersionControl();
```

## C++

```
DKString itemTypeName="book"; //пример типа элементов
//Задаем ID части, для которой нужно получить правила поддержки версий
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM * itemType;
//Получаем объект, соответствующий типу элементов "book"
itemType = (DKItemTypeDefICM *)dsICM->datastoreDef()->
 retrieveEntity(itemTypeName);
//Получаем объект отношения для указанного ID части
DKItemTypeRelationDefICM * itemTypeRelation =
 (DKItemTypeRelationDefICM*)itemType->retrieveItemTypeRelation(partId);
//Получаем правила поддержки версий для указанной части
int versionControlPolicy = itemTypeRelation->getVersionControl();
```

## Работа с транзакциями

Транзакции позволяют системе Content Manager поддерживать согласованность между библиотечным сервером и любым присоединенным менеджером ресурсов. Транзакция - это определенная пользователем обратимая единица работы, состоящая из одного или нескольких вызовов API в одном соединении с библиотечным сервером. Эта последовательность вызовов метода DKDatastoreICM выполняется напрямую или опосредованно через объекты DDO и XDO.

По умолчанию область действия транзакции и объем выполняемых в ней операций соответствует одному вызову метода API (неявная транзакция). Такой тип транзакций рекомендуется; это область действий дает оптимальную производительность. Однако вы можете изменить область действия единицы работы, включив в нее несколько вызовов методов (явная транзакция), хотя использование транзакций такого типа может привести к росту накладных расходов.

Когда транзакция завершается, она либо принимается вся целиком, либо выполняется ее откат. Если транзакция принимается, все изменения на сервере Content Manager, внесенные вызовами API транзакции, становятся постоянными. Если производится откат транзакции или она завершается неудачно, все произведенные за транзакцию изменения в процессе отката обращаются.

Для неявных транзакций принятие и откат выполняются автоматически. Если используются явные транзакции, принятием транзакции управляет прикладная программа, откат же может быть инициирован программой или системой Content Manager (автоматический откат). Система Content Manager инициирует откат транзакции, если произошла серьезная ошибка или если необходимо разрешить тупиковую ситуацию между библиотечным сервером и базой данных.



Внутри транзакции не принятые изменения в менеджере ресурсов не видны прикладной программе, которая производит эти изменения, пока эта транзакция не будет принята. Допустим, вы внесите изменения в элемент менеджера ресурсов и сохраняете его. Если вы получите этот элемент прежде, чем транзакция будет принята, вы не увидите изменений, которые только что внесли. Изменения станут видны вам только после принятия транзакции.

Параллельные или перекрывающиеся транзакции через одно соединение библиотечного сервера не поддерживаются. Чтобы поддерживать параллельные транзакции, надо создать несколько соединений библиотечного сервера с базой данных, или, при работе с программой клиента, запустить несколько клиентских процессов или потоков. Прикладные программы типа IBM WebSphere<sup>®</sup> Application Server управляют процессами, соединениями и сеансами.

Методы `execute()` и `executeWithCallback()` в `DKDatastoreICM` при вызове автоматически создают дополнительное соединение с базой данных. Это новое соединение затем используется для выполнения запроса. Поскольку запросы используют отдельное соединение с базой данных, для их транзакций используется и отдельная (от других операций контент-сервера) область действия. Это соединение с базой данных закрывается (или возвращается в пул, если разрешено объединение соединений в пул), когда закрывается `DKResultSetCursor`.

### **Что надо учитывать при проектировании транзакций в собственных прикладных программах**

Если на узле клиента или на библиотечном сервере до принятия транзакции произошел отказ, функция восстановления базы данных выполнит откат транзакции на библиотечном сервере. Если и узел клиента, и менеджер ресурсов активны, изменения на менеджере ресурсов, внесенные перед отказом, отменяются немедленно. Если отказал узел клиента, для менеджера ресурсов надо выполнить цикл утилиты асинхронного восстановления, чтобы восстановить согласованность менеджера ресурсов и библиотечного сервера. До запуска этой утилиты целостность данных на серверах сохраняется. Затронуты будут только текущие операции над элементами, выполнявшиеся в момент отказа; они будут отклоняться до восстановления менеджера ресурсов. Отклонение текущих операций над объектами предотвращает новые изменения того же объекта до преодоления последствий первого отказа.

Если отказал менеджер ресурсов, надо запустить утилиту асинхронного восстановления для устранения несогласованности. В OS/390 у менеджера ресурсов есть встроенная поддержка транзакций, такая как Object Access Method (OAM), которая упрощает восстановление.

### **Что надо учитывать при использовании явных транзакций**

Для явных транзакций, при которых вы управляете областью действия транзакции при помощи `DKDatastoreICM.startTransaction()` и

DKDatastoreICM.commit(), необходима осторожность при разработке программ, работающих с документами Content Manager с частями и выполняющих операции DKLobiCM создания, получения, изменения и удаления. Выполнение этих операций надо помещать как можно ближе к концу транзакции. Кроме того, транзакции следует делать короче, поскольку длинная транзакция увеличивает риск потенциальных проблем с блокировкой.

Проблемы с блокировкой чаще всего возникают, если элемент изменен, а программа не выполняет принятие транзакции немедленно. Пока транзакция не принята, измененный элемент виден для других программ. Но когда пользователь пытается получить этот элемент или посмотреть его содержимое, он вынужден ждать завершения транзакции изменения. Та же проблема (блокировка базы данных) возникает при создании новых элементов в папке. Если сама папка видна другому пользователю, и он пытается получить новый элемент, ему приходится ждать завершения транзакции. Чем больше времени проходит до момента принятия транзакции, тем дольше должен ждать пользователь.

Лучший подход для избежания блокировок - выполнять принятие транзакций часто и избегать длинных транзакций. Если в транзакции должны выполняться операции создания, получения, изменения или удаления, лучше выполнять их в такой момент, когда к ним не могут обратиться другие.

## **Применение резервирования и активирования в транзакциях**

Content Manager поддерживает операции резервирования и активирования для элементов. Операция резервирования вызывается, чтобы получить постоянную блокировку записи для элементов. Когда элемент резервируется пользователем, другие пользователи не могут изменять его, хотя по-прежнему могут видеть и получать его. Резервирование необходимо перед изменением или переиндексацией элемента, независимо от режима (явная или неявная) используемой транзакции. Когда работа с элементом закончена, вызовите операцию активирования, чтобы снять постоянную блокировку и сделать этот элемент доступным для изменения другими пользователями. После создания элемента можно оставить его в резервированном состоянии, чтобы другие пользователи не изменяли его, пока вы не закончите работу. Если вы выполняете резервирование (или активирование) элемента в явной транзакции, это резервирование отменяется при откате транзакции. Если вы резервируете элемент в неявной транзакции, резервирование принимается. Активирование этого элемента должна выполнить прикладная программа при помощи опций или методов checkin.

## **Обработка транзакций**

Областью действия транзакции можно управлять при помощи вызовов API клиента, однако это надо тщательно продумать. Чтобы сгруппировать набор вызовов API в транзакцию, ее надо построить явно, выполнив следующие шаги:

1. Вызовите метод `startTransaction()` класса `DKDatastoreICM`. Вы работаете с методами класса `DKDatastoreICM` для выполнения каждого шага транзакции.
2. Вызовите все нужные API в том порядке, в котором хотите включить их в транзакцию.
3. Вызовите метод `commit` или `rollback`, чтобы завершить транзакцию.

Все вызовы API между `startTransaction()` и `commit()` или `rollback()` рассматриваются как одна транзакция.

В транзакцию можно включать любые API, если не оговорено обратное. Подробности смотрите в электронном справочнике [Online API Reference](#). Некоторые API управления нельзя включать в явные транзакции. Пример таких API - определение или изменение типов элементов.

Ниже приводится список всех методов классов, участвующих в транзакциях Content Manager для создания и изменения элемента:

**DKDatastoreICM.startTransaction()**

Запускает явную транзакцию.

**DKDatastoreICM.commit()**

Принимает изменения транзакции в базе данных.

**DKDatastoreICM.rollback()**

Выполняет откат, то есть удаление изменение транзакции из базы данных.

**DKDatastoreICM.checkOut()**

Запрашивает постоянную блокировку записи для элемента.

**DKDatastoreICM.checkIn()**

Снимает затребованную ранее постоянную блокировку записи.

**DKDatastoreICM.add()**

Создает новый элемент в базе данных.

**DKDatastoreICM.updateObject()**

Изменяет элемент. Перед вызовом этого метода элемент должен быть зарезервирован.

**DKDatastoreICM.retrieveObject()**

Получает элемент из базы данных.

**DKDatastoreICM.deleteObject()**

Удаляет элемент из базы данных.

**DKDatastoreICM.moveObject()**

Переиндексирует элемент. Перемещает элемент из одного типа элементов в другой. Перед вызовом этого метода элемент должен быть зарезервирован.

Кроме того, много информации о транзакциях есть в примере SItemUpdateICM sample.

---

## Маршрутизация документа по процессу

Система Content Manager предоставляет интегрированную службу маршрутизации документов, которая помогает направлять документы по бизнес-процессу. API маршрутизации документов позволяют создавать новые прикладные программы, использующие маршрутизацию документов, или добавлять маршрутизацию документов в уже существующие программы. Маршрутизацией документов обеспечиваются следующие возможности:

- Синхронизация всех элементов в процессе маршрутизации документов, поскольку функции маршрутизации документов включаются в транзакции Content Manager.
- Пользователю показывается только та работа, к которой он имеет доступ.
- Один файл аудита содержит записи о создании документа, его изменении и маршрутизации.

Основные понятия и термины маршрутизации документов смотрите в *Руководстве администратора системы*. Кроме того, много дополнительной информации о маршрутизации документов есть в примерах.

## Процесс маршрутизации документов

Маршрутизация документов работает с процессами, рабочими узлами, рабочими списками и рабочими пакетами. Рабочие узлы, процессы и рабочие списки создает администратор системы при помощи клиента управления системой. Процесс состоит из рабочих узлов. Каждый рабочий узел в процессе - это отдельный шаг процесса. Можно создавать процессы, ветвящиеся в нескольких направлениях. Пользователь определяет, к узлу какой ветви надо переходить. Пользователь может выбирать из списка возможных значений, который определяет системный администратор. При определении рабочего узла вы можете задать обработчик сервера. Обработчик сервера можно задать для входа на рабочий узел, выхода с рабочего узла и для уведомления пользователя при достижении границы перегрузки. Рабочий пакет создается, когда запускается процесс. Рабочий пакет - это элемент маршрутизации, содержащий атрибуты работы. В атрибуты рабочего пакета входят PID элемента, приоритет, владелец и т.п.

Точки сбора - это рабочие узлы с дополнительной функцией. Рабочий пакет в узле точки сбора передается на следующий рабочий узел в процессе, как только в указанной папке будет достигнуто заданное число элементов заданных типов. Рабочие списки определяют рабочие пакеты, назначенные пользователю. У вас может быть один или несколько рабочих списков. Каждый рабочий список может включать один или несколько рабочих узлов. В рабочем списке можно

задать порядок рабочих пакетов по приоритету или дате. Кроме того, в рабочем списке можно определить порядок рабочих узлов.

При получении рабочих списков можно фильтровать результаты, включая или исключая приостановленные работы. Рабочие пакеты могут также находиться в состоянии уведомления. Состояние *уведомления* означает, что рабочий пакет находится на узле дольше срока, заданного администратором. Запомните, что один рабочий узел может быть в нескольких рабочих списках. Число пакетов, возвращаемых в рабочем списке, определяется системным администратором.

С помощью маршрутизации документов можно выполнять основные операции, например:

- Запустить процесс
- Завершить процесс
- Продолжить процесс
- Приостановить процесс
- Возобновить процесс
- Получить работу из рабочего списка
- Получить следующий элемент из рабочего списка
- Определить, изменить и удалить процесс
- Определить, изменить и удалить рабочий узел
- Определить, изменить и удалить рабочий список

## **Задание процесса маршрутизации документов**

Есть девять API, которые можно использовать для маршрутизации документов в прикладной программе. Подробности об этих API и методах можно найти в *электронном справочнике API*. Девять API маршрутизации документов:

### **DKDocRoutingServiceICM**

Этот класс содержит методы для управления процессом: start, terminate, continue, suspend и resume (запустить, прервать, продолжить, приостановить и возобновить соответственно).

### **DKDocRoutingServiceMgmtICM**

Этот класс содержит методы для управления классами поддержки: DKProcessICM, DKWorkNodeICM и DKWorkListICM. Вы можете обращаться к объекту DKDocRoutingServiceMgmtICM из объекта DKDocRoutingServiceICM.

### **DKProcessICM**

Этот класс представляет на библиотечном сервере процесс.

### **DKWorkNodeICM**

Этот класс представляет на библиотечном сервере рабочий узел.

### **DKWorkListICM**

Этот класс представляет на библиотечном сервере рабочий список.

### **DKRouteListEntryICM**

Этот класс определяет маршрут, по которому может пойти процесс (исходный узел и узел назначения). Объект процесса (то есть DKProcessICM) содержит собрание объектов записей маршрута (то есть DKRouteListEntryICM).

### **DKCollectionResumeListEntryICM**

Этот класс представляет для рабочих узлов запись в списке возобновления. Рабочий узел может содержать собрание объектов DKCollectionResumeListEntryICM.

### **DKWorkPackageICM**

Этот класс представляет на библиотечном сервере рабочий пакет. Когда запускается процесс, создается рабочий пакет.

### **DKResumeListEntryICM**

Этот класс представляет список возобновления. Рабочий пакет может содержать собрание списков возобновления.

## **Как создать документ с объектами службы маршрутизации**

Ниже в примерах показано, как создать объект маршрутизации документа.

#### **Java**

```
//Объект DKDocRoutingServiceMgmtICM - это вспомогательный класс с методами
// для управления DKProcessICM, DKWorkNodeICM и DKWorkListICM
// и метаданными, требуемыми для определения этих объектов
DKDocRoutingServiceMgmtICM routingMgmt = new
 DKDocRoutingServiceMgmtICM(dsICM);

//Класс DKDocRoutingServiceICM поддерживает основные службы маршрутизации,
//такие как запуск, прекращение, продолжение, приостановка и возобновление
//процесса.

DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);
```

## C++

```
// для управления DKProcessICM, DKWorkNodeICM и DKWorkListICM
// и метаданными, требуемыми для определения этих объектов
DKDocRoutingServiceMgmtICM* routingMgmt = new
 DKDocRoutingServiceMgmtICM(dsICM);

//Класс DKDocRoutingServiceICM поддерживает основные службы маршрутизации,
//такие как запуск, прекращение, продолжение, приостановка и возобновление
//процесса.
DKDocRoutingServiceICM* routingService = new
 DKDocRoutingServiceICM(dsICM);
```

Полный пример смотрите в примере SDocRoutingDefinitionCreationICM.

## Как определить новый обычный рабочий узел

Рабочий узел представляет собой шаг в определении процесса маршрутизации документа. Выход с узла может быть выполнен пользовательской программой в любое время. Пользовательская программа отвечает за проверку своего собственного критерия выхода.

## Java

```
// Создаем новый объект рабочего узла.
DKWorkNodeICM workNode1 = new DKWorkNodeICM();
//Выбираем имя не длиннее 15 символов
workNode1.setName("S_fillClaim");
//Задаем содержательное описание
workNode1.setDescription("Клиент заполняет страховую иск");
// Задаем максимальное время, которое элемент может находиться на этом
// рабочем узле (в минутах).
workNode1.setTimeLimit(100);
// Задаем максимальное число рабочих пакетов, которые могут находиться
// на этом узле одновременно
workNode1.setOverloadLimit(200);

// Задаем тип рабочего узла - обычный
workNode1.setType(0);

//Добавляем определение нового рабочего узла в объект
//управления маршрутизацией документа
routingMgmt.add(workNode1);
```

**C++**

```
// Создаем новый объект рабочего узла.
DKWorkNodeICM* workNode1 = new DKWorkNodeICM();
//Выбираем имя не длиннее 15 символов
workNode1->setName("ValidateCreditCard");
// Задаем содержательное описание
workNode1->setDescription("Кредитная карта покупателя проверяется
 соответствующей организацией");

//Задаем максимальное время в минутах,
//в течение которого элемент может оставаться на этом рабочем узле.
workNode1->setTimeLimit(100);
// Задаем максимальное число рабочих пакетов, которые могут находиться
// на этом узле одновременно
workNode1->setOverloadLimit(200);

// Задаем тип рабочего узла - обычный
workNode1->setType(0);

//Добавляем определение нового рабочего узла в объект
//управления маршрутизацией
routingMgmt->add(workNode1);

//Освобождаем память. Этот объект больше не нужен.
delete(workNode1);
```

Полный пример создания рабочих узлов есть в примере  
SDocRoutingDefinitionCreationICM.

### **Как получить список рабочих узлов**

Метод `listWorkNodeNames` возвращает список имен всех рабочих узлов на библиотечном сервере, а метод `listWorkNodes` возвращает собрание объектов `DKWorkNodeICM`, представляющих рабочие узлы на библиотечном сервере.



## Java

```
// Получаем объект управления маршрутизацией документов.
// Получаем объект управления маршрутизацией.
DKDocRoutingServiceMgmtICM routingMgmt = new
 DKDocRoutingServiceMgmtICM(dsICM);

// Получаем все рабочие узлы в системе
dkCollection workNodes = routingMgmt.listWorkNodes();
System.out.println("Work Nodes in System: (" +workNodes.cardinality()+")");
dkIterator iter = workNodes.createIterator();
while (iter.more ())
{
 DKWorkNodeICM workNode = (DKWorkNodeICM) iter.next();
 if(workNode.getType()==0)
 System.out.println(" Normal Node - " +workNode.getName()+":
 "+workNode.getDescription());
 else
 System.out.println(" Collection Pt - " +workNode.getName()+":
 "+workNode.getDescription());
}
```

## C++

```
// Получаем объект управления маршрутизацией документов.
DKDocRoutingServiceMgmtICM* routingMgmt = new
 DKDocRoutingServiceMgmtICM(dsICM);

// Получаем собрание, содержащее все рабочие узлы в системе.
dkCollection* workNodes = routingMgmt->listWorkNodes();
if (workNodes && (workNodes->cardinality(>0)))
{
 cout << "Число рабочих узлов в системе: ("
 << workNodes->cardinality() << ")" << endl;
 dkIterator* iter = workNodes->createIterator();
 while(iter->more())
 {
 DKWorkNodeICM* workNode = (DKWorkNodeICM*) iter->next()->value();
 if(workNode->getType()==0)
 {
 cout << " Обычный узел - " << workNode->getName() << ": " <<
 workNode->getDescription() << endl;
 }
 else
 {
 cout << " Точка сбора - " << workNode->getName() << ": " <<
 workNode->getDescription() << endl;
 }
 delete(workNode);
 }
 delete(iter);
 delete(workNodes);
}
delete(routingMgmt);
```

Дополнительную информацию о выводе списка рабочих узлов смотрите в примере SDocRoutingListingICM.

### Как задать новую точку сбора

Точка сбора - это рабочий узел с критерием выхода, заданным системой; такой узел специально предназначен для маршрутизации папок. Можно задать набор требований, которые должны быть выполнены до того, как процесс сможет продолжаться дальше этой точки.

## Java

```
// Создаем новый объект рабочего узла. Это будет
//точка сбора
DKWorkNodeICM collectionPoint = new DKWorkNodeICM();

// Выберите имя не длиннее 15 символов.
collectionPoint.setName("S_gatherAll");

// Задаем содержательное описание
collectionPoint.setDescription("Gather Claim,Police Report,Policy,& Photos");

// Задаем максимальное время, которое элемент может находиться на этом
// рабочем узле (в минутах).
collectionPoint.setTimeLimit(100);

// Задаем максимальное число рабочих пакетов, которые могут
// находиться на этом узле.
collectionPoint.setOverloadLimit(200);
// Задаем тип узла - точка сбора.
collectionPoint.setType(1);

// Создаем список продолжения - то есть список типов документов,
//которых должен ждать, прежде чем перейти на следующий узел.
//Будет создан список "записей продолжения" - описаний требований,
//которые должны быть выполнены, прежде чем процесс будет продолжен.
// Создаем список - собрание для записей продолжения.
dkCollection resumeList = new DKSequentialCollection();
// Создаем требования, то есть "записи списка продолжения", задающие,
//элементы какого типа мы должны ждать. Процесс не может уйти с
//точки сбора, пока заданное число элементов
//(DDO) заданного типа не поступит в эту точку сбора.
DKCollectionResumeListEntryICM resumeRequirement = new
 DKCollectionResumeListEntryICM();
// Задаем элемент - папку с именами типов элементов.
resumeRequirement.setFolderItemTypeName("S_simple");
// Собрание должно ждать добавления элемента заданного типа
//в папку, прежде чем обработка будет продолжена.
resumeRequirement.setRequiredItemTypeName("S_autoClaim");
// Задаем число элементов заданного типа, которые
//мы должны ждать.
resumeRequirement.setQuantityNeeded(1);
// Добавляем требование (запись) в список требований (список продолжения)
resumeList.addElement(resumeRequirement);
resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement.setFolderItemTypeName("S_simple");
resumeRequirement.setRequiredItemTypeName("S_policeReport");
resumeRequirement.setQuantityNeeded(1);
// Когда все требования (записи списка продолжения) добавлены в
//список требований (список продолжения), задаем этот список продолжения для
//точки сбора.
collectionPoint.setCollectionResumeList(resumeList);
//продолжение следует . . .
```

### Java (продолжение)

```
// Добавляем определение новой точки сбора к объекту управления
// маршрутизацией документов
routingMgmt.add(collectionPoint);
```

### C++

```
// Создаем новый объект рабочего узла. Это будет точка сбора
DKWorkNodeICM* collectionPoint = new DKWorkNodeICM();
// Выберите имя не длиннее 15 символов.
collectionPoint->setName("GatherOrderDetails");
// Задаем содержательное описание
collectionPoint->setDescription("Собираем всю информацию о
 заказе, способе доставки и адресе доставки");
// Задаем максимальное время, которое элемент может находиться на этом
// рабочем узле (в минутах).
collectionPoint->setTimeLimit(100);
// Задаем максимальное число рабочих пакетов, которые могут находиться
// на этом узле.
collectionPoint->setOverloadLimit(200);

// Задаем тип узла - точка сбора.
collectionPoint->setType(1);
// Создаем список продолжения - то есть список типов документов,
// которых должен ждать, прежде чем перейти на следующий узел.
// Будет создан список "записей продолжения" - описаний требований,
// которые должны быть удовлетворены для продолжения процесса.

// Создаем список / собрание для записей возобновления.
dkCollection* resumeList = new DKSequentialCollection();

// Создаем нужное число требований ("записей списка продолжения"),
// указывающих типы элементов, которые нужно ждать. Процесс не сможет
// пройти эту точку сбора, пока не будет собрано указанное число
// элементов (DDO) указанных типов элементов.
DKCollectionResumeListEntryICM* resumeRequirement = new
 DKCollectionResumeListEntryICM();
// Задаем элемент - папку с именами типов элементов.
resumeRequirement->setFolderItemTypeName("book");

// Задаем в собрании, что перед продолжением обработки папки нужно
// дождаться добавления в нее элемента заданного типа.
resumeRequirement->setRequiredItemTypeName("AnItemType");

// Задаем число элементов указанного типа, добавления
// которых нужно ждать.
resumeRequirement->setQuantityNeeded(1);

//продолжение следует . . .
```

### С++ (продолжение)

```
// Добавляем требование (запись) в список требований (список продолжения)
resumeList->addElement(resumeRequirement);
resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement->setFolderItemTypeName("book");
resumeRequirement->setRequiredItemTypeName("AnotherItemType");
resumeRequirement->setQuantityNeeded(1);
resumeList->addElement(resumeRequirement);

// Когда все требования (записи списка продолжения) добавлены
// в список требований (список возобновления), задаем этот список
// точки сбора.
collectionPoint->setCollectionResumeList(resumeList);
// Добавляем определение новой точки сбора к объекту управления
// маршрутизацией документов
routingMgmt->add(collectionPoint);

// Освобождаем память, связанную с этой точкой сбора.
delete(collectionPoint);
```

Дополнительную информацию об определении точек сбора смотрите в примере SDocRoutingDefinitionCreationICM.

### Как задать рабочий список

Рабочий список - это один или несколько рабочих узлов, из которых пользователь получает список рабочих пакетов или "следующий" рабочий пакет. Один рабочий узел может входить в несколько рабочих списков. Использование рабочих списков позволяет администратору или пользовательской прикладной программе динамически переназначать работу, не связываясь с пользователями.

## Java

```
// Создаем новый рабочий список
DKWorkListICM workList = new DKWorkListICM();
// Выберите имя не длиннее 15 символов.
workList.setName("S_fillClaimWL");
workList.setDescription("Рабочий список узла заполнения/принятия исков.");
//Задаем, что рабочие пакеты возвращаются рабочим списком
//отсортированными по времени
workList.setSelectionOrder(DKConstantICM.DK_ICM_DR_SELECTION_ORDER_TIME);
//Задаем, что будут возвращаться рабочие пакеты, которые не находятся
// в состоянии приостановки
workList.setSelectionFilterOnSuspend
 (DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

//Задаем, что будут возвращаться рабочие пакеты, которые не находятся
//в состоянии уведомления
workList.setSelectionFilterOnNotify
 (DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

// Задаем, что в рабочем списке надо указывать
// до 100 рабочих пакетов
workList.setMaxResult(100);
String[] wnNames = {"S_fillClaim"};
workList.setWorkNodeNames(wnNames);

//Добавляем определение нового рабочего списка в объект
//управления маршрутизацией
routingMgmt.add(workList);
```

## C++

```
// Создаем новый рабочий список
DKWorkListICM* workList = new DKWorkListICM();
// Выберите имя не длиннее 15 символов.
workList->setName("ValidateWorkList");
workList->setDescription("Рабочий список, включающий рабочий узел
 проверки кредитных карт.");

// Задаем, что рабочие пакеты, возвращаемые этим рабочим потоком,
// будут отсортированы по времени
workList->setSelectionOrder(DK_ICM_DR_SELECTION_ORDER_TIME);

//Задаем, что будут возвращаться рабочие пакеты, которые не находятся
// в состоянии приостановки
workList->setSelectionFilterOnSuspend(DK_ICM_DR_SELECTION_FILTER_NO);

//Задаем, что будут возвращаться рабочие пакеты, которые не находятся
// в состоянии уведомления
workList->setSelectionFilterOnNotify(DK_ICM_DR_SELECTION_FILTER_NO);

// Задаем, что этот рабочий список может содержать не более 100 рабочих
// пакетов
workList->setMaxResult(100);

DKString* wnNames = new DKString[1];
wnNames[0] = "ValidateCreditCard";
// Добавляем рабочий узел в рабочий список
workList->setWorkNodeNames(wnNames,1);

// Добавляем это новое определение рабочего списка к объекту
// управления маршрутизацией документов
routingMgmt->add(workList);

// Освобождаем память, связанную с этим рабочим списком.
delete(workList);
```

Дополнительную информацию об определении рабочих списков смотрите в примере SDocRoutingDefinitionCreationICM.

### Получение рабочих списков

Метод `listWorkListNames` возвращает список имен всех рабочих списков на библиотечном сервере, а метод `listWorkLists` возвращает собрание объектов `DKWorkListICM`, представляющих рабочие списки на библиотечном сервере.

## Java

```
// Получаем объект управления маршрутизацией документов.
// Получаем объект управления маршрутизацией.
DKDocRoutingServiceMgmtICM routingMgmt =
 new DKDocRoutingServiceMgmtICM(dsICM);
// Получаем все рабочие списки в системе
dkCollection workLists = routingMgmt.listWorkLists();
System.out.println("Work Lists in System: (" +workLists.cardinality()+")");
dkIterator iter = workLists.createIterator();
while (iter.more ())
{
 DKWorkListICM workList = (DKWorkListICM) iter.next();
 System.out.println(" - "+workList.getName()+":
 "+workList.getDescription());
}
```

## C++

```
dkCollection* workLists = routingMgmt->listWorkLists(); // Получаем все
// рабочие списки в системе

if (workLists && (workLists->cardinality())>0))
{
 cout<<"Work Lists in System: ("<<workLists->cardinality()<<")"<<endl;
 dkIterator* iter = workLists->createIterator();
 while(iter->more()){
 DKWorkListICM* workList = (DKWorkListICM*) iter->next()->value();
 cout << " - " << workList->getName() << ": "
 << workList->getDescription() << endl;
 delete(workList); // Освобождаем память
 }
 delete(iter); // Освобождаем память
 delete(workLists);
}
```

Полный пример смотрите в примере SDocRoutingListingICM.

### Как задать новый процесс и связанный с ним маршрут

Процесс маршрутизации документов определяет маршрут, по которому направляется рабочий пакет. Несколько процессов маршрутизации могут использовать одни и те же узлы, а между одними и теми же узлами можно использовать различные маршруты.



## Java

```
// Создаем новое определение процесса
DKProcessICM process = new DKProcessICM();
process.setName("S_claimProcess");
process.setDescription("Процесс для страхового иска");

// Определяем все возможные маршруты.

// Создаем список всех возможных маршрутов между узлами.
dkCollection routes = new DKSequentialCollection();

// Соединяем рабочие узлы при помощи записей списка маршрутов. Простой
// маршрут между узлами задается при указании рабочего узла 'From'
// и рабочего узла 'To'.
// Запись списка маршрутов просто соединяет два узла с указанием направления.
// Между узлами могут существовать несколько маршрутов. Определенный маршрут
// можно пометить заданным пользователем словом, например, "Продолжение",
// "Переход", "Принимаем", "Отклоняем", "Готово" и т.п.
// Создаем новое соединение между двумя узлами
DKRouteListEntryICM nodeRoute = new DKRouteListEntryICM();

// Каждый процесс должен начинаться с начального узла.
nodeRoute.setFrom(DKConstantICM.DK_ICM_DR_START_NODE);
nodeRoute.setTo("S_fillClaim");
// Задайте любое слово для действия, которое вызывает
//переход с первого узла на второй
nodeRoute.setSelection("Продолжение");

// Добавим один маршрут в собрание всех возможных маршрутов.
routes.addElement(nodeRoute);
nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_fillClaim");
nodeRoute.setTo("S_gatherAll");
// Задайте любое слово для действия, которое вызывает
// переход.
nodeRoute.setSelection("Продолжение");

// Добавим один маршрут в собрание всех возможных маршрутов.
routes.addElement(nodeRoute);
nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_gatherAll");
nodeRoute.setTo(DKConstantICM.DK_ICM_DR_END_NODE);

// Задайте любое слово для действия, которое вызывает
//переход.
nodeRoute.setSelection("Готово");
// Добавим один маршрут в собрание всех возможных маршрутов.
routes.addElement(nodeRoute);
// Задаем маршрут для процесса.
process.setRoute(routes);
// Добавляем процесс в объект управления маршрутизацией.
routingMgmt.add(process);
```



## C++

```
// Процесс маршрутизации документов определяет маршруты, по которым будут
// следовать рабочие пакеты. Одни и те же узлы могут входить в несколько
// процессов маршрутизации, а между двумя узлами может быть несколько различных
// маршрутов.

// Создаем новое определение процесса
DKProcessICM* process = new DKProcessICM();
process->setName("Buy_Book");
process->setDescription("Покупка книги через Интернет");
// Определяем все возможные маршруты.

// Создаем список всех возможных маршрутов между узлами.
dkCollection* routes = new DKSequentialCollection();

// Соединяем рабочие узлы при помощи записей списка маршрутов. Простой
// маршрут. Для задания простого маршрута нужно связать рабочий узел
// 'From' (Откуда) и рабочий узел 'To' (Куда).
// Запись списка маршрутов просто соединяет два узла с указанием
// направления. Между узлами могут существовать несколько маршрутов.
// Определенный маршрут можно пометить заданным пользователем словом,
// например, "Продолжение", "Переход", "Принимаем", "Отклоняем",
// "Готово" и т.п.

// Создаем новое соединение между двумя узлами
DKRouteListEntryICM* nodeRoute = new DKRouteListEntryICM();

// Каждый процесс должен начинаться с начального узла.
nodeRoute->setFrom(DK_ICM_DR_START_NODE);
nodeRoute->setTo("ValidateCreditCard");
// Задаем какое-либо пользовательское имя для действия, вызывающего
// переход от первого узла ко второму
nodeRoute->setSelection("Continue");

// Добавим один маршрут в собрание всех возможных маршрутов.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("ValidateCreditCard");
nodeRoute->setTo("GatherShippingDetails");

// Задаем какое-либо пользовательское имя для действия, вызывающего
// переход к другому узлу.
nodeRoute->setSelection("Continue");

// Добавим один маршрут в собрание всех возможных маршрутов.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("GatherOrderDetails");
nodeRoute->setTo(DK_ICM_DR_END_NODE);
//продолжение следует . . .
```

### **C++ (продолжение)**

```
// Задаем какое-либо пользовательское имя для действия, вызывающего
// переход к другому узлу.
nodeRoute->setSelection("Complete");

// Добавим один маршрут в собрание всех возможных маршрутов.
routes->addElement(nodeRoute);

// Задаем маршрут для процесса.
process->setRoute(routes);

// Добавляем процесс в объект управления маршрутизацией.
routingMgmt->add(process);

delete(process);
```

Полный пример смотрите в примере SDocRoutingDefinitionCreationICM.

### **Запуск процесса маршрутизации документов**

Ниже в примере показано, как запустить процесс маршрутизации документа.

### **Java**

```
//Сначала создадим документ или папку, для которой будет выполняться
//маршрутизация. Тип элементов под именем "s_simple" должен быть определен
//до того, как можно будет создать этот DDO.

DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);

// Сохраняем созданную папку на постоянном складе данных
ddoFolder.add();

//Создаем основной объект службы маршрутизации документов.
DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);

//Запускаем процесс с именем "S_claimProcess" (он должен быть уже определен).

//Этот вызов возвращает строку PID рабочего пакета, для которого
//выполняется маршрутизация.
String workPackagePidStr = routingService.startProcess
 ("S_claimProcess", ddoFolder.getPidObject().pidString(),1,"icmadmin");
```

## C++

```
//Сначала создадим документ или папку, для которой будет выполняться
// маршрутизация. Тип элементов с именем "book" должен быть
// определен заранее, до создания DDO этого типа.
DKDDO* ddoFolder = dsICM->createDDO("book", DK_CM_FOLDER);

// Сохраняем созданную папку на постоянном складе данных
ddoFolder->add();

// Задаем приоритет для этого процесса маршрутизации документов.
int Priority = 1;

//Создаем основной объект службы маршрутизации документов.
DKDocRoutingServiceICM* routingService = new DKDocRoutingServiceICM(dsICM);

// Запускаем процесс с именем "Buy_Book" (он должен быть определен заранее).
//Этот вызов возвращает строку PID рабочего пакета, для которого
//выполняется маршрутизация.
DKString workPackagePidStr = routingService->startProcess("Buy_Book",
 ((DKPidICM*) ddoFolder->getPidObject())->pidString(), Priority,
 "icmadmin");
```

Полный пример смотрите в примере SDocRoutingProcessingICM.

## Завершение процесса

Процесс можно явно прервать до того, как он достигнет конечного узла. Если вы прервали процесс, соответствующий рабочий пакет удаляется из системы. Чтобы прервать процесс, надо знать строку PID рабочего пакета, который маршрутизируется этим экземпляром процесса.

## Java

```
routingService.terminateProcess(workPackagePidStr);
```

## C++

```
routingService->terminateProcess(workPackagePidStr);
```

Дополнительную информацию о прерывании процесса смотрите в примере SDocRoutingProcessingICM.

## Продолжение процесса

Метод continueProcess() направляет элемент, на который ссылается PID элемента в указанном рабочем пакете, от текущего рабочего узла к следующему узлу,

определяемому выбором. Указанный рабочий пакет удаляется с библиотечного сервера, и создается новый рабочий пакет для указанного владельца. Если элемент, на который ссылается этот PID элемента, был зарезервирован, он активируется. Возвращается PID нового рабочего пакета. Если процесс завершился, возвращается пустое значение.

Ниже в фрагменте кода имя выбора для перехода от текущего рабочего узла к следующему - "Continue". Обратите внимание на то, что строка PID текущего рабочего пакета задается в вызове метода. Вызов метода возвращает строку PID нового рабочего пакета

#### Java

```
workPackagePidStr = routingService.continueProcess
 (workPackagePidStr, "Продолжение", "icmadmin");
```

#### C++

```
char * userName = "icmadmin";

workPackagePidStr = routingService->continueProcess(workPackagePidStr,
 "Continue", userName);
```

Полный пример смотрите в примере SDocRoutingProcessingICM.

### Приостановка процесса

Процесс маршрутизации экземпляра документа можно приостановить на заданное время (в минутах) или до выполнения определенного условия. Это никак не связано с процессами и потоками в программной среде. Поток или процесс в среде выполнения C++ при это не приостанавливается.

#### Java

```
dkCollection requirements = new DKSequentialCollection();
//Процесс будет приостановлен на 2 минуты.
routingService.suspendProcess(workPackagePidStr, 2, requirements);
```

**C++**

```
dkCollection * requirements = new DKSequentialCollection();
// Если требования не заданы и процесс нужно приостановить только
// на определенный период времени, можно передать этому методу
// пустое собрание (NULL).
//dkCollection * requirements = NULL;

//Процесс будет приостановлен на 2 минуты.
routingService->suspendProcess(workPackagePidStr, 2, requirements);
delete(requirements);
```

Полный пример смотрите в примере SDocRoutingProcessingICM.

### Возобновление процесса

Приостановленный процесс (то есть процесс, находящийся в состоянии приостановки) может быть возобновлен явно или неявно, по истечении заданного времени или при выполнении заданных условий. При этом процесс выводится из состояния приостановки и продолжает нормальную работу. Метод `resumeProcess` сбрасывает флаг приостановки заданного рабочего пакета до истечения заданного времени и до выполнения условий из списка возобновления. Перенаправление или резервирование рабочего элемента при этом не выполняется.

**Java**

```
routingService.resumeProcess(workPackagePidStr);
```

**C++**

```
routingService->resumeProcess(workPackagePidStr);
```

Дополнительную информацию о возобновлении процесса смотрите в примере SDocRoutingProcessingICM.

### Как в рабочем списке задать список постоянный строки идентификации рабочего пакета

В следующем примере кода показано, как получить список строк PID для всех рабочих пакетов в рабочем списке.

### Java

```
String[] workPackagePIDs =
 routingService.listWorkPackagePidStrings(workListName,processOwner);

// Печатаем PID рабочих пакетов.
System.out.println("Work Packages in Work List: (+workPackagePIDs.length+));
for(int i=0; i< workPackagePIDs.length; i++)
 System.out.println(" - PID: +workPackagePIDs[i]);
```

### C++

```
long arraySize = -1; // Размер задается API.
DKString* workPackagePIDs = routingService->
 listWorkPackagePidStrings("workListName",processOwner,arraySize);

// Печатаем PID рабочих пакетов.
cout << "Число рабочих пакетов в списке: (" << arraySize << ")" << endl;
for(int i=0; i< arraySize; i++)
 cout << " - PID: " << workPackagePIDs[i] << endl;

delete[] workPackagePIDs; // Освобождаем память
```

Полный пример смотрите в примере SDocRoutingListingICM.

### Как получить информацию о рабочем пакете

Когда экземпляр процесса маршрутизации документа выполняется, рабочий пакет как бы несет элемент (экземпляр типа элементов) по процессу маршрутизации. Рабочий пакет содержит всю необходимую информацию о процессе и элементе, который он перемещает. Прикладная операция выполняет необходимые операции именно над рабочим пакетом.

Метод `retrieveWorkPackage` возвращает объект `DKWorkPackageICM`, на который ссылается PID заданного рабочего пакета (`wpPidStringStr`).



## Java

```
//Используем заданную службу маршрутизации документов
//Указание false в вызове метода означает, что рабочий пакет
// не резервируется
DKWorkPackageICM workPackage =
 routingService.retrieveWorkPackage(workPackagePidStr,false);
System.out.println("-----");
System.out.println(" Рабочий пакет");
System.out.println("-----");
System.out.println(" Имя процесса: " + workPackage.getProcessName());
System.out.println(" имя рабочего узла: " + workPackage.getWorkNodeName());
System.out.println(" Владелец: " + workPackage.getOwner());
System.out.println(" Приоритет: " + workPackage.getPriority());
System.out.println(" Автор перемещения: " + workPackage.getUserLastMoved());
System.out.println(" Время перемещения: " + workPackage.getTimeLastMoved());
System.out.println(" Приостановлено: " + workPackage.getSuspendState());
System.out.println(" Уведомлено: " + workPackage.getNotifyState());
System.out.println(" Время уведомления: " + workPackage.getNotifyTime());
System.out.println("Время возобновления: " + workPackage.getResumeTime());
System.out.println("PID рабочего пакета: " + workPackage.getPidString());
System.out.println(" PID элемента: " + workPackage.getItemPidString());
```

## C++

```
cout << "-----" << endl;
cout << " Рабочий пакет" << endl;
cout << "-----" << endl;
cout << " Имя процесса: " << workPackage->getProcessName() << endl;
cout << " Имя рабочего узла: " << workPackage->getWorkNodeName() << endl;
cout << " Владелец: " << workPackage->getOwner() << endl;
cout << " Приоритет: " << workPackage->getPriority() << endl;
cout << " Автор перемещения: " << workPackage->getUserLastMoved() << endl;
cout << " Время перемещения: " << workPackage->getTimeLastMoved() << endl;
cout << " Приостановлено: " << workPackage->getSuspendState() << endl;
cout << " Уведомлено: " << workPackage->getNotifyState() << endl;
cout << " Время уведомления: " << workPackage->getNotifyTime() << endl;
cout << " Время возобновления: " << workPackage->getResumeTime() << endl;
cout << " PID рабочего пакета: " << workPackage->getPidString() << endl;
cout << " PID элемента: " << workPackage->getItemPidString() << endl;
```

Полный пример смотрите в примере SDocRoutingProcessingICM.

### Как получить список процессов маршрутизации документа

Ниже в примере показано, как получить список процессов маршрутизации документов.

## Java

```
//Метод listProcessNames возвращает список имен всех процессов
//на библиотечном сервере, а метод listProcesses возвращает собрание
//объектов DKProcessICM, представляющих процесс на библиотечном сервере.
// Получаем объект управления маршрутизацией документов.
DKDocRoutingServiceMgmtICM routingMgmt =
 new DKDocRoutingServiceMgmtICM(dsICM);
// Получаем список всех процессов маршрутизации документов
// Получаем объект управления маршрутизацией.
DKDocRoutingServiceMgmtICM routingMgmt =
 new DKDocRoutingServiceMgmtICM(dsICM);
// Получаем список всех работающих процессов маршрутизации
dkCollection processes = routingMgmt.listProcesses();
System.out.println("Работающие процессы: (" + processes.cardinality() + ")");

dkIterator iter = processes.createIterator();
while (iter.more ())
{
 // Передвигаем указатель на следующий элемент и получаем этот элемент.
 DKProcessICM proc = (DKProcessICM) iter.next();
 System.out.println(" - " + proc.getName() + ": " + proc.getDescription());
}
```

## C++

```
// Получаем объект управления маршрутизацией документов.
DKDocRoutingServiceMgmtICM* routingMgmt =
 new DKDocRoutingServiceMgmtICM(dsICM);

// Получаем список всех процессов маршрутизации документов
dkCollection* processes = routingMgmt->listProcesses();
if (processes && (processes->cardinality())>0)
{
 cout << "Число выполняемых процессов: (" << processes->cardinality()
 << ")" << endl;
 dkIterator* iter = processes->createIterator();
 while(iter->more())
 {
 DKProcessICM* proc = (DKProcessICM*) iter->next()->value();
 cout << " - " << proc->getName() << ": " << proc->getDescription() <<
 endl;
 delete(proc);
 }
 delete(iter);
 delete(processes);
}
delete(routingMgmt);
```

Функция печати есть в примере SDocRoutingListingICM.

## Произвольная маршрутизация

Ниже приведен пример процедуры произвольной маршрутизации. В этом примере для задания рабочих узлов, процессов и рабочих списков используется клиент администратора системы.

1. Создадим рабочие узлы, например, N1 и N2.
2. Создадим два одноузловых процесса, P1 и P2, причем, к P1 будет относиться рабочий узел N1, а к P2 - рабочий узел N2.

### P1 выглядит так:

| From: | Action:  | To: |
|-------|----------|-----|
| START | Continue | N1  |
| N1    | Continue | END |

### P2 выглядит так:

| From: | Action:  | To: |
|-------|----------|-----|
| START | Continue | N2  |
| N2    | Continue | END |

3. Создайте два рабочих списка, WL1 и WL2, таких что, WL1 содержит один рабочий узел N1, а WL2 содержит один рабочий узел N2.

Чтобы применить промежуточную маршрутизацию, выполняем во время работы следующие действия:

1. Запускаем процесс P1 с PID документа (например, ABC). Создается рабочий пакет WP1. В рабочем списке WL1 этот рабочий пакет WP1 будет показан на рабочем узле N1.
2. Чтобы переместить ABC документа из процесса P1 в процесс P2, прерываем рабочий пакет WP1 и запускаем процесс P2 с тем же документом (ABC). Создается рабочий пакет WP2.

В рабочем списке WL2 рабочий пакет WP2 будет показан на рабочем узле N2.

Дополнительные примеры смотрите в примере SDocRoutingDefinitionCreationICM.

## Примеры запросов маршрутизации документов

Этот раздел содержит примеры запросов. Дополнительную информацию о написании запросов смотрите в разделе “Язык запросов” на стр. 227.

### Пример 1

Находит документы по автомобилям и возвращает только активные рабочие пакеты, связанные с ними.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY[@REFERENCER => WORKPACKAGE[@SUSPENDFLAG =
0]
```

### Пример 2

Находит документы об автомобилях и возвращает те связанные с ними рабочие пакеты, которые входят в процесс "AccidentInvestigation".

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]
```

### Пример 3

Находит документы об автомобилях марки "Honda" и возвращает те связанные с ними рабочие пакеты, которые входят в процесс "AccidentInvestigation".

```
/Car[@Name = "Honda" AND @VERSIONID = latest-version(.) AND
@SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER =>
WORKPACKAGE[@PROCESSITEMID = /ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]
```

### Пример 4

Находит документы об автомобилях и возвращает те связанные с ними рабочие страницы, которые входят в шаг "UnderReview" процесса "AccidentInvestigation".

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]/@ITEMID
AND ../@WORKNODENAME = "UnderReview"]
```

### Пример 5

Находит документы об автомобилях и возвращает только связанные с ними приостановленные рабочие страницы, которые входят в шаг "UnderReview" процесса "AccidentInvestigation".

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]
/@ITEMID AND ../@WORKNODENAME = "UnderReview" AND
@SUSPENDFLAG = 1]
```

## Предоставление привилегий на маршрутизацию документов

Чтобы пользователь мог выполнять операции по маршрутизации документов, у него должны быть соответствующие привилегии. Привилегии, относящиеся к маршрутизации документов, приведены в таблице ниже. Общие привилегии для элементов применяются для процессов, рабочих узлов и рабочих списков.

Таблица 16. Привилегии для маршрутизации документов

| Привилегия                  | Описание                                                                                                                                                                                                                                                                     | Соответствующие API                                                                                   |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| ICM_PRIV_ITEM_UPDATE_WORK   | Используется для просмотра, авторизован ли пользователь выполнять для рабочего пакета следующие операции:<br><div> <div>задавать приоритет</div> <div>задавать владельца</div> <div>задавать список возобновления</div> <div>задавать длительность приостановки</div> </div> | suspendProcess resumeProcess<br>setWorkPackagePriority<br>setWorkPackageOwner                         |
| ICM_PRIV_ITEM_ROUTE_START   | Используется для просмотра, уполномочен ли пользователь запускать процесс.                                                                                                                                                                                                   | startProcess                                                                                          |
| ICM_PRIV_ITEM_ROUTE_END     | Используется для просмотра, авторизован ли пользователь прекращать процесс.                                                                                                                                                                                                  | terminateProcess                                                                                      |
| ICM_PRIV_ITEM_GET_WORKLIST  | Определяет, разрешено ли пользователю узнавать число рабочих пакетов или получать список рабочих пакетов из рабочего списка.                                                                                                                                                 | getCount listWorkPackagePidStrings                                                                    |
| ICM_PRIV_ITEM_GET_WORK      | Используется для просмотра, авторизован ли пользователь получать рабочий пакет.                                                                                                                                                                                              | getNextWorkPackagePidString<br>getNextWorkPackage<br>checkOutItemInWorkPackage<br>retrieveWorkPackage |
| ICM_PRIV_ITEM_GET_ASGN_WORK | Используется для просмотра, авторизован ли пользователь получать рабочий пакет, которым владеет другой пользователь.                                                                                                                                                         | getNextWorkPackagePidString<br>getNextWorkPackage<br>getCount<br>listWorkPackagePidStrings            |
| ICM_PRIV_ITEM_ROUTE         | Используется для просмотра, авторизован ли пользователь маршрутизировать рабочий пакет.                                                                                                                                                                                      | continueProcess                                                                                       |

## Работа со списками управления доступом для маршрутизации документов

Когда для объекта маршрутизации документов (например, процесса, рабочего узла или рабочего списка) определен список управления доступом (ACL), он влияет на операции, разрешенные для этого объекта. Действие списков управления доступом на объекты маршрутизации документов Content Manager и связанные с ними привилегии перечислены в следующей таблице.

Таблица 17. Списки управления доступом и маршрутизация документов

| Привилегия     | Описание                    | Соответствующие API         |
|----------------|-----------------------------|-----------------------------|
| Процесс        | startProcess                | ICM_PRIV_ITEM_ROUTE_START   |
| Рабочий узел   | continueProcess             | ICM_PRIV_ITEM_ROUTE         |
|                | suspendProcess              | ICM_PRIV_ITEM_UPDATE_WORK   |
|                | resumeProcess               | ICM_PRIV_ITEM_UPDATE_WORK   |
|                | terminateProcess            | ICM_PRIV_ITEM_ROUTE_END     |
|                | setWorkPackagePriority      | ICM_PRIV_ITEM_UPDATE_WORK   |
| Рабочий список | setWorkPackageOwner         | ICM_PRIV_ITEM_UPDATE_WORK   |
|                | getNextWorkPackagePidString | ICM_PRIV_ITEM_GET_WORK      |
|                | getNextWorkPackage          | ICM_PRIV_ITEM_GET_ASGN_WORK |
|                | getCount                    | ICM_PRIV_ITEM_GET_WORK      |
|                | listWorkPackagePidStrings   | ICM_PRIV_ITEM_GET_ASGN_WORK |
|                | checkOutItemInWorkPackage   | ICM_PRIV_ITEM_GET_WORKLIST  |
|                |                             | ICM_PRIV_ITEM_GET_ASGN_WORK |
|                |                             | ICM_PRIV_ITEM_GET_WORKLIST  |
|                |                             | ICM_PRIV_ITEM_GET_ASGN_WORK |
|                |                             | ICM_PRIV_ITEM_GET_WORK      |

## Константы маршрутизации привилегий

Константы маршрутизации документов определяются в DKConstantlCM. Вот список констант маршрутизации документов:

- public final static int DK\_ICM\_DR\_SELECTION\_FILTER\_NO = 0;
- public final static int DK\_ICM\_DR\_SELECTION\_FILTER\_YES = 1;
- public final static int DK\_ICM\_DR\_SELECTION\_FILTER\_EITHER = 2;
- public final static int DK\_ICM\_DR\_SELECTION\_ORDER\_PRIORITY = 0;
- public final static int DK\_ICM\_DR\_SELECTION\_ORDER\_TIME = 1;
- public final static int DK\_ICM\_DR\_MAX\_RESULT\_ALL = 0;
- public final static String DK\_ICM\_DR\_START\_NODE = "START";
- public final static String DK\_ICM\_DR\_END\_NODE = "END";

---

## Работа с другими контент-серверами

Классы `dkDatastore` служат для определения соответствующего контент-сервера для контент-серверов в вашей прикладной программе. Контент-сервер - это первичный интерфейс для Enterprise Information Portal. У каждого контент-сервера есть отдельный класс контент-сервера.

Для создания контент-сервера используйте классы `DKDatastorexx`, где суффикс `xx` идентифицирует конкретный контент-сервер. Эти классы перечислены в Табл. 7 на стр. 44.

*Таблица 18. Тип сервера и терминология имен классов*

| Контент-сервер                                     | Имя класса                                 |
|----------------------------------------------------|--------------------------------------------|
| Content Manager Версия 8.2                         | DKDatastoreICM                             |
| Ранние версии Content Manager                      | DKDatastoreDL                              |
| Content Manager OnDemand                           | DKDatastoreOD                              |
| Content Manager for AS/400 (VisualInfo for AS/400) | DKDatastoreV4                              |
| Content Manager ImagePlus for OS/390               | DKDatastoreIP                              |
| Domino.Doc                                         | DKDatastoreDD                              |
| Extended Search                                    | DKDatastoreDES                             |
| Panagon Image Services (FileNET)                   | DKDatastoreFN                              |
| Реляционные базы данных                            | DKDatastoreDB2, DKDatastoreJDBC (для Java) |

При создании контент-сервера надо реализовать каждый из следующих классов и интерфейсов:

### **dkDatastore**

Представляет контент-сервер и управляет соединением, связью и выполнением команд контент-сервера. `dkDatastore` - абстрактная версия класса менеджера запросов. Он поддерживает метод `evaluate`.

### **dkDatastoreDef**

Использует методы доступа к элементам, хранящимся на контент-сервере, а также создает и удаляет объекты и получает их списки. Он обслуживает собрание определений `dkEntityDef`. Примеры конкретных классов для этого интерфейса:

- `DKDatastoreDefDL`
- `DKDatastoreDefOD`

## dkEntityDef

Использует методы доступа к информации объектов, а также создает и удаляет объекты и атрибуты. Методы этого класса поддерживают доступ к многоуровневым объектам. Если контент-сервер не поддерживает подобъекты, они генерируют объекты DKUsageError. Если контент-сервер поддерживает многоуровневые объекты, вы должны реализовать методы переопределения этих исключительных ситуаций для подклассов таких складов данных. Примеры конкретных классов для интерфейса dkEntityDef:

- DKIndexClassDefDL
- DKAppGrpDefOD

Иерархия классов для определения объекта показана на рис. 14:

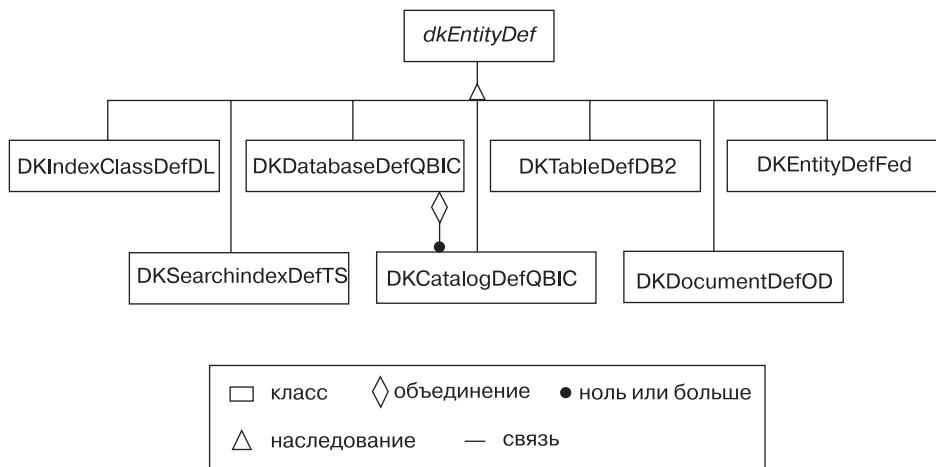


Рисунок 14. Иерархия классов

## dkAttrDef

Определяет методы доступа к информации атрибутов и создания и удаления атрибутов. Примеры конкретных классов для dkAttrDef:

- DKAttributeDefDL
- DKFieldDefOD

## dkServerDef

Определяет методы доступа к информации сервера. Примеры конкретных классов dkServerDef:

- DKServerDefDL
- DKServerDefOD



## dkResultSetCursor

Создает указатель контент-сервера, который управляет сборанием объектов DDO. Чтобы использовать методы `addObject`, `deleteObject` и `updateObject`, надо задать для опции контент-сервера `DK_CM_OPT_ACCESS_MODE` значение `DK_CM_READWRITE`.

## dkBlob

Объявляет общий общедоступный интерфейс для типов данных двоичного большого объекта (BLOB) на каждом контент-сервере. Конкретные классы, полученные на основе `dkBlob`, совместно используют общий интерфейс, что допускает обработку собраний двоичных больших объектов из разнородных контент-серверов. Примеры конкретных классов для `dkBlob`:

- `DKBlobDL`
- `DKBlobOD`

Эти классы определения данных и их иерархия показаны на рис. 15:

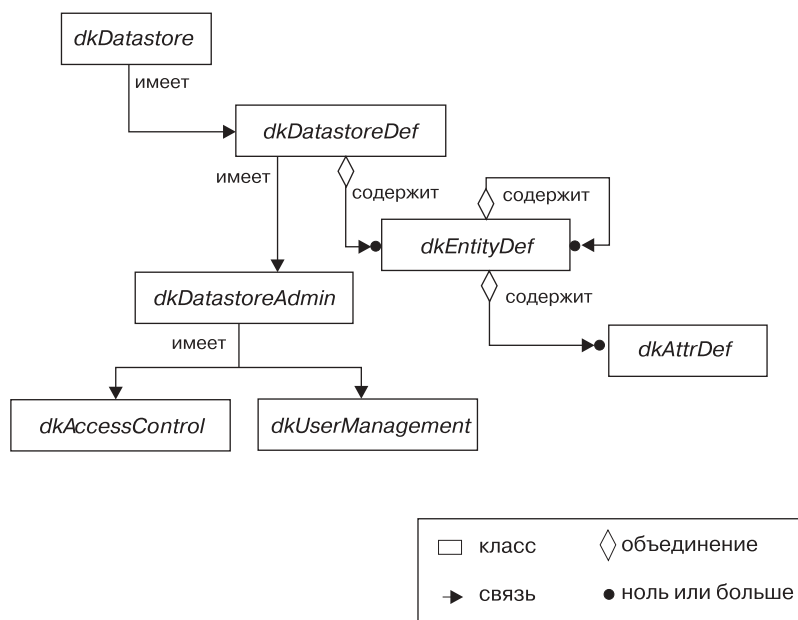


Рисунок 15. Иерархия классов определения данных

Дополнительную информацию о `dkDatastore` и других общих классах смотрите в разделе “Разработка пользовательских контент-серверов” на стр. 464.

---

## Работа с ранними версиями Content Manager

В этом разделе описано, как обращаться к данным на серверах Content Manager и как выполнять следующие задачи:

- Обрабатывать большие объекты
- Использовать объекты DDO
- Использовать объекты XDO в механизме поиска
- Использовать комбинированный запрос
- Использовать Механизм текстового поиска
- Использовать поиск изображений (QBIC)
- Использовать рабочие пакеты и рабочие комплекты.

### Обработка больших объектов

В ранних версиях соединителя Content Manager большие объекты можно получать частями, используя асинхронное получение. Смотрите программу примера TxdoAsyncRetDL в каталоге CMBROOT\dk\samples.

#### Задание размера кучи Java (только для Java)

В Java по умолчанию заданы начальный и максимальный размер кучи. Начальный размер кучи по умолчанию - 1 048 576 байт, максимальный размер кучи по умолчанию - 16 777 216 байт. Если программа Java пытается использовать объекты больше, чем размер кучи, при выполнении программы возникнет ошибка. Чтобы увеличить размер кучи для программы, используйте при запуске программы Java опцию -mx. Например:

#### Java

```
java -mx40000000 yourApplication
```

### Использование объектов DDO для содержания ранних версий Content Manager

В объекте DDO, связанном с DKDatastoreDL, есть особая информация для представления модели документа Enterprise Information Portal: документ, папка, части, элемент, ID элемента, ранг и т.п. В следующих разделах рассказано, как обращаться с этой информацией.

#### Свойства DDO

Тип элемента - документ или папка - это свойство с именем DK\_CM\_PROPERTY\_ITEM\_TYPE. Чтобы получить тип элемента объекта DDO, вызовите:

### Java

```
DKDDO addo = new DKDDO(dsDL, pid);
Object obj = addo.getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
if (obj != null) {
 short item_type = ((Short) obj).shortValue();
}
```

### C++

```
DKAny any = cddo->getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
if (!any.isNull()) {
 unsigned short item_type = (unsigned short) any;
} ... // какие-то действия
```

После вызова свойства переменная `item_type` равна `DK_CM_DOCUMENT` для документа или `DK_CM_FOLDER` для папки. Оператор `if` удостоверяет, что свойство существует. Дополнительные сведения смотрите в разделах “Добавление свойств в DDO” на стр. 53 и “Получение свойств DKDDO и атрибутов” на стр. 58.

### Постоянный идентификатор (Persistent identifier - PID)

PID содержит важные сведения, специфичные для Enterprise Information Portal - тип объекта, который указывает, какому индексному классу принадлежит DDO; PID содержит ID соответствующего элемента контент-сервера. Смотрите раздел “Создание постоянного идентификатора (PID)” на стр. 53.

### Представление документов

У объекта DDO, представляющего документ, свойство `DK_CM_PROPERTY_ITEM_TYPE` имеет значение `DK_CM_DOCUMENT`. Его PID содержит имя индексного класса в качестве типа объекта. ID в PID совпадает с ID элемента.

Части документа представлены как объекты `DKPartsDL`, представляющие собой собрания двоичных больших объектов, каждый из которых представлен как объект `DKBlobDL`.

DDO-документ имеет специальный атрибут под именем `DKPARTS`; его значение - объект `DKParts`.

Чтобы получить все части документа, получите сначала `DKParts`, а затем создайте итератор для перебора частей. Если в документе нет ни одной части, `DKParts` пуст или мощность `DKParts` равна нулю.

У документов, связанных с комбинированным запросом (сочетающим параметрический и текстовый запрос), может быть временный атрибут с именем DKRANK, значением которого является объект, содержащий целочисленный ранг, рассчитанный механизмом текстового поиска.

Дополнительную информацию о создании и обработке объекта DKParts смотрите в разделах “Создание, изменение и удаление документов и папок”, “Получение документа или папки” на стр. 314 и “Создание документов и использование атрибута DKPARTS” на стр. 107.

### **Представление папок**

У объекта DDO, представляющий папку, свойство DK\_CM\_PROPERTY\_ITEM\_TYPE имеет значение DK\_CM\_FOLDER. Его PID, как и у DDO-документа, содержит имя индексного класса в качестве типа объекта и ID элемента в качестве ID в PID.

Объект DKFolder представляет содержимое папки. Объект DKFolder - собрание DDO. Каждый DDO представляет элемент в папке - либо документ, либо другую папку. У DDO-папки есть атрибут с именем DKFOLDER, значение которого - объект DKFolder.

Чтобы получить все элементы папки, сначала получите DKFolder, а затем создайте итератор для перебора элементов. Если в папке нет ни одного элемента, DXFolderDL имеет пустое значение, но атрибут DKFOLDER всегда присутствует в DDO папки, созданной контент-сервером.

Дополнительную информацию о создании и обработке объекта DKFolder смотрите в разделах “Создание, изменение и удаление документов и папок”, “Получение документа или папки” на стр. 314 и “Создание папок и использование атрибута DKFOLDER” на стр. 111.

## **Создание, изменение и удаление документов и папок**

В этом разделе описаны процессы создания, изменение и удаление документов и папок.

### **Создание документа**

Чтобы создать документ и сохранить его постоянные данные на контент-сервере, надо создать объект DDO и задать все его атрибуты (и другую информацию), кроме ID элемента. ID элемента будет назначен и возвращен контент-сервером. Некоторые из предыдущих примеров сведены вместе в следующем примере:

## Java

```
// ----- Шаг 1: создать склад данных и соединиться с ним
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Шаг 2: создать DDO-документ (или DDO-папку)
// и задать все его атрибуты и другую необходимую информацию
DKPid pid = new DKPid();
pid.setObjectType("GRANDPA"); // задаем имя его индексного класса
DKDDO ddo = new DKDDO(dsDL,pid); // создаем DDO с PID и связываем
... // его с dsDL

// ----- Шаг 2.a: добавляем атрибуты для индексного класса GRANDPA
Object obj, vstr;
Boolean yes = new Boolean(true);
Boolean no = new Boolean(false);

short data_id = cddo.addData("Title"); // добавляем новый атрибут "Title"
vstr = new Short(DK_CM_DATAITEM_TYPE_STRING);
// ----- Добавляем свойства типа VSTRING и nullable
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstr);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, no);

data_id = cddo.addData("Subject"); // добавляем новый атрибут "Subject"
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstr);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, yes);

// ----- добавляем другие атрибуты, если нужно
....

// ----- Шаг 2.b: добавляем атрибут DKPARTS
DKParts parts = new DKParts(); // создаем новый DKParts - собрание
// частей
DKBlobDL blob = new DKBlobDL(dsDL); // создаем новый двоичный большой
// объект XDO
DKPidXDODL pidXDO = new DKPidXDODL(); // создаем PID для этого объекта XDO

pidXDO.setPartId(5); // задаем номер части 5
blob.setPidObject(pidXDO); // задаем PID для двоичного большого
// объекта XDO
blob.setContentClass(DK_DL_CC_GIF); // задаем тип класса содержимого GIF
blob.setRepType(DK_REP_NULL); // задаем тип представления для части
blob.setContentFromClientFile("choice.gif"); // задаем содержимое этого
// объекта
blob.setInstanceOpenHandler("xv"); // программа просмотра в AIX

parts.addElement(blob); // добавляем двоичный большой объект в собрание
// частей
.... // при необходимости создаем и добавляем в собрание
.... // еще несколько двоичных больших объектов

//продолжение следует . . .
```

### Java (продолжение)

```
// ----- Создаем атрибут DKPARTS и задаем его как ссылку на объект DKParts
short data_id = ddo.addData(DKPARTS); // добавляем атрибут "DKParts"
obj = new Short(DK_CM_COLLECTION_XDO); // добавляем свойство Type
ddo.addDataProperty(data_id,DK_CM_PROPERTY_TYPE,obj);
ddo.addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE,yes); // добавляем
// свойство Nullable
ddo.setData(data_id, parts); // задаем значение атрибута

// ----- Шаг 2.С: задает тип элемента: документ
obj = new Short(DK_CM_DOCUMENT);
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, obj);

// ----- Шаг 3: делаем элемент постоянным и добавляем его на склад данных
ddo.add(); // документ создан на складе данных
```

## C++

```
// шаг 1: создать склад данных и соединиться с ним
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// шаг 2: создать DDO-документ (или DDO-папку)
// и задать все его атрибуты и другую необходимую информацию
// смотрите раздел "Использование DDO"
DKPid pid;
// задать имя индексного класса, которому он принадлежит
pid.setObjectType("GRANDPA");
// создать объект DDO с PID и связать его с dsDL
DKDDO* ddo = new DKDDO(&dsDL,pid);

// шаг 2.a: добавить атрибуты для индексного класса GRANDPA
DKAny any;
DKBoolean yes = TRUE;
DKBoolean no = FALSE;
// добавить новый атрибут с именем "Title"
unsigned short data_id = cddo->addData("Title");
// добавить свойство типа VSTRING
any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
// добавить свойство допустимости пустых значений: nullable
any = no;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// добавить новый атрибут с именем "Subject"
data_id = cddo->addData("Subject");

any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
any = yes;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// добавить другие атрибуты, если нужно
// ...

// шаг 2.b: добавить атрибут DKPARTS
// создать новый двоичный большой объект XDO
DKParts* parts = new DKParts;
DKBlobDL* blob = new DKBlobDL(&dsDL);

DKPidXDODL pidXDO; // создать PID для этого объекта XDO

pidXDO.setPartId(5); // задаем номер части 5
blob->setPid(&pidXDO); // задаем PID для двоичного большого
 // объекта XDO
blob->setContentClass(DK_CC_GIF); // задаем тип класса содержимого GIF
blob->setRepType(DK_REP_NULL); // задаем тип представления для части
//продолжение следует . . .
```

### C++ (продолжение)

```
blob->setContentFromClientFile("choice.gif"); // задаем содержимое этого
 // объекта

DKAny any = (dkDataObjectBase*) blob;
parts->addElement(any); // добавляем двоичный большой объект
 // в собрание частей
... // при необходимости создать и добавить в собрание
... // еще несколько двоичных больших объектов
// создает атрибут DKPARTS и задает его как ссылку на объект DKParts
// добавить атрибут "DKParts"
unsigned short data_id = ddo->addData(DKPARTS);
any = DK_COLLECTION_XDO;
// добавить свойство типа
ddo->addDataProperty(data_id,DK_CM_PROPERTY_TYPE,any);
any = (DKBoolean) TRUE;
// добавить свойство допустимости пустых значений
ddo->addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE ,any);
any = (dkCollection*) parts;
// задает значение атрибута
ddo->setData(data_id, any);

// шаг 2.с: задает тип элемента: документ
any = DK_CM_DOCUMENT;
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);

// шаг 3: сделать его постоянным
// документ создается на складе данных
ddo->add();
```

Последним шагом обработки примера - создание документа (с информацией) на контент-сервере. Когда DDO-документ добавляется на контент-сервер, добавляются все его атрибуты, включая все части в собрании DKParts.

Тот же процесс используется и для добавления DDO-папки. Элементы собрания DKFOLDER добавляются на контент-сервер в качестве компонента папки. Такая папка содержит таблицу содержания элементов, то есть существующих документов и папок. Поэтому перед добавлением на контент-сервер DDO-папки создайте на нем все элементы папки.



## Java

Можно добавить один и тот же документ на разных контент-серверах одного типа. Чтобы добавить этот документ на сервер LIBSRVRN, имеющим индексный класс LIBSV2 с той же структурой, что и LIBSV, используйте следующий пример:

```
// ----- создаем склад данных и соединяемся с LIBSRVRN
DKDatastoreDL dsN = new DKDatastoreDL();
dsN.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Изменяем PID
pid = ddo.getPidObject();
pid.setObjectType("LIBSV2"); // задаем новый индексный класс
pid.setPrimaryId(""); // задаем пустой ID элемента
pid.setDatastoreName("LIBSRVRN"); // задаем новое имя склада данных
ddo.setPidObject(pid); // изменяем PID
ddo.setDatastore(dsN); // снова связываем DDO с dsN
ddo.add(); // добавляем DDO
```

## C++

Можно добавить один и тот же документ на разных контент-серверах одного типа. Например, чтобы добавить документ на сервере LIBSRVRN, имеющим индексный класс GRANDPA2 с той же структурой, что и GRANDPA:

```
// создать склад данных и соединиться с LIBSRVRN
DKDatastoreDL dsN;
dsN.connect("LIBSRVRN","FRNADMIN","PASSWORD");
// изменить PID
pid = ddo->getPid();
pid.setObjectType("GRANDPA2"); // задаем новый индексный класс
pid.setId(""); // задаем пустой ID элемента
pid.setDatastoreName("LIBSRVRN"); // задаем новое имя склада данных
ddo->setPid(pid); // изменяем PID
ddo->setDatastore(&dsN); // снова связываем его с dsN
ddo->add(); // добавляем его
```

## Изменение документа или папки

Чтобы изменить документ или папку:

1. Задайте ID элемента и тип объекта.
2. Измените соответствующие атрибуты или добавьте их в собрание DKParts.
3. Вызовите метод update для сохранения изменений.

#### Java

```
// ----- изменяем значение атрибута Title
String newTitle = "Accident Report";
short data_id = ddo.getDataByName("Title");
ddo.setData(data_id, newTitle);
ddo.update();
```

#### C++

```
// изменить значение атрибута Title
DKAny any = DKString("Догадайся, кто за всем этим стоит");
unsigned short data_id = ddo->getDataByName("Title");
ddo->setData(data_id, any);
ddo->update();
```

После вызова метода `update` значение атрибута `Title` на контент-сервере будет изменено. Части в этом документе не будут изменены, пока не изменится их содержимое. При вызове метода `update` должно существовать соответствующее соединение с сервером.

При изменении DDO-папки используются аналогичные шаги: изменение значения атрибутов или добавление и удаление элементов из `DKFolder`, затем вызов метода `update`.

### Изменение частей

Собрание частей в документе представляется при помощи объекта `DKParts`.

`DKParts` - это подкласс `DKSequentialCollection`. Помимо наследуемых функций последовательного собрания, `DKParts` имеет два дополнительных метода для добавления части в собрании и удаления части из собрания. Эти методы немедленно сохраняют изменения на контент-сервере.

Документ уже должен существовать на контент-сервере.

**Добавление и удаление элемента:** В следующих примерах в документе добавляется часть.

### Java

```
DKDDO addo = new DKDDO(); // создаем DDO-документ
DKBlobDL newPart = new DKBlobDL(); // создаем новую добавляемую часть
.... // инициализируем DDO и новую часть
DKParts parts = (DKParts) addo.getDataByName(DKPARTS); // получаем DKParts
parts.addMember(ddo, newPart); // предполагается, что среди этих
 // значений нет NULL
```

### C++

```
// DDO-документ
DKDDO* ddo;
// добавляемая часть
DKBlobDL* newPart;
// ddo и newPart
// инициализируются где-то здесь
...
...
// получить DKParts
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
// предполагается, что среди этих значений нет пустых (NULL)
parts->addMember(ddo, newPart);
```

Чтобы удалить newPart из собрания и с контент-сервера данных, используйте:

### Java

```
parts.removeMember(addo, newPart);
```

### C++

```
parts->removeMember(ddo, newPart);
```

Метод removeMember в DKParts действительно удаляет постоянную копию части с контент-сервера.

**Различия между изменением, добавлением и удалением DDO документа:** Методы addMember и removeMember удобно использовать для добавления и удаления части в собрании и на контент-сервере. Они быстрее метода update в DDO-документе. Метод update для DDO изменяет все атрибуты в объекте DDO, включая DKParts и все его изменяемые элементы. В примере выполняются следующие действия:

## Java

```
....
// ----- получаем DKParts; предполагается, что он существует и непуст
DKParts parts = (DKParts) addo.getDataByName(DKPARTS);
parts.addElement(newPart); // добавляем новую часть к частям
addo.update(); // изменяем ddo целиком
....
```

## C++

```
...
DKAny any = ddo->getDataByName(DKPARTS);
// получить DKParts; предполагается, что он существует
DKParts* parts = (DKParts*) any.value();
// предполагается, что он не пуст (не NULL)
any = (dkDataObjectBase*) newpart;
parts->addElement(any);
// изменяет весь DDO
ddo->update();
...
```

## Изменение папок

Собрание документов и папок в папке представляется как объект DKFolder. На контент-сервере в папке хранится таблица содержания со ссылками на объекты, а не сами объекты.

DKFolder - это подкласс DKSequentialCollection. Помимо наследуемых методов последовательного собрания, у него есть два дополнительных элемента для добавления элемента (документа или папки) в собрание и удаления элемента из собрания с немедленным сохранением этих изменений.

При добавлении или удалении документ или папка должны уже существовать на контент-сервере.

**Добавление и удаление элемента:** В следующем примере показано добавление в DDO-папку другого DDO (документа или папки):

### Java

```
DKDDO folderDDO = new DKDDO(); // создаем DDO-папку
DKDDO newMember = new DKDDO(); // создаем новый DDO для добавления
.... // инициализация DDO-папки и
.... // newMember
// ----- Получаем DKFolder в предположении что он существует и непуст
DKFolder folder = (DKFolder) folderDDO.getDataByName(DKFOLDER);
folder.addMember(folderDDO, newMember);
```

### C++

```
// DDO-папка
DKDDO* folderDDO;
// новый DDO, добавляемый как элемент этой папки
DKDDO* newMember;
... // folderDDO и newMember
... // инициализируются где-то здесь
DKAny any = folderDDO->getDataByName(DKFOLDER);
// получить объект DKFolder; предполагается, что он существует
DKFolder* folder = (DKFolder*) any.value();
// предполагается, что он не пуст
folder->addMember(folderDDO, newMember);
```

И newMember, и folderDDO должны существовать на контент-сервере, чтобы можно было выполнить добавление документа или папки.

Подобным образом, чтобы удалить newMember из собрания и с контент-сервера, введите:

### Java

```
folder.removeMember(folderDDO, newMember);
```

### C++

```
folder->removeMember(folderDDO, newMember);
```

**Внимание:** Удаление элемента из папки удаляет этот элемент только из таблицы содержания папки. При использовании removeElementAt функция не удаляет элемент из памяти или с контент-сервера.

**Различия между изменением, добавлением и удалением DDO папки:** Методы `addMember` и `removeMember` из `DKFolder` удобно использовать для добавления и удаления документа или папки в собрании и на контент-сервере. Они быстрее метода `update` в DDO-папке.

Метод `update` для DDO изменяет все атрибуты в объекте DDO, включая `DKFolder` и все его элементы, тогда как методы `addMember` и `removeMember` выполняют только добавление или удаление отдельного элемента в таблице содержания папки.

### Удаление документа или папки

Метод `del` в DDO служит для удаления документа или папки с контент-сервера.

#### Java

```
ddo.del();
```

#### C++

```
ddo->del();
```

У DDO должны быть заданы ID элемента и тип объекта, и должно быть соответствующее соединение с контент-сервером.

Используйте приведенный выше оператор и для удаления папки. Удаляются только постоянные данные, копия DDO в памяти не изменяется. Поэтому после удаления этот DDO можно снова добавить на тот же или на другой контент-сервер позже в программе. Дополнительную информацию смотрите в разделе “Создание документа” на стр. 304.

### Получение документа или папки

Чтобы получить документ из `DKDatastoreDL` (представляющего контент-сервер ранней версии `Content Manager`), надо знать имя индексного класса документа и его ID элемента. Необходимо также связать DDO с контент-сервером и установить соединение.

## Java

```
DKDDO ddo = new DKDDO(dsDL,pid);
// ----- Создаем склад данных и устанавливаем соединение
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

DKPid pid = new DKPid();
pid.setObjectType("Claim"); // задаем имя его индексного класса
pid.setPrimaryId("LN#U5K6ARLGM3DB4"); // задаем ID элемента
// ----- создаем DDO с PID и связываем его со складом данных

ddo.retrieve(); // получаем документ
```

## C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
DKPid pid;
// задать имя индексного класса, которому он принадлежит
pid.setObjectType("GRANDPA");
// задать ID элемента
pid.setId("LN#U5K6ARLGM3DB4");
// создать объект DDO с PID и связать его с dsDL
DKDDO* ddo = new DKDDO(&dsDL,pid);
// получить его
ddo->retrieve();
```

После вызова для получения все значения атрибутов DDO задаются равными значению постоянных данных, хранящихся на контент-сервере. Если в документе есть части, атрибут DKPARTS задается равным объекту DKParts. Но получения содержимого частей этого собрания не происходит. Поскольку часть может оказаться большой, не следует получать части в память все сразу. Лучше задавать получение той части, которая вам нужна.

Если DDO представляет собой результат параметрического запроса, выполненного с опцией запроса CONTENT=NO, этот DDO пуст (то есть не содержит значений атрибутов). Но вся необходимая информация для его получения уже задана.

### Получение частей

После получения DDO можно получить его части, которые хранятся в атрибуте DKPARTS, как показано ниже:

#### Java

```
DKParts parts = (DKParts) ddo.getDataByName(DKPARTS);
```

#### C++

```
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
```

В этом примере предполагается, что атрибут DKPARTS существует. Если он не существует, генерируется исключительная ситуация. В разделе “Получение папки” на стр. 317 приводится пример извлечения значения атрибута, для чего вначале получают ID данных и проверяют его на нулевое значение.

Для получения всех частей надо создать итератор для перебора собрания получения каждой части. Смотрите “Создание документов и использование атрибута DKPARTS” на стр. 107.

#### Java

```
// ----- создаем итератор и обрабатываем элементы собрания частей
if (parts != null) {
 DKBlobDL blob;
 dkIterator iter = parts.createIterator();
 while (iter.more()) {
 blob = (DKBlobDL) iter.next();
 if (blob != null) {
 blob.retrieve(); // получаем содержимое
 blob.open(); // двоичного большого объекта
 // другая обработка, если требуется
 }
 }
}
```



### C++

```
// создать итератор и обработать элементы собрания частей поочередно
if (parts != NULL) {
 DKAny* element;
 DKBlobDL* blob;
 dkIterator* iter = parts->createIterator();
 while (iter->more()) {
 element = iter->next();
 blob = (DKBlobDL*) element->value();
 if (blob != NULL) {
 // получаем содержимое двоичного большого объекта
 blob->retrieve();
 // другая обработка, если требуется
 blob->open();
 }
 }
 delete iter;
}
```

Как и объект DDO - результат параметрического запроса, каждая часть XDO в собрании DKParts пуста (у нее не задано содержание). Но у нее есть все необходимые данные для получения информации. Чтобы перенести ее содержимое и связанную информацию в память, вызовите метод retrieve:

### Java

```
blob.retrieve();
```

### C++

```
blob->retrieve();
```

## Получение папки

DDO-папку получают тем же способом, что и DDO-документ. После получения у DDO-папки все ее атрибуты заданы, включая и атрибут DKFOLDER. Значение этого атрибута задано равным объекту DKFolder - собранию элементов DDO в папке. Как и части в DKParts, эти DDO элементов содержат только информацию для получения этих элементов. Получить DDO-папку можно так:

## Java

```
data_id = ddo.dataId(DKFOLDER); // получаем ID данных DKFOLDER
if (data_id == 0) // папка не найдена
 throw new DKException(" folder data-item not found");

DKFolder fCol = (DKFolder) ddo.getData(data_id); // получаем собрание папки

// ----- создаем итератор и по очереди обрабатываем элементы собрания DDO
if (fCol != null) {
 DKDDO item;
 dkIterator iter = fCol.createIterator();
 while (iter.more()) {
 item = (DKDDO) iter.next();
 if (item != null) {
 item.retrieve(); // получаем DDO элемента
 // другая обработка
 }
 }
}
```

## C++

```
// получить ID данных DKFOLDER
data_id = ddo->dataId(DKFOLDER);
// папка не найдена
if (data_id == 0) {
 DKException exc(" folder data-item not found");
 DKTHROW exc;
}
// получить собрание папки
any = ddo->getData(data_id);
DKFolder* fCol = (DKFolder*) any.value();
// создаем итератор и обрабатываем элементы собрания DDO поочередно
if (fCol != NULL) {
 DKAny* element;
 DKDDO* item;
 dkIterator* iter = fCol->createIterator();
 while (iter->more()) {
 element = iter->next();
 item = (DKDDO*) element->value();
 if (item != NULL) {
 // получить DDO элемента
 item->retrieve();
 // другая обработка
 ...
 }
 }
 delete iter;
}
```

Смотрите также раздел “Создание папок и использование атрибута DKFOLDER” на стр. 111.

## О текстовом поиске (Механизм текстового поиска)

Продукт Механизм текстового поиска поддерживает различные типы запросов:

- “Логический запрос”
- “Свободный текстовый запрос” на стр. 320
- “Гибридный запрос” на стр. 320
- “Контекстный запрос” на стр. 320
- “Запрос глобального получения текста (GTR)” на стр. 321

У текстового поиска есть ID элемента, номер части и информация ранга (возвращенные запросом); их можно использовать для создания объекта XDO, который получает документ с сервера ранней версии Content Manager.

Объект DKDatastoreTS служит для представления механизма текстового поиска. Механизм текстового поиска сам не хранит данные, он только индексирует данные, хранящиеся в ранней версии Content Manager для поддержки текстового поиска этих данных. Результат текстового поиска - это идентификатор элемента, описывающий положение документа в Content Manager. Используйте эти идентификаторы для получения документа.

Объект DKDatastoreTS не поддерживает функции add, update, retrieve и delete. Этот контент-сервер можно запрашивать. В разделе “Загрузка данных для индексирования механизмом текстового поиска” на стр. 333 приведена информация о добавлении данных на Content Manager, проиндексированный механизмом текстового поиска.

### Логический запрос

Логический запрос состоит из слов и словосочетаний, разделенных логическими операциями. Словосочетания заключайте в кавычки. Словосочетания рассматриваются как литеральные строки.

В следующем примере создается строка поиска всех текстовых документов со словом `www` или словосочетанием `web site` в индексе текстового поиска `TMINDEX`:

#### Java

```
String cmd = "SEARCH=(COND=(www OR 'web site'));" +
 "OPTION=(SEARCH_INDEX=TMINDEX)";
```

#### C++

```
DKString cmd = "SEARCH=(COND=(www OR 'web site'))";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

### Свободный текстовый запрос

Свободный текстовый запрос состоит из слов, словосочетаний и предложений, заключенных в фигурные скобки ( { } ). Слова не обязательно должны быть соседними. В следующем примере создается строка поиска всех текстовых документов со свободным текстом `web site` в индексе текстового поиска `TMINDEX`:

#### Java

```
String cmd = "SEARCH=(COND=({web site}));" +
 "OPTION=(SEARCH_INDEX=TMINDEX)";
```

#### C++

```
DKString cmd = "SEARCH=(COND=({Web site}));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

### Гибридный запрос

Гибридный запрос состоит из логического запроса, за которым следует свободный текстовый запрос. В следующем примере создается строка поиска всех текстовых документов со словами `IBM` и `UNIX`, а также свободным текстом `Web site` в индексе текстового поиска `TMINDEX`:

#### Java

```
String cmd = "SEARCH=(COND=(IBM AND UNIX {web site}));" +
 "OPTION=(SEARCH_INDEX=TMINDEX)";
```

#### C++

```
DKString cmd = "SEARCH=(COND=(IBM AND UNIX {Web site}));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

### Контекстный запрос

В контекстном запросе выполняется поиск последовательности аргументов поиска в одном документе, в одном абзаце или в одном предложении. В

следующем примере создается строка поиска всех текстовых документов со словосочетанием `rational numbers` и словом `math` в одном абзаце при помощи индекса текстового поиска `TMINDEX`:

#### Java

```
String cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}));" +
 "OPTION=(SEARCH_INDEX=TMINDEX)";
```

#### C++

```
DKString cmd = "SEARCH=(COND=($PARA$ {'рациональные числа' математика}));";
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

В этом типе запроса используется не менее двух аргументов поиска.

### Запрос глобального получения текста (GTR)

Запрос GTR оптимален для языков с набором двухбайтных символов (DBCS), таких как японский и китайский. GTR поддерживает и языки с набором однобайтных символов (SBCS). Все двухбайтные символы заключайте в кавычки ('). Словосочетание поиска должно принадлежать заданному кодовому набору символов и языку.

В следующем примере выполняется поиск GTR всех текстовых документов с фразой `маркетинг IBM`. Ключевое слово `MATCH` задается для указания степени сходства с этой фразой.

#### Java

```
String cmd = "SEARCH=(COND=($CCSID=1251,LANG=6011,MATCH=1$ " +
 "'маркетинг IBM'))";
"OPTION=(SEARCH_INDEX=TMINDEX)";
```

#### C++

```
DKString cmd = "SEARCH=(COND=($CCSID=850, LANG=6011,MATCH=1$ ";
cmd += " 'маркетинг IBM'))";
cmd += "OPTION=(SEARCH_INDEX=TMGTRX)";
```

Убедитесь, что опции текстового поиска контент-сервера `DK_OPT_TS_CCSID` (идентификаторы кодовых наборов символов) и `DK_OPT_TS_LANG` (идентификаторы языка) заданы правильно. По умолчанию `DK_OPT_TS_CCSID` имеет значение `DK_CCSID_00850`, а `DK_OPT_TS_LANG` - значение `DK_LANG_ENU`. Эти

значения используются как глобальные значения по умолчанию для текстового запроса. Дополнительную информацию смотрите в электронном справочнике API.

Можно также ввести определенную информацию CCSID и LANG, как показано в следующем примере. Надо задать и CCSID , и LANG ; нельзя задать одно значение без другого.

### **Представление информации для механизма текстового поиска в виде объектов DDO**

Объект DDO (связанный с объектом DKDatastoreTS) используется для представления результатов текстового поиска.

У объекта DKDatastoreTS нет типа элемента свойства, который есть у объекта DKDatastoreDL. Формат его ID также отличается. DDO, созданный в результате текстового запроса, соответствует текстовой части в элементе. У него есть следующие стандартные атрибуты:

#### **DKDLITEMID**

ID элемента, частью которого является этот текст. Этот ID элемента служит для получения всего элемента с контент-сервера.

#### **DKPARTNO**

Целое, номер части этой текстовой части. Номер части вместе с ID элемента служит для получения текстовой части с контент-сервера.

#### **DKREPTYPE**

Тип представления этой текстовой части. Этот атрибут вместе с ID элемента и номером части служит для получения текстовой части с контент-сервера.

#### **DKRANK**

Целое, ранг релевантности этой части по отношению к результатам текстового запроса. Чем выше ранг, тем лучше соответствие. Дополнительную информацию смотрите в электронном справочнике API.

#### **DKDSIZE**

Целое, число вхождений слова (в результатах логических запросов). Дополнительную информацию смотрите в электронном справочнике API.

#### **DKRCNT**

Целое, число соответствий логического запроса. Дополнительную информацию смотрите в электронном справочнике API.

PID для DDO текстового поиска содержит следующую информацию:

**тип контент-сервера**  
TS.

**имя контент-сервера**

Имя, используемое для соединения с контент-сервером.

**тип объекта**

Индекс текстового поиска.

**ID**

ID документа механизма текстового поиска.

**Установление соединения**

Объект DKDatastoreTS поддерживает две функции соединения и функцию разъединения. Обычно вы создаете объект DKDatastoreTS, соединяетесь с ним, выполняете запрос, а после выполнения разъединяетесь. В следующем примере показана первая функция соединения, использующая сервер текстового поиска ТМ.

**Java**

```
// ----- Создаем склад данных
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("ТМ", "", "", "");
.... // выполняем запрос
dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TConnectTS.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
DKDatastoreTS dsTS;
dsTS.connect("ТМ", "", "", "");
... // выполняем какую-то работу
dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TConnectTS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

В следующем примере показана вторая функция соединения, использующая сервер текстового поиска с именем хоста `apollo`, номером порта `7502` и типом соединения `TCP/IP DK_CTYP_TCPIP`:

```
dsTS.connect("apollo", "7502", DK_CTYP_TCPIP);
```

В следующем примере показан первая функция соединения, использующая имя хоста сервера текстового поиска `apollo`, номером порта `7502` и тип соединения `T` (`TCP/IP`):

```
dsTS.connect("apollo", "", "", "PORT=7502; COMMTYPE=T");
```

В следующем примере показан первый метод соединения, использующий имя сервера текстового поиска TM, библиотечный сервер LIBSRVR2, ID пользователя FRNADMIN и пароль PASSWORD:

В следующем примере показан первый функция соединения, использующий имя сервера текстового поиска TM, библиотечный сервер LIBSRVRN, ID пользователя FRNADMIN и пароль PASSWORD:

```
dsTS.connect("TM", "", "", "LIBACCESS=(LIBSRVRN, FRNADMIN, PASSWORD)");
```

Последний параметр LIBACCESS (строка соединения) можно использовать для передачи последовательности параметров.

**Совет:** Чтобы предотвратить истечение срока соединения с механизмом текстового поиска, соединитесь с механизмом текстового поиска, выполните запросы и немедленно отсоединитесь. Не оставляйте соединение открытым.

### Получение и задание опций текстового поиска

У текстового поиска есть несколько опций, которые можно задавать и получать при помощи его функций. Список опций и их описания смотрите в электронном справочнике по API. В следующем примере показано, как задать и получить опцию набора кодов символов текстового поиска.

#### Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
Integer input_option = new Integer(DK_TS_CCSSID_00850);
Integer output_option = null;

dsTS.setOption(DK_TS_OPT_CCSSID, input_option);
output_option = (Integer) dsTS.getOption(DK_OPT_TS_CCSSID);
```

#### C++

```
DKAny input_option = DK_CCSSID_00850;
DKAny output_option;
dsTS.setOption(DK_OPT_TS_CCSSID, input_option);
dsTS.getOption(DK_OPT_TS_CCSSID, output_option);
```

output\_option - это объект, но обычно преобразуется к типу Integer.

**Советы:** Опции поиска CCSSID и LANG используются вместе. Если задана одна, надо задать и вторую. Значения по умолчанию CCSSID и LANG задаются опциями DKDatastoreTS, DK\_OPT\_TS\_CCSSID и DK\_OPT\_TS\_LANG. Списки опций для контент-серверов и их описания смотрите в электронном справочнике по API.



В одном условии поиска можно задать несколько опций поиска. Опции поиска разделяются запятыми. Пример нескольких условий поиска приводится в разделе “Запрос глобального получения текста (GTR)” на стр. 321.

Если вы задаете обе опции поиска SC (ровно один символ) и MC (последовательность необязательных символов), опцию SC надо задать первой. Например, \$SC=?,MC=\*\$ U?I\*.

### Получение списка серверов

Объект DKDatastoreTS поддерживает функцию для вывода списка серверов текстового поиска, с которыми он может соединяться. В следующем примере показано, как получить список серверов.

#### Java

```
DKServerDefTS pSV = null;
DKIndexTS pIndx = null;
String strServerName = null;
char chServerLocation = ' ';
String strLoc = null;
String strIndexName = null;
String strLibId = null;
int i = 0;
DKDatastoreTS dsTS = new DKDatastoreTS();
System.out.println("list servers");
pCol = (DKSequentialCollection)dsTS.listDataSources();
pIter = pCol.createIterator();
while (pIter.more() == true)
{
 i++;
 pSV = (DKServerDefTS)pIter.next();
 strServerName = pSV.getName();
 chServerLocation = pSV.getServerLocation();
 if (chServerLocation == DK_TS_SRV_LOCAL)
 strLoc = "LOCAL SERVER";
 else if (chServerLocation == DK_TS_SRV_REMOTE)
 strLoc = "REMOTE SERVER";
 System.out.println("Server Name [" + i + "] - " + strServerName +
 " Server Location - " + strLoc);
}
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogTS.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
DKDatastoreTS dsTS;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKString strServerName;
char chServerLocation = ' ';
DKString strLoc;
DKServerDefTS *pSV = 0;
long i = 0;
DKAny a;
cout << "list servers" << endl;
a = dsTS.listDataSources();
pCol = (DKSequentialCollection*)((dkCollection*)a);
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
 i++;
 pSV = (DKServerDefTS*)((void*)(*pIter->next()));
 strServerName = pSV->getName();
 chServerLocation = pSV->getServerLocation();
 if (chServerLocation == DK_SRV_LOCAL)
 {
 strLoc = "LOCAL SERVER";
 }
 else if (chServerLocation == DK_SRV_REMOTE)
 {
 strLoc = "REMOTE SERVER";
 }
 cout << "Server Name [" << i << "] - " << strServerName
 << " Server Location - " << strLoc << endl;
 delete pSV;
}
delete pIter;
delete pCol;
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogTS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

Список серверов возвращается в DKSequentialCollection - собрании объектов DKServerInfoTS. Получив объект DKServerInfoTS, вы можете получить имя и положение сервера. Затем имя сервера можно использовать для установления соединения с ним.

### **Получение списка для схемы**

У объекта DKDatastoreTS есть функции вывода схемы. Для текстового поиска есть индексы текстового поиска. В следующем примере показано, как получить список индексов.

Список индексов возвращается в объекте DKSequentialCollection - собрании объектов DKIndexTS. Получив объект DKIndexTS, вы можете получить информацию об индексе, включая его имя и ID библиотеки, и потом использовать эту информацию для создания запроса.

#### Java

```
tsCol = (DKSequentialCollection) dsTS.listEntities();
tsIter = pCol.createIterator();
int i = 0;
while (tsIter.more()) {
 i++;
 TsIndx = (DKSearchIndexDefTS)tsIter.next();
 strIndexName = TsIndx.getName();
 strLibId = TsIndx.getLibraryId();
 ... \\ Обрабатываем список, как требуется
}
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogTS.java), находится в каталоге CMBROOT\Samples\java\d1.

## C++

```
DKDatastoreTS dsTS;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKString strIndexName;
DKString strLibId;
DKServerDefTS *pSV = 0;
DKSearchIndexDefTS *pIndx = 0;
long i = 0;
DKAny a;
cout << "connecting to datastore" << endl;
dsTS.connect("TM","","");
cout << "list search indexes" << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsTS.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
 i++;
 pIndx = (DKSearchIndexDefTS*)((void*)(*pIter->next()));
 strIndexName = pIndx->getName();
 strLibId = pIndx->getLibraryId();
 cout << "index name [" << i << "] - " << strIndexName
 << " Library - " << strLibId << endl;
 delete pIndx;
}
delete pIter;
delete pCol;
dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogTS.cpp), находится в каталоге Cmbroot/Samples/cpp/dl. Смотрите также информацию об удалении собрания в разделе “Управление памятью в собраниях (только C++)” на стр. 121.

### Индексирование XDO механизмом поиска

Перед поиском элементов данных при помощи текстового поиска и перед поиском изображений нужно проиндексировать данные. Для индексов требуются три значения: SearchEngine, SearchIndex и SearchInfo.

Значение свойства SearchIndex - сочетание двух имен: имени службы поиска и имени индекса поиска. Например, если вы определили на клиенте администратора системы сервер текстового поиска под именем TM и связали с ним индекс поиска под именем TMINDEX, значение для SearchIndex будет: TM-TMINDEX.

Для объекта, который должен быть проиндексирован механизмом текстового поиска, значение SearchEngine должно быть SM, а для элемента данных,

индексируемого для запросов поискам изображений, значение SearchEngine должно быть QBIC (дополнительную информацию о поиске изображений смотрите в разделе “О понятиях и принципах поиска изображений” на стр. 347).

SearchIndex для QBIC - сочетание трех значений: имени базы данных QBIC, имени каталога QBIC и имени сервера QBIC. Например, если имя базы данных QBIC - SAMPLEDB, имя каталога QBIC - SAMPLECAT, и имя сервера QBIC - QBICSRV, правильное значение SearchIndex будет SAMPLEDB-SAMPLECAT-QBICSRV.

В фрагментах LoadSampleTSQBICDL и LoadFolderTSQBICDL в каталоге CMBROOT\Samples приведены примеры загрузки данных и создания папки с последующей загрузкой данных.

**Внимание:** При добавлении объекта части, который нужно индексировать механизмом текстового поиска, не задавайте тип представления. В настоящее время механизм текстового поиска работает только с типом представления по умолчанию - FRN\$NULL.

**Добавление XDO для индексирования механизмом текстового поиска:** В следующем примере показано, как добавить XDO для последующей индексации:

## Java

```
// ----- Определяем переменные для ID части, ID элемента и файла
int partId = 20;
String itemId = "CPPIORH4JBIXWIY0";
String fileName = "g:\\test\\testsrch.txt";
try {
 DKDatastoreDL dsDL = new DKDatastoreDL(); // создаем склад данных
 ... // соединяемся со складом данных
 dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
 DKBlobDL axdo = new DKBlobDL(dsDL); // создаем XDO
 DKPidXDODL apid = new DKPidXDODL(); // создаем PID
 apid.setPartId(partId); // задаем ID части
 apid.setPrimaryId(itemId); // задаем ID элемента
 axdo.setPidObject(apid); // задаем PID для XDO
 axdo.setContentClass(DK_DL_CC_ASCII); // задаем ContentClass как
 // текстовый

 // --- задаем searchEngine
 DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
 aSrchEx.setSearchEngine("SM");
 aSrchEx.setSearchIndex("TM-TMINDEX");
 aSrchEx.setSearchInfo("ENU");
 axdo->setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
 ...
 // ----- Задаем содержимое файла из буферной области
 axdo.setContentFromClientFile(fileName);
 axdo.add(); //добавить из буфера
 ...
 // ----- Выводим ID части после добавления
 System.out.println("after add partId = " + ((DKPidXDODL)
 (axdo.getPidObject())).getPartId());

 dsDL.disconnect(); //отсоединяемся от склада данных
 dsDL.destroy();
}
// ----- обработка исключительных ситуаций
catch (...)
```

**C++**

```
void main(int argc, char *argv[])
{
 DKDatastoreDL dsDL;
 DKString itemId, repType;
 int partId;
 itemId = "N2JJBERBQFK@WTVL";
 repType = "FRN$NULL";
 partId = 10;
 if (argc == 1)
 {
 cout<<"invoke: indexPartxsDL <partId> <repType> <itemId>"<<endl;
 cout<<" no parameter, following default will be provided:"<<endl;
 cout<<"The supplied default partId = "<<partId<<endl;
 cout<<"The supplied default repType = "<<repType<<endl;
 cout<<"The supplied default itemId = "<<itemId<<endl;
 }
 else if (argc == 2)
 {
 partId = atoi(argv[1]);
 cout<<"you enter: indexPartxsDL "<<argv[1]<<endl;
 cout<<"The supplied default repType = "<<repType<<endl;
 cout<<"The supplied default itemId = "<<itemId<<endl;
 }
 else if (argc == 3)
 {
 partId = atoi(argv[1]);
 repType = DKString(argv[2]);
 cout<<"you enter: indexPartxsDL "<<argv[1]<<" "<<argv[2]<<endl;
 cout<<"The supplied default itemId = "<<itemId<<endl;
 }
 else if (argc == 4)
 {
 partId = atoi(argv[1]);
 repType = DKString(argv[2]);
 itemId = DKString(argv[3]);
 cout<<"you enter: indexPartxsDL "<<argv[1]<<" "<<argv[2]<<" "
 <<argv[3]<<endl;
 }
 cout << "connecting Datastore" << endl;
 try
 {
 // подставьте в строку ниже библиотечный сервер,
 // ID пользователя, пароль
 dsDL.connect("LIBSRVN", "FRNADMIN", "PASSWORD");

 cout << "datastore connected" << endl;

 DKBlobDL* axdo = new DKBlobDL(&dsDL);
 DKPidXDODL* apid = new DKPidXDODL;
 }
 // продолжение следует...
```

## C++ (продолжение)

```
 apid ->setPartId(partId);
 apid ->setPrimaryId(itemId);
 apid ->setRepType(repType);
 axdo ->setPidObject(apid);
 cout<<"itemId= "<<axdo->getItemId()<<endl;
 cout<<"partId= "<<axdo->getPartId()<<endl;
 cout<<"repType= "<<axdo->getRepType()<<endl;

 //--- задать searchEngine ----
 cout<<"set search engine and setToBeIndexed()"<<endl;
 DKSearchEngineInfoDL aSrchEx;
 aSrchEx.setSearchEngine("SM");
 aSrchEx.setSearchIndex("TM-TMINDEX");
 aSrchEx.setSearchInfo("ENU");
 axdo->setExtension("DKSearchEngineInfoDL", (dkExtension*)&aSrchEx);
 axdo->setToBeIndexed();
 cout<<"setToBeIndexed() done..."<<endl;

 delete apid;
 delete axdo;
 dsDL.disconnect();
 cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
 cout << "Error id" << exc.errorId() << endl;
 cout << "Exception id " << exc.exceptionId() << endl;
 for(unsigned long i=0;i< exc.textCount();i++)
 {
 cout << "Error text:" << exc.text(i) << endl;
 }
 for (unsigned long g=0;g< exc.locationCount();g++)
 {
 const DKExceptionLocation* p = exc.locationAtIndex(g);
 cout << "Filename: " << p->fileName() << endl;
 cout << "Function: " << p->functionName() << endl;
 cout << "LineNumber: " << p->lineNumber() << endl;
 }
 cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

**Внимание:** При добавлении объекта части, который нужно индексировать механизмом текстового поиска, не задавайте тип представления. Механизм текстового поиска работает только с типом представления по умолчанию - FRN\$NULL.



**Загрузка данных для индексирования механизмом текстового поиска:** Чтобы загрузить в Content Manager данные, которые надо индексировать механизмом текстового поиска, надо создать как индекс, так и индекс текстового поиска.

Чтобы можно было создать индекс текстового поиска, должен работать сервер текстового поиска. Убедитесь, что ваша среда правильно настроена. Для этого можно отредактировать и запустить примеры TListCatalogDL и TListCatalogTS каталога CMBROOT\Samples.

Как создать в Content Manager части, индексированные механизмом текстового поиска, описано в разделе “Работа с расширенными объектами данных (XDO)” на стр. 61.

После того, как данные будут загружены в Content Manager, используйте функцию wakeUpService в классе DKDatastoreDL, чтобы поместить документы в очередь документов. Эта функция принимает имя механизма поиска как параметр.

Затем в окне Управление текстовым поиском Content Manager выполните следующие действия:

1. Щелкните дважды по серверу текстового поиска.
2. Щелкните дважды по индексу текстового поиска.
3. Нажмите кнопку **Индекс**.

Будут проиндексированы документы из очереди документов. По завершении индексации можно будет выполнять запросы текстового поиска.

Дополнительную информацию об управлении текстовым поиском смотрите в книге *Руководство администратора системы*.

## **Использование поддержки текстового структурированного документа**

Структурированные текстовые документы (например, файлы HTML) состоят из текста и разметки. Модель документа задает структуру документа, и текстовый поиск позволяет выполнять поиск по словам или словосочетаниям между тегами.

Чтобы выполнить текстовый запрос структурированных документов:

1. Создайте модель документа. Модель документа содержит разделы; каждый раздел включает имя раздела и используемый тег документа. Например:

```
<HTML>
<HEAD>
<TITLE>Acme Corp
</TITLE>
</HEAD>
<BODY>
<H1>Acme Corp
</H1>
<P>Acme Corp


```

```
<P>John Smith

<P><ADDRESS>Acme Corporation
</ADDRESS>
<HR>
<H2>Acme Corp - Цели бизнеса</H2>
<HR>
<P>
<H2>Маркетинг</H2>
<P>Нам надо увеличить время маркетинга на 25%.
<P>Нам надо удовлетворять потребности клиентов.
</BODY>
</HTML>
```

2. Создайте индекс текстового поиска, использующий эту модель документа Content Manager.
3. Задайте правила индексации для индекса текстового поиска и формат документа по умолчанию (например, DK\_TS\_DOCFMT\_HTML для файлов HTML)
4. Добавьте объекты частей на сервер Content Manager.
5. Запустите процесс индексации для индекса текстового поиска.

В этом примере показано, как вывести список моделей документов, определенных в вашей системе.

## Java

```
// ----- Инициализируем документы
DKSequentialCollection pCol = null;
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;
dkIterator pIter = null;
DKDocModelTS pDocModel = null;
int ccsid = 0;
String strDocModelName = null;
int i = 0;

// ----- Создаем склад данных и соединяемся с ним
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect(srchSrv,"",' ');

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- получаем список моделей документов
pCol = (DKSequentialCollection) dsAdmin.listDocModels("");
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
 i++;
 pDocModel = (DKDocModelTS)pIter.next();
 strDocModelName = pDocModel.getName();
 ccsid = pDocModel.getCCSID();
}
dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TListDocModelsTS.java), находится в каталоге CMBR00T\Samples\java\d1.

**C++**

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKDocModelTS* docModel = 0;
DKSequentialCollection *pCol = 0;
long ccsid = 0;
DKString strDocModelName;
dkIterator *pIter = 0;
long i = 0;
DKAny a;

dsTS.connect(srchSrv, "", "");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// получить список моделей документов
pCol = (DKSequentialCollection*)dsAdmin->listDocModels("");
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
 i++;
 docModel = (DKDocModelTS*)((void*)(*pIter->next()));
 strDocModelName = docModel->getName();
 ccsid = docModel->getCCSID();
 delete docModel;
}
delete pIter;
delete pCol;

dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TListDocModelsTS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

В следующем примере показано, как создать модель документа:

## Java

```
// ----- Создаем склад данных и соединяемся с ним
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;

// ----- создаем экземпляр объекта модели документа
DKDocModelTS docModel = new DKDocModelTS();

// ----- создаем 2 экземпляра объектов раздела документа для модели
DKDocSectionTS docSection = new DKDocSectionTS();
DKDocSectionTS docSection2 = new DKDocSectionTS();

// ----- Описываем модель документов для структуры текстового документа
// для файлов, подобных приведенному выше tstruct.htm
docModel.setCCSID(DK_TS_CCSID_00850);
docModel.setName(docModelName);
docSection.setName("SAMPITITLE");
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPCORPBODY");
docSection2.setTag("BODY");
docModel.addDocSection(docSection2);

dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- создаем модель документа
dsAdmin.createDocModel("", docModel);

dsTS.disconnect();
dsTS.destroy();
```

Другие примеры смотрите в файлах TCreateDocModelTS.java и TCreateStructDocIndexTS.java в каталоге CMBROOT\Samples\java\dl.

**C++**

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;

// создать экземпляр объекта модели документа
DKDocModelTS* docModel = new DKDocModelTS();

// создать 2 экземпляра объектов раздела документа для модели
DKDocSectionTS* docSection = new DKDocSectionTS();
DKDocSectionTS* docSection2 = new DKDocSectionTS();

// Описывает модель документов для структуры текстовых документов
// для файлов, подобных приведенному выше tstruct.htm

docModel->setCCSID(DK_TS_CCSID_00850);
docModel->setName("SAMPCORPMOD");
docSection->setName("SAMPCORPTITLE");
docSection->setTag("TITLE");
docModel->addDocSection(docSection);
docSection2->setName("SAMPCORPBODY");
docSection2->setTag("BODY");
docModel->addDocSection(docSection2);

dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// создать модель документа
dsAdmin->createDocModel("", docModel);

// удалить модель и разделы документа
delete docModel;

dsTS.disconnect();
```

Полные программы примеров, из которых взяты приведенные фрагменты (TCreateDocModelTS.cpp и TCreateStructDocIndexTS.cpp), находятся в каталоге Cmbroot/Samples/cpp/dl.

В следующем примере показано, как создать и задать правила индексации для индекса текстового поиска, чтобы он использовал модель документа:

## Java

```
// ----- Создаем склад данных и объект правил индексирования
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;
DKIndexingRulesTS indexRules = new DKIndexingRulesTS();

// ----- создаем экземпляр объекта модели документа
DKDocModelTS docModel = new DKDocModelTS();

// ----- создаем 2 экземпляра объектов раздела документа для модели
DKDocSectionTS docSection = new DKDocSectionTS();
DKDocSectionTS docSection2 = new DKDocSectionTS();

// ----- создаем экземпляр модели документа для правил индексирования
DKDocModelTS docModel2 = new DKDocModelTS();
docModel2.setCCSID(DK_TS_CCSID_00850);
docModel2.setName("SAMPCORPMOD");

// ----- Описываем модель документов для структуры текстового документа
// для файлов, подобных приведенному выше tstruct.htm
docModel.setCCSID(DK_TS_CCSID_00850);
docModel.setName("SAMPCORPMOD");
docSection.setName("SAMPTITLE");
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPCORPBODY");
docSection2.setTag("BODY");
docModel.addDocSection(docSection2);

// ----- Соединяемся со складом данных
dsTS.connect("TMMUF","", "", "");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

DKSearchIndexDefTS pEnt = new DKSearchIndexDefTS((dkDatastore)dsTS);
pEnt.setName("TSTRUCT");
pEnt.setIndexType(DK_TS_INDEX_TYPE_PRECISE);
pEnt.setIndexProperty(DK_TS_PROPERTY_SECTIONS_ENABLED);
pEnt.setLibraryId("LIBSUM");
pEnt.setLibraryClientServices("IMLLSCDL");
pEnt.setLibraryServerServices("IMLLSSDL");
String strIndexFileDir = "e:\\tsindex\\index\\tstruct";
// ----- В AIX используйте следующую форму для файла
// String strIndexFileDir = "/home/cltadmin/tsindex/tstruct";
pEnt.setIndexDataArea(strIndexFileDir);
String strWorkFileDir = "e:\\tsindex\\work\\tstruct";
// продолжение следует...
```

### Java (продолжение)

```
// ----- В AIX используйте следующую форму для файла
// String strWorkFileDir = "/home/cltadmin/work/tstruct";
pEnt.setIndexWorkArea(strWorkFileDir);

// ----- связываем модель документа с индексом
pEnt.addDocModel(docModel);

// ----- создаем индекс тестового поиска, поддерживающий разделы
dsDef.add((dkEntityDef)pEnt);

indexRules.setIndexName("TSTRUCT");
indexRules.setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules.setDefaultDocModel(docModel2);

dsAdmin.setIndexingRules(indexRules);

dsTS.disconnect();
dsTS.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TCreateStructDocIndexTS.java), находится в каталоге CMBROOT\Samples\java\dl.



**C++**

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKIndexingRulesTS* indexRules = new DKIndexingRulesTS();

// создать экземпляр объекта модели документа
DKDocModelTS* docModel = new DKDocModelTS();

// создать 2 экземпляра объектов раздела документа для модели
DKDocSectionTS* docSection = new DKDocSectionTS();
DKDocSectionTS* docSection2 = new DKDocSectionTS();

// экземпляр модели документа для правил индексации
DKDocModelTS* docModel2 = new DKDocModelTS();
docModel2->setCCSID(DK_TS_CCSID_00850);
docModel2->setName("SAMP_CORPMOD");

// Описывает модель документов для структуры текстовых документов
// для файлов, подобных приведенному выше tstruct.htm

docModel->setCCSID(DK_TS_CCSID_00850);
docModel->setName("SAMP_CORPMOD");
docSection->setName("SAMP_CORPTITLE");
docSection->setTag("TITLE");
docModel->addDocSection(docSection);
docSection2->setName("SAMP_CORPBODY");
docSection2->setTag("BODY");
docModel->addDocSection(docSection2);

// продолжение следует...
```

### C++ (продолжение)

```
dsTS.connect("TMMUF","","");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

DKSearchIndexDefTS* pEnt = new DKSearchIndexDefTS(&dsTS);

pEnt->setName("TSTRUCT");
pEnt->setIndexType(DK_TS_INDEX_TYPE_PRECISE);

// В этом индексе разрешены разделы структурированного текстового документа
pEnt->setIndexProperty(DK_TS_PROPERTY_SECTIONS_ENABLED);

pEnt->setLibraryId("LIBSUM");
pEnt->setLibraryClientServices("IMLLSCDL");
pEnt->setLibraryServerServices("IMLLSSDL");
DKString strIndexFileDir = "e:\\tsindex\\index\\tstruct";
//**** для AIX ****
//DKString strIndexFileDir = "/home/cltadmin/tsindex/index/tstruct";
pEnt->setIndexDataArea(strIndexFileDir);
DKString strWorkFileDir = "e:\\tsindex\\work\\tstruct";
//**** для AIX ****
//DKString strWorkFileDir = "/home/cltadmin/tsindex/work/tstruct";
pEnt->setIndexWorkArea(strWorkFileDir);

// Связать модель документа с индексом
pEnt->addDocModel(docModel);

// Создать индекс тестового поиска, поддерживающий разделы
dsDef->add(pEnt);

delete pEnt;

indexRules->setIndexName("TSTRUCT");
indexRules->setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules->setDefaultDocModel(docModel2);
dsAdmin->setIndexingRules(indexRules);

delete indexRules;

dsTS.disconnect();

Полная программа примера, из которой взят приведенный фрагмент
(TCreateStructDocIndexTS.cpp), находится в каталоге
Cmbroot/Samples/cpp/dl.
```

В следующем примере показано, как запустить асинхронный процесс индексации. При помощи программы управления системой можно запустить процесс индексации и проверить состояние индексации.

## Java

```
// ----- объявляем склад данных и управление
DKIndexFuncStatusTS pIndexFuncStatus = null;
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;

dsTS.connect("TMMUF","","","");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Запускаем процесс индексации
dsAdmin.startUpdateIndex(indexName);

// ----- Получаем состояние индексации
pIndexFuncStatus = dsAdmin.getIndexFunctionStatus(indexName,
 DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS);
.... // обрабатываем состояние, как требуется

// ----- Выводим очередь запланированных документов
System.out.println("*getScheduledDocs "
 + pIndexFuncStatus.getScheduledDocs());

// ----- Выводим первичную очередь документов
System.out.println("*getDocsInPrimaryIndex "
 + pIndexFuncStatus.getDocsInPrimaryIndex());

// ----- Выводим вторичную очередь документов
System.out.println("*getDocsInSecondaryIndex " +
 pIndexFuncStatus.getDocsInSecondaryIndex());
System.out.println("*getDocMessages "
 + pIndexFuncStatus.getDocMessages());
if (pIndexFuncStatus.isCompleted() == true)
{
 // ---- Обработка, если индексирование завершено
}

if (pIndexFuncStatus.getReasonCode() != 0)
{
 dsAdmin.setIndexFunctionStatus(indexName,
 DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS, DK_TS_INDEX_ACTID_RESET);
}

dsTS.disconnect();
dsTS.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TIndexingTS.java), находится в каталоге CMBROOT\Samples\java\d1.

## C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKIndexFuncStatusTS* pIndexFuncStatus = 0;

dsTS.connect(srchSrv,"","");
dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// запускает процесс индексирования
dsAdmin->startUpdateIndex(srchIndex);

// Получить состояние индексирования
pIndexFuncStatus = dsAdmin->getIndexFunctionStatus(srchIndex,
 DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS);
cout << "***** index status *****" << endl;
cout << "isCompleted " << pIndexFuncStatus->isCompleted() << endl;
cout << "getEnabledId " << pIndexFuncStatus->getEnabledId() << endl;
cout << "getReasonCode " << pIndexFuncStatus->getReasonCode()
 << endl;
cout << "getFuncStopped " << pIndexFuncStatus->getFunctionStopped()
 << endl;
cout << "getStartedLast " << pIndexFuncStatus->getStartedLast()
 << endl;
cout << "getEndedLast " << pIndexFuncStatus->getEndedLast() << endl;
cout << "getStartedExecution " << pIndexFuncStatus->getStartedExecution()
 << endl;
cout << "getScheduledDocs " << pIndexFuncStatus->getScheduledDocs()
 << endl;
cout << "getDocsInPrimaryIndex " << pIndexFuncStatus->getDocsInPrimaryIndex()
 << endl;
cout << "getDocsInSecondIndex " << pIndexFuncStatus->getDocsInSecondIndex()
 << endl;
cout << "getDocMessages " << pIndexFuncStatus->getDocMessages()
 << endl;
 if (pIndexFuncStatus->isCompleted() == TRUE)
 {
 // индексирование завершено
 }
 if (pIndexFuncStatus->getReasonCode() != 0)
 {
 dsAdmin->setIndexFunctionStatus(srchIndex,
 DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS,DK_TS_INDEX_ACTID_RESET);
 }
delete pIndexFuncStatus;
dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TIndexingTS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

Пример проверки состояния смотрите в фрагменте TCheckStatusTS в каталоге CMBR00T\Samples. Этот пример проверяет, не перемещен ли помещенный в очередь запрос из очереди запланированных документов в первичную или вторичную очередь. При возникновении ошибки индексации можно посмотреть файл журнала imldiag.log в каталоге экземпляра текстового поиска.

В следующем примере показано, как выполнить текстовый запрос структурированного документа на основе модели документа и индекса текстового поиска, определенных выше.

#### Java

```
// ----- Создаем склад данных
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- соединяемся
dsTS.connect("TMMUF","","","");
// ----- генерируем строку запроса
String cmd = "SEARCH=(COND=($CCSID=850," +
 "DOCMOD=(DOCMODNAME=SAMPCORPMOD," +
 "SECLIST=(SAMPCORPTITLE,SAMPCORPBODY))$ Corp));" +
 "OPTION=(SEARCH_INDEX=TMSTRUCT;MAX_RESULTS=5)";
// ----- Выполняем запрос
pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);

// ----- Обработка результатов
.....
dsTS.disconnect();
dsTS.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteStructDocTS.java), находится в каталоге CMBR00T\Samples\java\d1.

**C++**

```
DKDatastoreTS dsTS;
dkResultSetCursor* pCur = 0;

dsTS.connect("TMMUF","","","");

DKString cmd = "SEARCH=(COND=($CCSID=850,";
cmd += "DOCMOD=(DOCMODNAME=SAMPCORPMOD,";
cmd += "SECLIST=(SAMPCORPTITLE,SAMPCORPBODY))$ Corp));";
cmd += "OPTION=(SEARCH_INDEX=TSTRUCT;MAX_RESULTS=5)";

pCur = dsTS.execute(cmd);

// обработать результаты

dsTS.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteStructDocTS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

## Поиск изображений по их содержанию

Сервер поиска изображений может выполнять поиск хранимых изображений, если задать тип изображения или пример изображения.

На рис. 16 на стр. 347 показан пример программы, которая соединяется с сервером поиска изображений. Сервер поиска изображений использует технологию запросов по содержанию изображения QBIC для поиска похожих цветов, цветовой композиции и текстур.

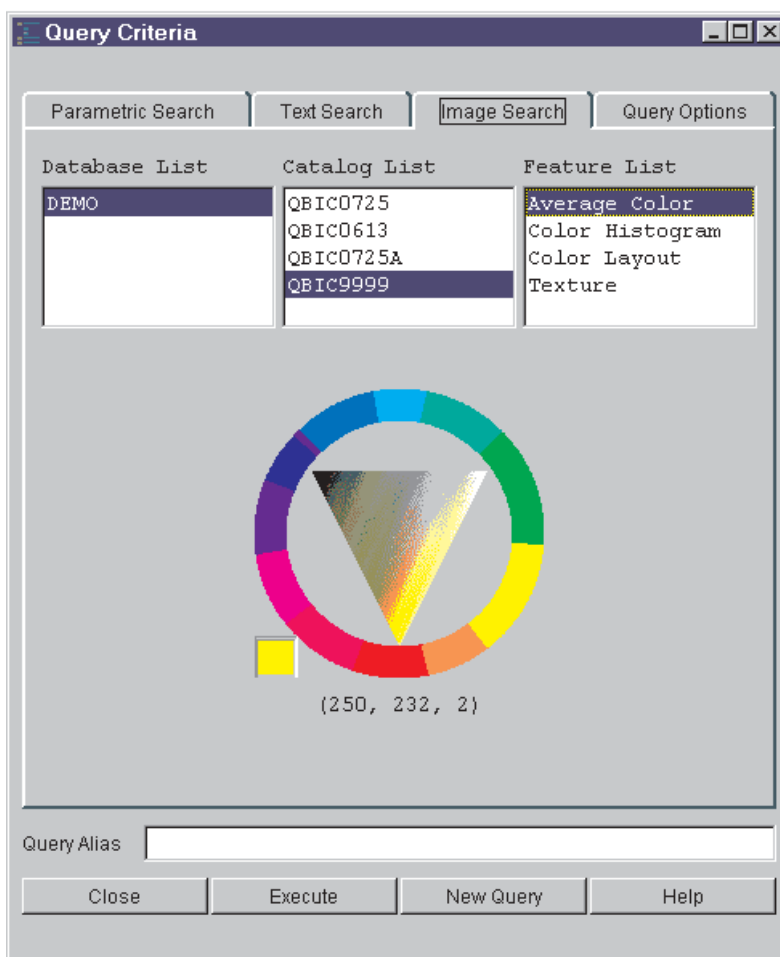


Рисунок 16. Пример клиента поиска изображений

### О понятиях и принципах поиска изображений

В этом разделе описаны компоненты поиска изображений: сервер, базы данных, каталоги и связь сервера поиска изображений с ранней версией системы Content Manager. Описаны также *характеристики* - пригодные для поиска визуальные свойства изображений.

**О серверах, базах данных и каталогах поиска изображений:** Для поиска изображений ранняя версия системы Content Manager использует сервер поиска изображений. Программы Content Manager хранят объекты изображений на

сервере объектов. Сервер поиска изображений анализирует информацию изображения и индексирует ее. Сервер поиска изображений не хранит сами изображения.

Сервер поиска изображений представляется контент-сервером, определяемым объектом DKDatastoreQBIC. Результаты поиска изображений включают идентификаторы (ID элементов), которые описывают положение изображения на сервере Content Manager. Для получения изображений вы можете использовать эти идентификаторы с другими результатами, такими как номера частей и RepType.

Вы можете выполнять запросы на контент-сервере. Однако контент-сервер для поиска изображений не поддерживает операции добавления, изменения, получения и удаления. На рис. 17 показан пример сервера поиска изображений.

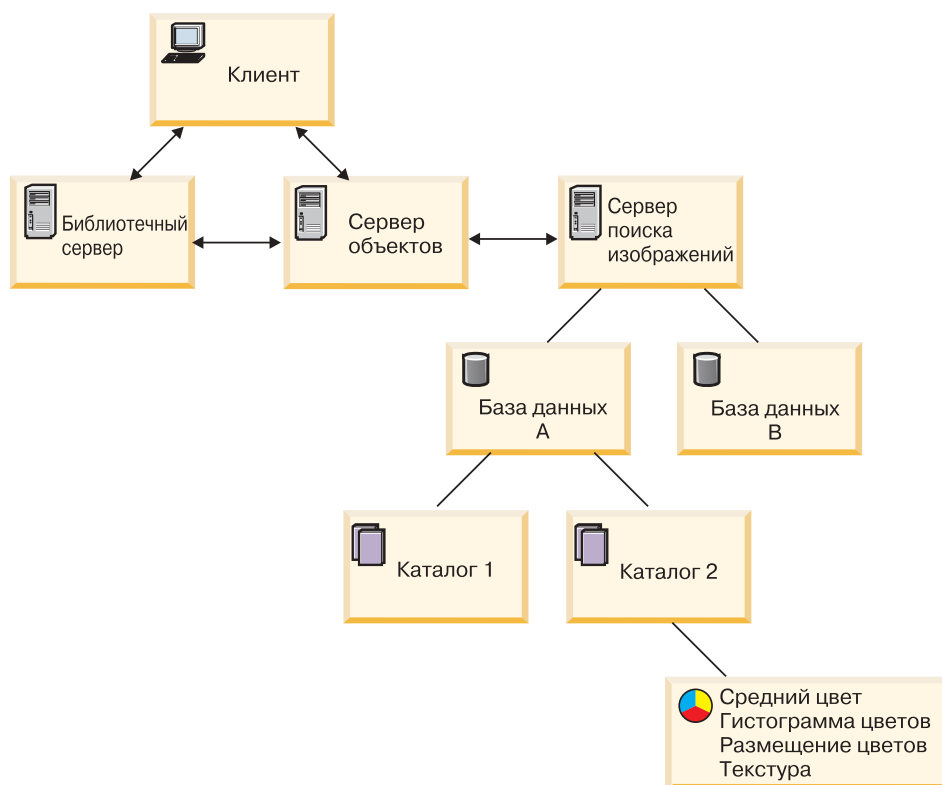


Рисунок 17. Сервер поиска изображений в ранних версиях системы Content Manager

Сервер поиска изображений может содержать одну или несколько баз данных. В каждой базе данных может размещаться один или несколько каталогов,



которые содержат информацию о визуальных характеристиках изображений. Используется четыре характеристики поиска изображений:

- Средний цвет.
- Гистограмма цветов.
- Размещение цветов (цветовая структура).
- Текстура.

**О поисковых характеристиках изображений:** Эти четыре характеристики поиска изображений и их назначение описаны в следующем разделе:

**Средний цвет** Средний цвет служит для поиска изображений, в которых преобладает какой-то определенный цвет. Изображения с близким господствующим цветом близки и по среднему цвету. Например, средний цвет изображений, содержащих равные доли красного и желтого - оранжевый.

Характеристика среднего цвета называется `QbColorFeatureClass`.

#### **Гистограмма цветов**

Мера долей площади, занимаемых цветами на изображении. Анализ гистограммы отдельно измеряет различные цвета в изображении. Например, у изображения сельского пейзажа будет гистограмма цветов, где часто встречаются синий, зеленый и серый.

Гистограмма цветов служит для поиска изображений с близким набором цветов. Характеристика гистограммы цветов называется `QbColorHistogramFeatureClass`.

#### **Размещение цветов**

Размещение цвета определяет средний цвет пикселей в заданной области изображения. Например, у изображений с ярко-красными объектами в середине есть локальный ярко-красный цвет.

Характеристика размещения цветов называется `QbDrawFeatureClass`.

#### **Текстура**

Текстура служит для поиска изображений с характерной текстурой. Текстура определяет детальность, контраст и направленность для изображения. Детальность определяет размер повторяющихся элементов изображения. Контраст определяет изменение яркости внутри изображения. Направленность определяет преобладающее направление в изображении. Например, разные изображения древесных волокон имеют сходную текстуру.

Характеристика текстуры называется `QbTextureFeatureClass`.

## Использование программ поиска изображений

Клиентские программы поиска изображений создают запросы по содержанию изображений, запускают их, а затем оценивают информацию, возвращенную сервером поиска изображений. Прежде чем программа сможет выполнить поиск по содержанию изображений, эти изображения должны быть проиндексированы, и информация о содержимом должна быть сохранена в базе данных поиска изображений.

**Ограничение:** Нельзя индексировать существующие изображения на вашем сервере объектов. Индексировать можно только те изображения, которые вы создадите на сервере объектов после установки сервера и клиента поиска изображений. На рис. 18 показан пример клиента и полученных изображений.

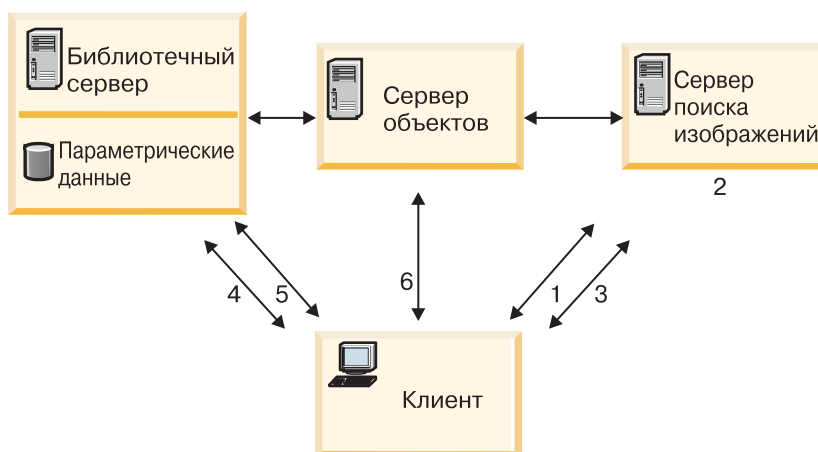


Рисунок 18. Как клиенты поиска изображений выполняют поиск и получение изображений

Чтобы выполнить поиск изображения:

1. Клиент строит строку запроса QVIC и посылает ее на сервер поиска изображений.
2. Сервер поиска изображений получает строку запроса и выполняет поиск соответствий среди каталогизированных изображений.
3. Клиент получает соответствия в виде списка идентификаторов. Идентификатор каждого изображения, для которого установлено соответствие, включает в себя:
  - ID элемента (ItemID).
  - Номер части (PartNum).
  - Тип представления (RepType).
  - Ранг (Rank).

4. Клиент запрашивает на библиотечном сервере часть, содержащую изображение, и индексную информацию.
5. Библиотечный сервер возвращает клиенту информацию индекса, например, текстовое описание. Библиотечный сервер также направляет серверу объектов требование послать клиенту заданные части изображения.
6. Сервер объектов посылает части изображения, и клиент подтверждает получение.

### **Создание запросов**

Создавая запросы, вы строите строку запроса, а программа передает ее серверу поиска изображений. Запрос изображения - это символьная строка, задающая критерии поиска. Критерий поиска содержит:

Запрос изображения - это символьная строка, задающая критерии поиска. Критерий поиска содержит:

#### **Имя характеристики**

Характеристики, используемые в поиске.

#### **Значение характеристики**

Значения этих характеристик. В Табл. 19 на стр. 352 приведены имена и значения признаков поиска изображений, которые можно передавать в строке запроса.

#### **Вес характеристики**

Относительный вес или важность данной характеристики среди прочих. Вес характеристики указывает на ее важность; он учитывается, когда сервер поиска изображений вычисляет меру подобия и возвращает результаты запроса. Чем больше вес, тем важнее считается характеристика.

#### **Максимальное число результатов**

Кроме определения типа изображений, которые вы ищете, можно задать в запросе максимальное количество возвращаемых изображений.

Строка запроса имеет следующую форму: имя\_характеристики значение, где имя\_характеристики - имя характеристики поиска изображений, а значение - значение, связанное с этой характеристикой. Если вы используете в одном запросе несколько характеристик, для каждой из них надо задать пару имя характеристики - значение. Пары разделяются строками "and".

Синтаксис запросов поиска изображений:

*имя\_характеристики значение*  
*имя\_характеристики значение вес*

Характеристика не может повторяться в одном запросе. Можно задать несколько характеристик в одном запросе. Если вы задаете в запросе несколько

характеристик, одной или нескольким из них можно приписать вес. Запросы, где выделена одна из характеристик, имеют вид: имя\_характеристики значение, где имя\_характеристики - имя характеристики поиска изображений, значение - значение, связанное с этой характеристикой, а вес - вес, приданный ей. вес задается сочетанием ключевого слова `weight`, знака равенства (=) и положительного действительного числа.

Можно также задать максимальное число соответствий, возвращаемых запросом. Чтобы задать максимальное число результатов, добавьте в запрос `and` число\_результатов. число\_результатов состоит из ключевого слова `max`, знака равенства (=) и целого числа больше 0. В Табл. 19 описаны имена и значения характеристик.

Таблица 19. Запрос поиска изображений: допустимые значения характеристик

| Имя характеристики                           | Значения                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QbColorFeatureClass или QbColor              | <p><b>color = &lt; значениеRGB , значениеRGB , значениеRGB &gt;</b><br/> где значениеRGB - целое от 0 до 255.</p> <p><b>file = &lt; положениеФайла , " имяФайла " &gt;</b><br/> где положениеФайла - <code>server</code> или <code>client</code>,<br/> имяФайла - полный путь файла в формате системы, в которой находится файл.<br/> Например, можно искать по среднему цвету и текстуре. Значение текстуры задается изображением в файле на клиенте. Вес текстуры вдвое больше, чем вес среднего цвета:</p> <p>QbColorFeatureClass color=<br/> &lt;50, 50, 50&gt; and QbTextureFeatureClass file=<br/> &lt;client, "\\patterns\pattern1.gif"&gt; weight=2.0</p> |
| QbColorHistogramFeatureClass или QbHistogram | <p><b>histogram = &lt; списокГистограмм &gt;</b><br/> где списокГистограмм состоит из одного или нескольких условий условиеГистограммы, разделенных запятыми (,).</p> <p>условиеГистограммы задается в виде (<br/> значениеГистограммы, значениеRGB ,<br/> значениеRGB , значениеRGB ), где<br/> значениеГистограммы - целое от 1 до 100 (в процентах), а значениеRGB - целое от 0 до 255.</p> <p><b>file = &lt; положениеФайла , " имяФайла " &gt;</b><br/> где положениеФайла - <code>server</code> или <code>client</code>,<br/> имяФайла - полный путь файла в формате системы, в которой находится файл.</p>                                                 |

Таблица 19. Запрос поиска изображений: допустимые значения характеристик (продолжение)

| Имя характеристики                  | Значения                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QbDrawFeatureClass или QbDraw       | <p><b>description = &lt; " строкаОписания " &gt;</b><br/> где строкаОписания - особая закодированная строка, описывающая файл размещения цветов. Формат строки описания:</p> <ol style="list-style-type: none"> <li>1. Dw,h задает внешние размеры самого изображения: w - ширина, h - высота.</li> <li>2. Rx,y,w,h,r,g,b задает, что прямоугольник шириной w и высотой h, левый верхний угол которого имеет координаты (x,y) относительно верхнего левого угла прямоугольника изображения, должен иметь значения цветов r (красный), g (зеленый) и b (синий).</li> <li>3. Двоеточие (:) используется как разделитель.</li> </ol> <p>Например, можно задать поиск размещение цветов (QbDrawFeatureClass) следующей строкой:</p> <p>QbDrawFeatureClass description= &lt;"D100,50:R0,0,50,50,255,0,0"</p> <p><b>file = &lt; положениеФайла , " имяФайла " &gt;</b><br/> где положениеФайла - server или client, имяФайла - полный путь файла в формате системы, в которой находится файл.</p> |
| QbTextureFeatureClass или QbTexture | <p><b>file = &lt; положениеФайла , " имяФайла " &gt;</b><br/> где положениеФайла - server или client, имяФайла - полный путь файла в формате системы, в которой находится файл.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

### Примеры запросов:

1. Поиск красного среднего цвета:  
QbColorFeatureClass color=<255,0,0>
2. Поиск с гистограммой из трех цветов - 10% красного, 50% зеленого и 40% синего:  
QbColorHistogramFeatureClass histogram= <(10, 255, 0, 0) (50, 0, 255, 0), (40, 0, 0, 255)>
3. Поиск среднего цвета и текстуры. Значение текстуры задается изображением в файле на клиенте. Вес текстуры вдвое больше, чем вес среднего цвета:

```
QbColorFeatureClass color=
<50, 50, 50> and QbTextureFeatureClass file=
<client, "\patterns\pattern1.gif"> weight=2.0
```

4. Поиск локального цвета:

```
QbDrawFeatureClass description=<"D100,50:R0,0,50,50,255,0,0">
```

5. Поиск красного среднего цвета и возврат не более пяти соответствий:

```
QbColorFeatureClass color=<255,0,0> and max=5
```

### Запуск запросов и оценка результатов поиска

Прикладные программы используют API поиска изображений для отправки запросов и оценки результатов поиска. Если информация в базе данных поиска изображений соответствует критерию поиска изображений, возвращается идентификатор соответствующих изображений. Этот идентификатор - динамический объект данных (DDO), соответствующий части изображения в объекте Content Manager.

## Установка соединения в QBIC

Поиск изображений содержит функции соединения с контент-сервером и отсоединения от него. В следующем примере показано, как соединиться с сервером поиска изображений с именем QBICSRV, используя ID пользователя QBICUSER и пароль PASSWORD.

### Java

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
... // необходимая обработка
dsQBIC.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TConnectQBIC.java), находится в каталоге CMBROOT\Samples\java\d1.

### C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
... // выполняем какую-то работу
dsQBIC.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TConnectQBIC.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

Соединение, предусмотренное для поиска изображений, позволяет прикладной программе соединиться с сервером поиска изображений.

После установления соединения ваша программа может использовать функции, требующие доступа к серверу поиска изображений, за исключением тех функций, которые не связаны с каталогами поиска изображений, например, `listDatabases`. Функция `openCatalog` требуется, чтобы открыть каталог и выполнить обработку. Функция `closeCatalog` вызывается по завершении обработки. В следующем примере показано, как соединиться, открыть каталог, закрыть каталог и отсоединиться.

#### Java

```
// ----- Создаем склад данных QBIC и соединяемся с ним
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
// ----- открываем каталог
dsQBIC.openCatalog("DEMO", "QBIC0725");
... // выполняем какую-либо обработку
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

#### C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "QBIC0725");
... // выполняем какую-то работу
dsQBIC.closeCatalog();
dsQBIC.disconnect();
```

## Получение списка серверов поиска изображений

У сервера поиска изображений есть функция получения списка серверов поиска изображений, с которыми он может соединяться. В следующем примере показано, как получить список серверов, содержащих объекты `DKServerInfoQBIC` (в виде объекта `DKSequentialCollection`). Получив объект `DKServerInfoQBIC`, вы можете затем получить имя сервера, имя хоста и номер порта.

## Java

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
.....
DKServerInfoQBIC pSV = null;
String strServerName = null;
String strHostName = null;
String strPortNumber = null;
pCol = (DKSequentialCollection)dsQBIC.listDataSources();
iter = pCol.createIterator();
while (iter.more()) {
 srvDef = (DKServerDefQBIC)iter.next();
 // обрабатываем каждый сервер, как требуется
}
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogQBIC.java), находится в каталоге CMBROOT\Samples\java\d1.

## C++

```
DKDatastoreQBIC dsQBIC;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKServerDefQBIC *pSV = 0;
DKString strServerName;
DKAny a;
long i = 0;
cout << "list servers" << endl;
a = dsQBIC.listDataSources();
pCol = (DKSequentialCollection*)((dkCollection*)a);
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
 i++;
 pSV = (DKServerDefQBIC*)((void*)(*pIter->next()));
 strServerName = pSV->getName();
 cout << "Server Name [" << i << "] - " << strServerName << endl;
 delete pSV;
}
delete pIter;
delete pCol;
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogQBIC.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.



## Получение списка баз данных, каталогов и характеристик поиска изображений

У DKDatastoreQBIC есть функция получения списка всех баз данных, каталогов и характеристик поиска изображений на сервере поиска изображений. Список возвращается в объекте DKSequentialCollection - собрании объектов DKIndexQBIC. Получив объект DKIndexQBIC, вы можете затем получить базу данных, каталог и имя характеристики. В следующем примере показано, как получить список баз данных, каталогов и характеристик.

### Java

```
// ----- Создаем склад данных и соединяемся с ним
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");

// ---- получаем список серверов
col = (DKSequentialCollection)dsQBIC.listDataSources();
iter = col.createIterator();
while (iter.more()) {
 srvDef = (DKServerDefQBIC)iter.next();
 // обрабатываем каждый сервер, как требуется
}

// ----- получаем список баз данных QBIC
col = (DKSequentialCollection)dsQBIC.listEntities();
iter = col.createIterator();
while (iter.more()) {
 dbDef = (DKDatabaseDefQBIC)iter.next();
 // ----- получаем список каталогов для базы данных
 col2 = (DKSequentialCollection)dbDef.listSubEntities();
 iter2 = col2.createIterator();
 while (iter2.more()){
 catDef = (DKCatalogDefQBIC)iter2.next();
 // ----- получаем список характеристик для каталога
 col3 = (DKSequentialCollection)catDef.listAttrs();
 iter3 = col3.createIterator();
 while (iter3.more()){
 featDef = (DKFeatureDefQBIC)iter3.next();
 // обрабатываем характеристики, как требуется
 }
 }
}
dsQBIC.disconnect();
dsQBIC.destroy();
.....
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogQBIC.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
DKDatastoreQBIC dsQBIC;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKSequentialCollection *pCol2 = 0;
dkIterator *pIter2 = 0;
DKSequentialCollection *pCol3 = 0;
dkIterator *pIter3 = 0;
DKDatabaseDefQBIC *pEntDB = 0;
DKCatalogDefQBIC *pEntCat = 0;
DKString strCatName;
DKString strDBName;
DKString strFeatName;
DKFeatureDefQBIC *pAttr = 0;
DKAny a;
DKAny *pA = 0;
long i = 0;
long j = 0;
long k = 0;
cout << "connecting to datastore" << endl;
dsQBIC.connect("QBICSRV","USERID","PW");
cout << "list databases " << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsQBIC.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
 i++;
 pEntDB = (DKDatabaseDefQBIC*)((void*)(*pIter->next()));
 strDBName = pEntDB->getName();
 cout << "database name [" << i << "] - " << strDBName << endl;
 cout << " list catalogs for DB " << strDBName << endl;
 pCol2 = (DKSequentialCollection*)((dkCollection*)pEntDB->listSubEntities());
 pIter2 = pCol2->createIterator();
 j = 0;
 while (pIter2->more() == TRUE)
 {
 j++;
 pA = pIter2->next();
 pEntCat = (DKCatalogDefQBIC*) pA->value();
 strCatName = pEntCat->getName();
 cout << "catalog name [" << j << "] - " << strCatName << endl;
 pCol3 = (DKSequentialCollection*)((dkCollection*)pEntCat->listAttrs());
 pIter3 = pCol3->createIterator();
 k = 0;
 while (pIter3->more() == TRUE)
 {
 k++;
 pA = pIter3->next();
 }
 }
}
// продолжение следует...
```

### С++ (продолжение)

```
pAttr = (DKFeatureDefQBIC*) pA->value();
cout << " Attribute name [" << k << "] - "
 << pAttr->getName() << endl;
cout << " datastoreName " << pAttr->datastoreName()
 << endl;
cout << " datastoreType " << pAttr->datastoreType()
 << endl;
cout << " attributeOf " << pAttr->getEntityName()
 << endl;
delete pAttr;
}
delete pIter3;
delete pCol3;
delete pEntCat;
}
cout << " " << j << " features listed for catalog: "
 << strCatName << endl;
delete pIter2;
delete pCol2;
delete pEntDB;
}
delete pIter;
delete pCol;
cout << i << " databases listed" << endl;
dsQBIC.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogQBIC.cpp), находится в каталоге Cmbroot/Samples/cpp/dl.

## Представление информации поиска изображений при помощи объекта DDO

Объект DDO, связанный с DKDatastoreQBIC, содержит определенную информацию, которая представляет результаты поиска изображений. DDO, создаваемый при обработке запроса изображений, соответствует части элемента, содержащей изображение; у него есть следующий набор стандартных атрибутов:

### DKDLITEMID

ID элемента, который содержит эту часть изображения. Этот ID элемента служит для получения всего элемента с контент-сервера.

### DKPARTNO

Целое, номер части этой части изображения. Вместе с ID элемента служит для получения этой части с контент-сервера.

### DKREPTYPE

Строка типа представления (RepType). Значение по умолчанию - FRN\$NULL. Этот атрибут зарезервирован.

## **DKRANK**

Целое, ранг релевантности этой части по отношению к результатам запроса изображения. Ранг принимает значения от 0 до 100. Чем выше ранг, тем лучше соответствие.

PID для DDO поиска изображений содержит следующую информацию:

**тип контент-сервера**

QBIC.

**имя контент-сервера**

Имя сервера, используемое для соединения с контент-сервером.

**ID**

Порядковый (начиная с нуля) номер DDO в наборе результатов.

По принятому соглашению, значение этого атрибута - всегда объект.

## **Работа с запросами изображений**

В этом разделе описано, как запускать и оценивать запросы изображений.

### **Запуск запроса изображений**

При помощи экземпляра `dkQuery` из `DKDatastoreQBICK` можно создать объект запроса для выполнения запроса и получения результатов. В следующем примере показано, как создать объект запроса изображений и затем запустить этот запрос. После выполнения запроса результаты возвращаются в собрании `DKResults`.

## Java

```
// ----- генерируем строку запроса, затем создаем склад данных и
// соединяемся с ним
String cmd = "QbColor color=<255, 0, 0>";
DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
// ----- открываем каталог
dsQBIC.openCatalog("DEMO", "qbic0725");

... // необходимая обработка

// ----- создаем запрос и запускаем его
dkQuery pQry = dsQBIC.createQuery(cmd, DK_IMAGE_QL_TYPE, parms);
pQry.execute(parms);
// ----- получаем и обрабатываем результаты
DKResults pResults = (DKResults)pQry.result();
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
 item = (DKDDO)pIter.next();
 // Обработка DKDDO
}
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
...
```

Полная программа примера, из которой взят приведенный фрагмент (SampleIQryQBIC.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
DKDatastoreQBIC* dsQBIC;
dsQBIC = new DKDatastoreQBIC();
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");
dsQBIC->openCatalog("DEMO", "qbic0725");
DKAny* element;
DKDDO* item;
DKString cmd = "QbColor color=<255, 0, 0>";
dkQuery* pQry = dsQBIC->createQuery(cmd);
pQry->execute();
DKAny any = pQry->result();
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more())
{
 element = pIter->next();
 item = (DKDDO*)element->value();
 // Обработка DKDDO
 ...
}
delete pIter;
delete pResults;
delete pQry;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TSampleIQryQBIC.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

### **Запуск запроса изображений с контент-сервера**

Другой вариант запуска запроса - использование функции execute объекта DKDatastoreQBIC. Результаты возвращаются в объекте dkResultSetCursor. В следующем примере показано, как запустить запрос изображений на контент-сервере. Результаты возвращаются в объекте dkResultSetCursor.

## Java

```
// ----- генерируем строку запроса, затем создаем склад данных и
// соединяемся с ним
String cmd = "QbColorFeatureClass color=<255, 0, 0>";

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");
// ----- Выполняем запрос со склада данных
dkResultSetCursor pCur = dsQBIC.execute(cmd, DK_IMAGE_QL_TYPE, parms);
while (pCur.isValid()) {
 item = pCur.fetchNext();
 // Обработка DKDDO
}
pCur.destroy();
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteQBIC.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
DKDatastoreQBIC* dsQBIC;
dsQBIC = new DKDatastoreQBIC();
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");
cout << "datastore connected" << endl;
dsQBIC->openCatalog("DEMO", "qbic0725");
DKString cmd = "QbColorFeatureClass color=<255, 0, 0>";
dkResultSetCursor* pCur = dsQBIC->execute(cmd);
DKDDO* item = 0;
while (pCur->isValid())
{
 item = pCur->fetchNext();
 if (item != 0)
 {
 // Обработка DKDDO
 ...
 delete item;
 }
}
delete pCur;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteQBIC.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

### **Оценка запроса изображений с контент-сервера**

DKDatastoreQBIC предоставляет также функцию оценки запроса. В следующем примере показано, как оценить запрос изображений с контент-сервера. Результаты возвращаются в собрании DKResults.



## Java

```
// ----- генерируем строку запроса, затем создаем склад данных и
// соединяемся с ним
String cmd = "QbColorFeatureClass color=<255, 0, 0>";
DKNameValuePair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");

// ----- используем evaluate для запуска запроса
DKResults pResults=(DKResults) dsQBIC.evaluate(cmd,DK_IMAGE_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
 item = (DKDDO)pIter.next();
 ... // Обработка DKDDO
}
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

## C++

```
DKDatastoreQBIC* dsQBIC;
dsQBIC = new DKDatastoreQBIC();
dsQBIC->connect("QBICSRV", "QBICUSER", "PASSWORD");
dsQBIC->openCatalog("DEMO", "qbic0725");
DKAny* element;
DKDDO* item;
DKString cmd = "QbColor color=<255, 0, 0>";
DKAny any = dsQBIC->evaluate(cmd);
DKResults* pResults = (DKResults*)((dkCollection*) any);
dkIterator* pIter = pResults->createIterator();
while (pIter->more())
{
 element = pIter->next();
 item = (DKDDO*)element->value();
 // Обработка DKDDO
 ...
}
delete pIter;
delete pResults;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

## Использование механизма поиска изображений

Сервер поиска изображений позволяет задать запрос на основе одной из следующих характеристик: средний цвет, доли цветов, размещение цветов и

текстуры. Можно также задать несколько характеристик в одном запросе. Результаты запроса содержат ID элемента, номер части, тип представления и информацию о ранге. Эта информация позволяет создать объект XDO для получения содержимого изображения.

### **Загрузка данных для индексирования и последующего поиска изображений**

Чтобы загрузить на сервер Content Manager данные, которые нужно индексировать при помощи сервера поиска изображений, нужно создать индексный класс Content Manager, базу данных поиска изображений и каталог поиска изображений. База данных - это собрание каталогов поиска изображений. Каталог содержит информацию о визуальных характеристиках изображений.

Для индексации в каталог надо добавить характеристики поиска изображений. Следует добавить в каталог все поддерживаемые характеристики.

Когда вы будете создавать базу данных и каталог поиска изображений, сервер поиска изображений должен быть запущен. Убедитесь, что ваша среда правильно настроена.

После загрузки данных в Content Manager вы сможете поместить изображение в очередь изображений. В программе управления системой выберите **Очередь обработки изображений**. По завершении индексирования можно будет выполнять поиск изображений.

### **Индексирование существующего XDO при помощи механизмов поиска**

Можно проиндексировать существующий XDO при помощи заданного механизма поиска. В следующем примере вызывается функция `setToBeIndexed` класса `DKBlobDL`.

## Java

```
try
{
 // ----- Создаем склад данных и соединяемся с ним
 DKDatastoreDL dsDL = new DKDatastoreDL();
 dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");

 // ----- создаем XDO и PID и задаем атрибуты
 DKBlobDL axdo = new DKBlobDL(dsDL);
 DKPidXDODL apid = new DKPidXDODL();
 apid.setPartId(partId);
 apid.setPrimaryId(itemId);
 axdo.setPidObject(apid);

 // ----- задаем информацию механизма поиска
 DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
 aSrchEx.setSearchEngine("SM");
 aSrchEx.setSearchIndex("TM-TMINDEX");
 aSrchEx.setSearchInfo("ENU");
 axdo.setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
 // ----- вызываем setToBeIndexed для XDO
 axdo.setToBeIndexed();

 dsDL.disconnect();
 dsDL.destroy();
}
catch(DKException exc)
{
 ... // обработка DKDDO
}
catch (Exception exc) {
 ... // обработка исключительных ситуаций
}
```

**C++**

```
void main(int argc, char *argv[])
{
 DKDatastoreDL dsDL;
 DKString itemId, repType;
 int partId;
 itemId = "N2JJBERBQFK@WTVL";
 repType = "FRN$NULL";
 partId = 10;
 if (argc == 1)
 {
 cout<<"invoke: indexPartxs <partId> <repType> <itemId>"<<endl;
 cout<<" no parameter, following default will be provided:"<<endl;
 cout<<"The supplied default partId = "<<partId<<endl;
 cout<<"The supplied default repType = "<<repType<<endl;
 cout<<"The supplied default itemId = "<<itemId<<endl;
 }
 else if (argc == 2)
 {
 partId = atoi(argv[1]);
 cout<<"you enter: indexPartxs "<<argv[1]<<endl;
 cout<<"The supplied default repType = "<<repType<<endl;
 cout<<"The supplied default itemId = "<<itemId<<endl;
 }
 else if (argc == 3)
 {
 partId = atoi(argv[1]);
 repType = DKString(argv[2]);
 cout<<"you enter: indexPartxs "<<argv[1]<<" "<<argv[2]<<endl;
 cout<<"The supplied default itemId = "<<itemId<<endl;
 }
 else if (argc == 4)
 {
 partId = atoi(argv[1]);
 repType = DKString(argv[2]);
 itemId = DKString(argv[3]);
 cout<<"you enter: indexPartxs "<<argv[1]<<" "<<argv[2]<<" "<<argv[3]<<endl;
 }

 cout << "connecting Datastore" << endl;
 try
 {
 // подставьте в следующую строку библиотечный сервер,
 // ID пользователя, пароль
 dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
 cout << "datastore connected" << endl;

 DKBlobDL* axdo = new DKBlobDL(&dsDL);
 DKPidXDODL* apid = new DKPidXDODL;
 apid ->setPartId(partId);
 apid ->setId(itemId);
 }
 // продолжение следует...
```

## C++ (продолжение)

```
axdo ->setPid(apid);
axdo ->setRepType(repType);
cout<<"itemId= "<<(axdo->getPid())->getId()<<endl;
cout<<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;
cout<<"repType= "<<axdo->getRepType()<<endl;

//--- задать searchEngine -----
cout<<"set search engine and setToBeIndexed()"<<endl;
DKSearchEngineInfoDL aSrchEx;
aSrchEx.setSearchEngine("SM");
aSrchEx.setSearchIndex("TM-TMINDEX");
aSrchEx.setSearchInfo("ENU");
axdo->setExtension("DKSearchEngineInfoDL", (dkExtension*)&aSrchEx);
axdo->setToBeIndexed();
cout<<"setToBeIndexed() done..."<<endl;

delete apid;
delete axdo;
dsDL.disconnect();
cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
 cout << "Error id" << exc.errorId() << endl;
 cout << "Exception id " << exc.exceptionId() << endl;
 for(unsigned long i=0;i< exc.textCount();i++)
 {
 cout << "Error text:" << exc.text(i) << endl;
 }
 for (unsigned long g=0;g< exc.locationCount();g++)
 {
 const DKExceptionLocation* p = exc.locationAtIndex(g);
 cout << "Filename: " << p->fileName() << endl;
 cout << "Function: " << p->functionName() << endl;
 cout << "LineNumber: " << p->lineNumber() << endl;
 }
 cout << "Exception Class Name: " << exc.name() << endl;
}
cout << "done ..." << endl;
}
```

## Использование комбинированного запроса

*Комбинированный запрос* представляет собой сочетание параметрических и текстовых запросов, с заданной сферой или без нее. *Сфера* - объект DKResults, сформированный в предыдущем параметрическом или текстовом запросе. Результат является пересечение сфер и результатов каждого запроса. Таким образом, при неточной формулировке запроса со сферой результаты отдельных запросов могут не пересекаться, и результат комбинированного поиска окажется пустым.

Если есть хотя бы один параметрический и один текстовый запрос, у полученного DDO будет атрибут DKRANK , означающий высший ранг соответствия части документа.

**Ограничение:** Для каждого запроса в комбинированном запросе надо использовать свое соединение с механизмом поиска; нельзя направлять множественные запросы через одно соединение.

### Комбинированные параметрические и текстовые запросы

Чтобы выполнить комбинированный запрос, составленный из одного параметрического и одного текстового запроса, без сферы, надо создать объект комбинированного запроса и передать два запроса как входные параметры для выполнения комбинированного запроса. Например:

#### Java

```
// ----- Создаем склад данных Content Manager до версии 8 и соединяемся
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Создаем склад данных текстового поиска и соединяемся с ним
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' '); // TM - локальный алиас
... // сервера механизма текстового поиска

// ----- генерируем строку параметрического поиска и создаем запрос
String pquery="SEARCH=(INDEX_CLASS=GRANDPA, COND=(DLSEARCH_Date > 1994));";
DKParametricQuery pq =
(DKParametricQuery) dsDL.createQuery(pquery, DK_CM_PARAMETRIC_QL_TYPE, null);

// ----- генерируем строку текстового поиска и создаем запрос
String tquery = "SEARCH=(COND=(Tivoli)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
(DKTextQuery) dsTS.createQuery(tquery, DK_CM_TEXT_QL_TYPE, null);

// ----- создаем комбинированный запрос
DKCombinedQuery cq = new DKCombinedQuery();

// ----- Включаем запросы в DKNVPair как входные параметры
DKNVPair par[] = new DKNVPair[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].setName(DK_PARM_END); // сигнал конца списка параметров

// ----- Выполняем комбинированный запрос
cq.execute(par);

// ----- получаем результаты
DKResults res = (DKResults) cq.result();
if (res != null) {
 ... // обрабатываем результаты
}
```

## C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

DKDatastoreTS dsTS;
// TM - локальный алиас для сервера механизма текстового поиска
dsTS.connect("TM","", ' ');
// создать параметрический запрос
DKString pquery="SEARCH=(INDEX_CLASS=GRANDPA,COND=(DLSEARCH_Date > 1994));";
DKParametricQuery* pq =
 (DKParametricQuery*) dsDL.createQuery(pquery,DK_PARAMETRIC_QL_TYPE, NULL);

// создать текстовый запрос
DKString tquery = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery* tq =
 (DKTextQuery*) dsTS.createQuery(tquery,DK_TEXT_QL_TYPE, NULL);

// создать комбинированный запрос
DKCombinedQuery* cq = new DKCombinedQuery();

// включить запросы в DKNVPair как входные параметры
DKNVPair par[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
// сигнал конца списка параметров
par[2].setName(DK_PARM_END);

// выполнить комбинированный запрос
cq->execute(par);

// получить результаты
DKAny any = cq->result();
DKResults* res = (DKResults*) any.value();
if (res != NULL) {
 // обработать результаты
 ...
}
```

Последний оператор `if` нужен для того, чтобы убедиться, что объект `DKResults` не пуст (`null`).

### Использование сферы

Если у вас есть объект `DKResults`, который вы хотите использовать в качестве области поиска, передайте его как дополнительный параметр запроса. В следующем примере показано использование объекта `DKResults` в качестве сферы поиска комбинированного запроса:

## Java

```
// ----- Эта область поиска -- результат параметрического запроса
DKResults scope;
// ----- Эта область поиска -- результат предыдущего текстового запроса
DKResults tscope;

// ----- Включаем запрос в массив DKNVPairs как входные параметры
DKNVPair par[] = new DKNVPair[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);

// ----- Выполняем комбинированный запрос
cq.execute(par);
....
```

## C++

```
DKResults* scope; // предполагается, что это сфера,
 // которая была инициализирована в результате
 // некоторого параметрического запроса
DKResults* tscope // предполагается, что это сфера,
 // которая была инициализирована в результате
 // некоторого текстового запроса

...
// включить запрос в DKNVPair как входные параметры
DKNVPair par[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);
// выполнить комбинированный запрос
cq->execute(par);
...
```

Кроме того, результаты одного комбинированного запроса можно использовать как сферу для другого комбинированного запроса; полученные результаты тоже иногда можно запрашивать.

## Ранжирование

Если комбинированный запрос содержит хотя бы один текстовый запрос, у полученного DDO будет атрибут DKRANK. Этот атрибут не сохраняется, а



вычисляется каждый раз механизмом текстового поиска. Это значение ранга отвечает высшему рангу части в документе, удовлетворяющей условиям текстового поиска.

### **Советы**

Если у вас есть несколько параметрических запросов и сфер, эффективнее выполнить один полный запрос. Это верно и для текстовых запросов.

Опция запроса "MAX\_RESULTS=nn" ограничивает число возвращаемых результатов. Обычно эта опция более применима к текстовым запросам, поскольку результат сортируется в порядке убывания ранга. Если эта опция задана равной, например, 10, это означает, что вызывающая программа запрашивает только 10 результатов с лучшими соответствиями.

При параметрических запросах смысл опции запроса "MAX\_RESULTS=nn" иной. Поскольку ранг при этом не определяется, вызывающая программа получит первые 10 результатов. Именно для этих результатов вычисляется пересечение с результатами текстового запроса. Поэтому в сочетании параметрических и текстовых запросов нежелательно задавать опцию запроса "MAX\_RESULTS=nn" для параметрического запроса.

## **О функциях рабочего потока и рабочего комплекта в ранних версиях Content Manager**

В этом разделе описываются функции рабочего потока и рабочего комплекта в ранних версиях Content Manager.

### **О службе рабочего потока в ранних версиях Content Manager**

*Рабочий комплект* - это контейнер с документами и папками, которые вы хотите обработать. *Рабочий поток* - это упорядоченный набор рабочих комплектов, представляющий определенный бизнес-процесс. Папки и документы переносятся между рабочими комплектами одного рабочего потока, позволяя реализовать в ваших программах простые бизнес-модели и задать последовательные этапы обработки до завершения работы.

Модель рабочего потока в Content Manager подчиняется следующим правилам:

- Рабочий комплект не обязательно находится в рабочем потоке.
- Рабочий комплект может находиться сразу в нескольких рабочих потоках.
- Рабочий комплект может несколько раз входить в один рабочий поток.
- Документ или папка могут находиться лишь в одном рабочем потоке в один момент времени.
- Документ или папка могут находиться в рабочем комплекте, не находящемся в рабочем потоке.

API Enterprise Information Portal поддерживают классы для работы с рабочими потоками Content Manager.

Класс DKWorkflowServiceDL представляет службу рабочих потоков Content Manager. Этот класс позволяет запускать, изменять, удалять и направлять документ или папку и завершать работу с документом или папкой в рабочем потоке. Кроме того, класс DKWorkflowServiceDL можно использовать для получения информации о рабочих комплектах и рабочих потоках.

Классы DKWorkflowDL и DKWorkBasketDL - объектно-ориентированные представления элемента рабочего потока и элемента рабочего комплекта соответственно.

### Установление соединения

До того, как можно будет использовать службу рабочих потоков, надо установить соединение с сервером Content Manager. У контент-сервера есть функции для соединения и разъединения.

В следующем примере показано, как соединиться с сервером Content Manager с именем LIBSRVRN, используя ID пользователя FRNADMIN и пароль PASSWORD.

#### Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
... // необходимая обработка
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TListWorkFlowWFS.java), находится в каталоге CMBROOT\Samples\java\d1.

#### C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
... // выполняем какую-то работу
dsDL.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TListWorkFlowWFS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

### Создание рабочего потока

DKWorkflowServiceDL служит для создания рабочего потока. В типичном случае для этого надо выполнить следующие шесть шагов:

1. Создайте экземпляр DKWorkflowDL.

2. Задайте имя рабочего потока ("GOLF").
3. Задайте пустую последовательность рабочих комплектов ("NULL"), показывающую, что в этом рабочем потоке нет рабочих комплектов.
4. Задайте привилегию ("Все привилегии").
5. Задайте диспозицию (DK\_WF\_SAVE\_HISTORY).
6. Вызовите функцию добавления add ().

В следующем примере выполняются шесть шагов по созданию рабочего потока.

#### Java

```
// ----- Создаем склад данных и службы рабочего потока CM
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);

// ----- задаем опции доступа и соединяемся
Object input_option = new Integer(DK_SS_CONFIG);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");

// ----- создаем рабочий поток CM
DKWorkflowDL newwf = new DKWorkflowDL(wfDL);
newwf.setName("Process claim");
newwf.setWorkBasketSequence((dkCollection *)NULL);
newwf.setAccessList("All Privileges");
newwf.setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf.add();
... // Необходимая обработка
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TCreateDelWorkflow.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_DL_OPT_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkflowDL * newwf = new DKWorkflowDL(&wfDL);
newwf->setName("GOLF");
newwf->setWorkBasketSequence((dkCollection *)NULL);
newwf->setAccessList("All Privileges");
newwf->setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf->add();
... // выполняем какую-то работу
dsDL.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TCreateDelWorkflowWFS.cpp), находится в каталоге Cmbroot/Samples/cpp/dl.

**Внимание:** Соединившись с контент-сервером как обычный пользователь (DK\_SS\_NORMAL), вы не сможете определить рабочий поток после соединения. Поэтому в этом примере используется DK\_SS\_CONFIG.

### Получение списка рабочих потоков

У DKWorkflowServiceDL есть функция получения списка рабочих потоков в системе, как показано в следующем примере. Список возвращается в последовательном собрании объектов DKWorkflowDL.

## Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- получаем список рабочих потоков CM
DKSequentialCollection wfList =
 (DKSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
{
 dkIterator pIter = wfList.createIterator();
 DKWorkflowDL pwf1;
 while (pIter.more())
 {
 pwf1 = (DKWorkflowDL)pIter.next();
 pwf1->retrieve();
 ... // необходимая обработка
 }
}
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TListWorkFlowWFS.java), находится в каталоге CMBROOT\Samples\java\d1.

## C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1 =
 (DKSequentialCollection *)wfDL.listWorkFlows();
if (wfList1 != NULL)
{
 dkIterator * pIter1 = wfList1->createIterator();
 DKWorkflowDL * pwf1;
 while (pIter1->more())
 {
 pwf1 = (DKWorkflowDL *)((void*)(*pIter1->next()));
 pwf1->retrieve();
 ... // выполняем какую-то работу
 delete pwf1;
 }
}
dsDL.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TListWorkFlowWFS.cpp), находится в каталоге Cmbroot/Samples/cpp/d1.

## Создание рабочего комплекта Content Manager

DKWorkflowServiceDL служит для создания рабочего комплекта. В типичном случае для этого надо выполнить следующие шаги:

1. Создание экземпляра DKWorkBasketDL.
2. Задайте имя рабочего комплекта (Hot Items).
3. Задайте привилегию (Все привилегии).
4. Вызовите функцию добавления (add).

В следующем примере выполняются эти шаги по созданию рабочего комплекта. Соединившись с контент-сервером как обычный пользователь (DK\_SS\_NORMAL), вы не сможете определить рабочий комплект после соединения. Поэтому в этом примере используется DK\_SS\_CONFIG.

### Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
Object input_option = new Integer(DK_SS_CONFIG);
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- создаем рабочий комплект CM и задаем свойства
DKWorkBasketDL newwb = new DKWorkBasketDL(wfDL);
newwb.setName("Hot Items");
newwb.setAccessList("All Privileges");
newwb.add();
... // необходимая обработка
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TCreateDelWorkBasket.java), находится в каталоге CMBROOT\Samples\java\dl.

## C++

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_DL_OPT_ACCESS,input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkBasketDL * newwb = new DKWorkBasketDL(&wfDL);
newwb->setName("Hot Items");
newwb->setAccessList("All Privileges");
newwb->add();
... // выполняем какую-то работу
dsDL.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TCreateDelWorkBasket.cpp), находится в каталоге CMBROOT\Samples\cpp\d1.

## Получение списка рабочих комплектов

У DKWorkflowServiceDL есть функция получения списка рабочих комплектов в системе, как показано в следующем примере. Список возвращается в последовательном собрании объектов DKWorkBasketDL.

## Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection wbList =
 (DKSequentialCollection)wfDL.listWorkBaskets();
if (wbList != null)
{
 dkIterator pIter = wbList.createIterator();
 DKWorkBasketDL pwbl;
 while (pIter.hasMore())
 {
 pwbl = (DKWorkBasketDL)pIter.next();
 pwbl->retrieve();
 ... // выполняем какую-то работу
 }
}
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TListWorkBasketWFS.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1=
 (DKSequentialCollection *)wfDL.listWorkBaskets();
if (wbList1 != NULL)
{
 dkIterator * pIter1 = wbList1->createIterator();
 DKWorkBasketDL * pwbl;
 while (pIter->more())
 {
 pwbl = (DKWorkBasketDL *)((void*)(*pIter1->next()));
 pwbl->retrieve();
 ... // выполняем какую-то работу
 delete pwbl;
 }
}
dsDL.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TListWorkBasketWFS.cpp), находится в каталоге Cmbroot/Samples/cpp/dl.

### **Вывод списка элементов в рабочем потоке ранней версии Content Manager**

У DKWorkflowServiceDL есть функция получения списка ID элементов в рабочем потоке, как показано в следующем примере. Список возвращается в последовательном собрании объектов DKString.



## Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- получаем список рабочих потоков CM
KSequentialCollection wfList = (DKSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
{
 dkIterator pIter = wfList.createIterator();
 while (pIter.more())
 {
 DKWorkflowDL pwf1 = (DKWorkflowDL)pIter.next();
 // ----- получаем список элементов в рабочем потоке CM
 DKSequentialCollection itemList = (DKSequentialCollection)pwf1.listItemIDs();
 if (itemList != null)
 {
 dkIterator iter1 = itemList.createIterator();
 String itemid;
 while (iter1.more())
 {
 itemid = (String)iter1.next();
 // ----- обработка элементов с использованием ID элементов
 }
 }
 }
}
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TListItemWFS.java), находится в каталоге CMBROOT\Samples\java\dl.

**C++**

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkflowDL * wf = new DKWorkflowDL(&wfDL, (char *)itemIDWF);
wf->retrieve;
DKSequentialCollection * pColDoc1 = (DKSequentialCollection *)wf->listItemIDs();
if (pColDoc1 != NULL)
{
 dkIterator* pIterDoc1 = pColDoc1->createIterator();
 DKString DocID1;
 while (pIterDoc1->more() == TRUE)
 {
 DocID1 = (DKString)(*pIterDoc1->next());
 ... // выполняем какую-то работу
 }
}
dsDL.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TListItemWFS.cpp), находится в каталоге Cmbroot/Samples/cpp/dl.

### **Выполнение рабочего потока ранней версии Content Manager**

У DKWorkflowServiceDL есть функции для выполнения рабочего потока. В следующем примере демонстрируется, как запустить элемент в рабочем потоке, как направить элемент в рабочий комплект и как завершить работу с элементом в рабочем потоке. Чтобы использовать этот пример, нужно заменить следующие значения:

- Задать нужный ID элемента вместо EP8L80R9MHH##QES.
- Задать нужный ID рабочего потока вместо HI7MOPALUPFQ1U47.
- Задать нужный ID рабочего комплекта вместо E3PP1UZ0ZUFQ1U3M.

## Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkflowServiceDL wfDL = new DKWorkflowServiceDL(dsDL);
DKString itemID = new String("EP8L80R9MHH##QES");
DKString itemIDWF = new String("HI7MOPALUPFQ1U47");
DKString itemIDWB = new String("E3PP1UZ0ZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkflowItem(itemID, // ID элемента
 itemIDWF, // IDWB элемента
 NULL, // по умолчанию (первый рабочий комплект)
 TRUE, // перезагрузка
 DK_WIP_DEFAULT_PRIORITY // начальный приоритет
);
...
wfDL.routeWipItem(itemID, // ID элемента
 itemIDWF, // IDWB элемента
 TRUE, // перезагрузка
 DK_NO_PRIORITY_CHANGE // начальный приоритет
);
...
wfDL.completeWorkflowItem(itemID);
dsDL.disconnect();
dsDL.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TProcessWFS.java), находится в каталоге CMBROOT\Samples\java\d1.

## C++

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
DKString itemID = DKString("EP8L80R9MHH##QES");
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
DKString itemIDWB = DKString("E3PP1UZ0ZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkflowItem(itemID, // ID элемента
 itemIDWF, // IDWB элемента
 NULL, // по умолчанию (первый рабочий комплект)
 TRUE, // перегрузка
 DK_WIP_DEFAULT_PRIORITY // начальный приоритет
);
...
wfDL.routeWipItem(itemID, // ID элемента
 itemIDWF, // IDWB элемента
 TRUE, // перегрузка
 DK_NO_PRIORITY_CHANGE // начальный приоритет
);
...
wfDL.completeWorkflowItem(itemID);
dsDL.disconnect();
```

Полная программа примера, из которой взят приведенный фрагмент (TProcessWFS.cpp), находится в каталоге Cmbroot/Samples/cpp/dl.

## Работа с OnDemand

Enterprise Information Portal поддерживает соединитель и соответствующие классы для доступа к содержимому серверов Content Manager OnDemand. Классы и API OnDemand позволяют:

- Соединяться с серверами OnDemand и отсоединяться от них.
- Получать списки групп программ и полей групп программ.
- Получать списки папок OnDemand.
- Запрашивать группу программ.
- Искать и получать документы с использованием интерфейса папок и групп программ.
- Рассматривать папки OnDemand как собственные объекты при объединенном поиске.
- Выполнять синхронный и асинхронный поиск в режиме групп программ или папок.
- Получать документы OnDemand целиком или их сегменты.
- Получать данные логического просмотра заданного документа OnDemand.
- Получать группу ресурсов заданного документа OnDemand.

- Получать данные комментариев заданного документа OnDemand.
- Создать и изменять комментарии.

**Ограничение:** OnDemand не поддерживает механизм текстового поиска, поиск QVIC и комбинированные запросы.

## Представление серверов и документов OnDemand

Контент-сервер Content Manager OnDemand в прикладной программе Enterprise Information Portal представляют при помощи DKDatastoreOD, а документ OnDemand - как объект DDO при помощи DKDDO. Объекты DDO OnDemand содержат следующую информацию:

- Имена и значения атрибутов документа
- Данные и комментарии документа (представленные как DKParts)
- Собрание логических представлений для документа
- Данные групп ресурсов

Атрибуты документа OnDemand хранятся в DKDDO как свойства. Сегменты и примечания документа OnDemand хранятся как DKParts.

Остальные данные документа (группы ресурсов и представления, фиксированные и логические) хранятся как особые свойства в объекте DDO OnDemand; их имена зарезервированы для OnDemand:

### DKViews

Собрание логических представлений

### DKFixedView

Содержит информацию фиксированного представления

### DKResource

Содержит данные групп ресурсов

### Примечания:

1. Администратор Enterprise Information Portal должен определить допустимый соединитель OnDemand, задав строку в поле **Дополнительные параметры** на странице **Параметры инициализации**. Эта строка должна иметь вид: `ТИП_ОБЪЕКТА=ШАБЛОНЫ;;` Обязательно введите две точки с запятой.
2. В Enterprise Information Portal Версии 8.2, DKViews, DKFixedView и DKResource заменяют соответственно DKViewDataOD, DKFixedViewDataOD и DKResourceGrpOD.

## Соединение с сервером OnDemand и отсоединение от него

Чтобы зарегистрироваться на сервере OnDemand, передайте ему при помощи метода connect имя сервера (например, `ODServer.mycompany.com`), ID пользователя и пароль.

#### Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
System.out.println("connecting to datastore...");
dsOD.connect(ODServer, UserID, Password, "");
```

#### C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();
cout << "соединение со складом данных ..." << endl;
dsOD->connect(ODServer, UserID, Password, "");
```

Отсоединитесь от сервера OnDemand, воспользовавшись методом `disconnect`.

#### Java

```
System.out.println("отсоединение от склада данных ...");
dsOD.disconnect();
dsOD.destroy(); // Завершаем работу со складом данных
```

#### C++

```
cout << "отсоединение от склада данных ..." << endl;
dsOD->disconnect();
delete dsOD;
```

## Получение список для OnDemand

Можно получить список групп программ и папок для серверов OnDemand.

### Вывод групп программ

Можно получить список групп программ в OnDemand при помощи метода `listEntities()` из `DKDatastoreOD`. В следующем примере показано, как применять этот метод:

## Java

```
...
pCol = (DKSequentialCollection) dsOD.listEntities(); // получаем группы
pIter = pCol.createIterator(); // программ
i = 0;
while (pIter.more() == true)
{
 i++;
 agDef = (DKAppGrpDefOD)pIter.next();
 strAppGrp = agDef.getName();
 System.out.println(" app grp name[" + i + "]: " + strAppGrp);
 System.out.println(" show attributes for " + strAppGrp + " app grp - ");
}
...
```

## C++

```
// ----- Вывод групп программ
// ----- Сначала получаем группы
pCol = (DKSequentialCollection*)dsOD.listEntities();
pIter = pCol->createIterator();
int i = 0;
// ----- Обрабатываем список
while (pIter->more() == TRUE)
{
 i++;
 agDef = (DKAppGrpDefOD*)((void*)(*pIter->next()));
 strAppGrp = agDef->getName();
 cout << " app group name[" << i << "]: ==>" << strAppGrp << endl;
}
...
```

В следующем примере показано получение информации атрибутов для каждой группы программ:

## Java

```
...
pCol2 = (DKSequentialCollection) dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2.createIterator();
j = 0;

while (pIter2.more() == true)
{
 j++;
 attrDef = (DKFieldDef0D)pIter2.next();
 System.out.println(" Имя атрибута[" + j + "]: " + attrDef.getName());
 System.out.println(" datastoreType: " + attrDef.datastoreType());
 System.out.println(" attributeOf: " + attrDef.getEntityName());
 System.out.println(" тип: " + attrDef.getType());
 System.out.println(" размер: " + attrDef.getSize());
 System.out.println(" id: " + attrDef.getId());
 System.out.println(" nullable: " + attrDef.isNullable());
 System.out.println(" точность: " + attrDef.getPrecision());
 System.out.println(" масштаб: " + attrDef.getScale());
 System.out.println(" stringType: " + attrDef.getStringType());
}

System.out.println(" " + j + " атрибутов указано для " +
 strAppGrp + " app grp\n");
...
```



## C++

```
// ----- Получаем атрибуты для каждого объекта (группы программ)
pCol2 = (DKSequentialCollection*)dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2->createIterator();
int j = 0;
// ----- Список атрибутов
while (pIter2->more() == TRUE)
{
 j++;
 attrDef = (DKFieldDefOD*)(void*)(*pIter2->next());
 cout << " attribute name[" << j << "]: ==>" << attrDef->getName()
 << endl;
 cout << " datastore type: " << attrDef->datastoreType() << endl;
 cout << " attribute of: " << attrDef->getEntityName() << endl;
 cout << " type: " << attrDef->getType() << endl;
 cout << " size: " << attrDef->getSize() << endl;
 cout << " ID: " << attrDef->getId() << endl;
 cout << " precision: " << attrDef->getPrecision() << endl;
 cout << " scale: " << attrDef->getScale() << endl;
 cout << " stringType: " << attrDef->getStringType() << endl;
 cout << " nullable: " << attrDef->isNullable() << endl;
 cout << " queryable: " << attrDef->isQueryable() << endl;
 cout << " updatable: " << attrDef->isUpdatable() << endl;
 // ----- Очистка атрибута
 delete attrDef;
}
cout << " " << j << " attribute(s) listed for the " << strAppGrp
 << " app group\n" << endl;
// ----- Очистка итераторов и собраний
if (pIter2)
 delete pIter2;
if (pCol2)
 delete pCol2;
. . .
```

### Поучение списка папок OnDemand

Чтобы получить список папок на контент-сервере OnDemand, воспользуйтесь функцией `listSearchTemplates()`.

## Java

```
...
dsDef = (DKDatastoreDefOD)dsOD.datastoreDef();
pCol = (DKSequentialCollection) dsDef.listSearchTemplates();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
 i++;
 folderName = (String)pIter.next();
 // обрабатываем папку, как требуется
}
dsOD.disconnect();
dsOD.destroy();
```

## C++

```
. . .
// ----- Список папок
dsDef = (DKDatastoreDefOD*)dsOD.datastoreDef();
pCol = (DKSequentialCollection*)dsDef->listSearchTemplates();
pIter = pCol->createIterator();
i = 0;
// ----- Обрабатываем список папок
while (pIter->more() == TRUE)
{
 i++;
 folderName = (DKString)(*pIter->next());
 cout << "folder name [" << i << "] - " << folderName << endl;
}
// ----- Отсоединение
dsOD.disconnect();
. . .
```

## Получение документа OnDemand

На сервере OnDemand можно получать документы. Можно также выводить документы с их частями и атрибутами.

### Поиск определенного документа

Следующий пример выполняет поиск по группе программ (OnDemand Publications) на контент-сервере OnDemand.

## Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
String appgrp = "OnDemand Publications";
String SQLcmd = "where bookname LIKE 'A%'";

DKNVPair[] parms = new DKNVPair[3];
parms[0] = new DKNVPair("APPL_GROUP", appgrp);
parms[1] = new DKNVPair("MAX_RESULTS", new String(Integer.toString(5)));
parms[2] = new DKNVPair("CONTENT", new String("ATTRONLY"));

System.out.println("executing query");
dkResultSetCursor pCur = dsOD.execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
System.out.println("datastore executed query");
```

## C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();

cout << "connecting to datastore ..." << endl;
dsOD->connect(ODServer, UserID, Password, "");

DKString appgrp = "OnDemand Publications";
DKString SQLcmd = "where bookname LIKE 'A%'";

DKNVPair parms[4];
parms[0] = DKNVPair("APPL_GROUP", appgrp);
parms[1] = DKNVPair("MAX_RESULTS", DKString(5));
parms[2] = DKNVPair("CONTENT", DKString("ATTRONLY"));
parms[3] = DKNVPair(DK_CM_PARM_END, DKAny((long)0));

cout << "executing query" << endl;
dkResultSetCursor* pCur = dsOD->execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
cout << "datastore executed query" << endl;
if (pCur != 0)
 delete pCur;
```

Следующий пример выполняет поиск по группе программ (OnDemand Publications) и получает возвращенные документы.

## Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
String appgrp = "OnDemand Publications";
String SQLcmd = "where bookname LIKE 'A%'";

DKNVPair[] parms = new DKNVPair[3];
parms[0] = new DKNVPair("APPL_GROUP", appgrp);
parms[1] = new DKNVPair("MAX_RESULTS", new String(Integer.toString(5)));
parms[2] = new DKNVPair("CONTENT", new String("ATTRONLY"));

System.out.println("executing query");
dkResultSetCursor pCur = dsOD.execute(SQLcmd, DK_CM_SQL_QL_TYPE, parms);
System.out.println("datastore executed query");

while (pCur.isValid()) {
 DKDDO p = pCur.fetchNext();
 if (p != null)
 {
 String idstr = ((DKPid)p.getPidObject()).pidString();
 System.out.println(" pidString : " + idstr);
 DKPid pid = new DKPid (idstr);
 DKDDO ddoold = p;
 short id, docType = 0;
 if ((id = ddoold.propertyId(DK_CM_PROPERTY_ITEM_TYPE)) > 0)
 docType = ((Short)ddoold.getProperty(id)).shortValue();
 if (docType == DK_CM_DOCUMENT)
 {
 System.out.println("создать новый DDO с клоном pid для получения!");
 p = dsOD.createDDO(ddoold.getObjectType(), DK_CM_DOCUMENT);
 p.setPidObject(pid);
 try
 {
 dsOD.retrieveObject((dkDataObject)p);
 }
 catch(DKException exc)
 {
 System.out.println("Exception name " + exc.name());
 System.out.println("Exception message " + exc.getMessage());
 System.out.println("Exception error code " + exc.errorCode());
 System.out.println("Exception error state " + exc.errorState());
 exc.printStackTrace();
 }
 }
 }
}

pCur.destroy(); // завершив работу, удаляем указатель
```

## C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();
DKString appgrp = "OnDemand Publications";
DKString SQLcmd = "where bookname LIKE 'A%'";

DKNVPair parms[4];
parms[0] = DKNVPair("APPL_GROUP", appgrp);
parms[1] = DKNVPair("MAX_RESULTS", DKString(5));
parms[2] = DKNVPair("CONTENT", DKString("ATTRONLY"));
parms[3] = DKNVPair(DK_CM_PARM_END, DKAny((long)0));

cout << "executing query" << endl;
dkResultSetCursor* pCur = dsOD->execute(SQLcmd,DK_CM_SQL_QL_TYPE,parms);
cout << "datastore executed query" << endl;

if (pCur != 0)
{
 while (pCur->isValid())
 {
 DKDDO* p = pCur->fetchNext();
 DKDDO* ddoold = p;
 DKString pidStr = ((DKPid*)ddoold->getPidObject())->pidString();
 DKPid* pid = new DKPid(pidStr);
 short id, docType = 0;
 DKAny a;
 if ((id = ddoold->propertyId(DK_CM_PROPERTY_ITEM_TYPE)) > 0)
 {
 a = ddoold->getProperty(id);
 if (a.typeCode() == DKAny::tc_ushort)
 docType = (short)(USHORT)a;
 else
 docType = a;
 }
 if (docType == DK_CM_DOCUMENT)
 {
 cout << "create the DDO from the pidstring..." << endl;
 p = dsOD->createDDO(ddoold->getObjectType(), DK_CM_DOCUMENT);
 p->setPidObject(pid);

 dsOD->retrieveObject((dkDataObject*)p);
 }
 delete pid;
 delete ddoold;
 }
 delete pCur;
}
```

### Вывод документов, их частей и атрибутов

В следующем примере найденные документы выводятся с их частями и атрибутами:

## Java

```
//----- для каждого элемента данных получаем атрибуты
//----- numDataItems - число элементов данных
for (j = 1; j <= numDataItems; j++)
{
 a = p.getData(j)
 strDataName = p.getDataName(j);
 System.out.println(" " + j + ". Attribute Name: " + strDataName);
 System.out.println(" type: " + p.getDataPropertyByName
 (j,DK_PROPERTY_TYPE));

 System.out.println(" nullable: " +
 p.getDataPropertyByName (j,DK_PROPERTY_NULLABLE));
 if (strDataName.equals(DKPARTS) == false &&
 strDataName.equals("DKResource") == false &&
 strDataName.equals("DKViews") == false &&
 strDataName.equals("DKLargeObject") == false &&
 strDataName.equals("DKFixedView") == false &&
 strDataName.equals("DKAnnotations") == false)
 {
 System.out.println(" attribute id: " +
 p.getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID));
 }
 //----- проверяем тип атрибута
 if (a instanceof String)
 {
 System.out.println(" Attribute Value: " + a);
 }
 else if (a instanceof Integer)
 {
 System.out.println(" Attribute Value: " + a);
 }
 else if (a instanceof Short)
 {
 System.out.println(" Attribute Value: " + a);
 }
 else if (a instanceof DKDate)
 {
 System.out.println(" Attribute Value: " + a);
 }
 else if (a instanceof DKTime)
 {
 System.out.println(" Attribute Value: " + a);
 }
 else if (a instanceof DKTimestamp)
 {
 System.out.println(" Attribute Value: " + a);
 }
 else if (a instanceof dkCollection)
 {
 // продолжение следует...
 }
}
```

## Java (продолжение)

```
System.out.println(" Attribute Value is collection");
pCol = (dkCollection)a;
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
 i++;
 a = pIter.next();
 pD0 = (dkDataObjectBase)a;

 if (pD0.protocol() == DK_XD0)
 {
 System.out.println(" dkXD0 object " + i + " in collection");
 pXD0 = (dkXD0)pD0;
 DKPidXD0 pid2 = pXD0.getPidObject();
 System.out.println(" XD0 pid string: " +
 pid2.pidString());
 // ----- Получаем и открываем обработчик экземпляра для XD0
 pXD0.retrieve();
 // pXD0.open();
 }
}
else if (a != null)
{
 System.out.println(" Attribute Value: " + a.toString());
 if (strDataName.equals("DKResource") ||
 strDataName.equals("DKFixedView") ||
 strDataName.equals("DKLargeObject"))
 {
 pD0 = (dkDataObjectBase)a;

 if (pD0.protocol() == DK_XD0)
 {
 System.out.println(" dkXD0 object ");
 pXD0 = (dkXD0)pD0;
 DKPidXD0 pid2 = pXD0.getPidObject();
 System.out.println(" XD0 pid string: " +
 pid2.pidString());
 // получаем и открываем обработчик экземпляра для XD0
 pXD0.retrieve();
 // pXD0.open();
 }
 }
}
```

**C++**

```
DKDDO *p = 0;
DKAny a;
. . .
for (j = 1; j <= numDataItems; j++)
{
 a = p->getData(j);
 strDataName = p->getDataName(j);

 cout << " " << j << ". Attribute Name: " << strDataName << endl;
 cout << " type: " << p->getDataPropertyByName(j,DK_PROPERTY_TYPE) << endl;
 cout << " nullable: "
 << p->getDataPropertyByName(j,DK_PROPERTY_NULLABLE) << endl;

 if (strDataName != DK_CM_DKPARTS &&
 strDataName != "DKResource" &&
 strDataName != "DKViews" &&
 strDataName != "DKLargeObject" &&
 strDataName != "DKPermissions" &&
 strDataName != "DKFixedView" &&
 strDataName != "DKAnnotations")
 {
 cout << " attribute ID: "
 << p->getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID) << endl;
 }

 if (a.typeCode() == DKAny::tc_string)
 {
 DKString astring = a;
 cout << " attribute Value (string): " << astring << endl;
 }
 else if . . .
 {
 // ----- Обработываем каждый из остальных типов
 }
 else if (a.typeCode() != DKAny::tc_null)
 {
 cout << " Attribute Value (non NULL): " << a << endl;
 if (strDataName == "DKResource" ||
 strDataName == "DKFixedView" ||
 strDataName == "DKLargeObject")
 {
 pDO = (dkDataObjectBase*)a;
 if (pDO->protocol() == DK_XDO)
 {
 cout << " dkXDO object " << endl;
 pXDO = (dkXDO*)pDO;
 pidXDO = (DKPidXDO00*)pXDO->getPid();
 cout << " XDO PID string: " << pidXDO->pidString() << endl;
 }
 }
 }
}
// продолжение следует...
```



### С++ (продолжение)

```
 // ----- Получаем и открываем обработчик экземпляра для XD0
 pXD0->retrieve();
 }
}
else cout << " Attribute Value is NULL" << endl;
```

Полная программа приведена в файле TRetrieveOD.cpp в каталоге CMBROOT\samples\cpp\od.

## Включение режима папок OnDemand

Чтобы включить режим папок OnDemand, надо передать строку  
ENTITY\_TYPE=TEMPLATES

соединителю OnDemand как часть строки соединения или строки конфигурации.  
Пример строки конфигурации может выглядеть так:

### Java

```
DKDatastoreOD dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

### С++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

Пример строки соединения может выглядеть так:

### Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
dsOD.connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

### С++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
dsOD->connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

## Асинхронный поиск

Соединитель OnDemand поддерживает асинхронный поиск, как объединенный, так и прямой. Асинхронный поиск не встраивается в основной поток; такой поиск можно в любое время отменить, нажав кнопку **Остановить поиск** в окне шаблона поиска. Свойство максимального числа соответствий в шаблоне поиска ограничивает максимальное число возвращаемых результатов.

Кроме того, соединитель OnDemand поддерживает синхронные и асинхронные поиски в режиме групповых прикладных программ от клиента AIX.

Если вы используете такой критерий поиска, как WHERE userid LIKE '%', полученные на клиенте документы могут занять всю доступную на компьютере память. Запуская асинхронный поиск при помощи метода executeWithCallback(), вы можете задать максимальное количество возвращаемых документов и отменить поиск в любое время.

Если множество результатов слишком велико, возможно, придется увеличить размер стека по умолчанию Java Virtual Machine (JVM). Размер стека по умолчанию для каждого треда Java - 400 Кбайт, что позволяет возвращать 3920 элементов без переполнения стека. Увеличение размера стека JVM до 800 Кбайт увеличивает это число вдвое (то есть до 7840 элементов). Если необходимо, можно задать и больший размер стека JVM.

Чтобы увеличить размер стека JVM, воспользуйтесь опцией командной строки Java -oss, после нее - nnnK или nnM, где K - Кбайт, а M - Мбайт.

Примеры использования асинхронного поиска смотрите в программах примеров TRetrieveWithCallbackOD, TRetrieveFolderWithCallbackOD и TCallbackOD.

## Папки OnDemand как шаблоны поиска

Трое из визуальных JavaBeans EIP - CMBSearchTemplateList, CMBSearchTemplateViewer и CMBSearchResultsView - используют шаблоны поиска EIP. Эти функции beans можно использовать для объединенного поиска, если задать для CMBCConnection dsType значение Fed.

Эти функции beans можно использовать и для прямого поиска на серверах OnDemand. Перед регистрацией задайте следующие свойства:

```
connection.setDsType("OD");
connection.setServerName(<odserver>);
connection.setConnectionString("ENTITY_TYPE=TEMPLATES");
```

## Папки OnDemand как собственные объекты

Соединитель OnDemand может также отобразить папки OnDemand как собственные объекты, если задать строку соединения "ENTITY\_TYPE=TEMPLATES". Использование папок OnDemand как объектов сможет быть полезным для

объединенного поиска, когда может быть проще работать с определениями папок, чем с группами программ OnDemand, которые считаются собственными объектами OnDemand по умолчанию.

Для объединенного поиска задайте определение сервера в управлении Enterprise Information Portal. Затем можно определить и отобразить объекты объединения на папки на сервере OnDemand.

## **Создание и изменение комментариев**

При использовании программы просмотра OnDemand, которую запускает функция bean CMBDocumentViewer, вы можете создавать, модифицировать и удалять комментарии для документов OnDemand, используя функцию bean CMBDataManagement и соответствующий класс CMBAnnotation.

## **Трассировка**

Вы можете проводить трассировку событий при помощи соединителя OnDemand. Чтобы включить трассировку API Java соединителя, поместите файл INI трассировки (cmbodtrace.ini - для Java или cmbodCtrase.ini - для C++) в корневой каталог диска C (C:\) или в каталог, заданный в переменной CMBROOT. Для AIX поместите этот файл в каталог /usr/lpp/cmb/cmgmt. Для Solaris поместите его в каталог /opt/IBMcmb/cmgmt.

В качестве каталога для вывода файлов трассировки по умолчанию используется C:\Ctrase. Чтобы записать информацию трассировки в другое место, отредактируйте файл INI трассировки. Имейте в виду, что для AIX все имена файлов должны быть в нижнем регистре.

Убедитесь, что в файле трассировки указано правильное полное имя и что строка, содержащая CMBODTRACEDIR, не начинается со знака #. Вот примеры файлов INI трассировки:

## Java

```
#=====
Это файл свойств Java, а не реальный файл INI!
*****#
Помните, что в системах Windows для разделения имен каталогов
следует использовать ДВЕ ОБРАТНЫХ КОСЫХ ЧЕРТЫ (\\)!!!
=====
*****#
#
***** В системах Windows этот файл должен находиться в c:\ *****
#
Свойство Имя каталога файла трассировки OD - CMBODTRACEDIR
#
Свойство CMBODTRACEDIR определяет каталог, в который будут записаны
файлы трассировки. Если такого каталога не существует, он будет
создан.
#
Убедитесь, что имена каталогов разделяются двумя обратными
косыми чертами, чтобы избежать нежелательных результатов.
#
Проверьте, что полное имя не совпадает с именем существующего файла.
В противном случае файлы трассировки не будут созданы.
#
Имя каталога вывода трассировки можно изменить, чтобы использовать диск,
на котором больше свободного места. При этом не рекомендуется изменять
имя каталога вывода трассировки в процессе активного сеанса трассировки
#
CMBODTRACESCOPE управляет количеством генерируемой информации трассировки.
#
CMBODTRACESCOPE=ENTRY_EXIT_JNI_ONLY
Трассируются точки входа и выхода только в JNI. Генерируется минимальный
объем данных трассировки.
#
CMBODTRACESCOPE=ENTRY_EXIT_ONLY
Трассируются точки входа и выхода в методах Java и функциях JNI.
#
CMBODTRACESCOPE=JNI_ONLY
Полная трассировка только для функций JNI.
#
Если CMBODTRACESCOPE отсутствует или для него установлено любое другое
значение, будет выполняться полная трассировка.
#
Чтобы выключить трассировку, добавьте символ # в столбец 1.
#
В AIX: измените следующую строку на CMBODTRACEDIR=/usr/lpp/cmb/cmgmt/trace
В Sun: измените следующую строку на CMBODTRACEDIR=/opt/IBMcmb/cmgmt/trace
CMBODTRACEDIR=c:\\trace
```

## C++

```
#=====
Файл INI трассировки OnDemand
#
Ключ имени каталога файлов трассировки OnDemand - CMBODTRACEDIR
#
Ключ CMBODTRACEDIR определяет каталог, в который будут записаны файлы
трассировки. Если такого каталога не существует, он будет создан.
#
Проверьте, что полное имя не совпадает с именем существующего файла.
В противном случае файлы трассировки не будут созданы.
#
Имя каталога вывода трассировки может быть изменено, чтобы указывать на
диск, на котором больше свободного места. При этом не рекомендуется
изменять имя каталога вывода трассировки в процессе сеанса активной
трассировки.
#
CMBODTRACESCOPE управляет количеством генерируемой информации трассировки.
#
CMBODTRACESCOPE=ENTRY_EXIT_ONLY
Трассировать только вход и выход всех методов и функций C++.
#
Если CMBODTRACESCOPE отсутствует или для него установлено любое другое
значение, выполняется полная трассировка.
#
Чтобы выключить трассировку, добавьте символ # в столбец 1
для строки CMBODTRACEDIR.
#
[ODCTRACE]
В AIX: измените строку ниже на CMBODTRACEDIR=/usr/lpp/cmb/cmgmt/ctrace
CMBODTRACEDIR=D:\Ctrace
#CMBODTRACESCOPE=ENTRY_EXIT_ONLY
```

## Работа с сервером Content Manager ImagePlus for OS/390

API Enterprise Information Portal при работе с контент-серверами Content Manager ImagePlus for OS/390 поддерживают следующие возможности:

- Соединение с одним или несколькими серверами ImagePlus и отсоединение от них.
- Получение категорий.
- Получение полей атрибутов.
- Получение папок.
- Получение документов.

В вашей программе сервер ImagePlus for OS/390 content представляется как DKDatastoreIP.

**Ограничение:** ImagePlus for OS/390 не поддерживает:

- Механизм текстового поиска и поиск QBIC
- Комбинированные запросы
- Рабочие комплекты и рабочие потоки

## Получение списка объектов и атрибутов

После создания контент-сервера ImagePlus for OS/390 в виде объекта DKDatastoreIP и соединения с ним для этого контент-сервера можно проверить объекты и атрибуты. В следующем примере выводится список всех объектов для контент-сервера ImagePlus for OS/390:

### Java

```
// ----- После создания склада данных и соединения
// dsIP - объект DKDatastoreIP
DKEntityDefIP entDef = null;
DKAttrDefIP attrDef = null;

DKSequentialCollection pCol = (DKSequentialCollection)dsIP.listEntities();
dkIterator pIter = null;

if (pCol == null)
{
 // ----- Обработка, если собрание объектов пусто
}
else
{
 ... // ----- необходимая обработка
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogIP.java), находится в каталоге CMBROOT\Samples\java\d1.

## C++

```
// Получить список объектов...
DKEntityDefIP* docDef = 0;
DKAttrDefIP* attrDef = 0;

cout << "---List entities---" << endl;
DKSequentialCollection* pCol =
 (DKSequentialCollection*)(dsIP.listEntities());
dkIterator *pIter = 0;

if (pCol == 0)
{
 cout << "collection of entities is null!" << endl;
}
else
{
 ...
}
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogIP.cpp), находится в каталоге Cmbroot/Samples/cpp/ip.

Следующий пример получает список всех атрибутов, связанных с каждым объектом, при помощи функций `getAttr` и `listAttrNames` класса `DKEntityDefIP`.

## Java

```
// ----- получить список атрибутов при помощи методов listAttrNames и getAttr

pIter = pCol.createIterator();
while (pIter.more())
{
 // ----- Итерация по каждому объекту
 entDef = (DKEntityDefIP)pIter.next();
 System.out.println(" Entity type : " + entDef.getType());
 System.out.println(" Entity type name: " + entDef.getName());

 // ----- получаем список атрибутов объекта
 String[] attrNames = entDef.listAttrNames();
 int count = attrNames.length;
 for (int i = 0; i < count; i++)
 {
 attrDef = (DKAttrDefIP)entDef.getAttr(attrNames[i]);
 System.out.println(" Attr name : " + attrDef.getName());
 System.out.println(" Attr id : " + attrDef.getId());
 System.out.println(" Entity name : " + attrDef.getEntityName());
 System.out.println(" Datastore name: " + attrDef.datastoreName());
 System.out.println(" Attr type : " + attrDef.getType());
 System.out.println(" Attr restrict : " + attrDef.getStringType());
 System.out.println(" Attr min val : " + attrDef.getMin());
 System.out.println(" Attr max val : " + attrDef.getMax());
 System.out.println(" Attr display : " + attrDef.getSize());
 System.out.println(" Attr precision: " + attrDef.getPrecision());
 System.out.println(" Attr scale : " + attrDef.getScale());
 System.out.println(" Attr update ? : " + attrDef.isUpdatable());
 System.out.println(" Attr nullable ? : " + attrDef.isNullable());
 System.out.println(" Attr queryable? : " + attrDef.isQueryable());
 System.out.println("");
 }
}
```



## C++

```
// Метод 1:
cout <<
 "Получение списка атрибутов с помощью функций listAttrNames и getAttr"
 << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
 docDef = (DKEntityDefIP*)(pIter->next()->value());
 cout << " Document type : " << docDef->getType() << endl;
 cout << " Document type name: " << docDef->getName() << endl;

 long tmpCount;
 DKString* attrNames;

 // После возврата tmpCount содержит число элементов списка.
 attrNames = docDef->listAttrNames(tmpCount);
 for (int i=0; i<tmpcoun; i++)
 {
 cout << " Attr name before lookup " << attrNames[i] << endl;
 attrDef = (DKAttrDefIP*)(docDef->getAttr(attrNames[i]));
 cout << " Attr name [" << i << "] : " << attrDef->getName() << endl;
 cout << " Attr id : " << attrDef->getId() << endl;
 cout << " Entity name : " << attrDef->getEntityName() << endl;
 cout << " Datastore name: " << attrDef->datastoreName() << endl;
 cout << " Attr type : " << attrDef->getType() << endl;
 cout << " Attr restrict : " << attrDef->getStringType() << endl;
 cout << " Attr min val : " << attrDef->getMin() << endl;
 cout << " Attr max val : " << attrDef->getMax() << endl;
 cout << " Attr display : " << attrDef->getSize() << endl;
 cout << " Attr precision: " << attrDef->getPrecision() << endl;
 cout << " Attr scale : " << attrDef->getScale() << endl;
 cout << " Attr update ? " << attrDef->isUpdatable() << endl;
 cout << " Attr nullable ? " << attrDef->isNullable() << endl;
 cout << " Attr queryable? " << attrDef->isQueryable() << endl;
 cout << "" << endl;
 delete attrDef;
 } // end for

 delete [] attrNames;

} // end while
delete pIter;
```

В следующем примере показан другой способ получения списка атрибутов, связанных с каждым объектом, при помощи метода `listEntityAttrs` класса `DKDatastoreIP`.

## Java

```
// --- получение списка атрибутов при помощи метода listEntityAttrs
pIter = pCol.createIterator();
while (pIter.more())
{
 entDef = (DKEntityDefIP)pIter.next();
 System.out.println(" Entity type : " + entDef.getType());
 System.out.println(" Entity type name: " + entDef.getName());

 DKSequentialCollection pAttrCol =
 (DKSequentialCollection)dsIP.listEntityAttrs(entDef.getName());
 if (pAttrCol == null)
 {
 // ----- Обработка, если собрание атрибутов пусто
 }
 else
 {
 dkIterator pAttrIter = pAttrCol.createIterator();
 while (pAttrIter.more())
 {
 attrDef = (DKAttrDefIP)pAttrIter.next();
 System.out.println(" Attr name : " + attrDef.getName());
 System.out.println(" Attr id : " + attrDef.getId());
 System.out.println(" Entity name : " + attrDef.getEntityName());
 System.out.println(" Datastore name: " + attrDef.datastoreName());
 System.out.println(" Attr type : " + attrDef.getType());
 System.out.println(" Attr restrict : " + attrDef.getStringType());
 System.out.println(" Attr min val : " + attrDef.getMin());
 System.out.println(" Attr max val : " + attrDef.getMax());
 System.out.println(" Attr display : " + attrDef.getSize());
 System.out.println(" Attr precision: " + attrDef.getPrecision());
 System.out.println(" Attr scale : " + attrDef.getScale());
 System.out.println(" Attr update ? " + attrDef.isUpdatable());
 System.out.println(" Attr nullable ? " + attrDef.isNullable());
 System.out.println(" Attr queryable? " + attrDef.isQueryable());
 System.out.println("");
 }
 }
}
```

## C++

```
// Метод 2:
cout << "---List attributes using listEntityAttrs function---" << endl;
pIter = pCol->createIterator();
while (pIter->more())
{
 // итератор возвращает DKAny*
 docDef = (DKEntityDefIP*)(pIter->next()->value());
 cout << " Document type : " << docDef->getType() << endl;
 cout << " Document type name: " << docDef->getName() << endl;
 DKSequentialCollection* pAttrCol = (DKSequentialCollection*)
 (dsIP.listEntityAttrs(docDef->getName()));

 if (pAttrCol == 0)
 {
 cout << "collection of entity attrs is null for entity "
 << docDef->getName() << endl;
 }
 else
 {
 int i=0;
 dkIterator* pAttrIter = pAttrCol->createIterator();
 while (pAttrIter->more())
 {
 i++;
 // ----- Итератор возвращает указатель на DKAny
 attrDef = (DKAttrDefIP*)(pAttrIter->next()->value());
 cout << " Attr name [" << i << "] : " << attrDef->getName() << endl;
 cout << " Attr id : " << attrDef->getId() << endl;
 cout << " Entity name : " << attrDef->getEntityName() << endl;
 cout << " Datastore name: " << attrDef->datastoreName() << endl;
 cout << " Attr type : " << attrDef->getType() << endl;
 cout << " Attr restrict : " << attrDef->getStringType() << endl;
 cout << " Attr min val : " << attrDef->getMin() << endl;
 cout << " Attr max val : " << attrDef->getMax() << endl;
 cout << " Attr display : " << attrDef->getSize() << endl;
 cout << " Attr precision: " << attrDef->getPrecision() << endl;
 cout << " Attr scale : " << attrDef->getScale() << endl;
 cout << " Attr update ? " << attrDef->isUpdatable() << endl;
 cout << " Attr nullable ? " << attrDef->isNullable() << endl;
 cout << " Attr queryable? " << attrDef->isQueryable() << endl;
 cout << " " << endl;
 delete attrDef;
 } // end while
 delete pAttrIter;
 }
 delete pAttrCol;
 delete docDef;
} // end while
delete pIter;
}
```

## Синтаксис запросов ImagePlus for OS/390

В следующем примере показан синтаксис запроса для ImagePlus for OS/390:

### Java

```
SEARCH = (COND=(выражение_поиска), ENTITY={имя_объекта | отображенное_имя}
[,MAX_RESULTS=число_результатов]);
[OPTION=([CONTENT={YES | ATTRONLY | NO};][PENDING={YES | NO};])]
```

### C++

```
SEARCH=(COND=(выражение_поиска),ENTITY={имя_объекта | отображенное_имя}
[,MAX_RESULTS=число_результатов]);
[OPTION=([CONTENT={YES | ATTRONLY | NO};][PENDING={YES | NO};])]
```

В запросе используются следующие параметры:

#### выражение\_поиска

Каждое выражение поиска состоит из одного или нескольких критериев поиска. Между критериями поиска можно использовать только логическую операцию AND.

Критерий поиска имеет вид:

*{имя\_атрибута | отображенное\_имя\_атрибута} операция литерал*

где:

#### имя\_атрибута

Имя атрибута объекта, по которому выполняется поиск.

#### отображенное\_имя\_атрибута

Имя атрибута, отображенное при помощи атрибута, по которому выполняется поиск.

#### операция

Для всех атрибутов поддерживается равенство (==). Для атрибутов типа DATE можно использовать следующие дополнительные операции:

- >        больше, чем
- <        меньше, чем
- >=      больше или равно
- <=      меньше или равно

#### литерал

Литерал. Не используйте для числовых атрибутов кавычки ("), например:

FolderType == 9

Для атрибутов даты, времени и отметки времени кавычки или апострофы (') не обязательны, но допускаются, например:

ReceiveDate == 1999-03-08  
ReceiveDate == '1999-03-08'

Для строчных атрибутов кавычки или апострофы (') не обязательны, но допускаются. Если строка содержит апостроф ('), он должен задаваться двумя апострофами, например для значения Folder'1 задайте:

FolderId == 'Folder'1'

**имя\_объекта**

Имя объекта для поиска.

**отображенное\_имя\_объекта**

Имя объекта, отображенного на объект для поиска.

**число\_результатов**

Максимальное число возвращаемых результатов.

Необязательные ключевые слова:

- CONTENT**      Управляет объемом информации, возвращаемой в результатах
- YES (по умолчанию)**  
        Возвращает PID, атрибуты и их значения для документа или папки. Если в документе есть части, возвращаются PID объектов XDO. Если в папке есть документы, возвращаются PID документов.
- NO**          Возвращает только PID документов или папок.
- ATTRONLY**  
        Возвращает только PID, атрибуты и их значения для документа или папки.
- PENDING**      Управляет включением отложенных документов, который не содержат ни одной части. Эта опция применима, только когда значение ENTITY - DOCUMENT или объект отображен в DOCUMENT.
- YES**          Включает в результаты незаконченные документы.
- NO (по умолчанию)**  
        Не включает в результаты незаконченные документы.

---

## Работа с Content Manager for AS/400

Классы API, поддерживаемые для Content Manager for AS/400 VisualInfo for AS/400, подобны классам для Content Manager.

**Ограничение:** Content Manager for AS/400 не поддерживает:

- Механизм текстового поиска и поиск QBIC
- Комбинированные запросы
- Рабочие комплекты и рабочие потоки

### Вывод списка объектов (индексных классов) и атрибутов

Контент-сервер Content Manager for AS/400 представляется как объект DKDatastoreV4. После создания контент-сервера и соединения с ним можно вывести список объектов (индексных классов) и атрибутов для сервера Content Manager for AS/400 (смотрите пример).

#### Java

```
// ----- После создания склада данных (dsV4) и соединения
// ----- получаем индексные классы
pCol = (DKSequentialCollection) dsV4.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
 i++;
 icDef = (DKIndexClassDefV4)pIter.next();
 strIndexClass = icDef.getName();
 ... // ---- обрабатываем индексные классы, как требуется
 // ----- получаем атрибуты
 pCol2 = (DKSequentialCollection) dsV4.listEntityAttrs(strIndexClass);
 pIter2 = pCol2.createIterator();
 j = 0;

 while (pIter2.more() == true)
 {
 j++;
 attrDef = (DKAttrDefV4)pIter2.next();
 ... // ----- обрабатываем атрибуты
 }
}

dsV4.disconnect();
dsV4.destroy();
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogV4.java), находится в каталоге CMBR00T\Samples\java\d1.

## C++

```
cout << "list index class(es)..." << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsV4.listSchema());
pIter = pCol->createIterator();
i = 0;

while (pIter->more() == TRUE)
{
 i++;
 a = (*pIter->next());
 strIndexClass = a;
 cout << "index class name [" << i << "] - " << strIndexClass << endl;
 cout << " list attribute(s) for " << strIndexClass << " index class:" << endl;
 pCol2 =

(DKSequentialCollection*)((dkCollection*)dsV4.listSchemaAttributes(strIndexClass));
 pIter2 = pCol2->createIterator();
 j = 0;

 while (pIter2->more() == TRUE)
 {
 j++;
 pA = pIter2->next();
 pDef = (DKAttributeDef*) pA->value();
 cout << " Attribute name [" << j << "] - " << pDef->name << endl;
 cout << " datastoreType - " << pDef->datastoreType << endl;
 cout << " attributeOf - " << pDef->attributeOf << endl;
 cout << " type - " << pDef->type << endl;
 cout << " size - " << pDef->size << endl;
 cout << " id - " << pDef->id << endl;
 cout << " nullable - " << pDef->nullable << endl;
 cout << " precision - " << pDef->precision << endl;
 cout << " scale - " << pDef->scale << endl;
 cout << " string type - " << pDef->stringType << endl;
 }

 cout << " " << j << " attribute(s) listed for "
 << strIndexClass << " index class" << endl;
 pCol2->apply(deleteDKAttributeDef);
 delete pIter2;
 delete pCol2;
}

delete pIter;
delete pCol;
cout << i << " index class(es) listed" << endl;
dsV4.disconnect();
cout << "datastore disconnected" << endl;
```

Полная программа примера, из которой взята приведенная программа (TListCatalogV4.cpp), находится в каталоге Cmbroot/Samples/cpp/v4.

## Выполнение запроса

Следующий пример выполняет запрос в Content Manager for AS/400 и обрабатывает его результаты.

## Java

```
// ----- После создания склада данных (dsV4) и соединения строим
// запрос и параметры и выполняем его
pCur = dsV4.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
...

if (pCur == null)
{
 // ----- обработка, если указатель пуст
}

while (pCur.isValid()) {
 p = pCur.fetchNext();
 if (p != null)
 {
 cnt++;
 i = pCur.getPosition();
 System.out.println("\n====> Item " + i + " <=====");
 numDataItems = p.dataCount();
 DKPid pid = p.getPid();
 System.out.println(" pid string: " + pid.pidString());
 k = p.propertyId(DK_CM_PROPERTY_ITEM_TYPE);

 if (k > 0)
 {
 Short sVal = (Short)p.getProperty(k);
 j = sVal.shortValue();
 switch (j)
 {
 case DK_CM_DOCUMENT:
 {
 ... // обработка, если элемент - документ
 break;
 }
 case DK_CM_FOLDER:
 {
 ... // обработка, если элемент - папка
 break;
 }
 }
 }
 }
}

for (j = 1; j <= numDataItems; j++)
{
 a = p.getData(j);
 strDataName = p.getDataName(j);
 ... // обрабатываем атрибуты, как требуется
 if (strDataName.equals(DKPARTS) == false
 && strDataName.equals(DKFOLDER) == false)
 {
 // продолжение следует...
 }
}
```



## Java (продолжение)

```
 System.out.println(" attribute id: "
 + p.getDataPropertyByName(j,DK_CM_PROPERTY_ATTRIBUTE_ID));
 }
 if (a instanceof String)
 {
 System.out.println(" Attribute Value: " + a);
 }
 else if (a instanceof Integer)
 ... // ---- Обрабатываем каждый тип для атрибута
 else if (a instanceof dkCollection)
 {
 // ---- обработка, если значение атрибута - собрание
 pCol = (dkCollection)a;
 pIter = pCol.createIterator();
 i = 0;
 while (pIter.more() == true)
 {
 i++;
 a = pIter.next();
 pD0 = (dkDataObjectBase)a;

 if (pD0.protocol() == DK_CM_PDD0)
 {
 // обрабатываем DD0
 pDD0 = (DKDD0)pD0;
 ...
 }
 else if (pD0.protocol() == DK_CM_XD0)
 {
 // Обработка XD0
 pXD0 = (dkXD0)pD0;
 DKPidXD0 pid2 = pXD0.getPid();
 ...
 }
 }
 }
 else if (a != null)
 {
 // обработка атрибута
 }
 else ... // обработка, если атрибут пуст
 }
}
pCur.destroy(); // завершив работу, удаляем указатель
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteV4.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
cout << "executing query..." << endl;
...
pCur = dsV4.execute(cmd);
cout << " query executed" << endl;
...
cout << "\n..... Displaying query results \n\n";

...
while (pCur->isValid())
{
 p = pCur->fetchNext();

 if (p != 0)
 {
 cout << "=====> " << "Item " << cnt << " <=====" << endl;
 numDataItems = p->dataCount();
 pid = p->getPid();
 cout << " Pid String: " << pid.pidString() << endl;
 k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);

 if (k > 0)
 {
 a = p->getProperty(k);
 val = a;
 cout << " *****" << endl;

 switch (val)
 {
 case DK_CM_DOCUMENT:
 {
 cout << " Item is a document " << endl;
 break;
 }
 case DK_CM_FOLDER:
 {
 cout << " Item is a folder " << endl;
 break;
 }
 }

 cout << " *****" << endl;
 }

 cout << " Number of Data Items: " << numDataItems << endl;

 for (j = 1; j <= numDataItems; j++)
 {
 a = p->getData(j);
 strDataName = p->getDataName(j);

 // продолжение следует...
```



## C++ (продолжение)

```
switch (a.typeCode())
{
 case DKAny::tc_string :
 {
 strData = a;
 cout << " attribute name: " << strDataName
 << ", value: " << strData << endl;
 break;
 }
 case DKAny::tc_long :
 {
 lVal = a;
 cout << " attribute name: " << strDataName
 << ", value: " << lVal << endl;
 break;
 }

 case DKAny::tc_null :
 {
 cout << " attribute name: " << strDataName << ", value: NULL " << endl;
 break;
 }

 case DKAny::tc_collection :
 {
 pdCol = a;
 cout<<strDataName<<" collection name: "<<strDataName << endl;
 cout << "-----" << endl;
 pdIter = pdCol->createIterator();
 ushort b = 0;

 while (pdIter->more() == TRUE)
 {
 b++;
 cout << " -----" << endl;
 a = *(pdIter->next());
 pDOBase = a;

 if (pDOBase->protocol() == DK_PDDO)
 {
 pPDDO = (DKDDO*)pDOBase;
 cout << " DKDDO object " << b << " in " << strDataName
 << " collection " << endl;
 k = pPDDO->propertyId(DK_CM_PROPERTY_ITEM_TYPE);

 if (k > 0)
 {
 a = pPDDO->getProperty(k);
 val = a;
 cout << " *****" << endl;
 }
 }
 }
 }
}

// продолжение следует...
```

## C++ (продолжение)

```
 switch (val)
 {
 case DK_CM_DOCUMENT:
 {
 cout << " Item is a document " << endl;
 break;
 }
 case DK_CM_FOLDER:
 {
 cout << " Item is a folder " << endl;
 break;
 }
 }
 cout << " *****" << endl;
 }
 else if (pDOBase->protocol() == DK_XDO)
 {
 pXDO = (dkXDO*)pDOBase;
 cout << " dkXDO object " << b << " in " << strDataName
 << " collection " << endl;

 }

 if (pdIter != 0)
 {
 delete pdIter;
 }

 if (b == 0)
 {
 cout << strDataName << " collection has no elements " << endl;
 }

 cout << " -----" << endl;
 break;
}

default :
 cout << "Type is not supported\n";
}
cout << " type: " << p->getDataPropertyByName(j,DK_CM_PROPERTY_TYPE)
 << endl;
cout << " nullable: " <<
 p->getDataPropertyByName(j,DK_CM_PROPERTY_NULLABLE) << endl;

// продолжение следует...
```

### С++ (продолжение)

```
 if (strDataName != DKPARTS && strDataName != DKFOLDER)
 {
 cout << " attribute id: "
 << p->getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID) << endl;
 }
 }
 cnt++;
 delete p;
}
}
cout << "Total Item count is " << cnt-1 << endl;

if (pCur != 0)
 delete pCur;
```

Полная программа примера, из которой взята приведенная программа (TExecuteV4.cpp), находится в каталоге Cmbroot/Samples/cpp/v4.

## Выполнение параметрического запроса

В следующем примере выполняется параметрический запрос.

### Java

```
// ----- создаем строку запроса и объект запроса
String cmd = "SEARCH=(INDEX_CLASS=V4DEMO)";
pQry = dsV4.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Выполняем запрос
pQry.execute(parms);

System.out.println("number of query results = " + pQry.numberOfResults());

// ----- обработка результатов запроса
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
...
```

Полная программа примера, из которой взят приведенный фрагмент (TSamplePQryV4.java), находится в каталоге CMBROOT\Samples\java\d1.

## C++

```
cout << "query string: " << cmd << endl;
cout << "creating query..." << endl;
pQry = dsV4.createQuery(cmd);
cout << "executing query..." << endl;
pQry->execute();
cout << "query executed" << endl;
cout << "getting query results..." << endl;
any = pQry->result();
pResults = (DKResults*)((dkCollection*) any);

processResults(pResults);

dsV4.disconnect();
cout << "datastore disconnected" << endl;
delete pQry;
delete pResults;
```

Полная программа примера, из которой взята приведенная программа (TSamplePQryV4.cpp), находится в каталоге Cmbroot/Samples/cpp/v4.

## Работа с Domino.Doc

Domino.Doc - это решение Lotus Domino для организации, управления и хранения деловой документации и доступа к ней изнутри и извне фирмы.

Domino.Doc поддерживает API открытого управления документами (ODMA), так что вы можете создавать, сохранять и получать документы при помощи программ, поддерживающих ODMA. ODMA соединяется с сервером Domino.Doc при помощи протокола HTTP или Lotus Notes.

Domino.Doc поддерживает следующие возможности:

- Соединение с одним или несколькими серверами Domino.Doc и отсоединение от них.
- Поиск папок.
- Получение документов.
- Совместимость ODMA, позволяющая пользователям использовать привычные им программы.

**Ограничение:** Domino.Doc не поддерживает:

- Методы добавления, изменения и удаления (add, update и delete) документов.
- Механизм текстового поиска и поиск QWIC.
- Комбинированные запросы.

- Рабочие комплекты и рабочие потоки.

При использовании API для работы с объектом Domino.Doc надо построить выражение для возврата таких объектов. В этом разделе объясняется структура API Domino.Doc, распределение объектов в иерархии и построение выражений. На рис. 19 на стр. 421 показана связь модели объекта Domino.Doc с ее компонентами.



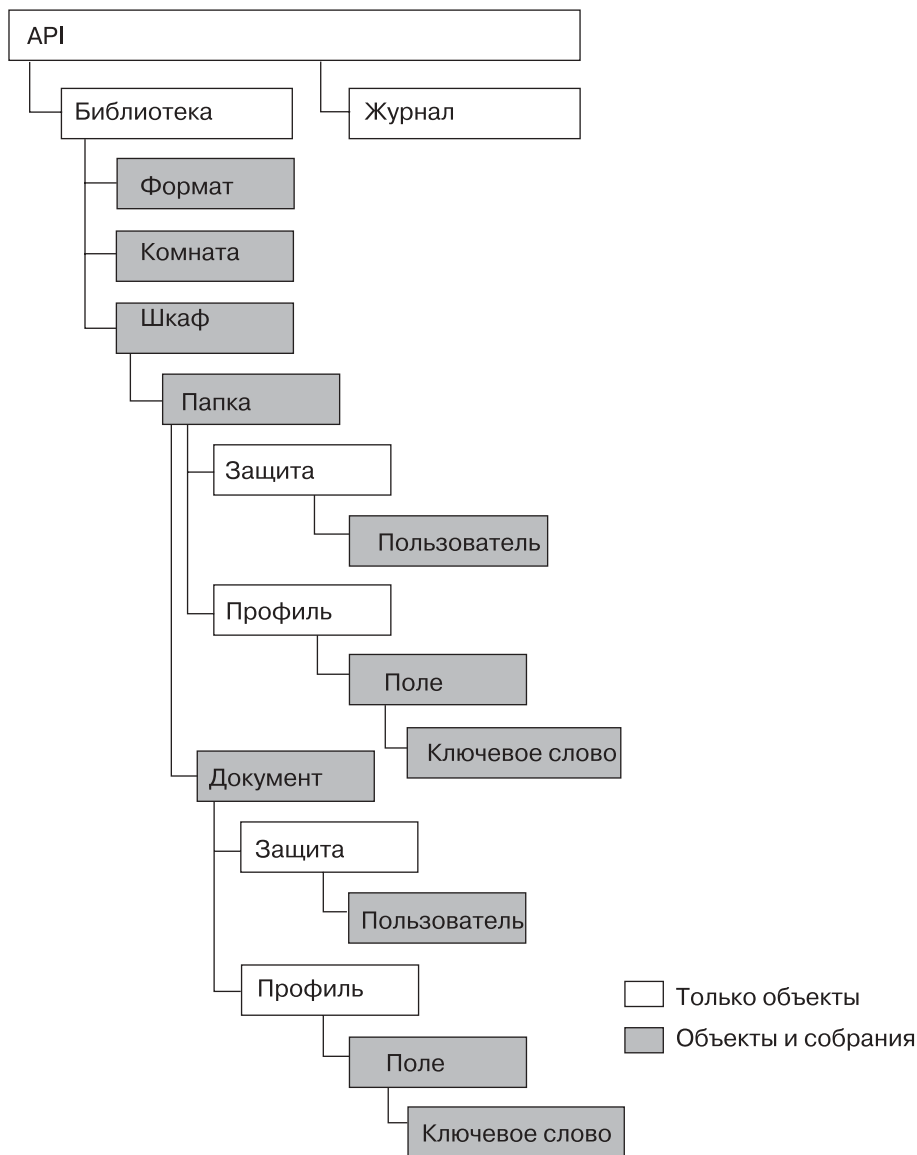


Рисунок 19. Модель объекта Domino.Doc

Элементы, содержащиеся в Domino.Doc (и представляющие их API) организованы так:

- Библиотека содержит залы (объекты DKRoomDefDD) и шкафы (объекты DKCabinetDefDD).
- Каждый шкаф содержит папки (объекты DKBinderDefDD).
- Каждая папка содержит профиль (объект DKAttrProfileDefDD) и защиту.

- Каждая папка содержит документы (объекты DKDocumentDefDD).
- Каждый документ содержит профиль (объект DKAttrProfileDefDD) и защиту.
- Каждый профиль содержит поля (объекты DKAttrFieldDefDD).
- Каждое поле может содержать ключевые слова (объекты DKAttrKeywordDefDD).

## Получение списка объектов и подобъектов

Следующий пример выводит список объектов (в этом примере они называются залы) в Domino.Doc и их подобъекты (в этом примере - шкафы, папки и документы).

### Java

```
...
// ----- получаем список залов
dkCollection rooms = dsDD.listEntities();
// ----- цикл по залам и их подобъектам
dkIterator itRooms = rooms.createIterator();
itRooms.reset();
while(itRooms.more()) {
 ... // обработка залов и объектов
```

Полная программа примера, из которой взят приведенный фрагмент (TListSubEntitiesDD.java), находится в каталоге CMBROOT\Samples\java\dd.

## C++

```
dkCollection* pColl = domDoc.listEntities();

long nbrEnts = pColl->cardinality();

dkIterator* itEnts = pColl-> createIterator();
while(itEnts->more())
{ // Для каждого возвращенного dkEntityDef...
 DKRoomDefDD* pEnt = (DKRoomDefDD*)itEnts->next()->value();
 cout << "Room title: " << pEnt->getName() << endl;
 cout << " Has SubEntities: " << pEnt->hasSubEntities() << endl;

 // печатаем подобъекты (Шкафы->Папки->Документы)
 printSubEnts(pEnt, domDoc, 1);

 delete pEnt;
}
delete itEnts;
delete pColl;
```

Полная программа, из которой взят приведенный фрагмент (TListEntitiesDD.cpp), находится в каталоге Cmbroot/Samples/cpp/dd.

В следующем примере производится получение списка атрибутов и ключевых слов документа:

## Java

```
...
DKAttrProfileDefDD profile = aDocument.getProfile();
dkCollection fields = profile.getFields();

if((fields != null) &&(fields.cardinality() > 0))
{
 dkIterator itFields = fields.createIterator();
 while(itFields.more())
 {
 DKAttrDefDD aField = (DKAttrDefDD)itFields.next();
 // ---- получаем ключевые слова
 dkCollection keywords = ((DKAttrFieldDefDD)aField).getKeywords();
 if(keywords != null)
 {
 if(keywords.cardinality() > 0)
 {
 dkIterator itKeywords = keywords.createIterator();
 while(itKeywords.more())
 {
 DKAttrDefDD aKeyword = (DKAttrDefDD)itKeywords.next();
 // ----- обрабатываем ключевое слово
 }
 }
 }
 }
}
...
```

В следующем примере перечисляются подобъекты (шкафы, папки и документы), связанные с объектом (залом, в данном случае).

**C++**

```
void printSubEnts(DKEntityDefDD* pEnt, DKDatastoreDD& domDoc, int indents)
{
 // отступы: 1=Шкафы; 2=Папки; 3=Документы
 DKString indentation = "";

 for(int i = 0; i < indents; i++)
 {
 indentation += " ";
 }

 if(pEnt->hasSubEntities())
 {
 dkCollection* pColl = pEnt->listSubEntities();
 long nbrEnts = pColl->cardinality();
 dkIterator* itEnts = pColl-> createIterator();
 while(itEnts->more())
 {
 DKEntityDefDD* pEnt = (DKEntityDefDD*)itEnts->next()->value();
 cout<< indentation << "SubEntity title: " << pEnt->getName() << endl;
 printSubEnts(pEnt, domDoc, indents+1);
 delete pEnt;
 }
 delete itEnts;
 delete pColl;
 }
 return;
}
```

## Получение списка атрибутов шкафа

Шкафы - единственные элементы, содержащие полезные атрибуты. При попытке вывести список атрибутов объекта для залов в собрании ничего не появится. Поэтому, когда DKDatastoreDD выводит список объектов поиска, он выводит только список шкафов.

В следующем примере производится получение списка шкафов и их атрибутов.

## Java

```
...
dkCollection cabinets = dsDD.listSearchableEntities();
dkIterator itCabinets = cabinets.createIterator();
while(itCabinets.more())
{
 // ----- Для каждого шкафа получаем список его атрибутов
 dkEntityDef aCabinet = (dkEntityDef)itCabinets.next();
 cabinetName = aCabinet.getName();
 // ----- Получаем список профилей документов с податрибутами
 System.out.println("\n" + Me + ": calling listAttrs for" + cabinetName);
 DKSequentialCollection coll =
 (DKSequentialCollection) aCabinet.listAttrs();
 ...
}
```

Полная программа примера, из которой взят приведенный фрагмент (TListAttributes.java), находится в каталоге CMBROOT\Samples\java\dd.

## Построение запросов в Domino.Doc

Если вы хотите ограничить запрос одним шкафом, строка запроса должна начинаться словом ENTITY=. Если параметр ENTITY и его значение отсутствуют, поиск будет выполняться по всей библиотеке. Кроме того, это значение должно быть заключено в кавычки ("). Например, "Diane Cabinet".

QUERY= - обязательный параметр.

В Domino.Doc строка запроса выглядит так:

```
"ENTITY=<"cabinetTitle"> QUERY=<"lotusQueryString">"
```

Функция FTSearch служит для запроса контент-сервера Domino.Doc. Чтобы эта функция эффективно работала, контент-сервер Domino.Doc должен быть полностью проиндексирован для текстового поиска. Для проверки наличия индекса служит свойство IsFTIndexed. Чтобы создать индекс, используйте функцию UpdateFTIndex.

Функция FTSearch выполняет поиск всех документов на контент-сервере - для поиска документов в отдельной производной таблице используйте функцию FTSearch в NotesView. Для поиска документов в отдельном собрании документов используйте функцию FTSearch в NotesDocumentCollection.

Если не задать опцию сортировки, документы будут отсортированы по релевантности. Если вам нужна сортировка по дате, данных о релевантности при сортировке результатов не будет. Если передать полученное DocumentCollection экземпляра NotesNewsletter, результаты будут

отсортированы либо по дате создания документа, либо по показателю релевантности, в зависимости от заданной опции сортировки.

## Использование синтаксиса запроса

Синтаксические правила запроса перечислены ниже. Для изменения порядка и группировки операций используйте круглые скобки.

### Простой текст

Простой текст служит для поиска слова или фразы в неизменной форме. Ключевые слова и обозначения поиска заключайте в апострофы ('). Если вы работаете внутри литерала LotusScript, не забудьте использовать кавычки (").

### Символы подстановки

Знак вопроса (?) в любом месте слова соответствует любому отдельному символу. Звездочка (\*) в любом месте слова заменяет от 0 до  $n$  (где  $n$  - любое число) символов.

### Логические операции

Логические операции служат для расширения или сужения поиска. Операции и их приоритеты:

1. ! (не)
2. & (и)
3. , (накопление)
4. | (или)

Можно использовать ключевое слово или обозначение.

### Операции близости

Операции близости служат для поиска слов, расположенных близко друг к другу. Для этих операций требуется включение в полнотекстовый индекс разделителей слов, предложений и абзацев. Есть следующие операции:

- near (вблизи)
- sentence (в предложении)
- paragraph (в абзаце)

### Операция поля

Операция поля служит для ограничения поиска заданным полем.

Синтаксис: FIELD *имя-поля* *операция* , где *операция* - ключевое слово CONTAINS (содержит) для полей текста и rtf или один из следующих символов для числового поля и поля даты: =, >, >=, <, <=

### exactcase - операция точного регистра

Операция exactcase служит для поиска следующего за ней выражения в заданном регистре.

### **termweight - операция веса условия**

Операция `termweight n` служит для настройки ранга релевантности следующего за ней выражения, где  $n$  от 0 до 100.

---

## **Работа с расширенным поиском - Extended Search (ES) Domino**

Enterprise Information Portal поддерживает IBM Extended Search 7.1, при этом поддержка ES 3 ограничена. Extended Search поддерживает запросы и получение документов из:

- Баз данных Lotus Notes.
- Баз данных NotesPump.
- Файловых систем.
- Механизмов поиска Web.

Классы и API Enterprise Information Portal поддерживают следующие возможности Extended Search:

- Соединение с одним или несколькими серверами ES и отсоединение от них.
- Получение списка серверов ES.
- Получение списка баз данных и полей.
- Использование для поиска языка Generalized Query Language (GQL).
- Получение документов.
- Использование в AIX классов и API C++ ES.
- Выполнение параметрического поиска на уровне объединения и через прямое соединение с ES.
- Использование операций текстового поиска `CONTAINS_TEXT` и `CONTAINS_TEXT_IN_CONTENT` на уровне объединения.
- Использование функций JavaBeans Enterprise Information Portal.

**Ограничение:** ES не поддерживает:

- Добавление, изменение и удаление документов.
- Механизм текстового поиска и поиск QBIC.
- Комбинированные запросы.
- Рабочие комплекты и рабочие потоки.

Все возможности ES предоставляются и управляются через базу данных конфигурации ES. База данных конфигурации служит для задания определений источников данных поиска, сетевых адресов, информации управления доступом и другой информации.

### **Получение списка серверов Extended Search**

Чтобы обеспечить доступ к нескольким серверам ES, можно создать файл `ctmbdes.ini`, который содержит информацию о серверах. Сохраните этот файл в



каталоге `C:\CMBROOT` (где *C* - буква диска). Файл `cmbdes.ini` должен содержать по одной строке для каждого сервера в следующем формате:  
`DATASOURCE=TCP/IP адрес;PORT=номер порта`

где *TCP/IP* - IP-адрес сервера ES, а *номер порта* - номер порта, задаваемый для доступа к серверу (например, `PORT=80`).

### **Получение списка объектов (баз данных) и атрибутов (полей)**

При построении запроса для поиска сервера ES надо знать имена доступных баз данных и полей. У объекта `DKDatastoreDES` есть функция `listEntities` для получения списка баз данных и функция `listEntityAttrs` для получения списка полей каждой базы данных. В следующем примере показано, как получить список баз данных и их полей.

## Java

```
try {
 DKSequentialCollection pCol = null;
 dkIterator pIter = null;
 DKSequentialCollection pCol2 = null;
 dkIterator pIter2 = null;
 String strDatabase = null;
 DKDatabaseDefDES dbDef = null;
 DKFieldDefDES attrDef = null;
 int i = 0;
 int j = 0;
 DKDatastoreDES dsDES = new DKDatastoreDES();
 System.out.println("connecting to datastore");
 dsDES.connect(libSrv,userid,pw,connect_string);
 System.out.println("datastore connected libSrv " + libSrv + " userid " +
 userid);
 System.out.println("list DES databases");
 pCol = (DKSequentialCollection) dsDES.listEntities();
 pIter = pCol.createIterator();
 i = 0;
 while (pIter.more() == true)
 {
 i++;
 dbDef = (DKDatabaseDefDES)pIter.next();
 strDatabase = dbDef.getName();
 System.out.println("database name [" + i + "] - " + strDatabase);
 System.out.println(" list attributes for " + strDatabase + " database");
 pCol2 = (DKSequentialCollection) dsDES.listEntityAttrs(strDatabase);
 pIter2 = pCol2.createIterator();
 j = 0;
 while (pIter2.more() == true)
 {
 j++;
 attrDef = (DKFieldDefDES)pIter2.next();
 System.out.println("Attr name [" + j + "] - " + attrDef.getName());
 System.out.println(" datastoreType " + attrDef.datastoreType());
 System.out.println(" attributeOf " + attrDef.getEntityName());
 System.out.println(" type " + attrDef.getType());
 System.out.println(" size " + attrDef.getSize());
 System.out.println(" id " + attrDef.getId());
 System.out.println(" nullable " + attrDef.isNullable());
 System.out.println(" precision " + attrDef.getPrecision());
 System.out.println(" scale " + attrDef.getScale());
 System.out.println(" stringType " + attrDef.getStringType());
 System.out.println(" queryable " + attrDef.isQueryable());
 System.out.println(" displayName " + attrDef.getDisplayName());
 System.out.println(" helpText " + attrDef.getHelpText());
 System.out.println(" language " + attrDef.getLanguage());
 System.out.println(" valueCount " + attrDef.getNumVals());
 }
 }
 // продолжение следует...
```

### Java (продолжение)

```
 System.out.println(" " + j + " attributes listed for
 " + strDatabase + " database");
 }
 System.out.println(i + " databases listed");
 dsDES.disconnect();
 System.out.println("datastore disconnected");
 }
 catch(DKException exc)
 {
 System.out.println("Exception name " + exc.name());
 System.out.println("Exception message " + exc.getMessage());
 System.out.println("Exception error code " + exc.errorCode());
 System.out.println("Exception error state " + exc.errorState());
 exc.printStackTrace();
 }
}
```

Полная программа примера, из которой взят приведенный фрагмент (TListAttributes.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
...
cout << "list entities" << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsDES.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
 i++;
 pEnt = (DKDatabaseDefDES*)((void*)(*pIter->next()));
 strDBName = pEnt->getName();
 cout << "\ndatabase name [" << i << "] - " << strDBName << endl;
 cout << "dispname: " << pEnt->getDisplayName() << endl;
 cout << "helptext: " << pEnt->getHelpText() << endl;
 cout << "lang: " << pEnt->getLanguage() << endl;
 int iVCount = pEnt->getNumVals();
 cout << "NumVals: " << iVCount << endl;
 cout << "datatype: " << pEnt->getDataType() << endl;
 cout << "searchable:" << pEnt->isSearchable() << endl;
 cout << "retrievable" << pEnt->isRetrievable() << endl;
 cout << "\n list attributes for " << strDBName << " database name" << endl;
 pCol2 =
 (DKSequentialCollection*)((dkCollection*)dsDES.listEntityAttrs(strDBName));
 pIter2 = pCol2->createIterator();
 j = 0;
 while (pIter2->more() == TRUE)
 {
 j++;
 pA = pIter2->next();
 pAttr = (DKFieldDefDES*) pA->value();
 cout << "Attr name [" << j << "] - " << pAttr->getName() << endl;
 cout << " datastoreName " << pAttr->datastoreName() << endl;
 cout << " datastoreType " << pAttr->datastoreType() << endl;
 cout << " attributeOf " << pAttr->getEntityName() << endl;
 cout << " type " << pAttr->getType() << endl;
 cout << " size " << pAttr->getSize() << endl;
 cout << " id " << pAttr->getId() << endl;
 cout << " nullable " << pAttr->isNullable() << endl;
 cout << " precision " << pAttr->getPrecision() << endl;
 cout << " scale " << pAttr->getScale() << endl;
 cout << " string type " << pAttr->getStringType() << endl;
 cout << " display name " << pAttr->getDisplayName() << endl;
 cout << " help text " << pAttr->getHelpText() << endl;
 cout << " language " << pAttr->getLanguage() << endl;
 cout << " isQueryable " << pAttr->isQueryable() << endl;
 cout << " isRetrievable " << pAttr->isRetrievable() << endl;
 delete pAttr;
 }
 cout << " " << j << " attributes listed for "
 << strDBName << " database name" << endl;
 // продолжение следует...
```

### С++ (продолжение)

```
 delete pIter2;
 delete pCol2;
 delete pEnt;
 }
 delete pIter;
 delete pCol;
 cout << i << " entities listed\n" << endl;
 ...
}
```

Полная программа примера, из которой взят приведенный фрагмент (TListCatalogDES.cpp), находится в каталоге Cmbroot/Samples/cpp/ES.

## Использование языка GQL (Generalized Query Language - обобщенный язык запросов)

Extended Search использует для поиска язык Generalized Query Language (GQL). В Табл. 20 приведены примеры допустимых выражений GQL.

Таблица 20. Выражения GQL

| Выражение GQL               | Описание                                                                              |
|-----------------------------|---------------------------------------------------------------------------------------|
| "программы"                 | Выполняет поиск документов, содержащих слово программы.                               |
| (TOKEN:WILD "ехес*")        | Выполняет поиск документов, содержащих любое слово, начинающееся с ехес.              |
| (AND "программы" "IBM")     | Выполняет поиск документов, содержащих одновременно слова программы и IBM.            |
| (START "View" "How")        | Выполняет поиск документов, в которых поле View начинается со слова Как.              |
| (EQ "View" "Как сделать?")  | Выполняет поиск документов, в которых поле View содержит точную строку Как сделать?.  |
| (GT "BIRTHDATE" "19330804") | Выполняет поиск документов, в которых поле BIRTHDATE больше, чем 4 августа 1933 года. |

ES использует тип запроса DK\_DES\_GQL\_QL\_TYPE. Синтаксис такого запроса:

```
SEARCH=(DATABASE=(имя_бд | список_имен_бд | ALL);
 COND=(выражение GQL));
[OPTION=([SEARCHABLE_FIELD=(имя_поля, ...);]
 [RETRIEVABLE_FIELD=(имя_поля, ...);]
 [MAX_RESULTS=число_результатов;]
 [TIME_LIMIT=время])]
```

где список\_имен\_бд - список имен баз данных (имя\_бд - одно имя), разделенных запятыми; ALL означает поиск во всех доступных базах данных. Лимит времени по умолчанию 30 секунд.

В этом примере строка запроса используется для поиска в базе данных Notes Help документов, где поле View содержит текст Как сделать?, причем максимальное число ожидаемых результатов - пять.

```
String cmd = "SEARCH=(DATABASE=(Notes Help);" +
 "COND=(EQ \"View\" \"Как сделать?\");" +
 "OPTION=(MAX_RESULTS=5)"
```

В этом примере выполняется запрос GQL для ES. После выполнения запроса результаты возвращаются в объекте dkResultSetCursor.

#### Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
System.out.println("connecting to datastore");
dsDES.connect(libSrv,userid,pw,connect_string);
System.out.println("datastore connected libSrv " + libSrv + " userid " +
 userid);
String cmd = "SEARCH=(DATABASE=(Notes Help);" +
 "COND=(EQ \"View\" \"Как сделать?\");" +
 "OPTION=(MAX_RESULTS=5)";
DKDDO ddo = null;
System.out.println("query string " + cmd);
System.out.println("executing query");
pCur = dsDES.execute(cmd,DK_DES_GQL_QL_TYPE,parms);

System.out.println("cardinality " + pCur.cardinality());
System.out.println("datastore executed query");
System.out.println("process query results");
...
pCur.destroy(); // завершив работу, удаляем указатель
System.out.println("query results processed");
dsDES.disconnect();
System.out.println("datastore disconnected");
```

Полная программа примера, из которой взят приведенный фрагмент (TExecuteDES.java), находится в каталоге CMBROOT\Samples\java\dl.

### C++

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore ES created" << endl;
cout << "connecting to datastore" << endl;
dsDES.connect(libsrv,userid,pw,str);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;

DKString cmd = "SEARCH=(DATABASE=(Notes Help));";
cmd += "COND=((IN \"Subject\" \"your\"));";
cmd += "OPTION=(MAX_RESULTS=2;TIME_LIMIT=10);";

cout << "query string " << cmd << endl;
cout << "executing query" << endl;
pCur = dsDES.execute(cmd);
cout << "query executed" << endl;
...
```

Полная программа примера, из которой взят приведенный фрагмент (TExecutedES.cpp), находится в каталоге Cmbroot/Samples/cpp/ES.

## Идентификация типа элемента объекта DDO в Extended Search

Объект DDO в ES всегда имеет тип DK\_CM\_DOCUMENT. Чтобы получить тип элемента объекта DDO, вызовите:

### Java

```
Object obj = ddo.getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
short type = ((Short) obj).shortValue();
```

### C++

```
DKDDO *p = 0;
ushort k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (k > 0)
{
 DKAny a = p->getProperty(k);
 ushort val = a; // val = DK_CM_DOCUMENT
}
```

## Создание PID в Extended Search

Постоянный идентификатор (PID) содержит определенную информацию о документе. Тип объекта идентифицирует базу данных, где документ был найден. PID создается из имени базы данных, за которым следует символ | и ID документа, например:

```
database name|documentId ()
```

Дополнительную информацию об идентификаторах PID смотрите в разделах “Постоянные идентификаторы (PID)” на стр. 17 и “Создание постоянного идентификатора (PID)” на стр. 53.

## Обработка содержимого документа Extended Search

Каждый элемент в DDO представляет поле, собрание или объект DKParts.

**Поле** Имя поля для отдельного поля находится внутри имени элемента. Значение поля также находится внутри значения элемента. Возможные свойства поля:

- DK\_CM\_VSTRING
- DK\_CM\_FLOAT
- DK\_CM\_XDOOBJECT
- DK\_CM\_DATE
- DK\_CM\_SHORT

### Собрание

Когда у поля есть несколько значений, имя поля находится в имени элемента. Значение элемента - объект DKSequentialCollection. Свойство может быть DK\_CM\_COLLECTION или DK\_CM\_COLLECTION\_XDO, если поле - двоичный большой объект.

### DKParts

DDO-документ имеет специальный атрибут - зарезервированное имя DKPARTS. Его значение - объект DKParts. Объект DKPARTS может хранить информацию адреса Web (URL) о документе. DKPARTS может также содержать XDO и его содержимое как строку, представляющую URL документа.

В этом примере обрабатывается содержимое объекта DDO:



## Java

```
public static void displayDDO(DKDDO ddo)
 throws DKException, Exception
{
 dkXDO pXDO = null;
 int i = 0;
 int numDataItems = 0;
 short k = 0;
 short j = 0;
 Integer valueCount = null;
 Object value = null;
 String dataName = null;
 dkCollection pCol = null;
 dkIterator pIter = null;
 Object anObject = null;
 numDataItems = ddo.dataCount();
 DKPid pid = ddo.getPidObject();
 System.out.println("pid string " + pid.pidString());

 System.out.println("Number of Attributes " + numDataItems);
 for (j = 1; j <= numDataItems; j++)
 {
 anObject = ddo.getData(j);
 dataName = ddo.getDataName(j);
 System.out.println(j + ": Name " + dataName);
 // определяем, имеет ли элемент данных одно или много значений
 Short type = (Short)ddo.getDataPropertyByName(j, DK_CM_PROPERTY_TYPE);
 k = type.shortValue();
 if (k == DK_CM_COLLECTION)
 {
 pCol = (dkCollection)anObject;
 pIter = pCol.createIterator();
 i = 0;
 while (pIter.more() == true)
 {
 i++;
 value = pIter.next();
 System.out.println(" Value" + i + " " + value);
 }
 }
 else if (k == DK_CM_COLLECTION_XDO)
 {
 pCol = (dkCollection)anObject;
 pIter = pCol.createIterator();
 i = 0;
 while (pIter.more() == true)
 {
 i++;
 blob = (DKBlobDES)pIter.next();
 }
 }
 }
 // продолжение следует...
```

#### Java (продолжение)

```
 System.out.println(" Value" + i + " " + blob.getContent());
 }

 else
 System.out.println(" Value " + anObject);
}
```

Полная программа примера, из которой взят приведенный фрагмент (TExecutedES.java), находится в каталоге CMBROOT\Samples\java\d1.

**C++**

```
DKDDO *p = 0;
dkDataObjectBase *pdOBase = 0;
DKDDO *pDDO = 0;
dkXDO *pXDO = 0;
DKAny a;
ushort j = 0;
ushort k = 0;
ushort val = 0;
ushort cnt = 1;
DKString strData = "";
DKString strDataName = "";
dkCollection* pdCol = 0;
dkIterator* pdIter = 0;
ushort numDataItems = 0;
DKPidXDOES *pidXDO = 0;
DKPid *pid = 0;
DKString strPid;
long pidIdCnt = 0;
long pidIndex = 0;
while (pCur->isValid())
{
 p = pCur->fetchNext();
 if (p != 0)
 {
 cout << "=====> " << "Item " << cnt << " <=====" << endl;
 numDataItems = p->dataCount();
 pid = (DKPid*)p->getPidObject();
 strPid = pid->pidString();
 cout << "pid string " << strPid << endl;
 cout << "pid id string " << pid->getId() << endl;
 strPid = pid->getIdString();
 cout << "pid idString " << strPid << endl;
 pidIdCnt = pid->getIdStringCount();
 cout << "pid idString cnt " << pidIdCnt << endl;
 strPid = pid->getPrimaryId();
 cout << "pid primary id " << strPid << endl;
 pidIndex = 0;
 strPid = pid->getIdString(pidIndex);
 cout << "pid item id " << strPid << endl;
 k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
 if (k > 0)
 {
 a = p->getProperty(k);
 val = a;
 cout << "*****" << endl;
 switch (val)
 {
 case DK_CM_DOCUMENT:
 cout << "Item is a document " << endl;
 break;
 }
 }
 }
}
// продолжение следует...
```



## C++ (продолжение)

```
default:
 cout << " Item is not recognized " << endl;
 break;
}
cout << "*****" << endl;
}
cout << "Number of Data Items " << numDataItems << endl;
for (j = 1; j <= numDataItems; j++)
{
 a = p->getData(j);
 strDataName = p->getDataName(j);
 switch (a.typeCode())
 {
 case DKAny::tc_string :
 {
 strData = a;
 cout << "attribute name : " << strDataName << " value : "
 << strData << endl;
 }
 break;
 case DKAny::tc_long :
 {
 long l = a;
 cout << "attribute name : " << strDataName << " value : " << l
 << endl;
 }
 break;
 case DKAny::tc_double :
 {
 double db = a;
 cout << "attribute name : " << strDataName << " value : " << db
 << endl;
 }
 break;
 case DKAny::tc_timestamp :
 {
 DKTimestamp tt = a;
 cout << "attribute name : " << strDataName << " value : "
 << tt.getMonth() << "/" << tt.getDay() << "/" << tt.getYear() << " "
 << tt.getHours() << ":" << tt.getMinutes() << ":" << tt.getSeconds()
 << endl;
 }
 break;
 case DKAny::tc_dobase :
 {
 pDOBase = a;
 pXDO = (dkXDO*)pDOBase;
 cout << "attribute name : " << strDataName << " value : " << endl;
 pidXDO = (DKPidXDOES*)pXDO->getPid();
 cout << "XDO pid database name " << pidXDO->getDatabaseName() << endl;
 // продолжение следует...
```

### C++ (продолжение)

```
cout << "XDO pid docId " << pidXDO->getDocId() << endl;
cout << "XDO mimetype " << pXDO->getMimeType() << endl;
((DKBlobDES*)pXDO)->getContentToClientFile("c:\\temp\\temp.html", 1);
}
break;
case DKAny::tc_collection :
{
 pdCol = a;
 cout << strDataName << " collection name : " << strDataName
 << endl;
 cout << "-----" << endl;
 pdIter = pdCol->createIterator();
 ushort b = 0;
 while (pdIter->more() == TRUE)
 {
 b++;
 cout << " -----" << endl;
 a = *(pdIter->next());
 switch (a.typeCode())
 {
 case DKAny::tc_string :
 {
 strData = a;
 cout << "attribute name : " << strDataName << " value : " << strData << endl;
 }
 break;
 case DKAny::tc_long :
 {
 long l = a;
 cout << "attribute name : " << strDataName << " value : " << l << endl;
 }
 break;
 case DKAny::tc_double :
 {
 double db = a;
 cout << "attribute name : " << strDataName << " value : " << db << endl;
 }
 break;
 case DKAny::tc_timestamp :
 {
 DKTimestamp tt = a;
 cout << "attribute name : " << strDataName << " value : "
 << tt.getMonth() << "/" << tt.getDay() << "/" << tt.getYear() << " "
 << tt.getHours() << ":" << tt.getMinutes() << ":" << tt.getSeconds() << endl;
 }
 }
 }
 // продолжение следует...
```

## C++ (продолжение)

```
 break;
 case DKAny::tc_dobase :
 {
 pDOBase = a;
 pXDO = (dkXDO*)pDOBase;
 cout << "attribute name : " << strDataName << " value : " << endl;
 pidXDO = (DKPidXDOES*)pXDO->getPid();
 cout << "XDO pid database name " << pidXDO->getDatabaseName() << endl;
 cout << "XDO pid docId " << pidXDO->getDocId() << endl;
 cout << "XDO mimetype " << pXDO->getMimeType() << endl;
 DKString str = "c:\\temp\\temp";
 DKString strT = b;
 str = str + strT + ".html";
 ((DKBlobDES*)pXDO)->getContentToClientFile(str, 1);
 }
 break;
 }
 ushort usCount = p->dataPropertyCount(j);
 for (ushort k = 1; k <= usCount; k++)
 {
 a = p->getDataProperty(j, k);
 cout << " property " << k << " " << a << endl;
 }
 if (b == 0)
 {
 cout << strDataName << " collection has no elements " << endl;
 }
 cout << " -----" << endl;
 break;
}
}
ushort usCount = p->dataPropertyCount(j);
for (ushort k = 1; k <= usCount; k++)
{
 a = p->getDataProperty(j, k);
 cout << " property " << k << " " << a << endl;
}
}
cnt++;
delete p;
```

Полная программа примера, из которой взят приведенный фрагмент (TExecutedES.cpp), находится в каталоге Cmbroot/Samples/cpp/ES.

## Получение документа

Чтобы получить документ из объекта DKDatastoreDES, надо знать имя базы данных, которая содержит документ, и ID документа. Надо также связать объект DDO с контент-сервером и установить соединение. В этом примере выполняется получение документа:

### Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dsDES.connect(libSrv, userid, pw, connect_string);
DKPid pid = new DKPid();
// ----- Первичный ID - это 'имя базы данных', за которым идет
// символ '|' и ID документа
pid.setPrimaryId("Notes Help" + DK_DES_ITEMID_SEPARATOR + "215e");
pid.setObjectType("Notes Help");
DKDDO ddo = new DKDDO(dsDES, pid);
ddo.retrieve();
```

Полная программа примера, из которой взят приведенный фрагмент (TRetrieveDDODES.java), находится в каталоге CMBROOT\Samples\java\d1.

### C++

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore ES created" << endl;
cout << "connecting to datastore" << endl;
dsDES.connect(libsrv,userid,pw,str);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;
...
p = new DKDDO(&dsDES, "");
DKPid pid2;
pid2.setDatastoreType(dsDES.datastoreType());
pid2.setDatastoreName(dsDES.datastoreName());
pid2.setId("Notes Help|215e");
pid2.setObjectType("");
p->setPidObject((DKPid*)&pid2);
p->retrieve();
...
```

Полная программа примера, из которой взят приведенный фрагмент (TRetrieveDDODES.cpp), находится в каталоге Cmbroot/Samples/cpp/ES.

## Получение двоичного большого объекта

Чтобы получить двоичный большой объект из объекта DKDatastoreDES, надо знать имя базы данных, где находится документ, ID документа, который



содержит двоичный большой объект, и имя поля, содержащего двоичный большой объект. Надо также связать объект DDO с контент-сервером и установить соединение.

В следующем примере база данных файловой системы под именем ES files содержит файл HTML D:\desdoc\README.html. Поле, которое содержит файл HTML, называется Doc\$Content. Следующий пример получает файл HTML и сохраняет его как D:\DESReadme.html.

#### Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dsDES.connect(libSrv, userid, pw, connect_string);
DKBlobDES xdo = new DKBlobDES(dsDES);
DKPidXDODES pid = new DKPidXDODES();
pid.setDocumentId("D:\\desdoc\\README.html");
pid.setDatabaseName("DES files");
pid.setFieldName("Doc$Content");
xdo.setPidObject(pid);
xdo.retrieve("c:\\DESReadme.html");
```

Полная программа примера, из которой взят приведенный фрагмент (TRetrieveXDODES.java), находится в каталоге CMBROOT\Samples\java\d1.

## C++

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore ES created" << endl;
cout << "connecting to datastore" << endl;
dsDES.connect(libsrv,userid,pw,str);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;
...
cout << "executing retrieve a XDO" << endl;

DKBlobDES* p = new DKBlobDES(&dsDES);
DKPidXDODES pid;
pid.setDocId("D:\\desdoc\\README.html");
pid.setDatabaseName("ES files");
pid.setFieldName("Doc$Content");
pid.setPrimaryId("ES files|D:\\desdoc\\README.html");
p->setPidObject((DKPidXDO*)&pid);

p->retrieve("c:\\temp\\DESReadme.html");

cout << "retrieve executed" << endl;
...
```

Полная программа примера, из которой взят приведенный фрагмент (TRetrieveXDODES.cpp), находится в каталоге Cmbroot/Samples/cpp/ES.

## Связывание типов MIME с документами

ES не поддерживает непосредственно идентификацию типов MIME (Multipurpose Internet Mail Extension). Однако если вы хотите вывести объект XDO при помощи браузера, знать его тип MIME необходимо.

Файл CMBCC2MIME.INI позволяет определить тип MIME документа. Когда запрос ES из баз данных NotesPump или FileSystem возвращает двоичный большой объект, выполняется поиск в файле CMBCC2MIME.INI, чтобы узнать тип MIME, который можно присвоить большому двоичному объекту. Тип MIME по умолчанию - text/html. Файл примера с именем cmbcc2mime.ini.samp находится в каталоге samples.

## Использование поиска в объединении в Extended Search

Синтаксис запросов объединения в ES аналогичен синтаксису SQL. Выражения запроса объединения перед передачей в ES преобразуются в синтаксис GQL. Поскольку грамматика SQL и GQL все же различна, только подмножество грамматики SQL поддерживается в среде Enterprise Information Portal.

В Табл. 21 на стр. 447 приводится сводка преобразования SQL в GQL для совместимых операций сравнения и логических операций.

Таблица 21. операции SQL и GQL

| операция SQL | операция GQL      |
|--------------|-------------------|
| AND          | AND               |
| OR           | OR                |
| NOT          | не поддерживается |
| IN           | не поддерживается |
| BETWEEN      | BETWEEN           |
| EQ           | EQ                |
| NEQ          | не поддерживается |
| GT           | GT                |
| LT           | LT                |
| LIKE         | не поддерживается |
| GEQ          | GTE               |
| LEQ          | LTE               |
| NOTLIKE      | не поддерживается |
| NOTIN        | не поддерживается |
| NOTBETWEEN   | не поддерживается |

## Работа с Panagon Image Services (только для Java)

Можно использовать классы API Enterprise Information Portal для Panagon Image Services (FileNET), чтобы обращаться к содержимому на серверах Panagon Image Services. Поддержка Panagon Image Services предоставляется только как специальная возможность EIP.

**Ограничение:** Panagon Image Services не поддерживает механизм текстового поиска, поиск QWIC и комбинированные запросы.

## Моделирование данных

В Enterprise Information Portal есть несколько понятий моделирования данных, которые необходимо сопоставить с соответствующими объектами на контент-сервере, поддерживаемом соединителем. В следующей таблице показаны модели данных EIP в среде FileNET.

| Понятия EIP    | FileNET                       |
|----------------|-------------------------------|
| контент-сервер | сервер Panagon Image Services |
| сервер         | сервер                        |
| объект         | класс документа               |

| Понятия EIP                      | FileNET                                                                                                                            |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| атрибут                          | индекс                                                                                                                             |
| DDO                              | документ (папка)<br><br>В FileNET папки используются для временной организации документов; в соединителе они пока не моделируются. |
| XDO                              | содержимое страницы, комментарий                                                                                                   |
| PID для DDO                      | id_документа как первичный ID                                                                                                      |
| PID для XDO содержимого страницы | Порядковый_номер_системы + ID_документа + номер_страницы                                                                           |
| PID для XDO аннотации            | Порядковый_номер_системы + ID_документа + номер_страницы + ID_аннотации                                                            |

Соединитель FileNET возвращает только пользовательские индексы (или атрибуты) в FileNET, которые можно использовать в отображениях объектов и атрибутов объединения и в списках вывода атрибутов результатов поиска. В следующей таблице показано отображение поддерживаемых типов данных индексов или атрибутов FileNET в соответствующие типы данных EIP в соединителе FileNET:

| Тип данных индекса FileNET | Тип данных атрибута EIP                                                                   |
|----------------------------|-------------------------------------------------------------------------------------------|
| числовой                   | double                                                                                    |
| дата                       | varchar, у которого max_length имеет максимальную длину, определенную для индекса FileNET |
| меню                       | varchar с max_length = 14 (максимум для названий меню)                                    |

Все объекты XDO в EIP имеют тип MIME. Клиенты EIP используют этот тип MIME для определения вывода документов. Соединитель для FileNET полагает, что у всех XDO одного документа один и тот же тип MIME. Тип MIME объекта XDO устанавливается согласно значению атрибута системы FileNET F\_DOCFORMAT того документа, которому принадлежит данный XDO. Если оно пусто, устанавливается тип MIME image/tiff, если атрибут системы FileNET F\_DOCTYPE указывает на тип image; иначе будет установлен тип MIME по умолчанию application/octet-stream.

## Документы и страницы в Panagon Image Services

Чтобы использовать DDO для представления документов в Panagon Image Services, необходимо правильно задать четыре поля PID для объектов DDO:

1. `DatastoreType` — константа `DK_FN_DSTYPE`, заданная в `DKConstantFN`. Это значение можно получить при помощи метода `datastoreType()` из `DKDatastoreFN`.
2. `DatastoreName` — это имя домена-организации FileNET. Это значение можно получить при помощи метода `datastoreName()` из `DKDatastoreFN`.
3. `ObjectType` — это имя класса документа FileNET (имя собственного объекта в EIP), к которому принадлежит документ.
4. `PrimaryID` — это номер документа FileNET, назначенный FileNET.

После того, как задан PID, программы могут получать атрибуты или содержимое DDO, вызывая метод `retrieve()` в классе `DKDatastoreFN` или `DKDDO`.

У DDO документа в EIP есть специальный атрибут с зарезервированным именем `DKPARTS`, значением которого является объект `DKParts`. Объект `DKParts` - это собрание двоичных больших объектов. Части FileNET (страницы/комментарии) в документе представляются как объекты `DKBlobFN` (подкласс `DKXDO`), являющиеся элементами в собрании `DKParts`.

Страница FileNET - это не вполне обычная страница; документ с единственной страницей (или единственной частью) в FileNET может содержать несколько физических страниц, но все они хранятся на одной странице в документе FileNET. В FileNet многостраничный документ содержит несколько частей или несколько файлов, создающихся с помощью Panagon Capture Software или Batch Entry System.

## Получение списка объектов и атрибутов

Сервер Panagon Image Services представляется как объект `DKDatastoreFN`. После создания контент-сервера и соединения с ним можно вывести список объектов (документов и классов) и атрибутов для сервера Panagon Image Services. Это показано в следующем примере:

## Java

```
DKDatastoreFN dsFN = null;
try {
 DKSequentialCollection pCol = null;
 dkIterator pIter = null;
 DKSequentialCollection pCol2 = null;
 dkIterator pIter2 = null;
 DKServerDefFN pSV = null;
 String strServerName = null;
 String strServerType = null;
 String strEntity = null;
 DKEntityDefFN entityDef = null;
 DKAttrDefFN attrDef = null;
 int i = 0;
 int j = 0;
 dsFN = new DKDatastoreFN();
// ----- соединяемся с сервером, используя имя сервера, ID пользователя и
// пароль
 dsFN.connect(libSrv, userid, pw, "");
 System.out.println("Datastore connected libSrv " + libSrv + " userid "
 + userid);
 System.out.println("List entities in server " + libSrv);
 pCol = (DKSequentialCollection) dsFN.listEntities();
 pIter = pCol.createIterator();
 i = 0;
 while (pIter.more() == true)
 {
 i++;
 entityDef = (DKEntityDefFN)pIter.next();
 strEntity = entityDef.getName();
 System.out.println("\nEntity name [" + i + "] - " + strEntity +
 ", id = " + entityDef.getId() + ", type = " + entityDef.getType());
 System.out.println(" List attr for the " + strEntity + " entity");
 pCol2 = (DKSequentialCollection) dsFN.listEntityAttrs(strEntity);
 pIter2 = pCol2.createIterator();
 j = 0;
 while (pIter2.more() == true)
 {
 j++;
 attrDef = (DKAttrDefFN)pIter2.next();
 System.out.println(" Attribute name [" + j + "] - " +
 attrDef.getName());
 System.out.println(" datastoreType = " +
 attrDef.datastoreType());
 System.out.println(" attributeOf = " +
 attrDef.getEntityName());
 System.out.println(" type = " + attrDef.getType());
 System.out.println(" size = " + attrDef.getSize());
 System.out.println(" id = " + attrDef.getId());
 }
 }
// продолжение следует...
```

### Java (продолжение)

```
 System.out.println(" nullable = " + attrDef.isNullable());
 System.out.println(" precision = " + attrDef.getPrecision());
 System.out.println(" scale = " + attrDef.getScale());
 System.out.println(" stringType = " + attrDef.getStringType());
 }
 System.out.println(" Total of " + j + " attributes listed for the "
 + strEntity + " entity");
}
System.out.println("\nTotal of " + i + " entities listed for server "
 + libSrv);
// ----- Отсоединение и очистка
dsFN.disconnect();
dsFN.destroy();
}
catch(DKException exc)
```

Смотрите программы примеров в каталоге CMBROOT\Samples\java\fn.

## Запросы

EIP поддерживает три основные формы запроса для Panagon Image Services:

- объект запроса
- командная строка
- DKCQExpr

Две формы запроса - объект запроса и командная строка - обычно используются в программах, обращающихся к серверу Panagon Image Services напрямую. На контент-сервере и в запросе объединения EIP для взаимодействия с отдельными контент-серверами используется только форма DKCQExpr.

Примеры прикладных программ для запросов к серверам Panagon Image Services находятся в каталоге CMBROOT\Samples\java\fn.

### Синтаксис и параметры строки запроса

Соединитель FileNET поддерживает использование DKParametricQuery для передачи и выполнения запроса. Запрос DKParametricQuery может быть построен с командной строкой. Командная строка представляет спецификацию запроса для условия фильтрации FileNET. Другие опции запросов (ENTITY\_NAME, MAX\_RESULTS, CONTENT) и ключевое условие FileNET (KEY) будут приниматься как параметры для методов evaluate() или execute() в парах (имя, значение), например, DKNVPair.

Обратите внимание на то, что тип значений - всегда String, а имена параметров определяются в DKConstant и DKConstantFN. Возвращаемые результаты - это

собрание объектов DDO документа для evaluate(), а указатель набора результатов указывает на результаты для execute().

За синтаксисом запроса FileNET будет следовать синтаксис выражения условий для команды и параметр KEY, поддерживаемые функцией FileNET WAL PRS\_Parse(): Он допускает следующие операции:

| Операция   | Применение                                                                                                                                                                                              |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| AND        | логическое И                                                                                                                                                                                            |
| OR         | логическое ИЛИ                                                                                                                                                                                          |
| NOT        | логическое НЕ                                                                                                                                                                                           |
| <          | меньше, чем                                                                                                                                                                                             |
| <=         | меньше или равно                                                                                                                                                                                        |
| =          | равно                                                                                                                                                                                                   |
| >          | больше, чем                                                                                                                                                                                             |
| >=         | больше или равно                                                                                                                                                                                        |
| !=         | не равно                                                                                                                                                                                                |
| +          | сложение                                                                                                                                                                                                |
| -          | вычитание                                                                                                                                                                                               |
| *          | умножение                                                                                                                                                                                               |
| /          | деление                                                                                                                                                                                                 |
| IN RANGE   | Диапазон задается в виде IN RANGEуслконстантауслконстанта, где усл - <, <=, > или >=, а константа - числовая или строчная константа                                                                     |
| LIKE       | Подобие. Шаблон справа от операции LIKE - это строка символов в одинарных кавычках, которая может содержать знаки подстановки ? (соответствует одному символу) и * (соответствует произвольной строке). |
| DEFINED(x) | Функция defined возвращает ненулевое значение, если для данного имени столбца x задано непустое значение, и ноль, если задано пустое значение.                                                          |

Операнды - это числовые константы, константы типа string и имена атрибутов. Поскольку командная строка используется для поиска контент-серверов FileNET напрямую, имена атрибутов - это все локальные имена атрибутов/индексов FileNET. Никакие имена атрибутов объединения и никакое отображение схем не поддерживаются и не требуются.



Числа задаются в формате `+ddd.dddE+eee` где `ddd` - 0 или больше десятичных цифр, `+` означает `+` или `-` (`+` можно опускать), `eee` - от 0 до трех цифр показателя степени, `.` - десятичная точка (необязательна), а `E` означает начало экспоненты (необязательно).

Строчные константы заключаются в одинарные кавычки, а знак одинарной кавычки внутри строки представляется двумя последовательными знаками одинарной кавычки.

Выражение условий для командной строки может содержать любое число операндов и операторов. В фильтре можно также использовать круглые скобки для задания порядка действий. Примеры условий фильтра:

```
PROD_PRICE > 100 AND (PROD_DATE > '10/15/92' OR PROD_NAME LIKE 'comp*')
PROD_PRICE >= 100 AND (NOT (PROD_ID < 30))
```

Выражение условий для параметра `KEY` должно содержать только одно имя атрибута, определяемое в качестве поискового ключа в FileNET, и только одно условие. Обратите внимание на то, что надо использовать имена ключей, определенные в FileNET; необращенное имя столбца использовать нельзя. Примеры строк ключей:

```
PROD_PRICE = 100000
PROD_PRICE IN RANGE >= 100000 <= 200000
```

### Дополнительные опции поиска

Кроме того, поддерживаются необязательные параметры для `evaluate()` и `execute()`:

#### (ENTITY\_NAME, имя\_объекта):

Задайте имя объекта (имя класса документа FileNET) в качестве области запроса. Пример:

```
parms[1] = new DKNVPair (DK_FN_PARM_ENTITY_NAME, имя_объекта);
```

К фактической строке запроса, переданной на сервер Panagon Image Services, будет добавлено `AND (F_DOCCLASSNUMBER = номер_класса_документа)`, где `= номер_класса_документа` - совпадающий номер класса документа для заданного имени объекта. При его отсутствии поиск будет производиться по всем классам документов FileNET.

#### (MAX\_RESULTS, число\_результатов)

В собрание результатов будет возвращено только заданное максимальное число результатов. Если это значение равно нулю или не задано, будет возвращено максимальное число результатов, которое допустимо в FileNET. Пример:

```
parms[0] = new DKNVPair(DK_CM_PARM_MAX_RESULTS, число_результатов);
```

### (CONTENT, YES / NO / ATTRONLY)

Этот параметр переопределяет набор опций DK\_CM\_OPT\_CONTENT на уровне контент-сервера.

- YES- документы результатов будут получены с их содержанием (по умолчанию).
- NO - в объектах DDO результатов задаются только ID документов.
- ATTRONLY - в объектах DDO результатов будут заданы ID документов, атрибуты данных и их значения.

Пример:

```
dsFN.setOption(DK_CM_OPT_CONTENT, DK_CM_CONTENT_YES);
```

Например, при значении содержимого YES в DDO документов результатов будут задаваться PID, тип объектов, свойства и весь набор атрибутов. Кроме того, задание этой опции приводит к тому, что для атрибута DKPARTS устанавливается значение для собрания частей содержимого (объекты XDO для частей и их аннотаций) с набором информации PID, но без реального содержимого. Если для CONTENT установить значение ATTRONLY, будут задаваться PID объектов DDO документов результатов, тип объектов, свойства и все атрибуты данных. Если для CONTENT установить значение NO, будут задаваться PID объектов DDO документов результатов, тип объектов и свойства. Часть документа или содержимое XDO при необходимости может быть получено явно с использованием метода retrieve() класса DKBlobFN.

### Обработка исключительных ситуаций и сообщения

Соединитель Panagon Image Services пользуется теми же классами исключительных ситуаций Java, определенных в EIP. В текстовом сообщении, включаемом в исключительную ситуацию, используется также (где это возможно) одно из общих определенных сообщений из DKMessageID.

Например, исключительная ситуация DKUsageError содержит сообщение (DKMessage.getMessage(DK\_CM\_MSG\_NOTIMP) + this.getClass().getName() + XXXX, DK\_CM\_MSG\_NOTIMP); где XXX - имя метода, который программа пыталась вызвать.

В исключительную ситуацию DKDatastoreAccessError будет включен текст сообщения FileNET в следующем формате:

```
WAL_имя_функции [IMS_SESSION= sss, ERR_CAT=ccc, ERR_FUN=fff, ERR_NUM=nnn]
```

Для дальнейшего вывода текста собственного сообщения об ошибке FileNET можно воспользоваться командой FileNET msg с тремя числами ccc, fff и nnn (команда находится в каталоге c:\fnsw\client\bin), например, msg ccc,fff,nnn.

ID сообщений EIP с 3401 по 3600 используются только для серверов Panagon Image Services.

---

## Работа с реляционными базами данных

Классы API Enterprise Information Portal поддерживают IBM DB2 Universal Database и другие реляционные базы данных при помощи Java Database Connectivity (JDBC), Open Database Connectivity (ODBC) for C++ или IBM DB2 DataJoiner.

### Соединение с реляционными базами данных

Чтобы создать представление базы данных, создайте объект DKDatastorexx, где xx - DB2 для базы данных DB2, JDBC для Java Database Connectivity и ODBC для Open Database Connectivity. В следующем примере выполняется соединение с базой данных примера DB2 с именем sample:

#### Java

```
dsDB2 = new DKDatastoreDB2();
dsDB2.connect("sample", "db2admin", "password", "");
.....
dsDB2.disconnect();
dsDB2.destroy();
```

#### C++

```
try { DKDatastoreDB2 dsDB2;
 dsDB2.connect("sample", userid, pw);
 . . .
 dsDB2.disconnect();
 }
 catch(DKException &exc) . . .
```

При соединении используйте имя базы данных.

### Строки соединения

При соединении с реляционной базой данных можно задать строку соединения и передать ее как параметр. Если вы задаете несколько строк соединения, разделяйте их точками с запятой (;). Строки соединения могут иметь вид:

#### Строки соединения для DB2, DataJoiner и ODBC:

NATIVECONNECTSTRING=(*собственная строка соединения*)

Задаёт собственную строку соединения для передачи базе данных при соединении. Посмотрите информацию о вашем контент-сервере, чтобы узнать допустимые собственные строки соединения.

SCHEMA=*имя схемы*

Задает имя схемы базы данных, используемое при запуске методов `listEntities`, `listEntityAttrs`, `listPrimaryKeyNames`, `listForeignKeyNames`.

#### **Строки соединения для JDBC:**

`SCHEMA=имя_схемы`

Задает имя схемы базы данных, используемое при запуске методов `listEntities`, `listEntityAttrs`, `listPrimaryKeyNames`, `listForeignKeyNames`.

#### **Строки конфигурации**

Можно задать строку конфигурации и передать ее как параметр для соответствующего метода конфигурации. Если вы задаете несколько строк конфигурации, разделяйте их точками с запятой (;). Строки конфигурации могут иметь вид:

#### **Строки конфигурации для DB2, DataJoiner и ODBC:**

`CC2MIMEURL=(URL)`

Задает файл `cmbscc2mime.ini` как адрес URL. Используйте эту форму строки конфигурации или `CC2MIMEFILE` в зависимости от положения файла.

`CC2MIMEFILE=(имяфайла)`

Задает файл `cmbscc2mime.ini` по имени.

`DSNAME=(имя_контент-сервера)`

Задает имя контент-сервера. Для запросов объединения и других функций объединения Enterprise Information Portal задает его автоматически.

`AUTOCOMMIT=ON | OFF`

Разрешает или запрещает автоматическое принятие. По умолчанию запрещено. Когда контент-сервер используется для запросов объединения и других функций объединения, автоматическое принятие по умолчанию разрешено.

#### **Строки конфигурации для JDBC:**

`CC2MIMEURL=(URL)`

Задает файл `cmbscc2mime.ini` как адрес URL. Используйте эту форму строки конфигурации или `CC2MIMEFILE` в зависимости от положения файла.

Задает файл `cmbscc2mime.ini` по имени.

`JDBC_SERVER_URL=(URL)`

Задает файл `cmbjdbsrvs.ini` в адресе URL. Этот файл содержит список серверов JDBC.

`JDBCSEVERSFIL`=(*имя\_файла*)

Задает файл `cmbjdbsrvs.ini`, содержащий список серверов JDBC в виде имен файлов.

`JDBC``DRIVER`=(*драйвер JDBC*)

Задает драйвер JDBC, который вы хотите использовать. Он устанавливается автоматически, когда вы используете клиентскую программу клиент системного администратора.

`DSNAME`=(*имя контент-сервера*)

Задает имя контент-сервера. Для запросов объединения и других функций объединения Enterprise Information Portal задает его автоматически.

`AUTO``COMMIT`=ON | OFF

Разрешает или запрещает автоматическое принятие. По умолчанию запрещено. Когда контент-сервер используется для запросов объединения и других функций объединения, автоматическое принятие по умолчанию разрешено.

## Получение списка объектов и списка атрибутов объекта

После создания контент-сервера для реляционной базы данных и соединения с ним можно вывести список объектов и список атрибутов объекта. Следующий пример показывает, как получить список серверов и перемещаться по нему:

## Java

```
// ----- После создания склада данных и соединения получаем индексные
// ----- классы
pCol = (DKSequentialCollection)dsDB2.listDataSources();
pIter = pCol.createIterator();
while (pIter.more() == true)
{
 i++;
 pSV = (DKServerDefDB2)pIter.next();
 strServerName = pSV.getName();
 // используем соответствующее имя сервера
}
// ----- соединяемся со складом данных
dsDB2.connect(db, userid, pw, "");
if (!schema.equals(""))
{
 dsDefDB2 = (DKDatastoreDefDB2)dsDB2.datastoreDef();
 dsDefDB2.setSchemaName(schema);
 schema = dsDefDB2.getSchemaName();
 System.out.println(" New Schema Name = [" + schema + "]");
}
// ----- список данных
pCol = (DKSequentialCollection) dsDB2.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
 i++;
 tableDef = (DKTableDefDB2)pIter.next();
 strTable = tableDef.getName();
 // ----- список атрибутов (столбцов таблицы)
 pCol2 = (DKSequentialCollection) dsDB2.listEntityAttrs(strTable);
 pIter2 = pCol2.createIterator();
 j = 0;
 while (pIter2.more() == true)
 {
 j++;
 colDef = (DKColumnDefDB2)pIter2.next();
 // обрабатываем информацию, как требуется
 }
}
// ----- выполняем принятие и отсоединяемся
dsDB2.commit();
dsDB2.disconnect();
dsDB2.destroy();
```

Посмотрите полные примеры в файлах TListCatalogDB2.java, TListCatalogJDBC.java и TListCatalogDJ.java в каталоге CMBROOT\Samples\java\dl.

## C++

```
try {
 DKDatastoreDB2 dsDB2;
 DKString schema;
 DKSequentialCollection *pCol2 = 0;
 dkIterator *pIter = 0;
 dkIterator *pIter2 = 0;
 DKTableDefDB2 *pEnt = 0;
 DKString strServerName;
 DKString strTable;
 DKColumnDefDB2 *pAttr = 0;
 DKDatastoreDefDB2 *dsDefDB2 = 0;
 DKAny a;
 DKAny *pA = 0;
 long i = 0;
 long j = 0;

 // ----- Соединяемся со складом данных и задаем значение для имени схемы
 . . .
 // ----- Создаем определение склада данных и задаем имя схемы
 dsDefDB2 = (DKDatastoreDefDB2 *) dsDB2.datastoreDef();
 if (schema != "")
 {
 dsDefDB2->setSchemaName(schema);
 }

 // ----- Получаем список объектов (таблиц)
 pCol = (DKSequentialCollection*)((dkCollection*)dsDB2.listEntities());
 pIter = pCol->createIterator();
 i = 0;
 // ----- Список аргументов (столбцов) для каждого объекта (таблицы)
 while (pIter->more() == TRUE)
 {
 i++;
 pEnt = (DKTableDefDB2*)((void*)(*pIter->next()));
 strTable = pEnt->getName();
 cout << "table name [" << i << "] - " << strTable << endl;
 cout << " list columns for " << strTable << " table" << endl;
 pCol2 =
 (DKSequentialCollection*)((dkCollection*)dsDB2.listEntityAttrs(strTable));
 pIter2 = pCol2->createIterator();
 j = 0;
 while (pIter2->more() == TRUE)
 {
 j++;
 pA = pIter2->next();
 pAttr = (DKColumnDefDB2*) pA->value();
 cout << " Attr name [" << j << "] - " << pAttr->getName() << endl;
 cout << " datastoreName " << pAttr->datastoreName() << endl;
 cout << " datastoreType " << pAttr->datastoreType() << endl;
 }
 }
 // продолжение следует...
```

### C++ (продолжение)

```
 cout << " attributeOf " << pAttr->getEntityName() << endl;
 cout << " type " << pAttr->getType() << endl;
 cout << " size " << pAttr->getSize() << endl;
 cout << " id " << pAttr->getId() << endl;
 cout << " nullable " << pAttr->isNullable() << endl;
 cout << " precision " << pAttr->getPrecision() << endl;
 cout << " scale " << pAttr->getScale() << endl;
 cout << " string type " << pAttr->getStringType() << endl;
 cout << " primary key " << pAttr->isPrimaryKey() << endl;
 cout << " foreign key " << pAttr->isForeignKey() << endl;
 delete pAttr;
 }
 delete pIter2;
 delete pCol2;
 delete pEnt;
}
delete pIter;
delete pCol;
dsDB2.disconnect();
}
catch(DKException &exc)
 . . .
```

В TListCatalogDB2.cpp, TListCatalogODBC.cpp и TListCatalogDJ.cpp в каталогах CMBROOT\Samples\cpp\db2, odbc и dj приведены полные примеры.

## Выполнение запроса

Для запуска запроса сначала нужно создать строку запроса, а затем выполнить запрос. В следующем примере выполняется запрос и обрабатываются его результаты.



## Java

```
// ----- После создания склада данных и соединения строим
// запрос и параметры и выполняем его
sDB2 = new DKDatastoreDB2();
dkResultSetCursor pCur = null;
DKNVPair parms[] = new DKNVPair[2];
String strMax = "5";
parms[0] = new DKNVPair(DK_CM_PARM_MAX_RESULTS,strMax);
parms[1] = new DKNVPair(DK_CM_PARM_END,null);
// ----- соединяемся со складом данных
dsDB2.connect(database,userId,pw,"");
// ----- создаем строку запроса
String cmd = "";
cmd = "SELECT * FROM EMPLOYEE";

DKDDO p = null;
DKDDO pDDO = null;
dkXDO pXDO = null;
DKPidXDO pidXDO = null;
int i = 0;
int numDataItems = 0;
short k = 0;
short j = 0;
String strDataName;
dkCollection pCol = null;
dkIterator pIter = null;
Object a = null;
dkDataObjectBase pDO = null;
int cnt = 0;

// ----- Выполняем запрос
pCur = dsDB2.execute(cmd,DK_CM_SQL_QL_TYPE,parms);
if (pCur == null)
{
 // обработка, если указатель пуст
}
while (pCur.isValid()) {
 p = pCur.fetchNext();
 if (p != null)
 {
 cnt++;
 i = pCur.getPosition();
 // получаем информацию элемента
 numDataItems = p.dataCount();
 DKPid pid = p.getPidObject();
 System.out.println("pid string " + pid.pidString());
 System.out.println("Number of Data Items " + numDataItems);
 for (j = 1; j <= numDataItems; j++)
 }
}
// продолжение следует...
```

## Java (продолжение)

```
{
 a = p.getData(j);
 strDataName = p.getDataName(j);
 // обрабатываем атрибуты
 if (a instanceof String)
 {
 System.out.println(" Attribute Value " + a);
 }
 // обрабатываем различные типы
 else if (a instanceof dkDataObjectBase)
 {
 pDO = (dkDataObjectBase)a;
 if (pDO.protocol() == DK_PDDO)
 {
 System.out.println(" DKDDO object ");
 pid = ((DKDDO)pDO).getPidObject();
 }
 else if (pDO.protocol() == DK_XDO)
 {
 // объект dkXDO
 pXDO = (dkXDO)pDO;
 pidXDO = pXDO.getPidObject();
 }
 }
 // обрабатываем различные типы
 {
 }
}
// завершив работу, удаляем указатель, выполняем принятие и отсоединяемся
pCur.destroy(); // удаляем указатель
dsDB2.commit();
dsDB2.disconnect();
dsDB2.destroy();
```

Посмотрите полные примеры в файлах TExecuteDB2.java, TExecuteJDBC.java и TExecuteDJ.java в каталоге CMBROOT\Samples\java\d1.

**C++**

```
try {
 DKDatastoreDB2 dsDB2;
 dkResultSetCursor* pCur = 0;
 DKNVPair par[2];
 DKAny anyValue;
 DKString strMax = "5";
 anyValue = strMax;
 par[0].set(DK_CM_PARM_MAX_RESULTS, anyValue);
 par[1].setName(DK_CM_PARM_END);
 // ---- Создаем склад данных и соединяемся
 . . .
 // ---- Создаем строку запроса с оператором select
 DKString qstrng = "SELECT * FROM EMPLOYEE";
 // ----- Выполняем запрос
 pCur = dsDB2.execute(qstrng, DK_CM_SQL_QL_TYPE, par);
 // ---- Declarations
 DKDDO *p = 0;
 dkDataObjectBase *pDOBase = 0;
 DKDDO *pDDO = 0;
 dkXDO *pXDO = 0;
 DKAny a;
 ushort j = 0;
 ushort k = 0;
 ushort val = 0;
 ushort cnt = 1;
 DKString strData = "";
 DKString strDataName = "";
 dkCollection* pdCol = 0;
 dkIterator* pdIter = 0;
 ushort numDataItems = 0;
 DKString strPid;
 DKPid* pid = 0;
 short sVal = 0;
 long lVal = 0;
 while (pCur->isValid())
 {
 p = pCur->fetchNext();
 if (p != 0)
 {
 cout << "======" << "Item " << cnt << " <======" << endl;
 numDataItems = p->dataCount();
 pid = (DKPid*)p->getPidObject();
 strPid = pid->pidString();
 cout << "pid string " << strPid << endl;
 k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
 if (k > 0)
 {
 a = p->getProperty(k);
 }
 }
 }
 // продолжение следует...
```

## C++ (продолжение)

```
 val = a;
 switch (val)
 {
 case DK_CM_DOCUMENT:
 {
 cout << "Item is document " << endl;
 break;
 }
 }
 }
 cout << "Number of Data Items " << numDataItems << endl;
 for (j = 1; j <= numDataItems; j++)
 {
 a = p->getData(j);
 strDataName = p->getDataName(j);
 switch (a.typeCode())
 {
 case DKAny::tc_string :
 {
 strData = a;
 cout << "attribute name : " << strDataName << " value : "
 << strData << endl;
 break;
 }
 // ---- Обрабатываем подобным образом каждый тип
 . . .
 }
 }
 // ----- Удаляем указатель и отсоединяемся
 if (pCur != 0)
 delete pCur;
 dsDB2.disconnect();
}
catch(DKException &exc)
 . . .
```

В TExecuteDB2.cpp, TExecuteODBC.cpp и TExecuteDJ.cpp в каталогах CMBROOT\Samples\cpp приведены полные примеры.

## Создание пользовательских контент-серверов

Можно создать свои собственные определения для пользовательских контент-серверов (не включенных в настоящее время в Enterprise Information Portal). Если вы интегрируете пользовательский сервер в EIP, надо задать свои классы Java или C++ для поддержки определения.

## Разработка пользовательских контент-серверов

Объектно-ориентированная структура API разработана со следующими целями:

- Добавление в структуру дополнительных систем хранения данных или контент-серверов.
- Отображение любых сложных типов данных контент-сервера.
- Реализация единой объектной модели для доступа к данным контент-сервера.
- Реализация гибких средств для использования сочетания разных типов механизмов поиска, таких как Механизм текстового поиска, поиск изображений (QVIC), и т.п.
- Реализация архитектуры клиент-сервер для пользователей прикладных программ Java.

Информацию о конкретных объектно-ориентированных API смотрите в электронном справочнике API.

Если вы интегрируете пользовательский контент-сервер в Enterprise Information Portal, надо:

- Импортировать пакет `com.ibm.mm.sdk.common`.
- **Только для Java:** Скомпоновать его с файлом `cmbscm81.jar` для обращения к общей структуре.
- **Только для C++:** скомпоновать его с файлами неотладочной версии `cmbscm817.dll` и отладочной версии `cmbscm8167.dll` для доступа к общей структуре.

### **Инфраструктура базы данных Enterprise Information Portal**

Классы `dkDatastore` служат в качестве первичного интерфейса между Enterprise Information Portal и контент-серверами. У каждого контент-сервера есть отдельный класс, который реализует класс `dkDatastore`, чтобы обеспечить реализацию информации для конкретного контент-сервера. Каждый тип контент-сервера представлен классом под именем `DKDatastorexx`, где `xx` идентифицирует имя или тип конкретного контент-сервера. В Табл. 7 на стр. 44 перечислены текущие контент-серверы, поддерживаемые Enterprise Information Portal.

При создании определения сервера надо задать класс `DKDatastore`, который вы создаете для контент-сервера на клиенте управления системой.

### **Общие классы в Enterprise Information Portal**

#### **dkDDO**

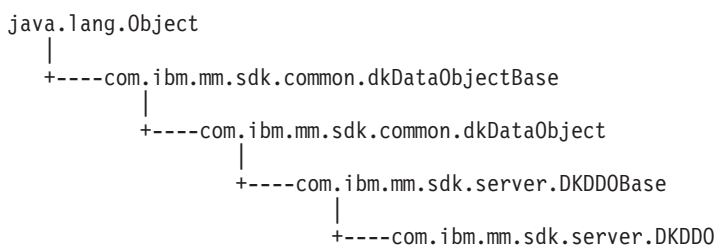
Класс `dkDDO` обеспечивает представление и протокол для определения объекта и доступа к его данным независимо от типа этого объекта. Протокол DDO реализован как набор функций для определения, добавления и доступа ко всем элементам данных объекта. Этот протокол можно использовать для динамического создания объекта и получения его с контент-сервера независимо от типа этого контент-сервера.

Реализуя контент-сервер, можно использовать информацию отображения схемы, зарегистрированную в классе контент-сервера. Схема отображает каждый отдельный элемент постоянных данных на его базовую репрезентацию на контент-сервере.

У объекта DDO есть набор атрибутов; с каждым атрибутом связаны тип, значение и свойства. Сам DDO может иметь свойства, принадлежащие DDO в целом. Например, можно отобразить каждый класс на элемент на контент-сервере Content Manager, или документ в OnDemand.

### Java

На следующей диаграмме показана иерархия для класса dkDDO:



### dkXDO

Класс dkXDO представляет сложные пользовательские типы или большие объекты (LOB), которые могут существовать самостоятельно или как часть объекта DDO. Таким образом, он имеет постоянный идентификатор (PID) и функции создания, получения, изменения и удаления.

Класс dkXDO расширяет dkXDOBase, класс типа public interface, определяя функции доступа, создания, получения, изменения и удаления для независимого контент-сервера. Эти функции позволяют программе сохранять на контент-сервере и получать с него данные объекта, обходясь без связанного объекта класса DDO и отдельного XDO.

Надо задать PID для отдельного XDO, чтобы найти его положение на контент-сервере. Если вы используете XDO с объектом DDO, PID задается автоматически. Например, можно отобразить каждый класс на элемент для контент-серверов Content Manager и на примечания для контент-серверов OnDemand.

### Java

Вот иерархия классов для класса dkXDO:

```
java.lang.Object
|
+----com.ibm.mm.sdk.server.dkDataObjectBase
|
+----com.ibm.mm.sdk.server.dkXDOBase
|
+----com.ibm.mm.sdk.server.dkXDO
```

### dkCollection

Класс dkCollection - собрание объектов. dkCollection не может оценивать запрос. Собрание может иметь имя (по умолчанию имя - пустая строка). Например, DKParts - это подкласс собрания DKSequentialCollection, которое в свою очередь есть подкласс собрания dkCollection.

### DKResults

DKResults - это подкласс собрания dkQueryableCollection и поэтому поддерживает сортировку и двунаправленные итераторы, а также допускает запросы. Элементарные объекты класса DKResults - это экземпляры класса dkDDO, которые представляют результаты запроса. Итератор, создаваемый этим классом - dkSequentialIterator.

### Java

Вот иерархия классов для класса DKResults:

```
java.lang.Object
|
+----com.ibm.mm.sdk.server.DKSequentialCollection
|
+----com.ibm.mm.sdk.server.dkQueryableCollection
|
+----com.ibm.mm.sdk.server.DKResults
```

### dkQuery

dkQuery - это интерфейс для объекта запроса, связанного с определенным контент-сервером. Объекты, которые реализуют этот интерфейс, создаются классами контент-сервера. Результаты запроса обычно представлены объектом DKResults. Примерами конкретной реализации интерфейса dkQuery служат DKParametricQuery, DKTextQuery и DKImageQuery, создаваемые связанными с ними контент-серверами.

### dkCQExpr

Класс dkCQExpr представляет составное или комбинированное выражение запроса. Он может содержать дерево выражений запроса

dkQExpr, которое может содержать комбинацию параметрического запроса, текстового запроса и запроса изображений. Если вы хотите, чтобы каждый контент-сервер допускал поиск объединения, контент-сервер должен быть в состоянии обрабатывать этот объект dkCQExpr.

### **dkSchemaMapping**

dkSchemaMapping - интерфейс, в котором определено ассоциативное отображение между объектом объединения и исходным объектом на контент-сервере. Контент-сервер должен понимать этот класс отображения, чтобы отображать объекты и атрибуты объединения на исходные объекты и атрибуты и обратно для запроса и для возвращения результатов.

### **dkDatastore и связанные классы**

Надо реализовать один конкретный класс для каждого из следующих классов или интерфейсов вашего контент-сервера. Например, на сервере OnDemand конкретный класс, реализующий интерфейс dkDatastore - это DKDatastoreOD.

#### **dkDatastore**

dkDatastore представляет соединение с контент-сервером и управляет этим соединением, его транзакциями и командами. Он поддерживает функцию evaluate, поэтому его можно считать подклассом менеджера запросов.

Основные методы в интерфейсе dkDatastore:

#### **connect()**

Соединяет с контент-сервером.

#### **disconnect()**

Отсоединяет от контент-сервера.

#### **evaluate(), execute(), executeWithCallback()**

Запрашивает контент-сервер.

#### **commit(), rollback()**

Выполняет транзакции на контент-сервере.

**Ограничение:** Некоторые контент-серверы не поддерживают эти функции.

#### **registerServices(), unregisterServices()**

Регистрирует механизмы поиска.

#### **changePassword(userid, oldPasswd, newPasswd)**

Заменяет пароль регистрации для текущего зарегистрированного ID пользователя на контент-сервере.

#### **listDataSources()**

Возвращает собрание объектов ID пользователей



контент-сервера, которые будут использованы для регистрации. Для использования этой функции не обязательно соединяться с контент-сервером.

**listDataSourceNames()**

Возвращает массив имен контент-серверов.

**getExtension(String)**

Получает с контент-сервера объект `dkExtension`. Если заданное расширение еще не существует, но поддерживается контент-сервером, возвращается вновь созданный объект, иначе возвращается пустое значение.

**addExtension(String, dkExtension)**

Добавляет на этот контент-сервер новый объект расширения (XDO).

**createDDO(String,int)**

Создает объект данных на основе заданного типа объекта и флага. У возвращаемого `CreateDDO` нового объекта `DKDDO` все свойства и атрибуты заданы. Вызывающая программа должна задать значения атрибутов для этого объекта данных.

Методы работы с объектом данных в интерфейсе `dkDatastore`:

**addObject(dkDataObject)**

Добавляет на контент-сервер новый документ или папку.

**retrieveObject(dkDataObject)**

Получает документ или папку с контент-сервера.

**deleteObject(dkDataObject)**

Удаляет документ или папку с контент-сервера.

**updateObject(dkDataObject)**

Изменяет документ или папку на контент-сервере.

**moveObject(dkDataObject, String)**

Перемещает папку или документ из одного объекта в другой.

Методы, связанные с отображением схемы в интерфейсе `dkDatastore`:

**registerMapping(DKNVPair)**

Регистрирует информацию отображения на этом контент-сервере.

**unRegisterMapping(String)**

Удаляет информацию отображения с этого контент-сервера.

**listMappingNames()**

Возвращает с этого контент-сервера массив имен отображений.

### **getMapping(String)**

Возвращает объект dkSchemaMapping.

### **dkDatastoreDef**

Интерфейс dkDatastoreDef определяет функции доступа к информации контент-сервера и создания, вывода списка и удаления его объектов. Он поддерживает собрание объектов dkEntityDef.

В Табл. 22 приведены примеры конкретных классов для интерфейса dkDatastoreDef.

*Таблица 22. Конкретные классы для dkDatastoreDef*

| Тип сервера                   | Имя класса        |
|-------------------------------|-------------------|
| Content Manager               | DKDatastoreDefICM |
| OnDemand                      | DKDatastoreDefOD  |
| Content Manager for AS/400    | DKDatastoreDefV4  |
| ImagePlus for OS/390          | DKDatastoreDefIP  |
| Domino.Doc                    | DKDatastoreDefDD  |
| Extended Search               | DKDatastoreDefDES |
| IBM DB2 Universal Database    | DKTableDefDB2     |
| JDBC                          | DKTableDefJDBC    |
| ODBC                          | DKTableDefODBC    |
| Ранние версии Content Manager | DKDatastoreDefDL  |

Основные методы в интерфейсе dkDatastoreDef:

#### **listEntities()**

Получает список объектов.

#### **listEntityAttrs()**

Получает список атрибутов объекта.

#### **addEntity()**

Добавляет объект.

#### **getEntity(name)**

Получает объект.

Кроме того, каждый конкретный класс может иметь собственные функции для определенного контент-сервера с именами, известными этому контент-серверу. Например, класс DKDatastoreDefDL содержит следующие особые функции:

- listIndexClassNames()
- listIndexClasses()

Класс DKDatastoreDefOD содержит такие особые функции:

- listAppGrps()
- listAppGrpNames()

### **dkEntityDef**

Класс dkEntityDef определяет функции для:

- Дают доступ к информации объекта.
- Создают и удаляют атрибуты.
- Создают и удаляют объект.

В классе dkEntityDef все функции, которые связаны с подобъектами, генерируют сообщение об ошибке DKUsageError, указывающее, что контент-сервер по умолчанию не поддерживает подобъекты. Но если контент-сервер поддерживает этот вид многоуровневого объекта - например, как Domino.Doc, - подкласс для этого контент-сервера должен реализовывать необходимые функции для переопределения исключительных ситуаций.

В Табл. 23 приведены примеры конкретных классов для класса dkEntityDef.

*Таблица 23. Конкретные классы для dkEntityDef*

| Тип сервера                   | Имя класса        |
|-------------------------------|-------------------|
| Content Manager               | DKItemTypeDefDL   |
| OnDemand                      | DKAppGrpDefOD     |
| Content Manager for AS/400    | DKIndexClassDefV4 |
| ImagePlus for OS/390          | DKEntityDefIP     |
| Domino.Doc                    | DKCabinetDefDD    |
| Extended Search               | DKDatabaseDefDES  |
| DB2 Universal Database        | DKTableDefDB2     |
| JDBC                          | DKTableDefJDBC    |
| ODBC                          | DKTableDefODBC    |
| Ранние версии Content Manager | DKIndexClassDefDL |

Основные функции в классе dkEntityDef:

#### **listAttrs()**

Получает список атрибутов объекта.

#### **getAttr(String attrName)**

Получает заданный атрибут объекта.

**addAttr(DKAttrDef)**

Добавляет к объекту атрибут.

**getName()**

Получает имя объекта.

**setName(String)**

Задаёт имя объекта.

**hasSubEntities()**

Выясняет, содержит ли объект подобъекты.

**getSubEntity(String)**

Получает подобъект.

**addSubEntity(dkEntityDef)**

Добавляет в объект подобъект.

**listSubEntities()**

Получает список подобъектов объекта.

**removeAttr(String)**

Удаляет подобъект из объекта.

**add()** Добавляет объект на контент-сервер.**update()**

Изменяет объект на контент-сервере.

**retrieve()**

Получает значения объекта с контент-сервера.

**del()** Удаляет объект с контент-сервера.**dkAttrDef**

Класс dkAttrDef определяет функции доступа к информации атрибута и создания и удаления атрибутов. В Табл. 24 приведены примеры конкретных классов для класса dkAttrDef.

Таблица 24. Конкретные классы для dkAttrDef

| Тип сервера                | Имя класса      |
|----------------------------|-----------------|
| Content Manager            | DKAttrDefDICM   |
| OnDemand                   | DKFieldDefOD    |
| Content Manager for AS/400 | DKAttrDefV4     |
| ImagePlus for OS/390       | DKAttrDefIP     |
| Domino.Doc                 | DKAttrDefDD     |
| Extended Search            | DKFieldDefDES   |
| DB2 Universal Database     | DKColumnDefDB2  |
| JDBC                       | DKColumnDefJDBC |

Таблица 24. Конкретные классы для *dkAttrDef* (продолжение)

| Тип сервера                   | Имя класса      |
|-------------------------------|-----------------|
| ODBC                          | DKColumnDefODBC |
| Ранние версии Content Manager | DKAttrDefDL     |

Основные методы в классе *dkAttrDef*:

**listAttrs()**

Получает список атрибутов.

**getAttr(String attrName)**

Получает заданный атрибут.

**getName()**

Получает имя атрибута.

**getDescription()**

Получает описание атрибута.

**add()** Добавляет объект на контент-сервер.

**dkServerDef**

Класс *dkServerDef* содержит информацию определения сервера для каждого контент-сервера. В Табл. 25 приведены примеры конкретных классов для класса *dkServerDef*.

Таблица 25. Конкретные классы для *dkServerDef*

| Тип сервера                   | Имя класса      |
|-------------------------------|-----------------|
| Content Manager               | DKServerDefICM  |
| OnDemand                      | DKServerDefOD   |
| Content Manager for AS/400    | DKServerDefV4   |
| Domino.Doc                    | DKServerDefDD   |
| Extended Search               | DKServerDefDES  |
| DB2 Universal Database        | DKServerDefDB2  |
| JDBC                          | DKServerDefJDBC |
| ODBC                          | DKServerDefODBC |
| Ранние версии Content Manager | DKServerDefDL   |

Основные функции в классе *dkServerDef*:

**setDatastore(dkDatastore ds)**

Задаёт ссылку на объект контент-сервера.

**getDatastore()**

Получает ссылку на объект контент-сервера.

**getName()**

Получает имя контент-сервера.

**setName(String name)**

Задает имя контент-сервера.

**datastoreType()**

Получает тип контент-сервера.

**dkResultSetCursor**

dkResultSetCursor - это указатель контент-сервера в наборе результатов запроса, который служит для управления виртуальным собранием объектов DDO. Это собрание - набор результатов запроса. Ни один элемент собрания не создается, пока не будет получен контент-сервером.

Основные функции в классе dkResultSetCursor:

**isScrollable()**

Возвращает TRUE, если указатель допускает прокрутку вперед и назад.

**isUpdatable()**

Возвращает TRUE, если указатель допускает изменение.

**isValid()**

Возвращает TRUE, если указатель допустим.

**isBegin()**

Возвращает TRUE, если указатель находится в начале набора результатов

**isEnd()**

Возвращает TRUE, если указатель находится в конце набора результатов.

**isInBetween()**

Возвращает TRUE, если указатель находится между элементами данных в наборе результатов.

**getPosition()**

Получает текущую позицию указателя.

**setPosition(int position, Object value)**

Устанавливает указатель в заданную позицию.

**setToNext()**

Устанавливает указатель на следующий элемент в наборе результата.

**fetchObject()**

Получает текущий элемент из набора результатов и возвращает его как объект DDO.

**fetchNext()**

Получает следующий элемент из набора результатов и возвращает его как объект DDO.

**findObject(int position, String predicate)**

Находит объект данных, удовлетворяющий заданному предикату, перемещает указатель в эту позицию и получает объект.

**addObject(DKDDO ddo)**

Добавляет на контент-сервер новый элемент типа, представленного заданным объектом DDO.

**deleteObject()**

Удаляет текущий элемент с контент-сервера.

**updateObject(DKDDO ddo)**

Изменяет текущий элемент в текущей позиции на контент-сервере, используя заданный DDO.

**newObject()**

Создает элемент того же типа и возвращает его как объект DDO.

**open()** Открывает указатель и, если необходимо, выполняет запрос для создания набора результатов.

**close()** Закрывает указатель и набор результатов.

**isOpen()**

Возвращает TRUE, если указатель открыт.

**destroy()**

Удаляет указатель; это позволяет освободить память до того, как указатель будет удален мусорщиком.

**datastoreName()**

Получает имя контент-сервера, которому принадлежит указатель.

**datastoreType()**

Получает тип контент-сервера, которому принадлежит указатель.

**handle(int type)**

Получает хэндл набора результатов, который связан с указателем набора результатов, для типа.

**Требование:** Чтобы использовать функции `addObject`, `deleteObject` и `updateObject`, надо задать для опции контент-сервера `DK_DL_OPT_ACCESS_MODE` значение `DK_READWRITE`.

## dkBlob

`dkBlob` - абстрактный класс, который объявляет общедоступный интерфейс для основных типов данных двоичного большого объекта (BLOB). Конкретные классы, происходящие от класса `dkBlob`, совместно используют этот общедоступный интерфейс, который допускает всевозможную обработку собраний двоичных больших объектов, происходящих с разнородных контент-серверов. Существуют также классы `dkClob` и `dkDBClob`, у которых могут быть конкретные классы.

В Табл. 26 приведены примеры конкретных классов для класса `dkBlob`.

Таблица 26. Конкретные классы для `dkBlob`

| Тип сервера                   | Имя класса             |
|-------------------------------|------------------------|
| Content Manager               | DKLobICM               |
| OnDemand                      | DKBlobOD               |
| Content Manager for AS/400    | DKBlobV4               |
| ImagePlus for OS/390          | DKBlobIP               |
| Domino.Doc                    | DKBlobDD               |
| Extended Search               | DKBlobDES              |
| DB2 Universal Database        | DBBlobDB2, DKBlobDB2   |
| JDBC                          | DKBlobJDBC, DKBlobJDBC |
| ODBC                          | DKBlobODBC, DKBlobODBC |
| Ранние версии Content Manager | DKBlobDL               |

Основные методы в классе `dkBlob`:

### **getContent()**

Возвращает массив байтов с данными заданного двоичного большого объекта.

### **getContentToClientFile(String afileName, int fileOption)**

Копирует данные из двоичного большого объекта в заданный файл.

### **setContent(byte[] aByteArr)**

Задаёт данные большого объекта равными содержимому байтового массива.

### **setContentFromClientFile(String afileName)**

Заменяет данные большого объекта содержимым файла *afileName*.



**add(String afileName)**

Добавляет содержимое заданного файла на контент-сервер.

**retrieve(String afileName)**

Получает содержимое контент-сервера в заданный файл.

**update(String afileName)**

Изменяет объект и контент-сервер при помощи содержимого заданного файла

**del(boolean flush)**

Удаляет данные объекта с контент-сервера, если *flush* - TRUE; иначе сохраняется текущее содержимое.

**concatReplace(dkBlob aBlob), concatReplace(byte[] aByteArr)**

Конкатенирует этот объект с другим объектом dkBlob или байтовым массивом.

**length()**

Возвращает длину содержимого заданного большого объекта.

**indexOf(String aString, int startPos), indexOf(dkBlob aBlob, int startPos)**

Начиная поиск с заданного смещения относительно начала, возвращает смещение в байтах первого вхождения аргумента поиска в этом объекте.

**subString(int startPos, int length)**

Возвращает строчный объект, который содержит подстроку данных этого большого объекта.

**remove(int startPos, int aLength)**

Начиная с позиции *startPos*, удаляет отрезок данных длины *aLength* (в байтах) из этого большого объекта.

**insert(String aString, int startPos), insert(dkBlob aBlob, int startPos)**

Вставляет данные аргумента, следующие за позицией *startPos* в данных этого большого объекта

**open(String afileName)**

Выгружает содержимое объекта в файл *afileName* и затем запускает обработчик файлов по умолчанию.

**setClassOpenHandler(String aHandler, boolean newSynchronousFlag)**

Задаёт имя исполняемой программы - обработчика файла для всего класса. Эта функция также указывает, синхронно или асинхронно выполнять обработчик для файлового объекта.

**setInstanceOpenHandler(String aHandler, boolean newSynchronousFlag)**

Задаёт имя исполняемой программы - обработчик файла и указывает, синхронно или асинхронно выполнять его для этого объекта.

**getOpenHandler()**

Получает имя исполняемой программы обработчика файла для всего класса.

**isOpenSynchronous()**

Возвращает текущее значение синхронности для обработчика файла.

**dkClob**

dkClob - абстрактный класс, который объявляет публичный интерфейс для хранения типов данных символьного большого объекта (CLOB), например, документов.

В Табл. 27 приведены примеры конкретных классов для класса dkClob.

Таблица 27. Конкретные классы для dkClob

| Тип сервера            | Имя класса |
|------------------------|------------|
| DB2 Universal Database | DKClobDB2  |
| ODBC                   | DKClobODBC |

Основные функции в классе dkClob:

**open()** Open() - элемент, наследуемый от dkXDOBase. Open() реализуется или переопределяется конкретными подклассами класса dkClob.

**Элементы dkXDO: dkXDO& add(), dkXDO retrieve(), dkXDO update(), dkXDO del()**

Наследуются как защищенные элементы из dkXDO. При необходимости эти защищенные элементы реализуются или переопределяются конкретными подклассами класса dkClob.

Ниже перечислены элементы, определенные dkClob:

**add(String afileName)**

Добавляет содержимое заданного файла на контент-сервер.

**retrieve(String afileName)**

Получает содержимое контент-сервера в заданный файл.

**update(String afileName)**

Изменяет объект и контент-сервер при помощи содержимого заданного файла.

**del(DKBoolean flush)**

Удаляет данные объекта с контент-сервера, если *flush* - TRUE; иначе сохраняется текущее содержимое.

**getContentToClientFile(String afileName, int fileOption)**

Копирует данные из символьного большого объекта в заданный файл.

**setContentFromClientFile(String afileName)**

Заменяет данные большого объекта содержимым файла *afileName*.

**indexOf(String& aString, long startPos=1), indexOf(dkClob& adkClob, long startPos=1)**

Начиная поиск с заданного смещения относительно начала, возвращает смещение в байтах первого вхождения аргумента поиска в этом объекте.

**subString(long startPos, long length)**

Возвращает строчный объект, который содержит подстроку данных этого большого объекта.

**remove(long startPos, long aLength)**

Начиная с позиции *startPos*, удаляет отрезок данных длины *aLength* (в байтах) из этого большого объекта.

**insert(DKString aString, long startPos), insert(dkClob& adkClob, long startPos)**

Вставляет данные аргумента за позицией *startPos* в данные этого символьного большого объекта

**open(String afileName)**

Выгружает содержимое объекта в файл *afileName* и затем запускает обработчик файлов по умолчанию.

**setInstanceOpenHandler(String ahandler, DKBoolean newSynchronousFlag)**

Задаёт имя исполняемой программы - обработчик файла и указывает, синхронно или асинхронно выполнять его для этого объекта.

**setClassOpenHandler(String ahandler, DKBoolean newSynchronousFlag)**

Задаёт имя исполняемой программы - обработчика файла для всего класса. Эта функция также указывает, синхронно или асинхронно выполнять обработчик для файлового объекта.

**getOpenHandler()**

Получает имя исполняемой программы обработчика файла для всего класса.

**isOpenSynchronous()**

Возвращает текущее значение синхронности для обработчика файла.

### **dkAnnotationExt**

dkAnnotationExt - это класс интерфейса для всех объектов комментариев. Если ваш контент-сервер поддерживает данные комментариев, надо реализовать этот интерфейс. Этот объект комментариев является расширением вашего класса DKBlobxx, где объект dkBlob - представление двоичных данных комментариев и собрания DKParts.

### **dkDatastoreExt**

Класс dkDatastoreExt определяет стандартные классы расширения контент-сервера.

В Табл. 28 приведены примеры конкретных классов для класса dkDatastoreExt.

*Таблица 28. Конкретные классы для dkDatastoreExt*

| Тип сервера                   | Имя класса         |
|-------------------------------|--------------------|
| Content Manager               | DKDatastoreExtICM  |
| OnDemand                      | DKDatastoreExtOD   |
| Content Manager for AS/400    | DKDatastoreExtV4   |
| ImagePlus for OS/390          | DKDatastoreExtIP   |
| Domino.Doc                    | DKDatastoreExtDD   |
| Extended Search               | DKDatastoreExtDES  |
| DB2 Universal Database        | DKDatastoreExtDB2  |
| JDBC                          | DKDatastoreExtJDBC |
| Ранние версии Content Manager | DKDatastoreExtDL   |

Основные функции в классе dkDatastoreExt:

#### **getDatastore()**

Получает ссылку на объект - владелец контент-сервера.

#### **setDatastore(dkDatastore ds)**

Задаёт ссылку на объект - владелец контент-сервера.

#### **isSupported(String functionName)**

Выясняет, поддерживается ли заданное имя функции этим расширением.

#### **listFunctions()**

Получает список всех имен функций, поддерживаемых расширением.

#### **addToFolder(dkDataObject folder, dkDataObject member)**

Добавляет элемент в эту папку и на контент-сервер.

**removeFromFolder(dkDataObject folder, dkDataObject member)**

Удаляет элемент из этой папки и с контент-сервера.

**checkOut(dkDataObject item)**

Резервирует элемент документа или папки на контент-сервере.

Пока элемент зарезервирован, у вас есть монопольные привилегии на его изменение, а другим пользователям он доступен только для чтения.

**checkIn(dkDataObject item)**

Активирует элемент документа или папки, ранее зарезервированный на контент-сервере. Активировав этой функцией файл, вы освобождаете все привилегии записи.

**getCommonPrivilege()**

Получает общую привилегию определенного контент-сервера.

**isCheckedOut(dkDataObject item)**

Выясняет, был ли зарезервирован элемент документа или папки на контент-сервере.

**checkedOutUserId(dkDataObject item)**

Получает ID пользователя, который зарезервировал элемент на контент-сервере.

**unlockCheckedOut(dkDataObject item)**

Разблокирует элемент на контент-сервере.

**changePassword (String userId, String oldPwd, String newPwd)**

Изменяет пароль на контент-сервере для заданного ID пользователя.

**moveObject (dkDataObject dataObj, String entityName)**

Перемещает объект *entityName* из одного объекта в другой.

**retrieveFormOverlay(String id)**

Получает объект оверлея формы.

**DKPidXDO**

Класс DKPidXDO представляет постоянную идентификацию данных двоичного большого объекта на контент-сервере.

В Табл. 29 приведены примеры конкретных классов для класса DKPidXDO.

*Таблица 29. Конкретные классы для DKPidXDO*

| Тип сервера                   | Имя класса |
|-------------------------------|------------|
| Ранние версии Content Manager | DKPidXDODL |
| OnDemand                      | DKPidXDOOD |
| Content Manager for AS/400    | DKPidXDOV4 |

Таблица 29. Конкретные классы для DKPidXDO (продолжение)

| Тип сервера            | Имя класса   |
|------------------------|--------------|
| ImagePlus for OS/390   | DKPidXDOIP   |
| Domino.Doc             | DKPidXDODD   |
| Extended Search        | DKPidXDODES  |
| DB2 Universal Database | DKPidXDODB2  |
| JDBC                   | DKPidXDOJDBC |
| ODBC                   | DKPidXDOODBC |

### dkUserManagement

Класс dkUserManagement представляет и обрабатывает все пользовательские функции управления контент-сервера.

В Табл. 30 приведены примеры конкретных классов для класса dkUserManagement.

Таблица 30. Конкретные классы для dkUserManagement

| Тип сервера                   | Имя класса    |
|-------------------------------|---------------|
| Content Manager               | DKUserMgmtICM |
| Content Manager for AS/400    | DKUserMgmtV4  |
| ImagePlus for OS/390          | DKUserMgmtIP  |
| Ранние версии Content Manager | DKUserMgmtDL  |

### DKConstant

Все общие константы определяются в классе DKConstant. У каждого контент-сервера есть свой класс DKConstantxx для определений своих констант.

**Рекомендация:** Все контент-серверы по возможности должны пользоваться общими сообщениями.

### DKMessageId

Все общие ID сообщений определяются в этом классе. У каждого контент-сервера есть свой класс DKMessageIdxx для определений своих ID сообщений.

**Рекомендация:** По возможности все контент-серверы должны использовать общие сообщения.

Следующие файлы свойств содержат общие предупреждения и сообщения об ошибке:

Для Java:

- DKMessage\_en.properties

- DKMessage\_es.properties

для C++:

- DKMessage\_en\_US.properties
- DKMessage\_es\_ES.properties

У каждого контент-сервера есть свои файлы

DKMessagexx\_yy\_zz.properties для своих предупреждений и сообщений об ошибках.

## Использование класса FeServerDefBase (только для Java)

Класс FeServerDefBase - абстрактный класс, который можно расширять для создания пользовательского определения сервера. У класса Java, который расширяет этот базовый класс, должен быть конструктор, который принимает и передает надклассу следующие параметры:

### **String connectString**

Строка соединения с сервером.

### **String[] serverList**

Список определенных серверов.

### **String[] associatedServerList**

Список серверов, связанных с данным сервером (пустой, если их нет).

### **String[] serverTypes**

Список определенных ID типов серверов.

### **String[] serverTypeDescriptions**

Список описаний определенных типов серверов.

Когда вы создаете класс Java, который расширяет класс FeServerDefBase, надо определить, как обрабатывать данные для диалога с новым сервером. Можно использовать тот же самый класс или класс отдельной модели. Если пользовательский контент-сервер требует не только полей строки соединения, надо использовать базу данных Enterprise Information Portal и API Java как модель для того, чтобы дополнительные функции работали правильно.

Когда в программе управления Enterprise Information Portal выбраны контент-серверы, меню Новый будет содержать список типов серверов, хранящийся в таблице FASERVERTYPES в базе данных Enterprise Information Portal. Эта таблица содержит имя класса Java, который реализуется при выборе пункта меню.

Если вы поддерживаете проверку паролей, надо поместить ваш класс Java в тот же каталог, что и файл .jar управления Enterprise Information Portal. Вы можете динамически реализовать этот класс Java и вызвать метод verify с введенным

пользователем паролем в качестве параметра. Метод `verify` вернет пустое значение для правильного пароля и массив строк с информацией для неправильного пароля.



---

## Построение программ рабочего потока EIP

При помощи классов EIP и API вы можете создавать или дополнять свои программы, поддерживающие рабочие потоки EIP. Обычно вы выполняете объединенный поиск и начинаете рабочий поток с результатов поиска (элемента содержимого или папки с несколькими элементами содержимого). Эти API используются для доступа к рабочему списку и вывода его содержания. После завершения каждого действия рабочий поток переходит к следующему действию в рабочем потоке.

---

### Соединение со службами рабочего потока

Чтобы использовать рабочий поток Enterprise Information Portal в своих программах, сначала создайте экземпляр DKWorkflowServicesFed. Потом установите соединение с ним. В следующем примере показан запуск служб рабочего потока:

#### Java

```
// ----- Создаем строки для имени
//службы, ID пользователя и пароля
String wfsrv = "icmnlsdb";
String userid = "icmadmin";
String pw = "password";
// ----- Создаем склад данных объединения
DKDatastoreFed dsFed = new DKDatastoreFed();
dsFed.connect(wfsrv, userid, pw,"");
//----- Создаем службу рабочего потока
DKWorkflowServiceFed svWF =new DKWorkflowServiceFed ();
// ----- Задаем склад данных в рабочем потоке
svWF.setDatastore(dsFed);
// ----- Соединяемся со службой
svWF.connect (wfsrv, userid, pw,"");
```

### C++

```
// ----- Создаем строку для названия службы, ID пользователя
// ----- и пароль
DKString wfsrv = "icmnlsdb";
DKString userid = "icmadmin";
DKString pw = "password";
// ----- Создаем склад данных объединения
DKDatastoreFed* dsFed = new DKDatastoreFed();
dsFed->connect(wfsrv, userid, pw, "");
//----- Создаем службу рабочего потока
DKWorkFlowServiceFed* svWF =new DKWorkFlowServiceFed ();
// ----- Задаем склад данных в рабочем потоке
svWF->setDatastore(dsFed);
// ----- Соединяемся со службой
svWF->connect (wfsrv, userid, pw, "");
```

Закончив использование службы рабочего потока, необходимо отсоединиться, вызвав функции `disconnect()` и `delete()`.

### Java

```
svWF.disconnect();
dsFed.disconnect();
svWF.destroy();
dsFed.destroy();
```

### C++

```
svWF->disconnect();
dsFed->disconnect();
delete svWF;
delete dsFed;
```

## Запуск рабочего потока

После создания рабочего потока запустите его. Для запуска рабочего потока выполните следующие действия:

1. Создайте объект `DKWorkFlowFed` и задайте имя рабочего потока
2. Создайте экземпляр рабочего потока при помощи подходящего шаблона, то есть определения рабочего потока из построителя рабочих потоков Enterprise Information Portal.
3. Задайте в контейнере постоянный идентификатор (PID) и приоритет.
4. Запустите рабочий поток.

В следующем примере выполняются эти шаги для запуска рабочего потока:

#### Java

```
// ----- Создаем объект DKWorkflowFed и задаем имя
DKWorkflowFed WF = new DKWorkflowFed(svWF);
WF.setName("wf1");
// ----- Создаем экз. рабочего потока с именем шаблона рабочего потока
WF.add("WD1");
// ----- Обновляем объект рабочего потока
WF.retrieve();
// ----- Строим объект контейнера для рабочего потока
DKWorkflowContainerFed con = WF.inContainer();
// ----- Получаем данные контейнера
con.retrieve();
// ----- Добавляем строку PID, указывающую на документ Extended Search
con.setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
con.setPriority(100);
// ----- Изменяем контейнер
con.update();
// ----- Запускаем рабочий поток
WF.start(con);
```

#### C++

```
// - Создаем объект DKWorkflowFed и задаем имя
DKWorkflowFed* WF = new DKWorkflowFed(svWF);
WF->setName("wf1");
// Создаем экземпляр рабочего потока с именем шаблона рабочего потока
WF->add("WD1");
// ----- Обновляем объект рабочего потока
WF->retrieve();
// ----- Строим объект контейнера для рабочего потока
DKWorkflowContainerFed* con = WF.inContainer();
// ----- Получаем данные контейнера
con->retrieve();
// Добавляем строку PID, указывающую на элемент контента Extended Search
con->setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
// ----- Задаем приоритет 100
con->setPriority(100);
// ----- Изменяем контейнер
con->update();
// ----- Запускаем рабочий поток
WF->start(con);
. . .
// ----- Завершив, очищаем все - удаляем контейнер и рабочий поток
delete con;
delete WF;
```

---

## Прекращение рабочего потока

Прекратить рабочий поток можно, вызвав методы `terminate()` или `del()`, как показано в следующем примере:

### Java

```
// -----Получаем состояние рабочего потока под именем WF
WF.retrieve();
int state = WF.state();
// -----Проверяем состояние и вызываем terminate или del
if (state ==DKConstantFed.DK_FED_FMC_PS_RUNNING ||
 state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDED ||
 state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDING)
{
 WF.terminate();
}
if (state == DKConstantFed.DK_FED_FMC_PS_READY ||
 state ==DKConstantFed.DK_FED_FMC_PS_FINISHED||
 state ==DKConstantFed.DK_FED_FMC_PS_TERMINATED)
{
 WF.del();
}
```

### C++

```
// - Строим экземпляра DKWorkFlowFed
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF, "Test");
// -----Получаем состояние рабочего потока под именем WF
WF->retrieve();
int state = WF->state();
//---Проверяем состояние и вызываем terminate или del
if (state == DK_FED_FMC_PS_RUNNING ||
 state == DK_FED_FMC_PS_SUSPENDED ||
 state == DK_FED_FMC_PS_SUSPENDING)
{
 WF->terminate();
}
if (state == DK_FED_FMC_PS_READY ||
 state == DK_FED_FMC_PS_FINISHED ||
 state == DK_FED_FMC_PS_TERMINATED)
{
 WF->del();
}
delete WF;
```

## Получение списка всех рабочих потоков

Можно получить список всех рабочих потоков в службе рабочего потока с помощью метода `listWorkFlows()`. В следующем примере показано, как получить список имен и описаний всех рабочих потоков в службе рабочего потока, на которые ссылается объект `DKWorkflowService svWF`

### Java

```
// ----- Вызываем метод listWorkFlows
DKSequentialCollection collWF = (DKSequentialCollection)svWF.listWorkFlows();
DKWorkflowFed WF = null;
if (collWF != null)
{
 dkIterator iterWF = collWF.createIterator();
 while (iterWF.more() == true)
 {
 WF = (DKWorkflowFed)iterWF.next();
 WF.retrieve();
 System.out.println("name = " + WF.getName() + " description = "
 + WF.getDescription());
 }
 iterWF = null;
}
```

### C++

```
// ----- Вызываем функцию listWorkFlows
DKSequentialCollection *collWF =
 (DKSequentialCollection*)svWF.listWorkFlows();
DKWorkflowFed *WF = NULL;
if (collWF != NULL)
{
 dkIterator *iterWF = collWF->createIterator();
 while (iterWF->more())
 {
 WF = (DKWorkflowFed*)(void*)(*iterWF->next());
 WF->retrieve();
 cout << "name = " + WF->getName()
 << " description = " << WF->getDescription() << endl;
 delete WF;
 }
 delete iterWF;
}
delete collWF;
```

---

## Приостановка рабочего потока

Выполняемый рабочий поток можно приостановить на определенное или неопределенное время. В следующем примере показано, как приостановить рабочий поток до определенного времени. При пустом значении DKTimestamp EIP приостанавливает рабочий поток на неопределенное время.

### Java

```
// ----- Строим объект DKWorkflowFed
DKWorkflowFed WF = new DKWorkflowFed(svWF, "Test");
WF.retrieve();
// ----- Вызываем метод suspend, если рабочий поток находится
// ----- в состоянии выполнения
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_RUNNING)
{
 // ----- Приостановлен до 2000-07-27-16.30.00.000000
 // ----- В этой отметке времени базовым годом является 1900; месяцы
 // ----- пронумерованы от 0 до 11
 DKTimestamp suspension = new DKTimestamp(100, 6, 27, 16, 30, 0, 0);
 WF.suspend(suspension);
}
```

### C++

```
// ----- Строим экземпляр DKWorkflowFed
DKWorkflowFed* WF = new DKWorkflowFed(svWF, "Test");
WF->retrieve();
// ----- Вызываем функцию suspend, если рабочий поток запущен
if (WF->state() == DK_FED_FMC_PS_RUNNING)
{
 // ----- Приостановлен до 2000-07-27-16.30.00.000000
 DKTimestamp* suspension = new DKTimestamp(2000, 7,
 27, 16, 30, 0, 0);
 WF->suspend(suspension);
 delete suspension;
}
delete WF;
```

---

## Возобновление рабочего потока

Приостановленный рабочий поток можно возобновить, вызвав функцию resume(). В следующем примере возобновляется приостановленный рабочий поток.

### Java

```
// ----- Строим объект DKWorkflowFed
DKWorkflowFed WF = new DKWorkflowFed(svWF, "Test");
WF.retrieve();
// ---- Вызываем метод resume(), если рабочий поток приостановлен
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_SUSPENDED)
{
 WF.resume();
}
```

### C++

```
// ----- Строим экземпляр DKWorkflowFed
DKWorkflowFed* WF = new DKWorkflowFed(svWF, "Test");
WF->retrieve();
// ----- Проверяем, приостановлен ли этот рабочий поток, и вызываем resume
if (WF->state() == DK_FED_FMC_PS_SUSPENDED)
{
 WF->resume();
}
delete WF;
```

---

## Получение списка всех рабочих списков

Список всех рабочих списков в службе рабочего потока можно получить с помощью функции `listWorkLists()` для этой службы. В следующем примере мы получаем список имен и описаний всех рабочих списков в службе рабочего потока, на которую ссылается экземпляр `DKWorkflowServiceFed` с именем `svWF`.

### Java

```
// ----- Вызываем метод listWorkLists
DKSequentialCollection collWL = (DKSequentialCollection)svWF.listWorkLists();
DKWorkListFed WL = null;
if (collWL != null)
{
 dkIterator iterWL = collWL.createIterator();
 while (iterWL.more() == true)
 {
 WL = (DKWorkListFed)iterWL.next();
 WL.retrieve();
 System.out.println("name = " + WL.getName() + " description = "
 + WL.getDescription());
 }
 iterWL = null;
}
```

## C++

```
// ----- Вызываем функцию listWorkLists
DKSequentialCollection *collWL =
 (DKSequentialCollection*)svWF.listWorkLists();
DKWorkListFed *WL = NULL;
if (collWL != NULL)
{
 dkIterator *iterWL = collWL->createIterator();
 while (iterWL->more())
 {
 WL = (DKWorkListFed*)(void*)(*iterWL->next());
 WL->retrieve();
 cout << "имя = " << WL->getName() << " описание = "
 << WL->getDescription() << endl;
 cout << "Порог = " << WL->getThreshold() << endl;
 delete WL;
 }
 delete iterWL;
}
delete collWL;
```

## Доступ к рабочему списку

Чтобы получить доступ к рабочему списку, надо создать экземпляр DKWorkListFed, который ссылается на рабочий список, созданный вами с помощью клиента администратора системы. В следующем примере показано обращение к рабочему списку с именем WL0712 и вывод информации, содержащейся в этом рабочем списке.

## Java

```
// ----- Создаем DKWorkListFed
DKWorkListFed WL = new DKWorkListFed(svWF, "WL0712");
WL.retrieve();
// ----- Выводим информацию об этом рабочем списке
System.out.println ("worklist name = " + WL.getName());
System.out.println ("description = " + WL.getDescription() +
 " owner = " + WL.getOwner() +
 " filter = " + WL.getFilter() +
 " threshold = " + WL.getThreshold() +
 " sort criteria = " + WL.getSortCriteria());
```



## C++

```
// ----- Создаем DKWorkListFed
DKWorkListFed* WL = new DKWorkListFed(svWF, "WL0712");
WL->retrieve();
// ----- Выводим информацию об этом рабочем списке
cout << "имя рабочего списка = " << WL->getName() << endl;
cout << "описание = " << WL->getDescription() <<
 " владелец = " << WL->getOwner() <<
 " фильтр = " << WL->getFilter() <<
 " порог = " << WL->getThreshold() <<
 " критерий сортировки = " << WL->getSortCriteria() << endl;
// ----- Закончив, удаляем рабочий список
delete WL;
```

## Доступ к рабочим элементам

После создания DKWorkListFed рабочие элементы можно получить в виде собрания. В следующем примере показано получение рабочих элементов.

## Java

```
// ----- Создаем собрание и итератор
DKSequentialCollection coll = (DKSequentialCollection)WL.listWorkItems();
dkIterator iter = (DKSequentialIterator) coll.createIterator ();
Object a;
DKWorkItemFed item;
String nodename;
String workflowname;

// ----- Перебор собраний
while (iter.more ())
{
 a = iter.next ();
 item = (DKWorkItemFed) a;
 if (item != null)
 {
 item.retrieve ();
 nodename = item.name ();
 workflowname = item.workflowName ();
 System.out.println ("workitem node = " + nodename +
 " workflow name = " + workflowname);
 }
}
iter = null;
```

**C++**

```
DKSequentialCollection *coll;
 dkIterator *iter;
 DKWorkItemFed* item;
 DKString nodename;
 DKString workflowname;
 // ----- Создаем собрание и итератор
 coll = (DKSequentialCollection*)WL->listWorkItems();

 if (coll != NULL)
 {
 iter = coll->createIterator();
 cout << "listWorkItems" << endl;
 // ----- Перебор собраний
 while (iter->more ())
 {
 item = (DKWorkItemFed*)((void*)(*iter->next()));

 if (item != NULL)
 {
 //item.retrieve ();
 nodename = item->name();
 workflowname = item->workFlowName();
 cout << "узел рабочего элемента = " << nodename
 << " имя рабочего потока = " << workflowname << endl;
 delete item;
 }
 }
 delete iter;
 delete coll;
 }
```

## Перемещение элементов в рабочем потоке

По мере выполнения рабочего потока вы перемещаете элементы от одной операции к другой, используя функции `checkOut()` и `checkIn()`. В следующем примере показано, как перемещать рабочие элементы. Обратите внимание на то, что резервировать и активировать рабочий элемент может только пользователь рабочего потока, которому в настоящее время назначено выполнение этого рабочего элемента.

### Java

```
DKWorkItemFed item =new DKWorkItemFed(svWF, "wf1", "node1", wfuser);
item.retrieve();
// ----- Вызываем метод checkOut для резервирования рабочего элемента
item.checkOut();
// ----- Вызываем метод checkIn
item.checkIn(null);
```

### C++

```
DKWorkItemFed* item =new DKWorkItemFed(svWF, "wf1", "node1", wfuser);
item->retrieve();
// ----- Вызываем метод checkOut для резервирования рабочего элемента
item->checkOut();
// ----- Вызываем метод checkIn
item->checkIn(NULL);
delete item;
```

## Получение списка всех шаблонов рабочих потоков

Список всех шаблонов рабочих потоков в службе рабочего потока можно получить с помощью функции `listWorkFlowTemplates()`. Ниже приводится пример вывода списка имен и описаний всех шаблонов рабочих потоков в службе рабочего потока, на которую ссылается объект `DKWorkFlowServiceFed svWF`.

### Java

```
// ----- Вызываем метод listWorkFlowTemplates
DKSequentialCollection collWT =
 (DKSequentialCollection)svWF.listWorkFlowTemplates();
DKWorkFlowTemplateFed WT = null;
if (collWT != null)
{
 dkIterator iterWT = collWT.createIterator();
 while (iterWT.more() == true)
 {
 WT = (DKWorkFlowTemplateFed)iterWT.next();
 WT.retrieve();
 System.out.println("name = " + WT.name() + " description = "
 + WT.description());
 }
 iterWT = null;
}
```

**C++**

```
// ----- Вызываем функцию listWorkFlowTemplates
DKSequentialCollection *collWT =
 (DKSequentialCollection*)svWF.listWorkFlowTemplates();
DKWorkFlowTemplateFed *WT = NULL;
if (collWT != NULL)
{
 dkIterator* iterWT = collWT->createIterator();
 while (iterWT->more())
 {
 WT = (DKWorkFlowTemplateFed*)(void*)(*iterWT->next());
 WT->retrieve();
 cout << "имя = " << WT->name() << " описание = "
 << WT->description() << endl;
 delete WT;
 }
 delete iterWT;
}
delete collWT;
```

## Создание ваших собственных действий (только Java)

Можно создавать собственные действия для использования в рабочем потоке. Вы определяете действия и добавляете их в списки действий в управлении Enterprise Information Portal. Для создания действий используйте объекты действий (то есть объекты `DKWorkFlowActionFed`). Объект действия - это контейнер метаданных с подробными инструкциями о том, каким образом определенная задача будет выполняться на узле клиента. Объекты действия (контейнеры метаданных) только содержат инструкции; они не инициализируют выполнение задач, которые описаны в метаданных действия.

Действия могут быть сгруппированы в список действий (`DKWorkFlowActionListFed`). Контейнер рабочего потока несет имя списка действий (а не содержание списка действий), в котором собран набор действий, относящихся к рабочему элементу. Клиент должен получать список действий и в цикле по элементам этого списка выполнять соответствующие действия. В приведенном ниже примере показано, как работать с действиями и списками действий. В примере показано выполнение следующих задач:

1. Получение рабочих элементов.
2. Получение контейнера, который маршрутизируется вместе с рабочим элементом.
3. Получение списка действий из контейнера.
4. Извлечение списка действий из списка действий.
5. Запуск этих действий.

```

wit.retrieve(); // wit - это объект DKWorkItemFed
DKWorkFlowContainerFed wcn = wit.inContainer();
wcn.retrieve();
String alName = wcn.getActionList();
DKWorkFlowActionListFed wal = new DKWorkFlowActionListFed(dsFed);
wal.setName(alName);
wal.retrieve();
dkIterator iter = null;
if ((coll!=null) && (coll.cardinality(>0))
{
 iter = coll.createIterator();
 while (iter.more ())
 {
 DKWorkFlowActionFed act = (DKWorkFlowActionFed) iter.next();
 System.out.println("ACTION = " + act.getCommand());
 Runtime.getRuntime().exec(act.getCommand());
 }
}
else
 System.out.println("NO ACTION DEFINED");

```



---

# Построение прикладных программ с невидимыми и видимыми JavaBeans

В этой главе описываются невидимые и видимые JavaBeans, входящие в Enterprise Information Portal.

Функции JavaBeans EIP можно подразделить на следующие категории:

**Невидимые функции bean** Невидимые функции bean можно использовать для построения прикладных программ Java и клиента Web, требующих настроенного пользовательского интерфейса. Невидимые функции bean поддерживают стандартную модель программирования bean с помощью конструкторов по умолчанию, свойств, событий и последовательного интерфейса. Их можно использовать в инструментах разработчика, поддерживающих интроспекцию.

**Видимые функции bean** Видимые функции bean - это настраиваемые компоненты графического пользовательского интерфейса на основе Swing. Видимые функции bean позволяют строить прикладные программы Java для Windows. Их можно помещать в окна прикладных программ на основе Java. Поскольку видимые функции bean построены с помощью невидимых bean (в качестве модели данных), при построении программ их надо использовать совместно с невидимыми bean.

**Функции bean, связанные с программой просмотра Java** Функции bean программы просмотра Java - это набор видимых и невидимых компонентов. С их помощью можно строить программы просмотра документов и преобразовывать документы. Функции bean программы просмотра Java используются как в апплете просмотра eClient, так и на промежуточном уровне eClient.

---

## Основные понятия, связанные с функциями bean

JavaBeans (далее называемые просто функциями bean) - это многократно используемые компоненты программного обеспечения, написанные на языке программирования Java и управляемые с помощью специальных инструментов строителя для функций bean. Многократное использование позволяет строить при помощи функций bean более сложные компоненты и новые прикладные программы, а также добавлять новые возможности к уже существующим программам. Все это можно делать в наглядном режиме с помощью строителя, или же вручную, вызывая методы функций bean из программ.

Функции bean - это, по существу, классы Java. Почти любой программный компонент и класс Java можно преобразовать в функцию bean. Вообще говоря,

любой класс Java, отвечающий конкретным соглашениям относительно определений интерфейса свойств и событий, можно рассматривать как функцию bean.

Функции bean определяют интерфейс времени разработки, позволяющий инструментам разработчика программ или инструментам построителя запрашивать компоненты для выяснения видов свойств, определяемых этими компонентами, и видов событий, которые эти компоненты генерируют или на которые отвечают. При работе с функциями bean нет необходимости в специальных инструментах интроспекции и построения. Сигнатуры четко определены и легко распознаваемы.

Функции bean обладают следующими характеристиками:

### **Интроспекция**

Интроспекция - это процесс, который позволяет построителю определить, как работает функция bean во время разработки и выполнения. Поскольку программный код функций bean соответствует предопределенным шаблонам для сигнатур их методов и определений классов, инструменты, распознающие эти шаблоны, могут "заглядывать внутрь" функций bean и определять их свойства и поведение. Каждой функции bean соответствует класс информации bean, содержащий информацию о свойствах, методах и событиях для самой функции bean. Каждый класс информации bean реализует интерфейс BeanInfo, где явным образом перечислены те возможности функций bean, с которыми будут работать инструменты построителя прикладных программ.

### **Свойства**

Свойства позволяют управлять видом и поведением функций bean. Инструменты построителя анализируют функцию bean для определения ее свойств и работы с этими свойствами. Это позволяет изменять свойства функций bean во время разработки.

### **Настройка**

Во время разработки можно настраивать допускающие это функции bean. Настройка позволяет изменить их вид и/или поведение. Она выполняется при помощи редакторов свойств или специальных сложных программ настройки bean.

### **События**

С помощью событий функции bean связываются с другими функциями bean. Они могут инициировать события, то есть одна функция может сгенерировать событие, на которое среагирует другая функция. Функция bean, инициировавшая событие, называется функцией-источником bean. Функция bean, получившая событие, называется ожидающей функцией bean. Ожидающая функция bean регистрирует свою "заинтересованность" в этом событии на функции-источнике bean.



Инструменты построителя определяют события, отправленные функцией bean, и события, полученные ей, с помощью интроспекции.

### **Восстановимость**

Реализуя интерфейс java.io.Serializable, функции bean используют преобразование объектов Java в последовательную форму, что позволяет сохранять и восстанавливать состояния, которые могли измениться в результате настройки. Например, состояние функции bean сохраняется при ее настройке прикладной программой в построителе программ, что позволяет позднее восстановить измененные свойства.

### **Методы**

Все методы функций bean идентичны методам других классов Java. Методы функций bean можно вызывать с помощью других bean или через языки сценариев. По умолчанию экспортируются все общедоступные методы bean.

---

## **Использование JavaBeans в построителях**

В этом разделе описано, как пользоваться JavaBeans в IBM Websphere Studio Application Developer и других построителях.

При использовании построителей, отличных от IBM Websphere Studio Application Developer, убедитесь, что построитель поддерживает Java 2. Чтобы добавить файлы .jar, указанные в приведенных ниже инструкциях, следуйте указаниям построителя по добавлению новых файлов .jar. Затем выполните указания построителя по добавлению функций bean из файла .jar, чтобы добавить функции bean для EIP из файла cmb81.jar.

**Внимание:** Для использования этих bean у вас должен быть JDK 1.3.1 или новее.

Каталог CMBROOT\Samples\java содержит примеры кодов невизуальных функций bean.

## **Использование IBM Websphere Studio Application Developer**

Невизуальные функции bean можно использовать для построения сервлетов и страниц JSP в Webphere Studio Application Developer; для этого надо выполнить следующие действия:

1. Создайте проект Web для вашей Web-программы.
2. В свойствах этого проекта в разделе Java Build Path | Libraries (Путь построения Java | Библиотеки) укажите следующие файлы JAR:

\CMBROOT\lib\cmb81.jar

\CMBROOT\lib\cmbview81.jar

\CMBROOT\lib\cmbsdk81.jar

\CMBROOT\lib\esclisrv.jar

\\SQLLIB\\java\\db2java.zip

3. Если вы собираетесь использовать сервлеты EIP и библиотеку тегов JSP, надо указать также следующие файлы:

\\CMBROOT\\lib\\cmbServlet81.jar

\\CMBROOT\\lib\\cmbtag81.jar

Кроме того, для библиотеки тегов надо импортировать в вашу Web-программу файл taglib.tld библиотеки тегов JSP для EIP:

- Скопируйте \\CMBROOT\\lib\\taglib.tld в каталог webApplication\\WEB-INF вашей Web-программы.
- Сконфигурируйте библиотеку тегов в файле webApplication\\WEB-INF\\web.xml для вашей Web-программы, добавив:

```
<taglib>
 <taglib-uri>cmb</taglib-uri>
 <taglib-location>/WEB-INF/taglib.tld</taglib-location>
</taglib>
```

4. Поскольку перечисленные выше файлы JAR содержат классы J2EE, надо включить JAR J2EE, который обычно находится в \\Program Files\\IBM\\Application Developer\\plugins\\com.ibm.etools.websphere.runtime\\lib\\j2ee.jar

---

## Вызов Java bean EIP

Функции bean уровня EIP можно вызвать одним из двух способов. Можно вызывать их напрямую через их общедоступные интерфейсы (общедоступные методы). В этом случае будут генерироваться явные исключительные ситуации Java, указывая на возможные ошибки.

Другой метод вызова функций bean уровня сеанса состоит в связывании всех экземпляров bean этого типа с другими функциями bean EIP с помощью событий требований и ответов. Пользуясь этим методом, надо помнить следующее:

- Функция bean CMBCConnection ожидает событий требований соединения и в ответ инициирует события ответов соединения.
- Функция bean CMBDataManagement ожидает событий требований данных и в ответ инициирует события ответов данных.
- Функция bean CMBSchemaManagement ожидает событий требований схемы и в ответ инициирует события ответов схемы.
- Функция bean CMBQueryService ожидает событий требований поиска и в ответ инициирует события ответов поиска.
- Функция bean CMBWorkflowDataManagement ожидает событий требований данных рабочего потока и в ответ инициирует события ответов данных рабочего потока.
- Функция bean CMBWorkflowQueryService ожидает событий требований рабочего списка и в ответ инициирует события ответов рабочего списка.

---

## Невизуальные функции bean

В EIP есть набор невизуальных JavaBeans, с помощью которых можно строить прикладные программы Java. Невизуальные функции bean представляют собой набор классов Java, отвечающих соглашениям о функциях bean. Они строятся с помощью классов соединителя Java EIP и CM. Обычно они применяются сервлетах или в Java Server Pages (JSP), хотя их можно использовать также в программах командной строки и в программах с графическим интерфейсом.

Выгоды использования невизуальных функций bean:

- Обеспечивается механизм доступа объединения и общая модель программирования для множества различных соединителей, поставляемых с EIP.
- Достигается более высокий уровень отвлеченности при программировании.
- Максимируются сложности и подробности конкретных соединителей.
- Появляется возможность реализовать преимущества технологии bean в большинстве распространенных сред разработки.

Применение функций bean упрощает создание основных прикладных программ, однако прежде чем начать их использовать, необходимо знать о некоторых ограничениях. Функции bean:

- Не поддерживают всех возможностей, доступных на уровне API Java, например, возможностей управления и конфигурирования.
- Не поддерживают пакетный импорт. Функции bean поддерживают только возможности импорта отдельных типов элементов. Пакетные процессы для импорта и экспорта больших объемов данных следует писать, используя интерфейсы соединителей.
- Обеспечивают не все функции всех серверов. Некоторые функциональные возможности специфичны для конкретных серверов. Например, поддержка функций, связанных с подэлементами элементов, ограничена функциями соединителя Content Manager Версии 8.

Перечисленные выше ограничения можно в некоторых случаях обойти с помощью методов, открывающих доступ к базовым API Java.

**Внимание:** Функции bean EIP следует отличать от функций Enterprise Java Bean (EJB). Эти функции не могут находиться непосредственно внутри управляемой среды таких контейнеров, как IBM Websphere. Однако ими можно воспользоваться изнутри EJB в качестве базового механизма соединения с репозиториями неструктурированных данных.

## Конфигурации невизуальных функций bean

Невизуальные функции bean могут иметь локальную, удаленную или динамическую конфигурацию.

#### **локальная**

Соединяется непосредственно с контент-сервером.

#### **удаленная**

Соединяется с контент-сервером через сервер RMI.

#### **динамическая**

Позволяет писать программы, динамически переключающиеся между локальным и удаленным режимами на основе файла `cmbscs.ini`. В файле `cmbscs.ini` задано, является контент-сервер локальным или удаленным.

### **Возможности невизуальных функций bean**

Функции bean EIP можно использовать в страницах JSP, поскольку их свойства обычно представляют собой простые типы, такие как строки или массивы. Они действуют, в сущности, как модельные компоненты для прикладных программ Web, так как смоделированы с использованием шаблона структуры Model View Controller (MVC). Учтите, что компонент представления обычно состоит из страниц JSP, а компонент контроллера - из сервлетов (подобных сервлетам из комплекта сервлетов EJB). Ниже перечислены возможности невизуальных функций bean EIP:

- Обеспечивают доступ к определениям схем на библиотечном сервере.
- Реализуют методы создания, получения, изменения и удаления для документов, простых (нересурсных) элементов и ресурсных элементов во всех репозиториях, поддерживаемых EIP.
- Поддерживают функции поиска и получения документов, простых элементов и ресурсных элементов во всех репозиториях, поддерживаемых EIP.
- Поддерживают преобразование типов данных в доступные для просмотра форматы.
- Выступают как объединяющий уровень, на котором для множества репозиториях управления содержимым, поддерживаемых EIP, вводится согласованный набор семантик.
- Интегрируют и представляют функции рабочего потока EIP и служб исследования информации.
- Обеспечивают работу служб извлечения и преобразования документов, а также поддержку управления комментариями к документам.
- Поддерживают функции сортировки и преобразования.
- Инициатируют события для ключевых действий, выполняемых входящими в них функциями bean, например, события соединения и отсоединения, события уведомления о результатах поиска и события уведомления об изменении содержимого.

### **Категории невизуальных функций bean**

Невизуальные функции bean можно подразделить на следующие категории:

#### **Функции bean склада данных**

Эти функции bean доступны на протяжении типичного

пользовательского сеанса и предоставляют пользователю специальные услуги. К функциям bean уровня сеанса относятся:

- **CMBConnection** Эта функция bean поддерживает соединение с сервером, который может быть собственным контент-сервером или сервером объединения. Она необходима для работы с любыми функциями JavaBeans.
- **CMBSchemaManagement** Используется для работы с метаданными репозитория.
- **CMBDataManagement** Используется для работы с данными репозитория.
- **CMBQueryService** и **CMBSearchResults** Используются для выполнения запросов и работы с их результатами.
- **CMBWorkflowDataManagement** и **CMBWorkflowQueryService** Используются для работы с процессами расширенного рабочего потока EIP.
- **CMBDocRoutingDataManagement** и **CMBDocRoutingQueryService** Используются для работы с процессами маршрутизации документов CM v8.
- **CMBDocumentServices** Используются для потоковой передачи документов и поддержки служб аннотирования.

#### Функции bean поддержки

Функции bean поддержки существуют в контексте одной или нескольких функций bean уровня сеанса и непосредственно служат для инкапсуляции значений данных и сервисной поддержки функций bean уровня сеанса. К доступным функциям bean поддержки относятся:

- **CMBEntity** Представляет определения элементов данных, доступные в репозиториях управления содержимым. Например, для репозитория CM v8 CMBEntity представляет определения и типов элементов, и дочерних компонентов; для репозитория CM v7 CMBEntity представляет индексный класс. CMBEntity - это класс поддержки для CMBSchemaManagement.
- **CMBAttribute** Представляет определения атрибутов в репозиториях. CMBAttribute - это класс поддержки для CMBSchemaManagement.
- **CMBSearchTemplate** Представляет шаблон объединенного поиска. CMBSearchTemplate - это класс поддержки для CMBSchemaManagement.
- **CMBSTCriterion** Представляет критерий поиска - часть шаблона объединенного поиска. CMBSTCriterion - это класс поддержки для CMBSchemaManagement.
- **CMBItem** Представляет экземпляры документов, ресурсных элементов и нересурсных элементов. CMBItem - это класс поддержки для CMBDataManagement.

- **CMBOject** Представляет экземпляры ресурсных элементов, базовых частей и частей типа журнал примечаний. CMBOject также служит для представления атрибутов двоичных больших объектов. CMBOject - это класс поддержки для CMBDDataManagement.
- **CMBAnnotation** Представляет экземпляры частей типа комментариев для репозитория CM v8 и примечания для репозитория OnDemand.
- **CMBPrivilege** Поддерживает функции извлечения информации, связанной с привилегиями, из репозитория, поддерживаемых CM или EIP.

### Вспомогательные функции bean

Вспомогательные функции bean не играют решающей роли в прикладных программах, но могут быть полезны для улучшения их работы. Ниже приводится список вспомогательных функций bean.

- **CMBConnectionPool** Используются для предоставления служб пула функциям bean MVBConnection. Класс API Java DKDatastorePool позволяет поддерживать экземпляры DKDatastore, используемые экземплярами MVBConnection. Перемещение служб пула на уровень API Java позволяет лучше управлять контент-серверами, так как серверы разумнее объединять в пул с учетом их типа.
- **CMBUserManagement** Используется в соединениях с репозиториями объединения для управления отображением пользователей системы объединения на пользователей собственных серверов.
- **CMBExceptionSupport** Обеспечивает структуру для общей обработки исключительных ситуаций.
- **CMBTraceLog** Обеспечивает структуру для общей обработки событий трассировки, а также возможности ожидания событий трассировки, инициированных другими функциями bean.

### Функции bean рабочего потока

Функции bean рабочего потока предоставляют службы рабочего потока. Службы рабочего потока, предоставляемые функциями bean EIP, поддерживают два типа систем рабочего потока: расширенный рабочий поток и маршрутизацию документов. Функции расширенных рабочих потоков строятся с помощью MQSeries Workflow, а маршрутизация документов обеспечивается системой рабочего потока, интегрированной в продукт CM V8 и набор API. Уровень функций bean поддерживает полный набор объектов, необходимых для создания определений рабочих потоков, выполнения экземпляров рабочих потоков на основе созданных определений и управления экземплярами запущенных рабочих потоков. Ниже перечислены функции bean - основные компоненты поддержки расширенных рабочих потоков:

- **CMBWorkflowDataManagement** Эта функция bean служит для создания экземпляров расширенных рабочих потоков и работы с

ними. Экземпляр этой функции можно получить из объекта `CMVConnection`. Эта функция bean поддерживает:

- Запуск, прекращение, приостановку и возобновление экземпляра рабочего потока.
- Передачу рабочих элементов и рабочих уведомлений от одного пользователя другому.
- Отмену рабочих уведомлений.
- **CMBWorkflowQueryService** Функция `CMBWorkflowQueryService` обеспечивает интерфейс для запросов информации, связанной с расширенными рабочими потоками. Экземпляр этой функции можно получить из объекта `CMVConnection`. Эта функция bean поддерживает получение следующей информации:
  - Информации о рабочих потоках в системе.
  - Информации о рабочих элементах, проходящих через систему в составе активных рабочих потоков.
  - Информации, связанной с рабочими списками.
  - Информации обо всех зарегистрированных рабочих уведомлениях.

Ниже перечислены функции bean - основные компоненты поддержки маршрутизации документов.

- **CMBDocRoutingManagementICM** Эта функция bean служит для создания процессов маршрутизации документов и управления ими. Экземпляр этой функции bean можно получить из экземпляра объекта `CMVConnection`. Эта функция bean поддерживает следующие возможности:
  - Запуск, прекращение, приостановку и возобновление экземпляра маршрутизации документов.
  - Резервирование элемента CM, содержащегося внутри рабочего пакета.
  - Задание свойств для рабочих пакетов, маршрутизируемых экземпляром маршрутизации документов.
- **CMBDocRoutingQueryServiceICM** Эта функция bean обеспечивает интерфейс для запросов информации, связанной с процессами маршрутизации документов. Экземпляр этой функции bean можно получить из экземпляра объекта `CMVConnection`. Эта функция bean поддерживает получение следующей информации:
  - Информации, связанной с рабочими пакетами, маршрутизируемыми через систему маршрутизации документов.
  - Информации, относящейся к процессам, которые в настоящий момент активны в системе.
  - Информации обо всех рабочих списках, присутствующих в системе.
  - Списка всех рабочих узлов, входящих в систему.

## Функции bean исследования информации

Функции bean исследования информации EIP позволяют встраивать в прикладные программы текстовый анализ и технологию исследования информации. Они предоставляют следующие возможности:

- Составление сводок и категоризация текстов, создание сводок для документов.
- Категоризацию, то есть определение категории документа.
- Поддержку извлечения существенной информации из документа.
- Поддержку распознавания языка, на котором написан документ.
- Поддержку кластеризации, то есть объединения сходных документов в собрание документов.
- Текстовый поиск документов по всему каталогу или же ограниченный документами определенной категории.
- Поддержку доступа к документам, постоянно получаемым из Web с помощью искателя Web IBM (IBM Web Crawler).

К функциям bean исследования информации относятся:

- **CMBCatalogService** Обеспечивает интерфейс для получения информации, относящейся к каталогу. Эта функция также поддерживает импорт на контент-сервер элементов, созданных в результате операций исследования информации. Использование операций исследования информации ограничено каталогом. Каждому каталогу соответствует таксономия, которая, в свою очередь, состоит из иерархии категорий.
- **CMBAdvancedSearchService** Поддерживает выполнение текстового поиска информации, содержащейся в каталоге. Методы этой функции bean позволяют управлять каталогом, в котором будет выполнен поиск, а также содержимым и размером результатов, сгенерированных текстовым поиском.
- **CMBCategorizationService** Эта функция bean позволяет определять категорию документов на основе заданного каталога.
- **CMBSummarizationService** Обеспечивает генерирование сводок документов.
- **CMBClusteringService** Служит для объединения документов в кластеры на основе сходства их содержимого.
- **CMBWebCrawlerService** Обеспечивает интерфейс для управления результатами работы искателя Web. Эта функция позволяет пользователю инициировать требования поиска Web в конкретных областях Web и управлять результатами поиска. Полученные результаты можно затем категоризировать, составлять для них сводки и импортировать на контент-сервер.

## Функции bean служб документов

Функции bean служб документов поддерживают потоковую передачу



документов и службы аннотирования документов. Перечисленные ниже функции bean - часть подраздела служб документов:

- **CMBDocumentServices** - Функция bean CMBDocumentServices обеспечивает услуги, необходимые при работе с документами, включая возможности визуализации, преобразования и восстановления страниц одного или нескольких документов.
- **CMBDocument** - Эта функция bean представляет объект, созданный при загрузке документа с помощью CMBDocumentServices. По существу, CMBDocument - это контейнер для страниц документа. CMBDocument также позволяет запрашивать и задавать свойства, управляющие характеристиками документа.
- **CMBPage** - Эта функция bean обеспечивает представление отдельной страницы документа. Возможности этого класса позволяют задавать свойства набора возможных изображений, которые можно построить для данной страницы, и управлять ими.
- **CMBPageAnnotation** Эта функция bean моделирует комментарий, который можно связать со страницей в документе. Все поддерживаемые комментарии моделируются подклассами этой функции bean. Кроме того, сама функция CMBPageAnnotation содержит свойства, такие как страница, где помещен комментарий, и тип этого комментария. К подклассам этой функции относятся:
  - CMBArrowAnnotation
  - CMBCircleAnnotation
  - CMBHighlightAnnotation
  - CMBLineAnnotation
  - CMBNoteAnnotation
  - CMBPenAnnotation
  - CMBRectAnnotation
  - CMBStampAnnotation
  - CMBTextAnnotation

### Другие классы функций bean

Классы функций bean, перечисленные в этом разделе, поддерживают разнообразные возможности, описанные ниже.

- **Классы BeanInfo** Позволяют явно выделить возможности функций bean EIP в отдельный связанный с ними класс, реализующий интерфейс BeanInfo.
- **Классы исключительных ситуаций** Используются для инкапсуляции исключительных ситуаций на уровне функций bean. Все классы исключительных ситуаций - производные базового класса CMBException. Каждый из подклассов CMBException указывает на конкретную ошибку. В каждом из случаев свойства объекта исключительной ситуации позволяют получить подробную

информацию об ошибке, вызвавшей эту ситуацию. Если функции bean управляются событиями, исключительные ситуации генерируются внутри событий.

- **Классы событий и ожидания** Реализуют стандартную для функций bean модель ожидания событий. Классы должны явно запрашивать события путем реализации интерфейса ожидания, связанного с событием, и регистрации этого интерфейса ожидания с объектом, генерирующим событие. На уровне функций bean EIP имеются готовые пары "событие - ожидание" для операций доступа к схемам, доступа к данным, рабочего потока и поиска.
- **Функции ожидания события сеанса** Есть классы ожидания, существующие на уровне всего сеанса. Среди функций bean EIP в настоящее время есть отслеживающие события требований соединений и события ответов соединений на уровне сеанса.

## Особенности использования невизуальных функций bean

Невизуальные функции bean дают возможность прикладным программам общего назначения обращаться к репозиториям управления содержимым, поддерживаемым EIP. Этот раздел содержит советы по некоторым шаблонам использования функций bean.

**Одиночные элементы в функциях bean** У CMBConnection есть методы получения доступа к экземплярам других функций bean EIP уровня сеанса. Полученные этим способом функции bean уровня сеанса, такие как CMBSchemaManagement и CMBDDataManagement, оказываются уже связанными с функцией CMBConnection (из которой они были получены) для получения сообщений о соединении и отсоединении и совместного использования обработчиков событий трассировки и исключительных ситуаций. При этом создается только один экземпляр каждой из других функций bean всего сеанса. При повторном вызове этих методов возвращается тот же самый экземпляр (одиночный шаблон структуры). Если функции bean уровня сеанса создаются в прикладной программе, а не с помощью функций bean CMBConnection, их надо связать с используемой CMBConnection.

### Особенности работы с потоками для функций bean

В любой момент времени один экземпляр функции bean CMBConnection может быть использован только в одном потоке. Это ограничение распространяется на все остальные функции bean, связанные с CMBConnection (через свойство Connection соответствующей функции bean). Это означает, что для каждого потока необходимо создать отдельное соединение. Другой вариант - получение и освобождение соединений при многопоточности с помощью функции bean CMBConnectionPool. В этом случае каждый поток получает, использует и освобождает соединение.

Все функции bean уровня сеанса сопряжены с экземпляром CMBConnection, из которого они получены или с которым были связаны после создания. Это значит, что экземпляр функции bean уровня сеанса, например,

CMBSchemaManagement в каждый момент времени может использоваться только одним рабочим потоком. Если экземпляр функции bean уровня сеанса используется несколькими потоками, надо обеспечить в вашей прикладной программе явную синхронизацию, чтобы убедиться, что экземпляр bean уровня сеанса активно используется в каждый момент времени только одним рабочим потоком.

Все функции bean уровня сеанса также ожидают событий ответов соединений, генерируемых CMBConnection. Это позволяет им узнать об изменениях, произошедших в базовом репозитории содержимого, с которым связан экземпляр функции bean CMBConnection, и предпринять соответствующие действия.

В отличие от CMBConnection, функция bean CMBConnectionPool рассчитана на использование в многопоточном режиме. В этом случае разные потоки могут одновременно вызывать методы, связанные с получением и освобождением объектов соединений. Любое соединение, полученное от функции bean пула соединений, представляет собой экземпляр CMBConnection и доступ к нему ограничен одним потоком. Его следует как можно быстрее вернуть в пул, чтобы сделать доступным для других потоков, которым может понадобиться соединение из этого пула.

## **Трассировка и регистрация в функциях bean**

На уровне функций bean EIP можно разрешить трассировку всех функций bean всего сеанса. При включении трассировки для экземпляра CMBConnection включается также трассировка всех функций bean EIP, полученных от этой функции bean соединения, включая функции bean управления схемами, управления данными, службы запроса и рабочих потоков.

При включении трассировки инициируются события трассировки. Утилита bean CMBTraceLog ожидает событий трассировки и записывает трассировку в заданный журнал stdout, stderr или же в окно (при работе с визуальными bean).

Все функции bean всего сеанса также ожидают событий трассировки. Функция трассировки в bean записывает информацию регистрации в тот же файл журнала, что и API Java, если log4j используется для регистрации.

## **Что такое свойства и события для невизуальных функций bean**

Каждая невизуальная функция bean поддерживает:

- Импортируемые свойства, блокируемые или нет

Значение такого свойства задается во время выполнения другими функциями bean при помощи событий PropertyChange или VetoableChange. Функция bean с импортируемыми свойствами должны принимать события PropertyChange или VetoableChange.

- Экспортируемые свойства, блокируемые или нет

У невидимой функции bean могут быть свойства, значения которых могут представлять интерес для других функций bean. При каждом изменении этого значения функция bean должна генерировать событие `PropertyChange` или `VetoableChange`.

- Независимые свойства  
Значение такого свойства неважно для других функций bean.
- События, генерируемые этой функцией bean
- События, интересующие эту функцию bean

## **Построение прикладной программы с использованием невидимых функций bean**

### **Пример прикладной программы без графического пользовательского интерфейса**

В этом разделе приводится пример создания прикладной программы без графического интерфейса с использованием невидимых функций bean. Эта программа включает все функции bean, кроме `CMBUserManagement`. Полный текст прикладной программы (`DemoSimpleApp1.java`), откуда взят приведенный пример, находится в каталоге `Cmbroot/Samples/java/beans`. В этом примере прикладной программы показано, как:

1. Соединиться с сервером Enterprise Information Portal (сервером объединения)
2. Получить список имен шаблонов поиска
3. Получить список критериев поиска по имени шаблона поиска
4. Выбрать шаблон поиска и получить для него критерии поиска
5. Задать все значения для поиска и послать запрос
6. Напечатать результат с помощью функции bean результатов поиска
7. Выбрать строку результата и вывести ее на экран
8. Отсоединиться от сервера

---

## **Работа с визуальными функциями bean**

Визуальные функции bean позволяют включить функции Enterprise Information Portal или другого контент-сервера в прикладные программы Java на основе Swing. Визуальные функции beans выполняют базовые задачи, общие для многих программ, такие как регистрация, поиск, вывод и просмотр результатов, изменение документов и просмотр информации о версиях.

У каждой визуальной функции bean есть свойство `Connection`. Значение этого свойства - ссылка на экземпляр `CMVConnection`, невидимой функции bean, которая поддерживает соединение с контент-серверами. Любая прикладная программа, построенная с помощью визуальных функций bean EIP, должна также содержать экземпляр невидимой функции bean `CMVConnection`.

### **CMBLogonPanel**

Эта функция bean выводит панель для регистрации в Enterprise Information Portal или на контент-серверах, таких как Content Manager Версии 8.2 (СМ 8.2). Она также выводит окно, где пользователи системы объединения могут изменять ID пользователя и пароли на контент-сервере.

### **CMBSearchResultsViewer**

Эта функция bean выводит результаты поиска. Если результат поиска - папки, с помощью функции bean CMBSearchResultsViewer можно раскрыть папку для просмотра ее содержимого. Элементы или папки в результатах поиска можно выбрать и открыть для просмотра в окне в стиле Проводника Windows.

### **CMBSearchTemplateList**

Для серверов, которые поддерживают шаблоны поиска, эта функция bean выводит на экран список доступных шаблонов поиска, позволяя выбрать нужный шаблон.

### **CMBSearchTemplateViewer**

Для серверов, которые поддерживают шаблоны поиска, эта функция bean выводит шаблон поиска и поля для ввода пользователями критериев поиска. Она выполняет поиск на основе этих критериев.

### **CMBSearchPanel**

Для любого сервера панель поиска содержит список доступных объектов и поля для ввода критериев поиска. На основе этих критериев выполняется поиск. Панель CMBSearchPanel полезна для поиска на контент-серверах, которые не поддерживают шаблоны поиска.

### **CMBFolderViewer**

Выводит содержимое одной или нескольких папок в окне в стиле Проводника Windows.

### **CMBItemAttributesEditor**

Выводит окно, где пользователи могут изменять индексный класс и атрибуты индексирования для элемента.

### **CMBDocumentViewer**

Выводит на экран один или несколько документов, запуская соответствующую программу просмотра.

### **CMBVersionsViewer**

Выводит информацию о версии документа, если версии поддерживаются.

## Функция bean CMBLogonPanel

Функция bean CMBLogonPanel (смотрите рис. 20) выводит окно, позволяющее пользователям регистрироваться на контент-сервере, изменить отображения пользователей и пароль.

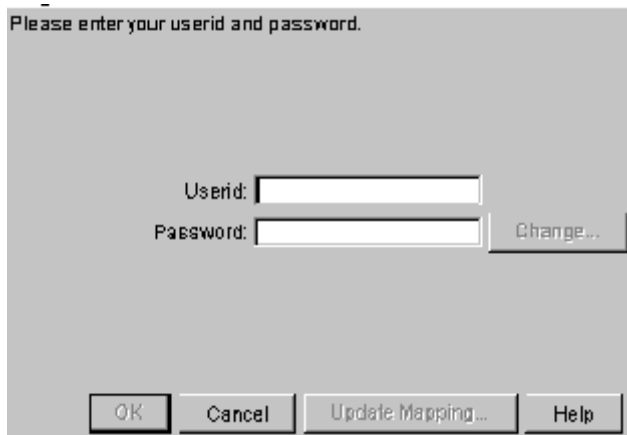


Рисунок 20. Окно функции bean CMBLogonPanel

В функции bean CMBLogonPanel при нажатии пользователем кнопки **Изменить** появляется окно **Изменение пароля** (смотрите рис. 21). Пользователь вводит старый пароль, а затем дважды - новый пароль.



Рисунок 21. Окно Изменение пароля

В функции bean CMBLogonPanel при нажатии пользователем кнопки **Изменить отображение** появляется окно **Изменение отображения пользователей** (смотрите рис. 22 на стр. 515). При изменении отображения вы изменяете заданные для

сервера ID пользователя и пароль. Эта функция доступна, только если вы зарегистрированы на базе данных объединения Enterprise Information Portal.

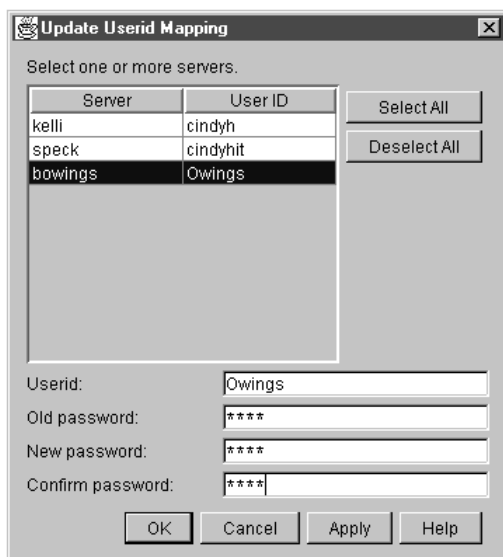


Рисунок 22. Окно Изменение отображения ID пользователей

В верхней части окна выводится список всех серверов и ID пользователя для каждого сервера. Пользователи могут выбрать в этом списке один или несколько серверов. Чтобы выбрать все серверы, нажмите кнопку **Выбрать все**. После того, как выбраны один или несколько серверов, можно задать новый ID пользователя и (необязательно) пароль. Если выбран один сервер, в поле **ID пользователя** выводится ID пользователя. Если выбрано несколько ID пользователя, поле **ID пользователя** остается пустым.

#### Отменить выбор всех

Отменяет выбор всех серверов.

#### Применить

Нажмите эту кнопку, чтобы применить изменения отображения и паролей, не закрывая окна.

**ОК** Нажмите, чтобы сохранить изменения и закрыть это окно.

#### Отмена

Нажмите, чтобы закрыть окно без сохранения изменений.

## Функция bean CMBSearchTemplateList

У функции bean CMBSearchTemplateList есть три формата вывода данных. В первом формате, показанном на рис. 23 на стр. 516, одно изображение используется в качестве фонового рисунка для выбранных элементов, а другое - для невыбранных элементов. На рис. 24 на стр. 516 показан формат с простым

списком шаблонов. На рис. 25 показан формат с выпадающим списком шаблонов.

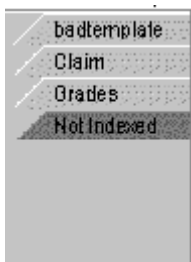


Рисунок 23. Формат со списком шаблонов изображений

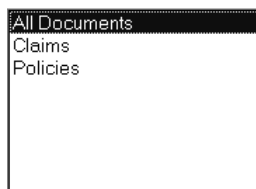


Рисунок 24. Формат с простым списком шаблонов

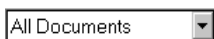


Рисунок 25. Формат с выпадающим списком шаблонов

## Функция bean CMBSearchTemplateViewer

Функция bean CMBSearchTemplateViewer (смотрите рис. 26) выводит окно, в котором можно задать критерии поиска в соответствии с шаблоном поиска, определенным системным администратором. Эта функция bean также запускает поиск и генерирует событие CMBSearchResults для возвращения результатов поиска.

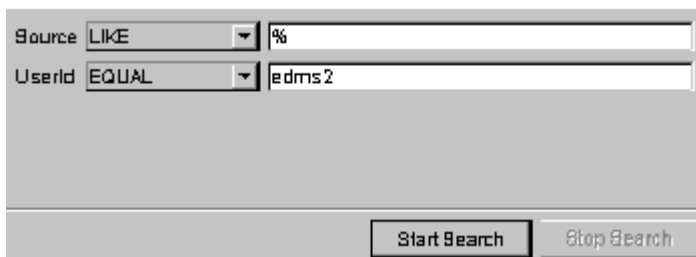


Рисунок 26. Функция bean CMBSearchTemplateViewer



Функция bean `CMBSearchTemplateViewer` выдает список критериев поиска, например, `Source` или `Userid`. У каждого критерия поиска есть метка, список операций и текстовое поле. Для операций `BETWEEN` и `NOTBETWEEN` выводится два текстовых поля. В операциях `IN` и `NOTIN` используется многострочная текстовая область. Каждое значение надо вводить на отдельной строке.

### **Области текстового поиска**

Функция bean `CMBSearchTemplateViewer` может содержать области, позволяющие пользователям выполнять поиск с полнотекстовыми или индексными атрибутами. Область полнотекстового поиска в шаблоне может быть просто текстовым полем с меткой.

При вводе строки запроса в текстовое поле синтаксис запроса должен соответствовать свободному или логическому текстовому поиску (смотрите описание класса `DKDatastoreTS`). Подробнее об этом смотрите в электронном справочнике API.

## **Проверка или редактирование полей в `CMBSearchTemplateViewer`**

Можно задать операторы для проверки полей в функции bean `CMBSearchTemplateViewer`, чтобы изменить введенные пользователем критерии поиска. Для этого создайте обработчик для события `CMBTemplateFieldChangedEvent`. Текущие значения критерия поиска сохраняются в `CMBTemplate`, возвращаемым методом `getTemplate` перед этим событием. Эти значения можно просмотреть и изменить. После завершения обработки события выводятся новые значения.

## **Функция bean `CMBSearchPanel`**

Функция bean `CMBSearchPanel` выводит окно, в котором пользователи могут задавать критерии поиска объектов, доступных на текущем контент-сервере. Эта функция bean также запускает поиск и генерирует событие `CMBSearchResultsEvent` для возвращения результатов поиска. В верхней части панели `CMBSearchPanel` есть выпадающий список доступных объектов. Если выбран объект, `CMBSearchPanel` выводит атрибуты этого объекта. У каждого атрибута есть метка, выпадающий список операций и текстовое поле. У операций диапазона, таких как `BETWEEN` или `NOTBETWEEN`, есть два текстовых поля. У операций со многими значениями, таких как `IN` или `NOTIN`, есть многострочное поле ввода. Каждое значение надо вводить на отдельной строке этой многострочной текстовой области.

## **Функция bean `CMBSearchResultsViewer`**

Функция bean `CMBSearchResultsViewer` выводит результаты поиска в окне, которое содержит панель дерева и панель подробностей. Можно изменить размеры этих панелей, перетаскив разделительную линию.

На рис. 27 показана функция bean CMBSearchResultsViewer с выбранной папкой **Результаты поиска**.

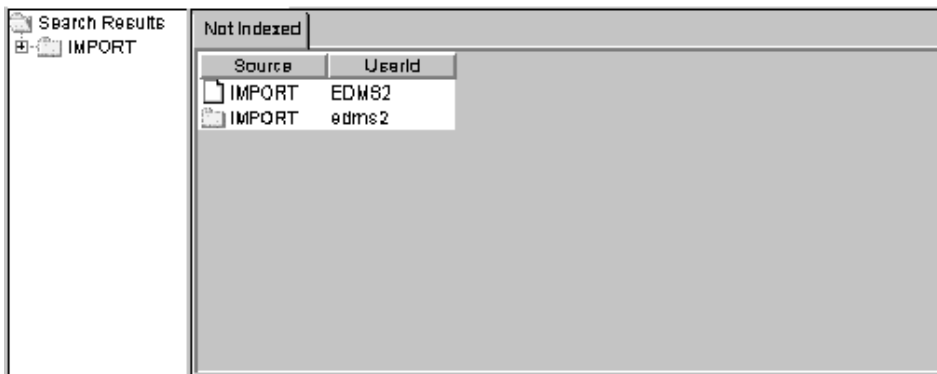


Рисунок 27. Функция bean CMBSearchResultsViewer

### Панель дерева CMBSearchResultsViewer

Панель дерева (слева) содержит главную папку с меткой **Результаты поиска**. Под этой папкой выводятся все папки, найденные при поиске. Это необязательная панель. Ее можно убрать с экрана, задав значение свойства TreePaneVisible: setTreePaneVisible(false).

### Панель подробностей CMBSearchResultsViewer

На панели подробностей выводится содержимое папки, выбранной на панели дерева. Если выбрана папка **Результаты поиска**, в записной книжке появляется вкладка с именем шаблона поиска. Если выбрана другая папка, в папке **Результаты поиска** появится одна или несколько вкладок, по одной для каждого индексного класса в этой папке. Имена этих вкладок имеют вид:

**индексный класс @ сервер**

где *индексный класс* - это имя индексного класса или типа элементов, а *сервер* - имя контент-сервера. Столбцы таблицы меняются в зависимости от выводимых атрибутов данного индексного класса или типа элементов. На панели подробностей можно выбирать несколько объектов. Выбор нескольких объектов можно отключить, задав для свойства MultiSelectEnabled значение setMultiSelectEnabled(false). Для иерархического типа элементов значения атрибутов дочерних компонентов выводятся в таблице с заголовками столбцов вида имя дочернего компонента/имя атрибута. Например, если у типа элементов Journal есть дочерний компонент под названием Author, а у дочернего компонента Author есть атрибут Last Name, заголовок столбца будет: Author/Last Name.

### Всплывающие меню

Всплывающие меню с опциями сортировки выводятся при щелчке правой кнопкой мыши по заголовку столбца таблицы. Если выбрать **Сортировать по возрастанию**, элементы в таблице будут отсортированы в порядке возрастания. Если выбрать **Сортировать по убыванию**, элементы будут отсортированы в порядке убывания. Другое всплывающее меню появится, если щелкнуть правой кнопкой мыши по другой папке (не **Результаты поиска**) или по папке или документу на панели подробностей. Всплывающее меню позволяет пользователям просматривать подробности папок на панели дерева или редактировать атрибуты папок.

**Необязательно:** Вместо вывода подробностей папки в функции bean `CMBSearchResultsViewer` можно воспользоваться `CMBViewFolderEvent`. С помощью этого события можно заставить функцию bean `CMBFolderViewer` вывести содержимое выбранной папки.

### Двойной щелчок мышью

Двойной щелчок по папке в панели дерева папок или по элементу на панели подробностей вызывает те же действия, что и выбор пункта **Вид** соответствующего всплывающего меню. Если вы запретили вывод всплывающего меню по умолчанию для элемента, генерируется событие `CMBItemActionEvent`.

## Переопределение всплывающих меню

Можно переопределить всплывающие меню для `CMBSearchResultsViewer` и `CMBFolderViewer`, задав другие всплывающие меню или отменив вывод всплывающего меню. Чтобы отключить меню по умолчанию, задайте `setDefaultPopupMenu(false)`.

Если щелкнуть правой кнопкой мыши по папке на панели дерева, генерируется событие `CMBFolderPopupEvent`. Если щелкнуть правой кнопкой мыши по элементу на панели подробностей, генерируется событие `CMBItemPopupEvent`. Для вывода других всплывающих меню можно использовать обработчики.

## Функция bean `CMBFolderViewer`

Функция bean `CMBFolderViewer` выводит панель дерева, которая выглядит как в функции bean `CMBSearchResultsViewer`. Однако при этом нет главной папки **Результаты поиска**. На рис. 28 на стр. 520 показаны панели дерева и подробностей в функции bean `CMBFolderViewer`.

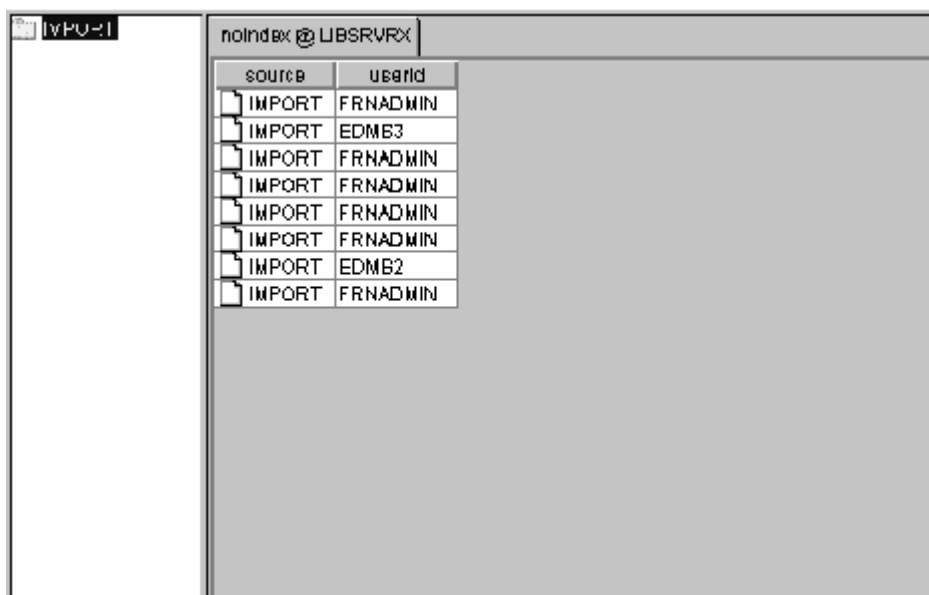


Рисунок 28. Функция bean CMBFolderViewer

В левой панели функция bean CMBFolderViewer выводит дерево папок. В правой панели выводится записная книжка таблиц для документов, которые содержатся в папке, выбранной на панели дерева. Между панелями дерева и записной книжки находится подвижный разделитель.

### Панель дерева CMBFolderViewer

Эта панель дерева содержит папки. Под каждой из них выводятся вложенные папки.

### Панель подробностей CMBFolderViewer

На панели подробностей выводится содержимое папки, выбранной в панели дерева. Это содержимое выводится в виде записной книжки со вкладками для каждого объекта (индексного класса, типа элементов и т.п.) и сервера, используемых для индексирования элементов таблиц. Имена вкладок имеют форму: индексный класс @ сервер, где *индексный класс* - это имя индексного класса, а *сервер* - имя сервера. На каждой странице записной книжки в таблице выводятся документы и папки, содержащиеся в выбранной папке. Столбцы таблицы меняются в зависимости от выводимых атрибутов данного индексного класса.

### Всплывающие меню

Поведение всплывающих меню для просмотра папок такое же, как и для просмотра результатов поиска.

## Двойной щелчок мышью

Действие двойного щелчка при просмотре папок такое же, как и при просмотре результатов поиска.

## Функция bean CMBDocumentViewer

Функция bean CMBDocumentViewer позволяет просматривать документы при помощи запускаемых или встроенных программ просмотра документов, определяемых их содержимым. Поддерживается два типа программ просмотра:

1. Программы просмотра на основе Java. Эти программы просмотра должны быть созданы на основе класса CMBJavaDocumentViewer.
2. Программы просмотра не на основе Java. В качестве программы просмотра для конкретного типа содержимого можно задать любой исполняемый файл.

Если для свойства Visible задано значение false, программа просмотра всегда запускается в отдельном окне. Если значение Visible - true, программа просмотра будет по возможности выводить изображение в области вывода функции bean CMBDocumentViewer. (Пока это возможно только для программ просмотра на основе Java.)

CMBJavaDocumentViewer - это абстрактный класс, на основе которого разработчики создают программы просмотра документов на основе Java, подключаемые к функции bean CMBDocumentViewer. Эти программы просмотра могут выводить документы в области вывода функции bean CMBDocumentViewer или в отдельном окне на экране.

Вызов CMBDocumentViewer terminate() ожидает завершения обработки всех закрытых событий документов. Если вы вызовете terminate() из обработчика закрытого события документа, может произойти тупиковая ситуация и программа зависнет. Чтобы избежать этого, при вызове terminate() из обработчика событий onDocumentClosed(CMBDocumentClosedEvent) вызывайте метод CMBDocumentViewer.terminate() при помощи SwingUtilities.invokeLater(Runnable). При этом вызов terminate() будет помещен в конец очереди; до его вызова будут обработаны все прочие события в этой очереди (такие как обработка событий закрытия других документов).

## Как задать программу просмотра

Есть два способа задать программу просмотра:

1. В EIP Administration программы просмотра задаются с помощью редактора связей типов MIME с прикладными программами. Для этого надо выбрать в меню **Инструменты** пункт **Редактор связей MIME**. Для программ просмотра на основе Java имя прикладной программы должно быть именем класса Java, включая суффикс **.class**. Для исполняемых файлов имя прикладной программы должно быть именем исполняемого файла.

2. С помощью свойства Mime2App в CMBDocumentViewer. В качестве значения этого свойства можно задать экземпляр объекта свойств, отображающего типы MIME на имена прикладных программ.

Когда программа просмотра задана для типа MIME и в EIP Administration, и с помощью свойства Mime2App, задание с помощью Mime2App будет иметь преимущество.

## Программы просмотра по умолчанию

Если программа просмотра для данного типа содержимого не указана, запускается программа просмотра по умолчанию. Для документов из OnDemand запускается клиент OnDemand (в режиме только для просмотра). Документы со всех остальных контент-серверов просматриваются с помощью программы просмотра Content Manager. Чтобы редактировать комментарии, выберите в меню Файл программы просмотра Редактировать документ.

## Запуск внешних программ просмотра

Прикладные программы, запускаемые для просмотра документов определенных типов MIME, можно задать с помощью свойства Mime2App в CMBDocumentViewer. Используйте для этого метод setMime2App, передавая ему в качестве аргумента объект свойств, в котором задано отображение имен типов MIME на имена выполняемых файлов.

## Функция bean CMBItemAttributesEditor

Функция bean CMBItemAttributesEditor (смотрите рис. 29) выводит на экран окно для просмотра и изменения индексного класса и атрибутов индексирования папки или документа.

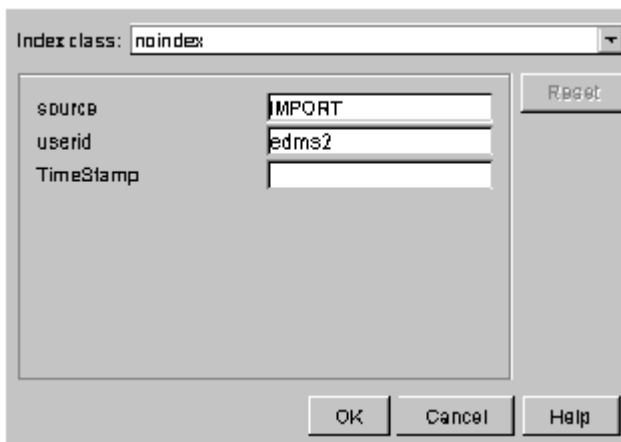


Рисунок 29. Функция bean CMBItemAttributesEditor

В верхней части окна выводится список всех доступных объектов. По умолчанию выбран текущий объект. Под списком объектов находится список

атрибутов для этого объекта. Текстовые поля (например, имя, фамилия и т.п.) изначально содержат текущие значения полей для этого элемента.

Если выбран новый объект, всем атрибутам нового объекта, имена которых совпадают с именами атрибутов ранее выбранного объекта, присваиваются значения аналогичных атрибутов предыдущего объекта.

Если нажать кнопку **Сброс**, объекту и атрибутам возвращаются их исходные значения.

Нажатие кнопки **ОК** приводит к изменению объекта и атрибутов и генерирует события перед изменением и после него. Событие, генерируемое перед изменением, можно использовать для проверки правильности содержания полей и заполнения пропущенных полей перед выполнением изменения. Это событие может заблокировать заданное изменение.

### **Блокировка изменений в CMBItemAttributesEditor**

Можно задать дополнительные операторы проверки для CMBItemAttributesEditor, проверяющие введенные пользователем значения атрибутов, изменяющие их или запрещающие изменение при недопустимости значений. Для этого создайте обработчик события CMBEditRequestedEvent.

### **Функция bean CMBVersionsViewer**

Функция bean CMBVersionsViewer выводит таблицу с атрибутами версий для отдельного документа или элемента. Выводятся следующие атрибуты версий: номер версии, ID пользователя оператора, отметка времени создания версии, ID последнего пользователя, изменявшего объект, и отметка времени последнего изменения. Из программы просмотра версий вы можете просматривать различные версии элемента или изменять его атрибуты.

### **Общие особенности поведения визуальных функций bean**

В следующих разделах описаны общие свойства и поведение визуальных функций bean.

#### **Свойства**

В этом разделе описываются три общих свойства визуальных функций bean.

#### **Connection**

У каждой функции bean есть свойство Connection, представляющее собой ссылку на экземпляр невидимой функции bean CMBConnection. Для правильной работы визуальной функции bean необходимо задать значение этого свойства.

#### **CollationStrength**

У каждой функции bean, выполняющей сортировку, есть свойство CollationStrength. Для свойства CollationStrength определены те же значения, что и для класса Java java.text.Collator.

### Скрытие/вывод кнопок

Кнопки всех визуальных функций bean можно сделать видимыми или скрытыми. Это достигается с помощью свойства *setNameButtonVisible*, где *Name* - имя кнопки.

### Сохранение/восстановление конфигурации

У функций bean *CMBSearchTemplateViewer*, *CMBSearchResultsViewer* и *CMBFolderViewer* есть два метода - *loadConfiguration* и *saveConfiguration* - которые можно использовать для сохранения значений полей и размеров столбцов между сеансами работы прикладной программы и последующего их восстановления. Аргумент этих методов - это объект свойств. Для всех трех этих функций bean можно использовать один объект свойств. Имена сохраняемых свойств уникальны для каждой функции bean.

### События справки

Когда пользователь запрашивает справку, нажав кнопку **Справка** или клавишу F1, каждая визуальная функция bean генерирует событие *CMBHelp*. При нажатии клавиши F1 или кнопки Справка из вторичных окон некоторые функции bean генерируют следующие события, связанные со справкой:

#### **CMBChangePasswordHelpEvent**

Если кнопка **Справка** нажата в окне Изменение пароля

#### **CMBUpdateMappingHelpEvent**

Если кнопка **Справка** нажата в окне Изменение отображения

#### **CMBLoginFailedHelpEvent**

Если кнопка **Справка** нажата в окне Регистрация на сервере завершилась неудачно

#### **CMBServerUnavailableHelpEvent**

Если кнопка **Справка** нажата в окне Сервер недоступен

**Совет:** Возможный путь обработки всех этих типов вызовов справки - создать один класс, реализующий ожидание этих событий. В методе *onHelp* может понадобиться задать дополнительные операторы для определения функции bean - источника события и вывода текста справки для этой функции bean.

### Замена визуальной функции bean

Можно заменить одну визуальную функцию bean на другую или на компонент Swing. Для этого новая функция bean должна содержать обработчики для событий замещаемой визуальной функции bean. Она должна также генерировать, как минимум, ключевые события замещаемой функции bean. Эти ключевые события описаны в Табл. 31.

Таблица 31. Визуальные функции bean и ключевые события

| Визуальная функция bean      | Ключевые события                |
|------------------------------|---------------------------------|
| <i>CMBSearchTemplateList</i> | <i>CMBTemplateSelectedEvent</i> |



Таблица 31. Визуальные функции bean и ключевые события (продолжение)

| Визуальная функция bean | Ключевые события                                                   |
|-------------------------|--------------------------------------------------------------------|
| CMBSearchTemplateViewer | CMBSearchStartedEvent CMBSearchResultsEvent                        |
| CMBSearchResultsViewer  | CMBViewDocumentEvent CMBViewFolderEvent-CMBEditItemAttributesEvent |
| CMBFolderViewer         | CMBViewDocumentEvent CMBEditItemAttributesEvent                    |
| CMBDocumentViewer       | Событие закрытия CMBDocumentOpenedEvent CMBDocument                |
| CMBItemAttributesEditor | нет                                                                |

Все данные, необходимые для реализации этих функций, можно получить или из обрабатываемых этой функцией bean событий, или из невидимой функции bean CMBConnection.

## Построение прикладной программы с использованием визуальных функций bean

Вы можете посмотреть пример клиентской прикладной программы, написанной с использованием визуальных функций bean. Исходные файлы для него находятся в каталоге: <cmbrroot>/samples/java/beans/gui. Прочтите также файл readme.html в этом каталоге, чтобы узнать подробности о клиенте в этом примере и требованиях к установке.

В следующих разделах показано использование визуальных функций bean при построении прикладной программы.

### Соединения визуальных функций bean

В этом разделе описан сценарий соединения визуальных функций bean для создания простой прикладной программы. За исключением кнопки **Поиск**, все функции bean соединены друг с другом, то есть каждое событие, генерируемого одной функцией bean, ожидает другая функция bean. Например, чтобы соединить SearchTemplateList с SearchTemplateViewer, достаточно одной строки программы. Добавить кнопку, запускающую поиск, можно с помощью стандартного компонента JButton. Создайте внутренний класс, который при нажатии этой кнопки будет генерировать событие, вызывающее соответствующий метод.

Линии на рис. 30 на стр. 526, идущие от каждой функции bean к bean соединения, показывают, что эта функция bean содержит ссылку на функцию bean соединения. Для создания таких ссылок у каждой функции bean задается свойство Connection. Например, для создания ссылки от функции bean панели регистрации к функции bean соединения достаточно одной строки программы.



На рис. 30 показано 9 функций bean. Все эти функции bean - дочерние для JFrame или других функций bean контейнеров. Возможная последовательность событий при выполнении:

1. Пользователь вводит в окне регистрации ID пользователя и пароль и нажимает кнопку **ОК**. Функция `bean CMBLogonPanel` вызывает метод `connect` из функции `bean CMBConnection` для установления соединения с сервером.
2. Функция `bean` соединения устанавливает соединение. После этого функция `bean CMBSearchTemplateList` получает и выводит на экран список шаблонов поиска для этого ID пользователя. (Для этого не требуется вызывать никаких методов. Функция `bean CMBSearchTemplateList` ожидает соответствующих событий от функции `bean CMBConnection`. `CMBSearchTemplateList` задает ожидающие функции `bean`, когда функция `bean CMBConnection` связывается с ней при помощи метода `setConnection()`.)
3. Пользователь выбирает из списка шаблон поиска. Функция `bean CMBSearchTemplateList` генерирует событие `CMBTemplateSelectedEvent`. Это событие ожидают функции `bean CMBSearchTemplateViewer` и `CMBSearchResultsViewer`. `CMBSearchTemplateViewer` выводит на экран соответствующий шаблон. `CMBSearchResultsViewer` очищает панель подробностей и выводит в ней столбцы, определенные этим шаблоном.
4. Пользователь заполняет шаблон и нажимает клавишу `Enter` или кнопку **Поиск**. Если пользователь нажимает кнопку **Поиск**, обработчик события для этого действия вызывает метод `startSearch`. Если пользователь нажимает клавишу `Enter`, метод `startSearch` вызывается неявно.
5. Функция `bean CMBSearchTemplateViewer` проверяет поля шаблона, определяя, можно ли начать поиск. Если поиск можно начать, генерируется событие `CMBSearchStartedEvent`. Это событие обрабатывает ожидающая его функция `bean CMBSearchResultsViewer`, которая очищает объекты результатов, подготавливая получение новых результатов поиска.

6. По мере хода поиска генерируются события `CMBSearchResultsEvent`, обеспечивающие поступление частичных результатов поиска в функцию bean `CMBSearchResultsViewer`. (Когда поиск завершен, генерируется событие `CMBSearchCompleted`. Это событие можно использовать для повторного активирования кнопки **Поиск**, если она была отключена в начале поиска.)
7. Пользователь может открыть папки в результатах поиска и выбрать документ или папку для просмотра. После этого генерируется событие `CMBViewFolderEvent` или `CMBViewDocumentEvent`. Эти события обрабатываются соответственно ожидающими их функциями bean `CMBFolderViewer` и `CMBDocumentViewer`, которые выводят на экран содержимое папки или документа.
8. В функции bean `CMBFolderViewer` можно выбрать документ для просмотра. При выборе документа для просмотра генерируется событие `CMBViewDocumentEvent`. Функция bean `CMBDocumentViewer` ожидает это событие и выводит документ на экран в соответствующей программе просмотра.
9. Можно выбрать атрибуты документа или папки для изменения из `CMBSearchResultsViewer` или `CMBFolderViewer`. При выборе документа генерируется событие `CMBEditItemAttributesEvent`.
10. Событие `CMBEditItemAttributesEvent` ожидает функция bean `CMBItemAttributesEditor`. Она выводит на экран объект и атрибуты для этого элемента. Затем можно изменить объект и атрибуты и нажать кнопку **ОК**, чтобы применить эти изменения.

### Использование функций bean в нескольких окнах или диалоговых панелях

Чтобы передать событие от функции bean в одном окне функции bean в другом окне, нужно задать дополнительные операторы. Обычно для вывода окна на экран достаточно факта отправки события. Окно `EditAttributesDialog` содержит функцию bean `ItemAttributesEditor`. `SearchFrame` создает это окно, когда сгенерировано событие `CMBEditItemAttributesEvent`:

```
// Вызываем вторичное диалоговое окно для редактирования атрибутов
searchResultsViewer.addEditItemAttributesListener(new
CMBEditItemAttributesListener() {
 public void onEditItemAttributes(CMBEditItemAttributesEvent event) {
 EditAttributesDialog editAttributesDialog = new
 EditAttributesDialog(SearchFrame.this,connection,event.getItem());
 editAttributesDialog.setVisible(true);
 }
});
```

Информация, которая обычно передается функции bean `CMBItemAttributesEditor`, вместо этого передается конструктору этого окна как аргументы. В конструкторе эта информация передается функции bean `CMBItemAttributesEditor` при задании его свойств:

```
itemAttributesEditor.setConnection(connection);

itemAttributesEditor.setItem(item);
```

---

## Работа с комплектом программы просмотра документов Java

Для доступа к документам на ваших контент-серверах и добавления к ним комментариев можно использовать программу просмотра документов. С помощью комплекта программы просмотра Java EIP можно создать пользовательскую программу просмотра документов. Кроме того, можно создать пользовательские апплеты и программы для интегрирования в EIP, а также автономные программы.

**Внимание:** Опции *viewdata* не поддерживаются механизмом OnDemand.

Классы комплекта программы просмотра Java содержат объекты действия, поддерживающие следующие функции:

- Опции просмотра страниц
  - Поворот документов: на 90 градусов по часовой стрелке, на 90 градусов против часовой стрелки, на 180 градусов
  - Увеличить, уменьшить
  - Масштаб: 25%, 50%, 100%, 150%, 200% и 400%
- Инвертирование
- Улучшение
- Печать
- Закрыть текущий документ
- Закрыть все документы
- Создать и редактировать комментарии
  - Записать
  - Выделить
  - Нарисовать рамку
  - Нарисовать окружность
  - Нарисовать линию
  - Нарисовать стрелку
  - Добавить текст
  - Штамп
  - Добавить примечание
  - Стереть
  - Спрятать или показать
  - Режим курсора по умолчанию

- Переместить комментарий на передний план
- Переместить комментарий на задний план
- Изменить свойства комментария
- Отменить или повторить операции с комментарием
- Вырезать, копировать, вставить или удалить комментарии
- Сохранить документ (сохраняются только комментарии)
- Перейти к документу или странице в пределах документа
  - Переходы по страницам: первая страница, предыдущая страница, перейти к странице, следующая страница, последняя страница
  - Переходы по документам: первый документ, предыдущий документ, перейти к документу, следующий документ, последний документ
- Миниизображения
  - Спрятать или показать миниизображения
  - Переход к страницам с помощью миниизображений
  - Панорамный вид и изменение масштаба

Комплект программы просмотра документов Java содержит много классов GUI, которые можно использовать для построения программ на основе Swing. Он содержит также классы без графического интерфейса, которые можно использовать для программ просмотра документов, не использующих Swing.

---

## Структура программы просмотра

Комплект программы просмотра Java содержит программу просмотра документов и функцию bean Document Services. Функции bean обеспечивают механизм интегрирования программы просмотра в прикладные программы на основе EIP.

Комплект программы просмотра содержит также классы общей программы просмотра документов, потоковых служб документов и служб аннотирования. Эти классы позволяют использовать программу просмотра в прикладных программах, когда нельзя использовать функции bean, например, в ситуациях автономного просмотра или распределенной обработки, где содержимое не является локальным.

Потоковые службы документов управляют набором механизмов обработки документов, которые анализируют документы, строят страницы и дают возможность работать со страницами документов. Эти механизмы обеспечивают анализ документов различных форматов, например, таких как TIFF и ЮСА, с изображениями-страницами, а также документов в текстовом формате, формате ttf и документов Office. Кроме того, можно написать

дополнительные механизмы документов и встроить их в архитектуру комплекта программы просмотра для поддержки других форматов или альтернативного построения для форматов документов.

Класс служб аннотирования позволяет работать с комментариями. Механизм аннотирования анализирует специальные форматы комментариев Content Manager. Можно написать дополнительные механизмы аннотирования.

## **Механизмы документов**

В EIP есть четыре механизма документов:

- Механизм документов MS-Tech: управляет обычными типами содержимого IBM Content Manager, строя страницы как изображения. Поддерживает типы документов TIFF, MO:DCA, а также GIF, JPEG и простой текст.
- Механизм документов INSO: поддерживает Microsoft Office, Lotus SmartSuite и другие форматы деловых документов.
- Механизм документов AFP2Web: понимает AFP и преобразует документы AFP в HTML или PDF.
- Механизм документов Java: преобразует документы-URL, в HTML со ссылкой на этот URL. Этот механизм также преобразует XML в HTML путем вызова XSLT.

Эти механизмы являются общими интерфейсами, но непосредственно включать их в программу не следует. Вместо этого используйте интерфейсы, предлагаемые функцией bean Document Services или классом потоковых служб документа.

Некоторые из этих механизмов используют не только чистые средства Java и имеют ограничения по переносимости. Это может ограничить использование комплекта на некоторых платформах. Механизмы MS-Tech и Java используют только чистые средства Java. Другие механизмы содержат логические средства, специфичные для платформы, и их можно использовать только на платформе Windows.

## **Механизм аннотирования**

EIP содержит механизм аннотирования MS-Tech для работы с комментариями Content Manager. Этот механизм поддерживает комментарии Content Manager Версии 8.2, Content Manager Версии 8.1, Content Manager Версии 7 и VI/400.

---

## **Создание общей программы просмотра документов**

При создании общей программы просмотра документов главным образом с классом CMBGenericDocViewer, который использует интерфейс CMBStreamingDocServices. CMBStreamingDocServices загружает документы и строит их изображения. CMBStreamingDocServices использует набор механизмов документов для преобразования разных форматов, таких как TIFF и MO:DCA.

Для загрузки, редактирования и сохранения комментариев в документах используйте интерфейс CMBAnnotationServices.

---

## Настройка общей программы просмотра документов

Можно настроить файл конфигурации по умолчанию -

CMBViewerConfiguration.properties, расположенный в архиве cmbview81.jar, или создать новый файл конфигурации. Создаете ли вы конфигурационный файл или настраиваете CMBViewerConfiguration.properties, необходимо сохранить прежнее имя файла и поместить его перед cmbview81.jar в каталоге класса.

Чтобы создать файл конфигурации, выполните следующие действия:

1. Для каждой панели инструментов надо создать запись с именем этой панели. На этом шаге задается положение панели инструментов в основном фрейме. Положение по умолчанию - NORTH. Укажите положение для каждой из перечисленных панелей инструментов.

```
Панели инструментов=[<имя_панели_инструментов>[,<имя_панели_инструментов2>[,<имя_панели_инструментов3>]...]
<имя_панели_инструментов>.position={NORTH|SOUTH|EAST|WEST}
```

2. Укажите действия, которые нужно добавить в заданную панель инструментов. Чтобы поставить разделитель между действиями на панели инструментов, используйте слово 'separator'.

```
<имя_панели_инструментов>.tools=[<имя_действия>[,<имя_действия2>[,<имя_действия3>]...]
```

```
<имя_действия>.label=<метка_действия>
<имя_действия>.tooltip=<пояснение_действия>
<имя_действия>.icon=<имя_файла_значка>
<имя_действия>.key=<код_ключа>
<имя_действия>.cursor=<файл_курсора>
<имя_действия>.hotspot=<x,y>
```

3. Нельзя добавлять новые всплывающие меню. Однако в три предварительно заданных всплывающих меню можно добавить подменю и пункты меню.

```
<имя_всплывающего_меню>.items=[<имя_пункта_меню>[,<имя_пункта_меню2>[,<имя_пункта_меню3>]...]
<имя_всплывающего_меню>.submenu=[<имя_подменю>[,<имя_подменю2>[,<имя_подменю3>]...]
<имя_подменю>.label=<метка_подменю>
```

Если вы задали соответствующий файл конфигурации, вы готовы построить автономную программу или апплет просмотра. При выполнении следующих шагов смотрите *Электронный справочник API*:

1. Создайте собственный класс, реализующий CMBStreamingDocServicesCallbacks.
2. Создайте CMBStreamingDocServices с обратными вызовами, реализованными на первом шаге.



3. Создайте собственный класс, реализующий `CMBAnnotationServicesCallbacks`.
4. Создайте `CMBAnnotationServices` с реализованными обратными вызовами.
5. Создайте экземпляр `CMBGenericDocViewer` и инициализируйте его с `CMBStreamingDocServices`, `CMBAnnotationServices` и файлом свойств конфигурации. Если вы просто хотите использовать файл конфигурации по умолчанию, передайте пустое значение для файла свойств.
6. Чтобы загрузить документ, вызовите `loadDocument()` в `CMBGenericDocViewer`. Это возвращает экземпляр `CMBDocument`.
7. Чтобы загрузить любые комментарии, вызовите `loadAnnotationSet()` в `CMBGenericDocViewer`. Возвращается объект `CMBAnnotationSet`. Используйте метод `setItemHandle(CMBAnnotationSet, CMBItem)` в `CMBAnnotationServices`.
8. Настройте внешний вид программы просмотра, используя различные методы - положение, размер миниизображений, окно MDI или SDI и так далее.
9. Добавьте эту программу просмотра в основной фрейм прикладной программы или апплета.
10. Подготовьте полосу меню, взяв действия из общей программы просмотра документов и добавив их к меню основного фрейма прикладных программ.
11. Вызовите `showDocument()` в общей программе просмотра документов, чтобы вывести этот документ на экран.
12. Вызовите `saveAnnotations(CMBDocument)` в `CMBGenericDocViewer`, чтобы сохранить комментарии в документе.
13. Вызовите `closeDocument(CMBDocument)`, чтобы закрыть документ, или `closeAllDocuments()`, чтобы закрыть все документы.

На рис. 31 на стр. 534 показан пример общей программы просмотра документов, использующей все параметры по умолчанию.



Рисунок 31. Общая программа просмотра документов

## Примеры прикладных программ

Чтобы помочь вам понять комплект программы просмотра документов Java, в этих разделах описываются пять примеров прикладных программ, которые можно создать. Этот комплект можно использовать и другими способами, помимо приведенных в примерах.

### Автономная программа просмотра

Для реализации автономной программы просмотра можно использовать общую программу просмотра документов. Автономную программу просмотра можно использовать для просмотра файлов или документов, получаемых по адресам URL. Автономные программы просмотра можно использовать в качестве апплетов на Web-страницах или для просмотра документов, получаемых, например, по электронной почте. На рис. 32 на стр. 535 показана структура автономной программы просмотра.

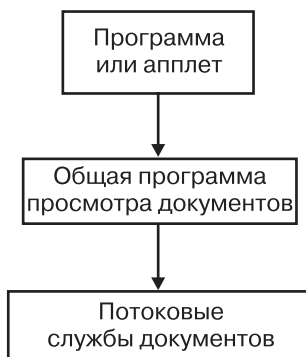


Рисунок 32. Автономная программа просмотра

## Программа Java

Чтобы создать рабочую прикладную программу Java, используйте визуальные функции bean EIP, в которых применяется визуальная функция bean `CMBDocumentViewer`. Для вывода эта функция документов использует встроенную общую программу просмотра документов. Кроме того, функция bean `CMBDocumentViewer` может запускать другие программы просмотра, чтобы просматривать документы. Однако помните, что эти программы просмотра могут зависеть от платформы. На рис. 33 показана возможная структура прикладной программы Java.

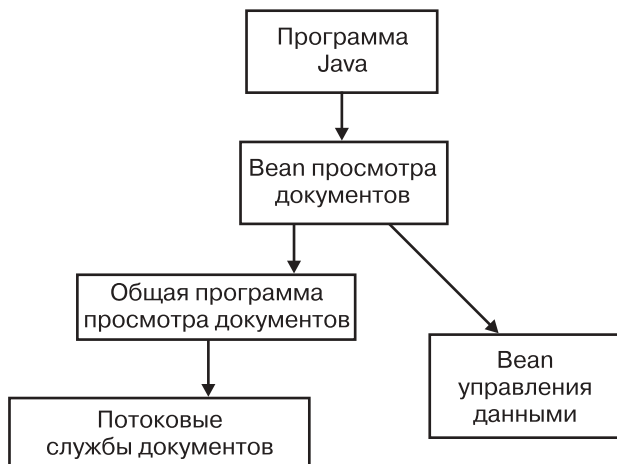


Рисунок 33. Прикладная программа Java

## Минимальный клиент

Функцию `bean CMBDocumentServices` можно использовать для преобразований документа на стороне сервера в прикладной программе Web. Можно преобразовывать документы типов содержимого, не обрабатываемых браузером (для таких документов требуется запуск дополнительного модуля или собственной программы), в типы содержимого, обрабатываемые самим браузером, такие как HTML, GIF, JPEG, или такие как PDF, для которых уже есть дополнительные программные модули. На рис. 34 показана структура минимального клиента.

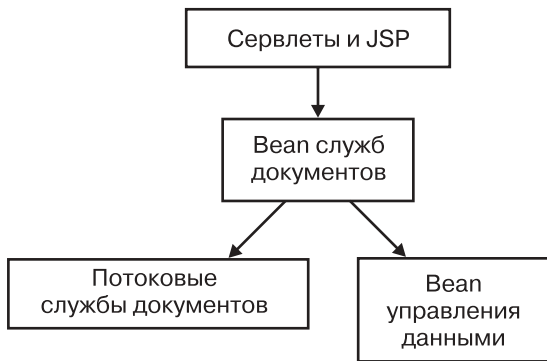


Рисунок 34. Минимальный клиент

## Апплет или сервлет

Прикладная программа Web может обеспечить возможности просмотра документов и редактирование комментария, используя технологию апплета или сервлета. Такую структуру использует апплет просмотра `eClient`. Для просмотра документа апплет может использовать общую программу просмотра документов. Документы некоторых типов хранятся на контент-сервере по частям, чтобы обеспечить более эффективное совместное использование общей информации, такой как формы фонов. При необходимости общая программа просмотра документов требует эти дополнительные части. Апплет, содержащий эту программу просмотра, удовлетворяет эти требования, посылая требования HTTP сервлету. На стороне сервлета контент-сервер, используя функцию `bean CMBDataManagement`, получает запрошенную информацию. На рис. 35 на стр. 537 показана структура апплета или сервлета.

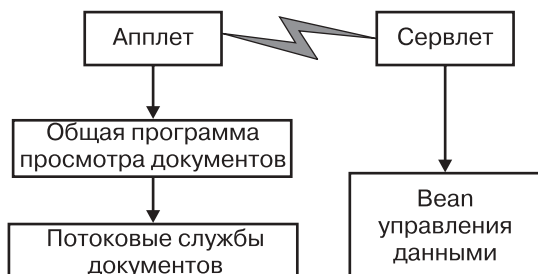


Рисунок 35. Апплет или сервлет

## Двойной режим, апплет или сервлет

Вы можете использовать вариант этих примеров для программ просмотра, базируемых в Web. На сервере и в апплете используйте CMBDocumentServices. Такой подход полезен, когда документы не подготавливаются для вывода в апплете, а преобразуются на сервере. Это может случиться, когда возможности базовых механизмов документов в апплете или на сервере различны.

Например, если сервер - это Windows NT или 2000, а апплет выполняется в OS/2, апплет может быть не в состоянии выводить все типы документов. Апплет выводит документы тех форматов, которые он поддерживает, например, TIFF. Для типов, которые апплет не может вывести, например для форматов Office, он запрашивает преобразование у сервлета. С точки зрения пользователя интерфейс и функциональность остаются теми же самыми. Однако производительность сервера по преобразованию документов на стороне сервера может быть ниже, чем для документов, построенных локально.

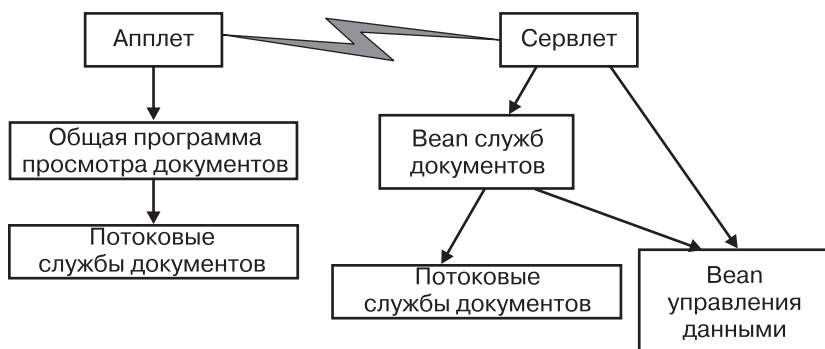


Рисунок 36. Двухрежимная программа просмотра

## Работа со службами аннотирования

Комплект программы просмотра Java Enterprise Information Portal 8.1 поддерживает возможности вывода и конвертирования комментариев документов. Подобно службам документов, подключаемые механизмы аннотирования обеспечивают дополнительные возможности, которые вы можете использовать в своих программах для работы с комментариями различных типов. На рис. 37 показано взаимодействие между общей программой просмотра документов, службами документов и службами аннотирования.

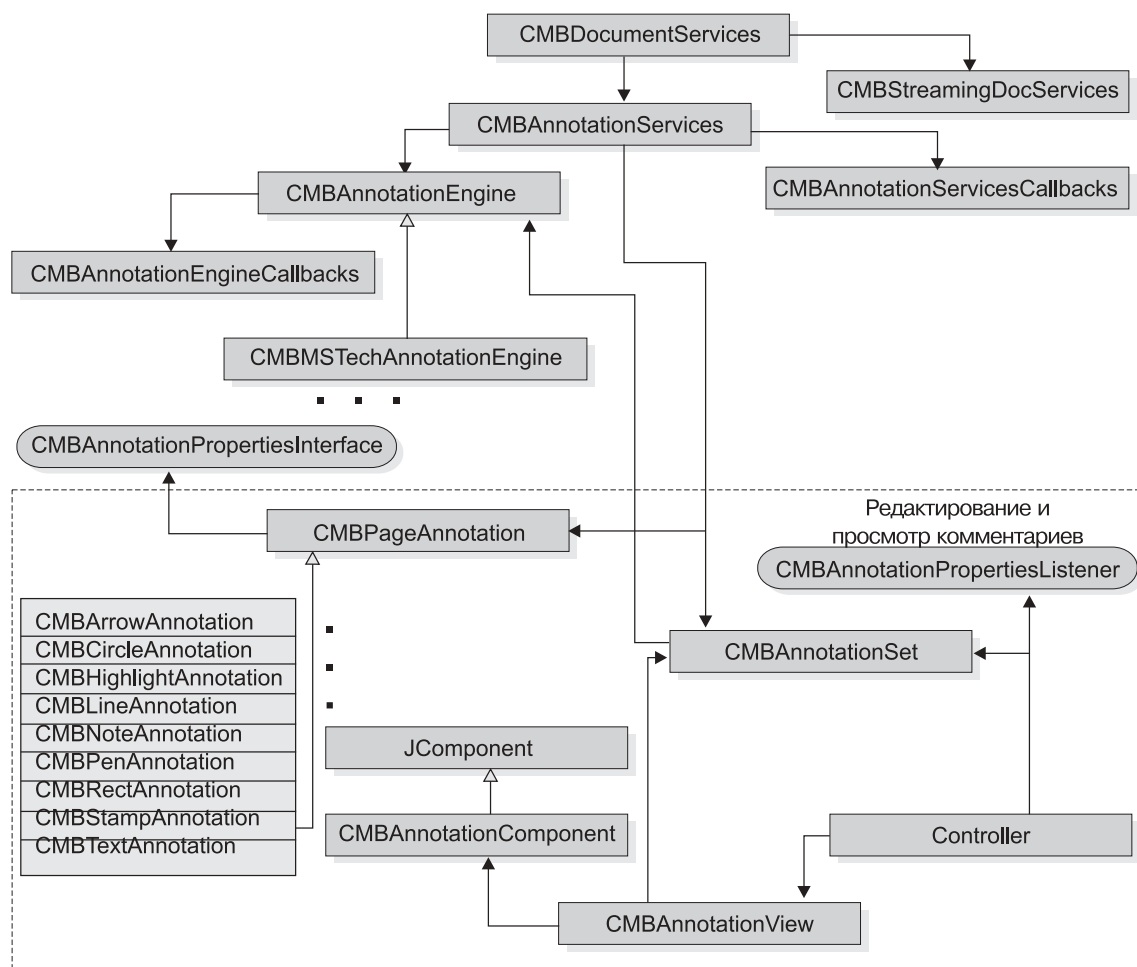


Рисунок 37. Связи между службами аннотирования и общей программой просмотра документов

## Использование интерфейсов со службами аннотирования

Основные интерфейсы, используемые в комплекте программы просмотра Java, предоставляет CMBAAnnotationServices. Службы аннотирования позволяют загружать, обрабатывать и сохранять объекты комментариев при помощи средств аннотирования, независимо от систем хранения баз данных. Чтобы работать с комментариями, надо включить подходящий механизм аннотирования, который преобразует объекты комментариев в экземпляры CMBPageAnnotation, и передать данные комментария как входящий поток. После этого вы сможете работать с комментариями и редактировать их, а потом сохранить эти комментарии обратно в исходную систему баз данных в исходном формате.

На рис. 38 показана диаграмма класса служб аннотирования.

Чтобы реализовать механизм аннотирования, надо расширить абстрактный класс CMBAnnotationEngine. Механизм аннотирования использует интерфейсы CMBAnnotationServicesCallbacks и CMBAnnotationEngineCallbacks, чтобы связываться с прикладной программой и службами аннотирования. Механизм аннотирования в EIP 8.1 понимает только формат комментариев Content Manager. Этот формат комментариев используется на компьютерах баз данных Content Manager Версии 7, Content Manager Версии 8. 1 и VI/400.

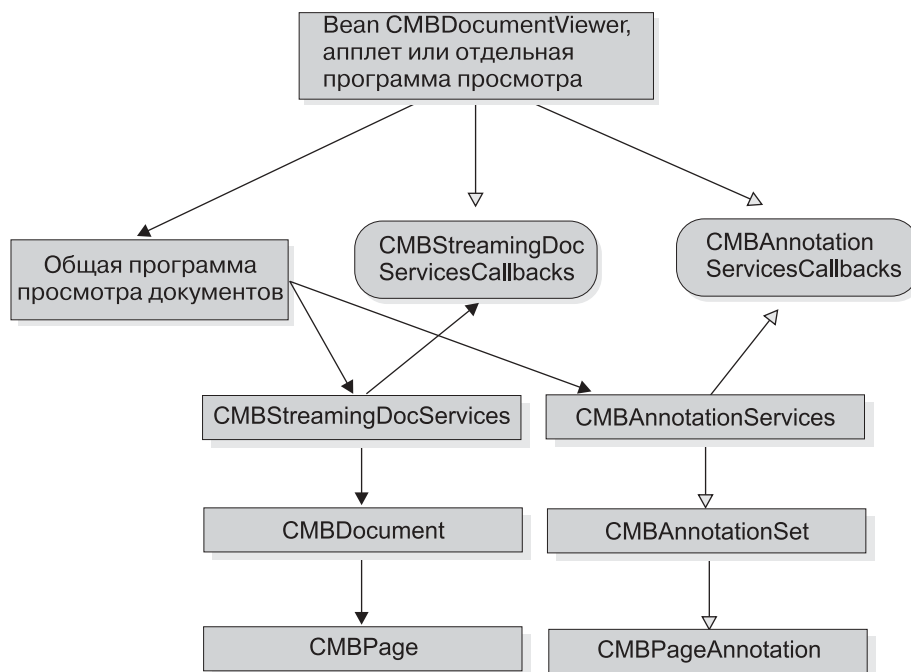


Рисунок 38. Диаграмма класса служб аннотирования

## Как работает поддержка редактирования аннотации

Для реализации возможности редактирования комментариев используется шаблон структуры Model View Controller (MVC). MVC действует как модель, которая представляет данные комментария. В `CMBAnnotationSet` есть методы работы с данными, но нет пользовательского интерфейса. Класс `CMBAnnotationSet` поддерживает список объектов `CMBPageAnnotation`. С каждым документом связан объект `CMBAnnotationSet`, который представляет его комментарии. `CMBAnnotationView` работает как средство, которое представляет данные модели пользователю. `CMBAnnotation` обрабатывает все связи комментариев для компонента представления (`JComponent`). `CMBAnnotationComponent` - это вспомогательный класс, который можно использовать как компонент представления для показа связей комментариев. Контроллер является внутренним для инструментария программы просмотра, он обрабатывает события мыши и клавиатуры для создания и редактирования комментариев.

`CMBPageAnnotation` - это базовый класс, который описывает отдельный комментарий на странице документа. Если вам нужно определить дополнительные типы графических комментариев, надо расширить класс `CMBPageAnnotation`. Можно создавать комментарии Content Manager девяти типов: `CMBArrowAnnotation`, `CMBCircleAnnotation`, `CMBHighlightAnnotation`, `CMBLineAnnotation`, `CMBNoteAnnotation`, `CMBPenAnnotation`, `CMBRectAnnotation`, `CMBStampAnnotation` и `CMBTextAnnotation`.

**Примечание:** Службы аннотирования можно использовать в программах без графического интерфейса.

## Построение прикладной программы с использованием служб аннотирования

В этом разделе описаны необходимые действия и интерфейсы API для построения прикладной программы с использованием служб аннотирования. Подробности использования API смотрите в электронном справочнике по API.

1. Создайте подкласс класса `CMBAnnotationServicesCallbacks`, чтобы реализовать абстрактные методы обработки обратных вызовов комментариев.
2. Создайте экземпляр `CMBAnnotationServices`.  

```
CMBAnnotationServices annoServices = new
CMBAnnotationServices(annoServicesCallbacks);
```
3. Получите экземпляр `CMBAnnotationSet`, загрузив поток комментария.  

```
CMBAnnotationSet annotationSet = annoServices.loadAnnotationSet(annoStream,
format, documentResolution, annotationPartNumber);
```
4. Приготовьте представление комментария.



```

CMBAnnotationComponent annoComponent = new CMBAnnotationComponent();
CMBAnnotationViewer annoView = annoServices.prepareAnnotationView(
annoComponent, annotationSet);
annoView.refreshEntireDrawingArea();

```

5. Добавьте компонент комментария в контейнер Swing как JFrame или JPanel прикладной программы.
6. Теперь вы сможете использовать интерфейсы API служб аннотирования, чтобы добавлять новые комментарии и редактировать или удалять уже существующие комментарии.

```

annoServices.prepareToAddAnnotation();
annoServices.addAnnotation()
annoServices.removeAnnotation();
annoServices.reorderAnnotation();
...

```
7. Сохраните измененные комментарии.

```

annoServices.saveAnnotationset(annotationSet);

```

Пример со службами аннотирования приводится в каталоге <CMBROOT>\Samples\java\viewer directory(TAnnotationEditor.java).

## Как добавить пользовательский тип аннотации в собственной прикладной программе

Чтобы добавить в службы аннотирования пользовательский тип комментария, выполните описанные ниже действия. Подробности использования API смотрите в электронном справочнике по API.

1. Создайте подкласс класса CMBPageAnnotation. `public class TImageAnnotation extends CMBPageAnnotation`
2. Определите константу для пользовательского комментария со значением больше 100. Значения от 1 до 99 зарезервированы.

```

public static final int ANN_IMAGE = 101;

```
3. Переопределите следующие методы CMBPageAnnotation:

```

public void draw(Graphics2D g2);
public void drawOutline(Graphics2D g2);
public CMBPropertiesPanel getAnnotationPropertiesPanel();

```
4. Реализуйте интерфейс CMBAnnotationPropertiesInterface, чтобы создать панель свойств. Панель свойств появляется, когда пользователь выбирает редактирование свойств пользовательского комментария.
5. Реализуйте методы set и get для свойств пользовательского комментария.
6. Добавьте пользовательский тип комментария.

```

annoServices.prepareToAddAnnotation(TImageAnnotation.ANN_IMAGE,
"ImageAnnotation",1);

```

Пример с пользовательским типом комментария TImageAnnotation содержится в каталоге <CMBROOT>\Samples\java\viewer. В этом примере показано, как добавить пользовательский тип комментария в службы аннотирования.



---

# Работа с библиотекой тегов и сервлетом контроллера Enterprise Information Portal

В Enterprise Information Portal входит библиотека тегов Java Server Pages и сервлет, которые можно использовать при создании JSP или сервлетов для программ Web. Использование библиотеки тегов снижает потребность в сценариях Java на страницах JSP, написанных с использованием JavaBeans EIP.

Эта библиотека тегов работает совместно с сервлетом, который может действовать как контроллер программы Web, разработанной по схеме модель-представление-контроллер, и выполняет инициализацию функций bean и другие действия.

---

## Установка библиотеки тегов и сервлета

Библиотеку тегов и сервлет необходимо установить на Web-сервер с IBM WebSphere Application Server и сконфигурировать этот Web-сервер, чтобы он использовал их. Информацию по установке и конфигурированию библиотеки тегов и сервлета, а также информацию о построении файлов WAR/EAR смотрите в книге *Планирование и установка Enterprise Information Portal*.

---

## Использование библиотеки тегов

В следующем примере JSP показано использование тега шаблонов поиска для получения списка шаблонов поиска:

```
<%@ taglib uri="cmb" prefix="cmb" %>
<%@ page import="com.ibm.mm.beans.*" %>
<jsp:useBean id="connection" scope="session"
 class="com.ibm.mm.beans.CMBConnection" />
<%
 CMBSchemaManagement schema = connection.getSchemaManagement();
 CMBSearchTemplate[] searchtemplates = schema.getSearchTemplate();
 request.setAttribute("searchtemplates",searchtemplates);
%>
<html>
<head>
<title>Search Templates Tag Test</title>
</head>

<body bgcolor="white">
<table border=2 cellpadding=3 cellspacing=3>
<tr>
<td>Доступные шаблоны поиска</td>
</tr>
<tr>
<td><cmb:searchtemplates>
</td>
</tr>
```

```

 <td><%= searchtemplate.getName() %></td>
 </tr>
</cmb:searchtemplates>
</table>
</body>
</html>

```

Директива `taglib` объявляет, что эта страница использует библиотеку тегов `EIP`, и связанный с ней префикс `cmb`. Затем вызывается тег `searchtemplates` и метод `getName()` возвращает имя каждого шаблона поиска.

## Соглашения, используемые в библиотеке тегов

У тегов JSP есть атрибуты для спецификации функций `bean`, которые они используют или генерируют. Если эти атрибуты не заданы, используются значения по умолчанию. Это необязательные параметры. Их значения берутся из локальных переменных и атрибутов в требовании и области действия сеанса. При использовании в сочетании с комплектом инструментов сервлетов локальные переменные, атрибуты требования или атрибуты сеанса содержат соответствующие умолчания. В Табл. 32 показаны функции `bean` по умолчанию и сферы действия, где они предполагаются или размещаются. Эти соглашения применяются и к сервлету; следуйте им и для других сервлетов, которые вы пишете при помощи библиотеки тегов.

Таблица 32. Соглашения библиотеки тегов

| Сфера действия       | Имя            | Тип                  | Описание                                                                           |
|----------------------|----------------|----------------------|------------------------------------------------------------------------------------|
| прикладная программа | connectionPool | CMBConnectionPool    | Bean пула соединений, совместно используемого сеансами                             |
| сеанс                | соединение     | CMBConnection        | Экземпляр <code>CMBConnection</code> для сеанса                                    |
| сеанс                | схема          | CMBSchemaManagement  | Bean управления схемой                                                             |
| сеанс                | data           | CMBDDataManagement   | Bean управления данными                                                            |
| сеанс                | user           | CMBUserManagement    | Bean управления пользователями                                                     |
| сеанс                | query          | CMBQuesryService     | Bean службы запросов                                                               |
| сеанс                | traceLog       | CMBTraceLog          | Bean журнала трассировки; все остальные bean посылают ему свои данные трассировки. |
| сеанс                | docservices    | CMBDDocumentServices | Bean служб документов                                                              |
| требование           | item           | CMBItem              | Последний обработанный элемент                                                     |

Таблица 32. Соглашения библиотеки тегов (продолжение)

| Сфера действия | Имя            | Тип               | Описание                                  |
|----------------|----------------|-------------------|-------------------------------------------|
| требование     | элементы       | CMBItem[ ]        | Последнее обработанное собрание элементов |
| требование     | searchTemplate | CMBSearchTemplate | Выбранный шаблон поиска                   |
| требование     | searchResults  | CMBSearchResults  | Результаты последнего поиска              |

## Сводка тегов

В следующих разделах дана сводка библиотеки тегов:

### Теги, связанные с соединениями

**<cmb:datasources connection="connection">datasource ... </cmb:datasources>**

Этот тег осуществляет перебор доступных источников данных.

*connection*

Задайте имя переменной типа CMBConnection, которая содержит соединение.

*datasource*

Строчная переменная, содержащая имя источника данных в виде строки.

### Теги, связанные с схемой

**<cmb:searchtemplates searchTemplates="searchTemplate"> ... </cmb:searchTemplates>**

Этот тег осуществляет перебор доступных шаблонов поиска.

*searchTemplates*

Укажите имя массива типа CMBSearchTemplate[], который будет содержать шаблоны поиска.

**searchTemplate**

Переменная типа CMBSearchTemplate, которая будет содержать шаблон поиска.

**<cmb:searchcriteria searchTemplate="searchtemplate">criteria ... </cmb:searchcriteria>**

Этот тег осуществляет перебор критериев поиска в шаблоне поиска.

*searchTemplate*

Задайте имя шаблона поиска.

### **criterion**

Переменная типа CMBSTCriterion, которая будет содержать критерий поиска.

**<cmb:displaycriteria searchTemplate="searchTemplate">criterion ...  
</cmb:displaycriteria>**

Этот тег осуществляет перебор критериев поиска, которые могут быть показаны для шаблона поиска.

*searchTemplate*

Задайте имя шаблона поиска.

### **criterion**

Переменная типа CMBSTCriterion, которая будет содержать критерий поиска.

**<cmb:allowedoperators criterion="criterion">operator ...  
</cmb:allowedoperators>**

Этот тег осуществляет перебор операций, допустимых для шаблона поиска.

*criterion*

Задайте имя переменной типа CMBSTCriterion, которая будет содержать критерий поиска.

### **операция**

Строка символов, которая содержит значение операции.

**<cmb:predefinedvalues criterion="criterion"> value... </cmb:predefinedvalues>**

Этот тег осуществляет перебор предопределенных значений критерия поиска.

*criterion*

Задайте имя переменной типа CMBSTCriterion, которая будет содержать критерий поиска.

**value** Строка символов, которая содержит предварительно определенное значение критерия поиска.

**<cmb:entities schema="schema">entity ... </cmb:entities>**

Этот тег осуществляет перебор доступных объектов объединения.

*schema* Задайте имя переменной типа CMBSchemaManagement, которая будет содержать схему.

**entity** Строка символов, которая содержит имя объекта.

**<cmb:attributes entity="entity" schema="schema">attribute ... </cmb:attributes>**

Этот тег осуществляет перебор атрибутов объединения в объекте объединения.

*schema* Задайте имя переменной типа CMBSchemaManagement, которая будет содержать схему.

**entity**    Задайте имя объекта.

*attribute*

Строка символов, содержащая имя переменной типа CMBAttribute, содержащей атрибут.

## Теги, связанные с поиском

**<cmb:searchresults searchresults="*searchResults*">item ... </cmb:searchresults>**

Этот тег осуществляет перебор результатов поиска.

*searchResults*

Задайте имя переменной типа CMBSearchResults, которая содержит результаты поиска.

**item**    Задайте имя переменной типа CMBItem, которая будет содержать объект из результатов поиска.

## Теги, связанные с элементом

**<cmb:itemattributes item="*item*"> attrname ... attrtype... attrvalue... </cmb:itemattributes>**

Этот тег осуществляет перебор атрибутов элемента.

*item*    Задайте имя переменной типа CMBItem, которая содержит элемент.

**attrname**

Строчная переменная, которая будет содержать имя атрибута.

**attrtype**

Строчная переменная, которая будет содержать тип атрибута.

**attrvalue**

Строчная переменная, которая будет содержать значение атрибута.

**<cmb:itemcontents " data="*data*" item="*item*"> content... </cmb:itemcontents>**

Этот тег осуществляет перебор содержания элемента.

*data*    Задайте имя переменной типа CMBDataManagement.

*item*    Задайте имя переменной типа CMBItem, которая содержит элемент.

**content**

Переменная типа CMBObject для содержимого объекта.

**<cmb:itemnotelogs " data="*data*" item="*item*">notelog ... </cmb:itemnotelogs>**

Этот тег осуществляет перебор журналов замечаний элемента.

*data*    Задайте имя переменной типа CMBDataManagement.

*item*    Задайте имя переменной типа CMBItem, которая содержит элемент.

**notelog**

Переменная типа CMBObject, которая будет содержать журнал замечаний объекта.

**<cmb:itemprivileges data="data" item="item">privilege ... </cmb:itemprivileges>**

Этот тег осуществляет перебор привилегий элемента.

*data*      Задайте имя переменной типа CMBDataManagement.

*item*      Задайте имя переменной типа CMBItem, которая содержит элемент.

**privilege**

Переменная типа CMBPrivilege, которая будет содержать привилегию объекта.

**<cmb:itemresources data="datamanagement" item="item"> resource... </cmb:itemresources>**

Этот тег осуществляет перебор ресурсов элемента.

*data*      Задайте имя переменной типа CMBDataManagement.

*item*      Задайте имя переменной типа CMBItem, которая содержит элемент.

**resource**

Переменная типа CMBResources, которая будет содержать ресурс объекта.

**<cmb:unmappeditem data="data" item="item">unmappeditem...</cmb:unmappeditem>**

Этот тег возвращает исходный элемент данного отображенного элемента.

*data*      Задайте имя переменной типа CMBDataManagement.

*item*      Задайте имя переменной типа CMBItem, которая содержит отображенный элемент.

**unmappeditem**

Переменная типа CMBItem, которая будет содержать неотображенный объект.

**<cmb:viewdata data="data " item="item">viewdata... </cmb:viewdata>**

Этот тег возвращает представление элемента.

*data*      Задайте имя переменной типа CMBDataManagement.

*item*      Задайте имя переменной типа CMBItem, которая содержит элемент.

**viewdata**

Переменная типа CMBViewData, которая будет содержать данные для вывода.



## Теги, связанные с папками

**<cmb:folderitems folder="folder"> item... </cmb:folderitems>**

Этот тег осуществляет перебор содержимого папки.

*folder*      Задайте имя переменной типа CMBItem, которая будет содержать содержимое папки.

**item**      Переменная типа CMBItem, которая представляет папку.

## Теги, связанные с документами

**<cmb:viewerdocuments docservices="docservices">document ... </cmb:viewerdocuments>**

Этот тег осуществляет перебор загруженных в данный момент документов.

*docservices*

Задайте имя переменной типа CMBDocumentServices.

**document**

Переменная типа CMBDocument, которая будет содержать документ.

**<cmb:documentpages document="document"> docpage... </cmb:documentpages>**

Этот тег осуществляет перебор страниц документа.

*document*

Задайте имя переменной типа CMBDocument, которая будет содержать документ.

**docPage**

Переменная типа CMBPage, которая будет содержать страницу.

---

## Сервлет контроллера EIP

EIP содержит сервлет с подключаемыми действиями, который можно использовать при построении прикладных программ Web. Этот сервлет действует как контроллер в программе Web, разработанной по схеме модель-представление-контроллер, выполняя действия и инициализируя функции bean (модель), к которым затем обращается JSP (представления), прямо или же косвенно через теги JSP.

Предлагаются действия для типичных прикладных задач:

- Регистрация и выход из системы.
- Поиск.
- Создание, получение, изменение и удаление документов.
- Создание папок, добавление документов в папку и удаление документов из папок.
- Запуск документов и страниц документов для их просмотра.

Кроме того, этот сервлет выполняет общие задачи перед действиями и после них, такие как управление соединением с контент-сервером. После каждого действия вызывается JSP для форматирования результатов и их отправки назад в браузер.

Вы можете настроить этот сервлет, чтобы добавить в него новые действия и связать с этими действиями JSP.

---

## Что может делать этот сервлет

Ниже перечислены свойства сервлета контроллера, которые вы можете использовать:

### Поддержка пула соединений

Сервлет контроллера использует пул соединений EIP для более эффективного управления соединениями. Соединение для сеанса может выделяться или в момент запроса, или в момент регистрации сеанса. Текущий пул соединений находится в области видимости прикладной программы.

### Регистрация сеансов с превышенным сроком ожидания

Если для сеанса превышен срок ожидания и сервлет получает запрос, на экран выводится JSP регистрации, позволяя пользователю повторить регистрацию. После успешной регистрации выполняется исходный запрос.

### Очистка при завершении сеанса

Этот сервлет выполняет соответствующую очистку при завершении сеанса - при его отключении от системы или при превышении срока ожидания. Это означает, что соединение разрушается или возвращается в пул соединений. Все другие функции bean EIP, созданные сервлетом, завершаются и их ресурсы освобождаются (не дожидаясь очередного цикла сборки мусора).

### Национальная версия

Этот сервлет гарантирует правильную настройку национальной версии для функций bean нижнего уровня, поэтому сообщения и символьные строки выводятся на национальном языке.

### Использование других наборов JSP

В файле свойств (его имя по умолчанию - `cmbServlet.jsp.properties`) описываются JSP, используемые для ответов на действия сервлета. Положение файла свойств задается параметром прикладной программы. Поэтому можно создать несколько различных прикладных программ Web, использующих разные наборы JSP.

### Расширение сервлета

Все известные сервлету действия определяются в файле свойств под

названием `cmbServlet.properties` (по умолчанию). Изменяя этот файл, можно добавить, изменить или удалить действия сервлета. Чтобы добавить новое действие:

1. Реализуйте класс, выполняющий это действие. Этот класс должен быть расширением класса `com.ibm.mm.servlets.CMBServletAction`.
2. Добавьте имя этого класса и имя действия в файл `cmbServlet.properties`. Используйте следующий синтаксис:  
`actions = список действий action.<имя_действия>.class = имя_класса`

В параметре `actions` перечисляются действия, известные сервлету. Для каждого действия в строке в файле свойств задается класс для этого действия. Например, чтобы добавить действие с именем `replay` из класса с именем `ReplayAction`:

```
actions = ... replay
action.replay.class = ReplayAction
```

Можно также заменить действие или задать собственное действие, выполняемое перед или после предопределенного действия.

Например, чтобы перед регистрацией выполнялось определенное вами действие дополнительной проверки:

```
action.logon.class = MyLogonAction com.ibm.mm.servlets.CMBLogonAction
```

Соглашение об именах предопределенных действий:

`com.ibm.mm.servlets.CMBдействиеAction`, где *действие* - имя действия с первой буквой в верхнем регистре.

## Обращение к сервлету

Для использования сервлета контроллера в прикладных программах применяются параметры программ, параметры запросов и файл свойств.

### Соглашения

Сервлет определяет следующие значения сеанса и запроса, которые могут использоваться в других JSP или сервлетах. Эти соглашения поддерживаются библиотекой тегов JSP. Эти соглашения совпадают с соглашениями для библиотеки тегов EIP.

### Параметры прикладной программы

Этот сервлет воспринимает следующие параметры прикладной программы (их можно также поместить в файл `cmbServlet.properties`).

| Параметр программы                | Значения | Описание                                           |
|-----------------------------------|----------|----------------------------------------------------|
| <code>servletPropertiesURL</code> | URL      | Положение файла <code>cmbServlet.properties</code> |

| Параметр программы | Значения                                 | Описание                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| defaultServerType  | Fed , ICM , OD, DL, DES, V4, IP, DD, ... | Информация регистрации по умолчанию. Вместе с параметрами defaultServer, defaultUserid и defaultPassword этот параметр может применяться при совместном использовании ID пользователя. Для регистрации вместо вывода страницы регистрации будет использоваться эта информация регистрации по умолчанию.         |
| defaultServer      |                                          | Информация регистрации по умолчанию.                                                                                                                                                                                                                                                                            |
| defaultUserid      |                                          | Информация регистрации по умолчанию.                                                                                                                                                                                                                                                                            |
| defaultPassword    |                                          | Информация регистрации по умолчанию.                                                                                                                                                                                                                                                                            |
| connectionpool     | логическое: true или false               | Разрешен ли пул соединений                                                                                                                                                                                                                                                                                      |
| maxfreeconnection  | целое                                    | Максимальное число соединений, доступных в пуле соединений.                                                                                                                                                                                                                                                     |
| minfreeconnection  | целое                                    | Минимальное число соединений, доступных в пуле соединений.                                                                                                                                                                                                                                                      |
| timeout            | целое                                    | Интервал (в миллисекундах), после которого бездействующее соединение будет разорвано и удалено.                                                                                                                                                                                                                 |
| noSessionPage      | URL                                      | Страница, выводимая для регистрации, когда сервлет запущен без установленного сеанса или соединения. Этот параметр можно использовать для запроса регистрации и возврата к исходному действию - тогда ссылки с закладками для обращения к EIP будут работать, даже если пользователь должен зарегистрироваться. |
| timedOutPage       | URL                                      | Страница, выводимая на экран, когда для сеанса истекает срок ожидания из-за его неактивности.                                                                                                                                                                                                                   |
| serverErrorPage    | URL                                      | Страница, выводимая на экран, когда возникает ошибка при обращении к серверу.                                                                                                                                                                                                                                   |

| Параметр программы    | Значения     | Описание                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| connectFailedPage     | URL          | Страница, выводимая на экран, когда возникает ошибка при соединении с сервером. В ней может выводиться запрос ввода правильных ID пользователя и пароля для этого сервера, после чего может быть произведена повторная попытка.                                                                                                                                                                                                                                  |
| tracelevel            | 0, 1 или 2   | <p>Задаёт уровень трассировки:</p> <ul style="list-style-type: none"> <li>• <b>0</b> - не сохранять информацию трассировки</li> <li>• <b>1</b> - сохранять информацию об исключительных ситуациях (значение по умолчанию)</li> <li>• <b>2</b> - сохранять информацию об исключительных ситуациях, сообщения оповещения, заголовки и атрибуты WebSphere Application Server, файлы ini EIP, свойства системы JVM, внутреннюю информацию трассировки EIP</li> </ul> |
| connectiontype        | 0, 1 или 2   | <p>Положение базы данных EIP и модулей времени выполнения контент-сервера:</p> <ul style="list-style-type: none"> <li>• <b>0</b> - локальное (по умолчанию)</li> <li>• <b>1</b> - удаленное</li> <li>• <b>2</b> - динамическое</li> </ul>                                                                                                                                                                                                                        |
| cmbsclient            | URL          | Положение файла cmbsclient.ini                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| cmbscs                | URL          | Положение файла cmbscs.ini                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| serviceconnectiontype | 0, 1 или 2   | <p>Положение модулей времени выполнения служб</p> <ul style="list-style-type: none"> <li>• <b>0</b> - локальное (по умолчанию)</li> <li>• <b>1</b> - удаленное</li> <li>• <b>2</b> - динамическое</li> </ul>                                                                                                                                                                                                                                                     |
| cmbsvclient           | URL          | Положение файла cmbsvclient.ini                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| cmbsvcs               | URL          | Положение файла cmbsvcs.ini                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| cmbcc2mime            | URL          | Положение файла cmbcc2mime.ini                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| cachedir              | имя каталога | Каталог для кэширования документов при преобразовании документов                                                                                                                                                                                                                                                                                                                                                                                                 |
| jnitrace              | имя файла    | Файл, в который сохраняется информация трассировки JNI для алгоритмов JNI, использованных при преобразовании документов (эта информация применяется IBM для диагностики)                                                                                                                                                                                                                                                                                         |

| Параметр программы    | Значения                          | Описание                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| conversion            | логическое: true <i>или</i> false | При значении true документы по возможности конвертируются в формат, который может быть выведен в браузере на системе промежуточного уровня. При значении false браузеру посылается исходный, непреобразованный документ.                                                                                                                                                                       |
| maxresults            | целое                             | Максимальное число возвращаемых результатов; значение -1 (значение по умолчанию) означает все результаты.                                                                                                                                                                                                                                                                                      |
| valuedelimiter        | символ                            | Определяет символ, разделяющий значения в критерии поиска. Значение по умолчанию зависит от национальной версии; для американского английского это запятая (,).                                                                                                                                                                                                                                |
| conversion.<mimetype> | <none   document   page >         | Опции преобразования для просмотра документов определенного типа mime. Они влияют на поведение сервлета viewDocument. page означает, что документ надо попытаться разбить на страницы. document означает, что документ надо преобразовать в вид, который сможет показать браузер. none означает, что преобразование выполнять не надо и документ должен быть возвращен в своей исходной форме. |
| nameseparator         | символ                            | Определяет символ, который будет отделять атрибут дочернего компонента от атрибута родительского компонента в специфицированных именах. Значение по умолчанию зависит от национальной версии; для американского английского это обратная дробная черта (/).                                                                                                                                    |

### Файл свойств

Сервлет ищет файл свойств `cmbervlet.properties`. В этом файле определяются действия, которые сервлет может использовать, включая действия, определенные в нем. В нем также определяются используемые файлы JSP.

Свойства сервлета можно также определить на сервере программ Web (механизме сервлетов). Используется тот же синтаксис, что и в файле свойств.

Содержимое `cmbServlet.properties` сохраняется сервлетом управления в объекте `Properties`. К нему можно обратиться при помощи атрибута прикладной программы `"cmbServletProperties"`, как показано в следующем примере.

```
// проверяем, разрешено ли объединение соединений в пул
String name = "connectionpool";
Properties props = (Properties) application.getAttribute
("cmbServletProperties");
String value = props.getProperty(name);
// "true" - разрешено, "false" - не разрешено
```

## Параметры требования

Сервлет воспринимает следующие параметры требования. Можно задавать дополнительные параметры, используемые в JSP ответа.

### *Общие*

#### **action=действие**

Выполняемое действие. Разрешены дополнительные параметры, зависящие от действия; они описаны ниже.

Это необязательный параметр. Если он не задан, страница ответа будет выполняться сервлетом после выполнения стандартных действий, таких как проверка соединения на превышение срока ожидания и регистрация.

#### **reply=<URL>**

(необязательный). Направляет на JSP, заданную в этом параметре, вместо JSP, определенной как ответ на действие в файле `cmbServlet.properties`. Если параметр действия не задан, а ответ задан, страница ответа будет выполняться сервлетом контроллера после выполнения стандартных действий, таких как проверка соединения на превышение срока ожидания и регистрация.

### *Относящиеся к соединению*

#### **action=logon serverType=<> server=<> userid=<> password=<> [connstring=<>] [configstring=<>]**

Регистрация на сервере. Нужно задать тип сервера, имя сервера, ID пользователя и пароль. Строку соединения и строку инициализации задавать не обязательно; они зависят от типа сервера.

#### **action=logoff [endSession=<true|false>]**

Отключение от сервера. По умолчанию также завершается сеанс.

### *Относящиеся к поиску*

**action=searchTemplate template=<> {<имя\_критерия>.op=<>  
<имя\_критерия>=<>}**

Выполнить поиск, используя заданные шаблон поиска и значения критерия поиска.

**action=searchEntity entity=<> {attribute.<имя\_атрибута>.op=<>  
attribute.<имя\_атрибута>=<>} [conjunction=<and|or>]**

Выполнить поиск, используя объект. Можно также задать значения атрибутов и операции. Если задаются несколько значений атрибутов, их отделяют разделителем значений, определенным в параметрах прикладной программы. При формировании запроса используется сочетание атрибутов с помощью операции and (по умолчанию) или or (операция задается параметром conjunction).

**action=searchQuery queryString=<>  
{queryParameter.<parametername>=<>}**

Выполнить поиск, используя заданную строку запроса. Синтаксис запроса зависит от сервера, на котором выполняется поиск.

Может быть задано любое число дополнительных параметров запроса. Они также зависят от типа сервера.

#### *Относящиеся к элементам*

**action=lock itemId=<>**

Блокировать элемент (обычно для монопольного доступа при изменении элемента).

**action=unlock itemId=<>**

Снять блокировку с заблокированного элемента.

**action=createItem type=<document|folder> entity=<>  
{attribute.<имя\_атрибута>=<значение\_атрибута>}**

Создать элемент. Если посылается содержимое, его также можно задать.

**action=retrieveItem itemId=<>**

Получить атрибуты и содержимое элемента. Может применяться, чтобы гарантировать использование самого нового содержимого, сохраненного на сервере.

**action=updateItem itemId=<> [entity=<>]  
{attribute.<имя\_атрибута>=<значение\_атрибута>}**

Изменить атрибуты элемента. Если задан объект, элемент переиндексируется. Содержимое также обновляется, если для обращения к сервлету используется команда post.

**action=deleteItem itemId=<>**

Удалить элемент.



**action=addContent itemId=<>**

Добавить часть содержимого в элемент. Посылаются данные содержимого.

**action=getContent itemId=<> contentIndex=<>**

Получает часть содержимого и возвращает ее браузеру.

**action=updateContent itemId=<> contentIndex=<>**

Изменить часть содержимого в элементе; посылаются данные содержимого. Если содержимое не существует, добавляется часть содержимого.

**action=deleteContent itemId=<> contentIndex=<>**

Удалить часть содержимого для указанного элемента.

**action=addNoteLog itemId=<>**

Изменить журнал примечаний для элемента. Посылается текст журнала примечаний.

**action=updateNoteLog itemId=<> notelogIndex=<>**

Изменить журнал примечаний для элемента; посылается текст журнала примечаний. Если журнал примечаний не существует, он добавляется.

**action=deleteNoteLog itemId=<> notelogIndex=<>**

Удалить текст журнала примечаний для элемента. Посылается текст журнала примечаний.

#### *Относящиеся к папкам*

**action=addItemToFolder itemId=<> folderId=<>**

Добавить указанный элемент в указанную папку.

**action=removeItemFromFolder itemId=<> folderId=<>**

Удалить указанный элемент из указанной папки.

#### *Относящиеся к документам*

**action=viewDocument itemId=<>**

Получить документ и вывести его в программе просмотра. Если документ разбит на страницы, это действие передается JSP, которая генерирует набор кадров программы просмотра. Если документ не разбит на страницы, это действие возвращает реальное содержимое документа.

**action=viewPage itemId=<> page=<> scale=<> rotation=<>  
annotations=<yes|no>**

Извлекает страницу документа.

## Матрица функций сервлетов комплекта инструментов

Таблица 33. Матрица функций сервлетов

| Действие             | CMv8 | CMv7 | VI/400 | IP/390 | OD/Wkstn | OD/390 | DD  |
|----------------------|------|------|--------|--------|----------|--------|-----|
| регистрация          | Да   | Да   | Да     | Да     | Да       | Да     | Да  |
| logoff               | Да   | Да   | Да     | Да     | Да       | Да     | Да  |
| searchTemplate       | Нет  | Нет  | Нет    | Нет    | Да       | Да     | Нет |
| searchEntity         | Да   | Да   | Да     | Да     | Да       | Да     | Нет |
| searchQuery          | Да   | Да   | Да     | Да     | Да       | Да     | Да  |
| lock                 | Да   | Да   | Да     | Да     | Нет      | Нет    | Нет |
| unlock               | Да   | Да   | Да     | Да     | Нет      | Нет    | Нет |
| createItem           | Да   | Да   | Нет    | Нет    | Нет      | Нет    | Нет |
| retrieveItem         | Да   | Да   | Да     | Да     | Да       | Да     | Да  |
| updateItem           | Да   | Да   | Да     | Да     | Нет      | Нет    | Нет |
| deleteItem           | Да   | Да   | Да     | Да     | Нет      | Нет    | Нет |
| addContent           | Да   | Да   | Нет    | Нет    | Нет      | Нет    | Нет |
| getContent           | Да   | Да   | Да     | Да     | Да       | Да     | Да  |
| updateContent        | Да   | Да   | Да     | Да     | Нет      | Нет    | Нет |
| deleteContent        | Да   | Да   | Да     | Да     | Нет      | Нет    | Нет |
| addNoteLog           | Да   | Да   | Да     | Нет    | Нет      | Нет    | Нет |
| updateNoteLog        | Да   | Да   | Да     | Нет    | Нет      | Нет    | Нет |
| deleteNoteLog        | Да   | Да   | Да     | Нет    | Нет      | Нет    | Нет |
| addItemToFolder      | Да   | Да   | Да     | Нет    | Нет      | Нет    | Нет |
| removeItemFromFolder | Да   | Да   | Да     | Нет    | Нет      | Нет    | Нет |
| viewDocument         | Да   | Да   | Да     | Да     | Да       | Да     | Да  |
| viewPage             | Да   | Да   | Да     | Да     | Да       | Да     | Да  |

Да - функция поддерживается Нет - функция не поддерживается **Примечание:** Действия logon, logoff, getContent, retrieveitem, viewDocument и viewPage поддерживаются на всех контент-серверах.

Таблица 34. Матрица функций сервлетов - продолжение

| Действие       | DES | DB2 | JDBC | IC  | Fed | FileNet |
|----------------|-----|-----|------|-----|-----|---------|
| регистрация    | Да  | Да  | Да   | Да  | Да  | Да      |
| logoff         | Да  | Да  | Да   | Да  | Да  | Да      |
| searchTemplate | Нет | Нет | Нет  | Нет | Да  | Нет     |
| searchEntity   | Нет | Да  | Да   | Нет | Да  | Нет     |

Таблица 34. Матрица функций сервлетов - продолжение (продолжение)

| Действие             | DES | DB2 | JDBC | IC  | Fed | FileNet |
|----------------------|-----|-----|------|-----|-----|---------|
| searchQuery          | Да  | Да  | Да   | Да  | Да  | Да      |
| lock                 | Нет | Нет | Нет  | Нет | Да  | Нет     |
| unlock               | Нет | Нет | Нет  | Нет | Да  | Нет     |
| createItem           | Нет | Да  | Да   | Нет | Нет | Нет     |
| retrieveItem         | Да  | Да  | Да   | Да  | Да  | Да      |
| updateItem           | Нет | Да  | Да   | Нет | Да  | Нет     |
| deleteItem           | Нет | Да  | Да   | Нет | Да  | Нет     |
| addContent           | Нет | Нет | Нет  | Нет | Да  | Нет     |
| getContent           | Да  | Да  | Да   | Да  | Да  | Да      |
| updateContent        | Нет | Нет | Нет  | Нет | Да  | Нет     |
| deleteContent        | Нет | Нет | Нет  | Нет | Да  | Нет     |
| addNoteLog           | Нет | Нет | Нет  | Нет | Да  | Нет     |
| updateNoteLog        | Нет | Нет | Нет  | Нет | Да  | Нет     |
| deleteNoteLog        | Нет | Нет | Нет  | Нет | Да  | Нет     |
| addItemToFolder      | Нет | Нет | Нет  | Нет | Нет | Нет     |
| removeItemFromFolder | Нет | Нет | Нет  | Нет | Нет | Нет     |
| viewDocument         | Да  | Да  | Да   | Да  | Да  | Да      |
| viewPage             | Да  | Да  | Да   | Да  | Да  | Да      |

**Да** - функция поддерживается **Нет** - функция не поддерживается **Примечание:** Действия logon, logoff, getContent, retrieveItem, viewDocument и viewPage поддерживаются на всех контент-серверах.



---

## Работа с исследованием информации

Этот раздел начинается с описания того, как построить прикладную программу с помощью функций bean исследования информации; далее следуют разделы, где описано построение собственного контент-провайдера, использование API служб исследования информации и работа с примером JSP.

---

### Построение прикладной программы исследования информации при помощи функций bean

Исследование информации позволяет создавать мощные программы на основе современных технологий анализа и исследования текстовой информации. Функции bean исследования информации обеспечивают:

- Составление сводок текста, то есть создание для документа краткой сводки
- Категоризацию, то есть определение категории документа
- Извлечение информации, то есть поиск и извлечение из документа нужных единиц информации
- Идентификация языка, то есть определение языка, на котором написан документ
- Кластеризация, то есть объединение сходных документов в собрание документов
- Текстовый поиск, который можно ограничивать документами отдельных категорий
- Доступ к документам, регулярно получаемым из Web с помощью искателя Web - программы IBM Web Crawler

#### Примечание

Перед использованием функций bean исследования информации надо понять различия между API служб и JavaBeans.

- JavaBeans Исследования информации - это программные компоненты для быстрой разработки прикладных программ, соответствующие требованиям JavaBeans. Некоторые элементы кода 'привинчиваются друг к другу' и не могут изменяться пользователем.
- API служб Java содержит все функции исследования информации в виде отдельных стандартных блоков для создания прикладных программ. Дополнительную информацию смотрите в книге "Использование API служб" на стр. 614.

Для возможности использования bean Исследование информации нужно установить и сконфигурировать компонент Enterprise Information Portal Исследование информации на компьютере, где будут запускаться эти функции.

Функции bean Исследование информации:

- **CMBSummarizationService**

Это невизуальная функция bean, которую можно использовать для создания краткой сводки документа. Ее можно запускать в трех различных режимах; можно задавать длину создаваемой сводки.

- **CMBCategorizationService**

Это невизуальная функция bean, которую можно использовать для определения категории документа. Она запускается на результатах, полученных с помощью Information Structuring Tool а именно, на созданной таксономии и метаданных, формулирующих условия, определенные как различительные для категории, и правила, описывающие каждую категорию. Документ может быть отнесен к одной или нескольким категориям. Результат категоризации содержит одну или несколько категорий и значения достоверности, указывающие, насколько хорошо соответствует документ каждой категории. Вы можете задать значение для этого показателя и максимальное число возвращаемых результатов.

- **CMBInformationExtractionService**

Это невизуальная функция bean, которую можно использовать для извлечения из документа имен, терминов и выражений.

- **CMBLanguageIdentificationService**

Это невизуальная функция bean, которую можно использовать для определения языка, на котором написан документ.

- **CMBClusteringService**

Это невизуальная функция bean, которую можно использовать для разбиения собрания документов на сходные наборы документов или кластеров.

- **CMBAdvancedSearchService**

CMBAdvancedSearchService - это невизуальная функция bean, выполняющая поиск. Сложный поиск выполняется только для элементов, хранящихся на складе данных исследования информации.

- **CMBCatalogService**

CMBCatalogService поддерживает постоянный склад данных исследования информации. Чтобы использовать эту службу, нужно создать каталог с помощью Information Structuring Tool. Эту функцию bean можно использовать для хранения результатов, создаваемых любыми службами исследования информации, и, конечно, для поиска и удаления этой информации.

- **CMBWebCrawlerService**

CMBWebCrawlerService следит за пространством Web, то есть за каталогом, куда Web crawler, запускаемый асинхронно, помещает все найденные

искателем документы HTML. Из этих файлов создаются объекты типа `CMBItem` (класс поддержки EIP, представляющий документ), которые можно затем использовать во всех службах исследования информации или просто импортировать на склад данных исследования информации при помощи `CMBCatalogService`.

- **CMBInfoMiningAdapter**

`CMBInfoMiningAdapter` - это переключатель между различными событиями исследования информации. Кроме того, эта функция bean поддерживает событие `CMBResultEvent`, которое обеспечивает работу функций bean Исследование информации совместно с другими функциями bean Enterprise Information Portal. `CMBInfoMiningAdapter` позволяет вам сочетать функции bean для построения разнообразных сценариев, как показано ниже на диаграмме.

На рис. 39 на стр. 564 показано использование функций bean исследования информации.

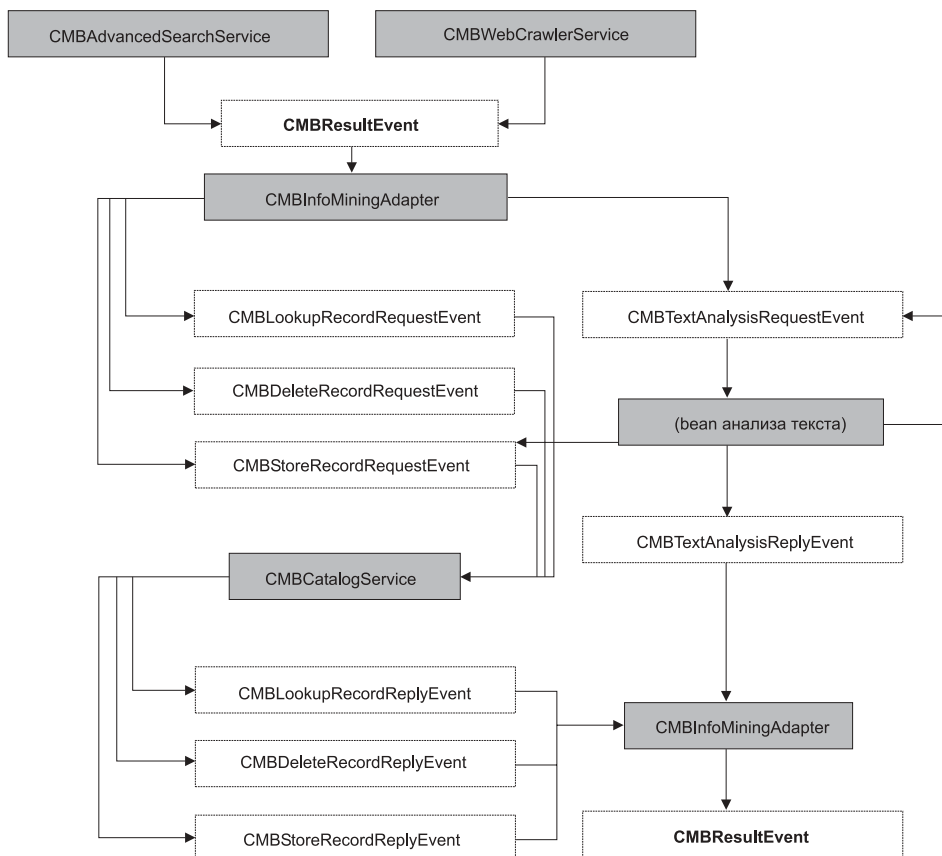


Рисунок 39. Функции bean исследования информации

В рис. 40 на стр. 565 перечислены функции bean текстового анализа.



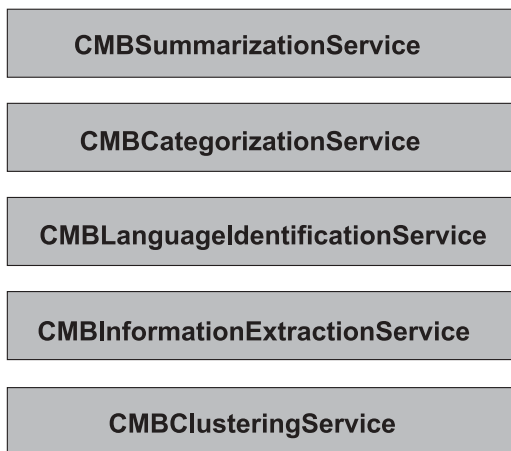


Рисунок 40. Функции bean текстового анализа

В следующих примерах показаны варианты использования функций bean исследования информации при построении прикладных программ:

- **Пример категоризации:** Сбор информации при подготовке к исследованию информации - обычная работа библиотекаря. Смотрите рис. 41 на стр. 567. В этом примере вы начинаете с простого поиска документов Enterprise Information Portal (EIP) на контент-сервере EIP. (Сбор документов в Web показан в примере Web Crawler.) Затем определяется язык документов, и документы на английском языке распределяются по категориям. Информация о категории хранится на складе метаданных.
- **Пример составления сводок:** Это другой типичный пример работы библиотекаря. Как и в примере категоризации, вы начинаете с простого поиска документов EIP на контент-сервере EIP, определяете документы на английском языке, а затем создаете для этих документов краткие сводки, сохраняя их на складе метаданных. Смотрите рис. 43 на стр. 577.
- **Пример извлечения информации:** Это этап исследования информации. В этом примере вы начинаете с простого поиска документов EIP на контент-сервере EIP. Затем из документов на английском языке извлекаются имена и термины. Извлеченная информация хранится на складе метаданных. Это показано на рис. 45 на стр. 584.
- **Пример кластеризации:** Это этап исследования информации. Вы начинаете с простого поиска документов EIP на контент-сервере EIP. Затем определяются документы на английском языке и вручную запускается служба кластеризации. Результаты выводятся на дисплей. На складе метаданных информация о кластеризации не сохраняется. Это показано на “Кластеризация” на стр. 591.
- **Пример Web Crawler:** Получение документов при помощи искателя Web и преобразование этой информации в доступную для поиска по отдельным

категориям (пример сложного поиска). Это тоже работа библиотекаря, как в примере категоризации и составления сводок, но здесь сбор информации выполняется в Интернете или внутренней сети, а не на контент-сервере EIP. Чтобы восстановить в результатах поиска категории и краткие сводки документов из каталога, документы нужно получать с помощью службы каталога.

Смотрите рис. 49 на стр. 597.

- **Пример сложного поиска:** Это этап исследования информации. Вы выполняете сложный поиск, позволяющий искать документы с помощью гибких запросов, которые ограничивают поиск конкретными категориями. Чтобы восстановить в результатах поиска категории и краткие сводки документов из каталога, документы нужно получать с помощью службы каталога.

## Положение файлов примеров

Файлы примеров, описанные в этой главе, поставляются в следующих каталогах:

| Пример | Положение |
|--------|-----------|
|--------|-----------|

|                                |  |
|--------------------------------|--|
| <b>Программа категоризации</b> |  |
|--------------------------------|--|

|  |                                                        |
|--|--------------------------------------------------------|
|  | <CMBROOT>\samples\java\beans\infomining\categorization |
|--|--------------------------------------------------------|

|                                     |  |
|-------------------------------------|--|
| <b>Программа составления сводок</b> |  |
|-------------------------------------|--|

|  |                                                       |
|--|-------------------------------------------------------|
|  | <CMBROOT>\samples\java\beans\infomining\summarization |
|--|-------------------------------------------------------|

|                                        |  |
|----------------------------------------|--|
| <b>Программа извлечения информации</b> |  |
|----------------------------------------|--|

|  |                                                               |
|--|---------------------------------------------------------------|
|  | <CMBROOT>\samples\java\beans\infomining\informationExtraction |
|--|---------------------------------------------------------------|

|                                |  |
|--------------------------------|--|
| <b>Программа кластеризации</b> |  |
|--------------------------------|--|

|  |                                                    |
|--|----------------------------------------------------|
|  | <CMBROOT>\samples\java\beans\infomining\clustering |
|--|----------------------------------------------------|

|                                  |  |
|----------------------------------|--|
| <b>Программа сложного поиска</b> |  |
|----------------------------------|--|

|  |                                                        |
|--|--------------------------------------------------------|
|  | <CMBROOT>\samples\java\beans\infomining\advancedSearch |
|--|--------------------------------------------------------|

|                              |  |
|------------------------------|--|
| <b>Программа Web Crawler</b> |  |
|------------------------------|--|

|  |                                                    |
|--|----------------------------------------------------|
|  | <CMBROOT>\samples\java\beans\infomining\webcrawler |
|--|----------------------------------------------------|

Каждый из этих каталогов содержит файл `compile.bat` для компиляции исходного кода. Кроме того, в каталогах программ примеров записан файл `run.bat` для запуска этих программы.

## Каталогизация документов

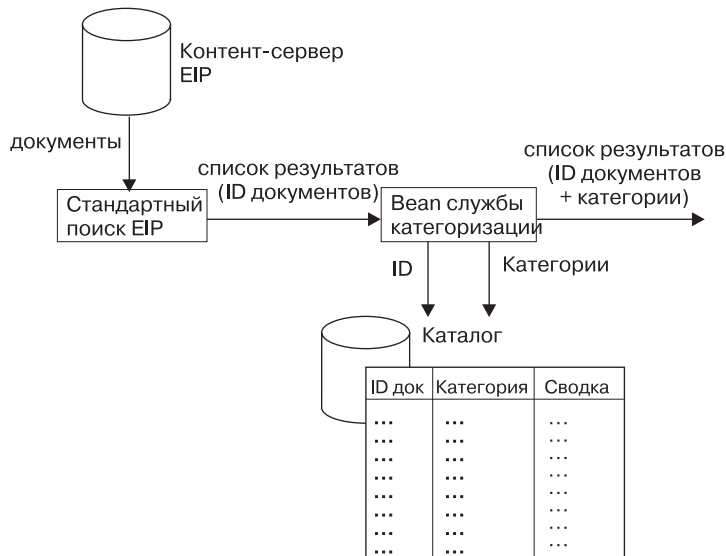


Рисунок 41. Пример каталогизации

Этот пример показывает, как каталогизировать документы, полученные в результате простого поиска. Результаты анализа сохраняются, а соответствующие документы становятся доступными для поиска по категориям.

На рис. 41 показан простой поиск документов, хранимых на контент-сервере EIP. Функция bean службы определения языка определяет язык, на котором написаны документы. Эта функция bean использует ID документов для получения содержимого документов и затем анализирует это содержимое для определения языка. Функция bean службы категоризации распределяет документы по категориям. Затем эта информация - ID документов и информация категорий - делается доступной для дальнейшей обработки.

В данном примере используются следующие функции bean:

- CMBConnection
- CMBQueryService (для простого поиска EIP)
- CMBSearchResults (для простого поиска EIP)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBCategorizationService
- CMBCatalogService

Программа этого примера:

1. Создает функции bean.
2. Настраивает функции bean.
3. Служба категоризации анализирует документы на английском языке. Результаты сохраняются в каталоге.
4. Запускает простой запрос EIP.
5. Выводит результаты поиска (список документов) и категории для проверки.

Ниже приводятся исходные коды Java.

### Полный исходный код Categorization.java

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBCategorizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class Categorization implements CMBResultListener, CMBExceptionListener
{
 String DEFAULT_CMDBNAME = "icmnlbdb";
 String DEFAULT_CMDBUSER = "icmadmin";
 String DEFAULT_CMDBPASSWORD = "password";
 String DEFAULT_SAMPDBNAME = "eipsampl";
 String DEFAULT_SAMPDBUSER = "icmadmin";
 String DEFAULT_SAMPDBPASSWORD = "password";

 String CATALOG_NAME = "Sample";
 String SEARCH_TEMPLATE = "SearchLongBySource";
 String SEARCH_VALUE = "ibmpress";
 String SEARCH_CRITERION = "source";

 public Categorization() throws Throwable
 {
```

```

// создание функций bean
CMBCConnection samp1Connection = new CMBCConnection();
CMBCConnection eipConnection = new CMBCConnection();
CMBQueryService queryService = new CMBQueryService();
CMBSearchResults searchResults = new CMBSearchResults();
CMBLanguageIdentificationService languageIdentificationService =
 new CMBLanguageIdentificationService();
CMBCategorizationService categorizationService = new CMBCategorizationService();
CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

// Чтение параметров
BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Sample database [" + DEFAULT_SAMPDBNAME + "] : ");
String sampDbName = din.readLine();
if (sampDbName.trim().equals("")) sampDbName = DEFAULT_SAMPDBNAME;

System.out.print("User ID for " + sampDbName +
 " [" + DEFAULT_SAMPDBUSER + "] : ");
String sampUserName = din.readLine();
if (sampUserName.trim().equals("")) sampUserName = DEFAULT_SAMPDBUSER;

System.out.print("Password for " + sampDbName +
 " [" + DEFAULT_SAMPDBPASSWORD + "] : ");
String sampPasswd = din.readLine();
if (sampPasswd.trim().equals("")) sampPasswd = DEFAULT_SAMPDBPASSWORD;

System.out.print("EIP database [" + DEFAULT_CMDBNAME + "] : ");
String eipDbName = din.readLine();
if (eipDbName.trim().equals("")) eipDbName = DEFAULT_CMDBNAME;

System.out.print("User ID for
 " + eipDbName + " [" + DEFAULT_CMDBUSER + "] : ");
String eipUserName = din.readLine();
if (eipUserName.trim().equals("")) eipUserName = DEFAULT_CMDBUSER;

System.out.print("Password for " + eipDbName +
 " [" + DEFAULT_CMDBPASSWORD + "] : ");
String eipPasswd = din.readLine();
if (eipPasswd.trim().equals("")) eipPasswd = DEFAULT_CMDBPASSWORD;

System.out.print("Catalog [" + CATALOG_NAME + "] : ");
String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG_NAME;

System.out.println("\n\nRunning Categorization with the
 following settings:");
System.out.println("Sample database = " + sampDbName);
System.out.println("User ID for " + sampDbName + " = " + sampUserName);
System.out.println("Password for " + sampDbName + " = " + sampPasswd);
System.out.println("EIP Database = " + eipDbName);
System.out.println("User ID for " + eipDbName + " = " + eipUserName);
System.out.println("Password for " + eipDbName + " = " + eipPasswd);

```

```

System.out.println("Catalog = " + catalog + "\n");

int key;
do {
 System.out.print("Continue (y/n)? ");
 InputStreamReader inReader = new InputStreamReader(System.in);
 key = inReader.read();
 if (key == 'n') System.exit(0); }
while (key != 'y');

// Настройка функций bean
samp1Connection.setServerName(sampDbName);
samp1Connection.setUserid(sampUserName);
samp1Connection.setPassword(sampPasswd);
samp1Connection.setConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);
samp1Connection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter1.setContentProvider
 (new CMBDefaultContentProvider());
adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// соединение функций bean
samp1Connection.addCMBConnectionReplyListener(queryService);
samp1Connection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter1);
eipConnection.addCMBConnectionReplyListener
 (languageIdentificationService);
eipConnection.addCMBConnectionReplyListener
 (categorizationService);
eipConnection.addCMBConnectionReplyListener(catalogService);
eipConnection.addCMBConnectionReplyListener(adapter2);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener
 (languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener
 (categorizationService);
categorizationService.addCMBStoreRecordRequestListener
 (catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);

```

```

adapter2.addCMBResultListener(this);

samlConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// запуск запроса
samlConnection.connect();
eipConnection.connect();
catalogService.setDefaultCategoryPath
 (catalogService.getTaxonomy().getRootCategory().getPathAsString());
CMBSchemaManagement schema = samplConnection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion
 (SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL, searchValues);

CMBSearchRequestEvent searchRequest = new
 CMBSearchRequestEvent(this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH,
 searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

samlConnection.disconnect();
eipConnection.disconnect();
}

// реализация com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
 if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
 return;

 Vector cmbItemVector = (Vector)e.getData();

 if(cmbItemVector == null)
 return;

 try {
 for(int i = 0; i < cmbItemVector.size(); i++)
 {
 CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

 CMBRecord currentRecord = currentItem.getInfoMiningRecord();

 System.out.println("\n\nPID : " + currentRecord.getPID());
 System.out.println("Categories : "
 + currentRecord.getValue("IKF_CATEGORIES"));
 } /* for */
 }
}

```

```

 catch (CMBNoSuchKeyException nske)
 { nske.printStackTrace();
 }
 }

 //реализация com.ibm.mm.beans.CMBExceptionListener
 public void onCMBException(CMBExceptionEvent e)
 {
 ((Exception)e.getData()).printStackTrace();
 }

 public static void main(String[] args)
 {
 try
 {
 new Categorization();
 System.exit(0);
 }
 catch(Throwable t)
 {
 t.printStackTrace();
 }
 }
}

```

### Создание функций bean

Чтобы выполнить поиск, нужно установить соединение с контент-сервером. Такое соединение может быть установлено с помощью функции bean `CMBConnection`. Для простого поиска EIP требуются функции bean `CMBQueryService` и `CMBSearchResults`. Язык документов определяется при помощи `CMBLanguageIdentificationService` и сохраняется в атрибуте `IKF_LANGUAGE` соответствующей записи. Документам присваивается одна или нескольких категорий с помощью функции `CMBCategorizationService`, которая также сохраняет информацию о категориях в соответствующих записях. `CMBCatalogService` используется для сохранения информации категорий элементов в каталоге и преобразования соответствующих документов в доступные для сложного поиска. Чтобы преобразовать события результатов поиска в события требований текстового анализа, а затем снова сохранить события ответов элементов в событиях результатов поиска, нужны два адаптера.

Требуемые функции bean создает следующий код:

```

CMBConnection samplConnection = new CMBConnection();
CMBConnection eipConnection = new CMBConnection();
CMBQueryService queryService = new CMBQueryService();
CMBSearchResults searchResults = new CMBSearchResults();
CMBLanguageIdentificationService languageIdentificationService =
 new CMBLanguageIdentificationService();
CMBCategorizationService categorizationService = new CMBCategorizationService();

```



```
CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();
```

### Настройка функций bean

Код, показанный в разделе настройки, нужно привести в соответствие с вашей установкой. Для функции bean соединения нужно указать имя контент-сервера соединения, ID пользователя и соответствующий пароль.

Прежде, чем запускать функций bean службы категоризации и службы каталога, их нужно привязать к существующему каталогу. Кроме того, для функции bean службы каталога необходима категория по умолчанию, которая будет назначаться элементам, не содержащим никакой информации категорий.

В этом примере настройку осуществляет следующий код:

```
samp1Connection.setServerName(sampDbName);
samp1Connection.setUserid(sampUserName);
samp1Connection.setPassword(sampPasswd);
samp1Connection.setConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);
samp1Connection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter1.setContentProvider
 (new CMBDefaultContentProvider());
adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);
```

### Соединение функций bean

На рис. 42 на стр. 574 показан поток событий по функциям bean.

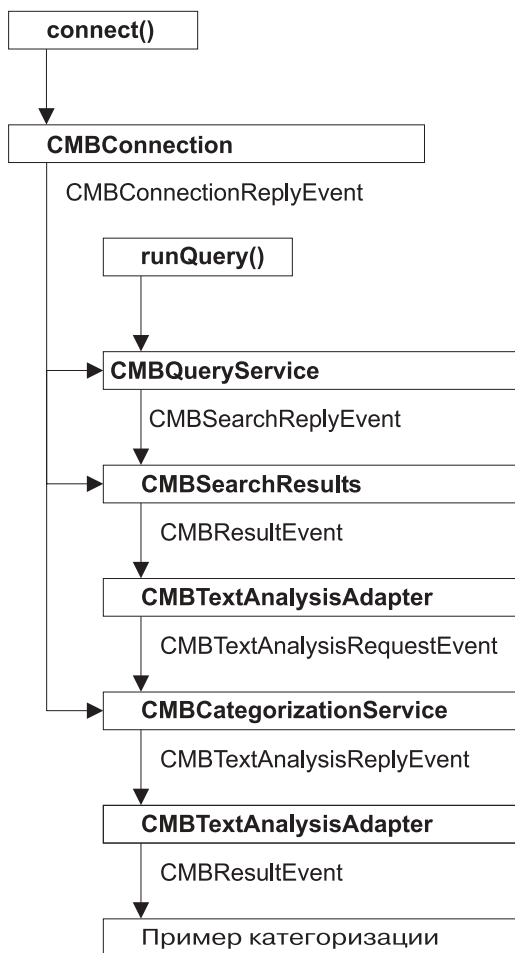


Рисунок 42. Пример каталогизации: Поток событий

Пять функций bean, используемых в этом примере, ожидают хэндл соединения от CMBConnectionReplyEvent. Функция bean CMBQueryService инициирует поиск, результатом которого служит событие, приводящее к запуску потока событий через другие функции bean.

Код соединения функций bean:

```

sampleConnection.addCMBConnectionReplyListener(queryService);
sampleConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter1);
eipConnection.addCMBConnectionReplyListener
 (languageIdentificationService);
eipConnection.addCMBConnectionReplyListener
 (categorizationService);

```

```

eipConnection.addCMBConnectionReplyListener/catalogService);
eipConnection.addCMBConnectionReplyListener(adapter2);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener
 (languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener
 (categorizationService);
categorizationService.addCMBStoreRecordRequestListener
 (catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);
adapter2.addCMBResultListener(this);

samlConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.
 addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

```

Поскольку исключительные ситуации тоже посылаются как события, класс Categorization должен обработать соответствующие события, реализовав интерфейс CMBExceptionListener, который соединяется с функциями bean для получения уведомления об исключительных ситуациях.

#### **Совет**

При анализе документов можно комбинировать функции bean каталогизации и составления сводок, сочетая их в любой последовательности. Как это сделать, описано в разделе “Импорт документов из пространства Web” на стр. 597.

### **Запуск запроса**

Для запуска запросов нужно установить соединение с контент-сервером, вызвав методы соединения функции bean CMBConnection:

```

samlConnection.connect();
eipConnection.connect();

```

Для запуска простого поиска EIP нужно задать шаблон поиска, критерий шаблона, операцию сравнения и, конечно, значение поиска.

Следующий код необходимо изменить в соответствии с вашей конфигурацией:

```

catalogService.setDefaultCategoryPath
 (catalogService.getTaxonomy().getRootCategory().getPathAsString());
CMBSchemaManagement schema = connection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL,
 searchValues);
CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent(this,
 CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

```

Чтобы закрыть текущее соединение, вызываются методы отсоединения:

```

samlConnection.disconnect();
eipConnection.disconnect();

```

### **Вывод результатов анализа текста**

В классе Categorization используется интерфейс CMBResultListener для вывода списка документов, найденных во время поиска, и для вывода информации о категориях каждого документа. Событие CMBResultEvent, полученное как аргумент для метода CMBResult, содержит вектор объектов CMBItem, где каждый объект CMBItem представляет документ:

```

Vector cmbItemVector = (Vector)e.getData();

```

В объект CMBItem встраивается постоянный ID документа:

```

CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);
CMBRecord currentRecord = currentItem.getInfoMiningRecord();
System.out.println("\n\nPID : " + currentRecord.getPID());

```

и результаты анализа текста:

```

System.out.println("Categories : " + currentRecord.getValue("IKF_CATEGORIES"));

```

## Составление сводок документов

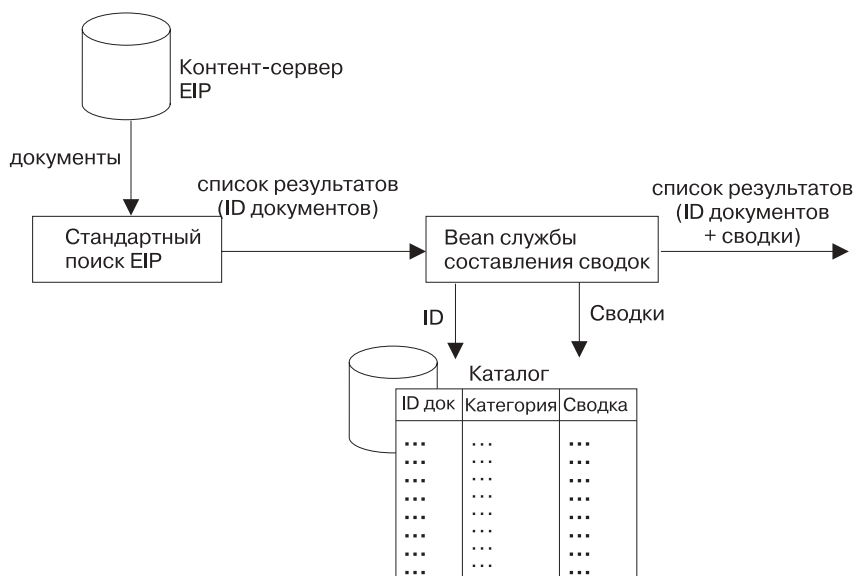


Рисунок 43. Пример составления сводок

В этом примере показано, как можно составлять сводки документов, полученных простым поиском EIP. Результаты анализа сохраняются, а соответствующие документы преобразуются в доступные для сложного поиска.

На рис. 43 показан простой поиск документов, хранимых на контент-сервере EIP. Функция bean службы определения языка определяет язык, на котором написаны документы. Эта функция bean использует ID документов для получения содержимого документов и затем анализирует это содержимое для определения языка. Служба составления сводок bean, используя ID документов на английском языке, получает содержимое документов, найденных с помощью контент-провайдера, и создает их краткие сводки. Затем служба каталога bean сохраняет сводки на складе метаданных. Затем эта информация, ID документов и сводки преобразуются в доступные для дальнейшей обработки. В данном примере используются следующие функции bean:

- CMBConnection
- CMBQueryService (для выполнения простого поиска EIP)
- CMBSearchResults (для выполнения простого поиска EIP)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBSummarizationService
- CMBCatalogService

## Полный исходный код Summarization.java

Вот полный исходный код примера составления сводок. Поскольку способ действия функций bean составления сводок аналогичен описанному для функций bean категоризации, подробности можно посмотреть в разделе “Каталогизация документов” на стр. 567.

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBSummarizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class Summarization implements CMBResultListener, CMBExceptionListener
{
 String DEFAULT_CMDBNAME = "icmnlbdb";
 String DEFAULT_CMDBUSER = "icmadmin";
 String DEFAULT_CMDBPASSWORD = "password";
 String DEFAULT_SAMPDBNAME = "eipsampl";
 String DEFAULT_SAMPDBUSER = "icmadmin";
 String DEFAULT_SAMPDBPASSWORD = "password";

 String CATALOG_NAME = "Sample";
 String SEARCH_TEMPLATE = "SearchLongBySource";
 String SEARCH_VALUE = "ibmpress";
 String SEARCH_CRITERION = "source";

 public Summarization() throws Throwable
 {
 // создание функций bean
 CMBConnection samplConnection = new CMBConnection();
 CMBConnection eipConnection = new CMBConnection();
 CMBQueryService queryService = new CMBQueryService();
 CMBSearchResults searchResults = new CMBSearchResults();
 CMBLanguageIdentificationService languageIdentificationService =
 new CMBLanguageIdentificationService();
 CMBSummarizationService summarizationService = new CMBSummarizationService();
 }
}
```

```

CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

// чтение параметров
BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Sample database [" + DEFAULT_SAMPDBNAME + "] : ");
String sampDbName = din.readLine();
if (sampDbName.trim().equals("")) sampDbName = DEFAULT_SAMPDBNAME;

System.out.print("User ID for " + sampDbName +
 " [" + DEFAULT_SAMPDBUSER + "] : ");
String sampUserName = din.readLine();
if (sampUserName.trim().equals("")) sampUserName = DEFAULT_SAMPDBUSER;

System.out.print("Password for " + sampDbName +
 " [" + DEFAULT_SAMPDBPASSWORD + "] : ");
String sampPasswd = din.readLine();
if (sampPasswd.trim().equals("")) sampPasswd = DEFAULT_SAMPDBPASSWORD;

System.out.print("EIP database [" + DEFAULT_CMDBNAME + "] : ");
String eipDbName = din.readLine();
if (eipDbName.trim().equals("")) eipDbName = DEFAULT_CMDBNAME;

System.out.print("User ID for " + eipDbName
 + " [" + DEFAULT_CMDBUSER + "] : ");
String eipUserName = din.readLine();
if (eipUserName.trim().equals("")) eipUserName = DEFAULT_CMDBUSER;

System.out.print("Password for " + eipDbName +
 " [" + DEFAULT_CMDBPASSWORD + "] : ");
String eipPasswd = din.readLine();
if (eipPasswd.trim().equals("")) eipPasswd = DEFAULT_CMDBPASSWORD;

System.out.print("Catalog [" + CATALOG_NAME + "] : ");
String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG_NAME;

System.out.println("\n\nRunning Summarization with the following settings:");
System.out.println("Sample database = " + sampDbName);
System.out.println("User ID for " + sampDbName + " = " + sampUserName);
System.out.println("Password for " + sampDbName + " = " + sampPasswd);
System.out.println("EIP Database = " + eipDbName);
System.out.println("User ID for " + eipDbName + " = " + eipUserName);
System.out.println("Password for " + eipDbName + " = " + eipPasswd);
System.out.println("Catalog = " + catalog + "\n");

int key;
do {
 System.out.print("Continue (y/n)? ");
 InputStreamReader inReader = new InputStreamReader(System.in);
 key = inReader.read();
 if (key == 'n') System.exit(0); }
while (key != 'y');

```

```

// Настройка функций bean
samlConnection.setServerName(sampDbName);
samlConnection.setUserid(sampUserName);
samlConnection.setPassword(sampPasswd);
samlConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samlConnection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter1.setContentProvider(new CMBDefaultContentProvider());
adapter1.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// соединение функций bean
samlConnection.addCMBConnectionReplyListener(queryService);
samlConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter1);
eipConnection.addCMBConnectionReplyListener(languageIdentificationService);
eipConnection.addCMBConnectionReplyListener(summarizationService);
eipConnection.addCMBConnectionReplyListener(catalogService);
eipConnection.addCMBConnectionReplyListener(adapter2);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener(summarizationService);
summarizationService.addCMBStoreRecordRequestListener(catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);

adapter2.addCMBResultListener(this);

samlConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// запуск запроса
samlConnection.connect();
eipConnection.connect();

```



```

catalogService.setDefaultCategoryPath(catalogService.getTaxonomy().
 getRootCategory().getPathAsString());

CMBSchemaManagement schema = samplConnection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL,
 searchValues);

CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent
 (this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH,
 searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

samplConnection.disconnect();
eipConnection.disconnect();
}

// реализация com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
 if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
 return;

 Vector cmbItemVector = (Vector)e.getData();

 if(cmbItemVector == null)
 return;

 try {
 for(int i = 0; i < cmbItemVector.size(); i++)
 {
 CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

 CMBRecord currentRecord = currentItem.getInfoMiningRecord();

 System.out.println("\n\nPID : " + currentRecord.getPID());
 System.out.println("Summary : " + currentRecord.getValue("IKF_SUMMARY"));
 } /* for */
 }
 catch (CMBNoSuchKeyException nske)
 {
 nske.printStackTrace();
 }
}

//реализация com.ibm.mm.beans.CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
 ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
 try
 {

```

```
 new Summarization();
 System.exit(0);
 }
 catch(Throwable t)
 {
 t.printStackTrace();
 }
}
```

Ниже показан поток событий по функциям bean.

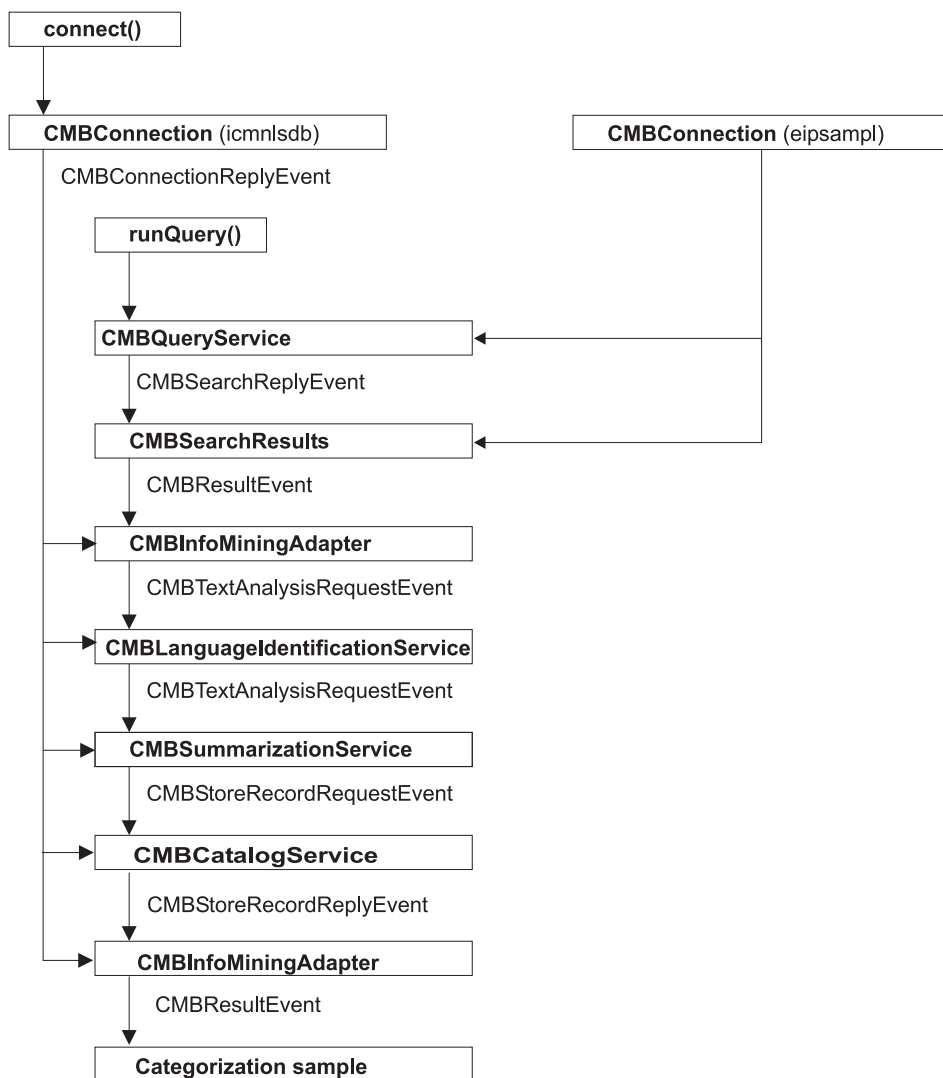


Рисунок 44. Пример составления сводок: Поток событий

## Извлечение информации

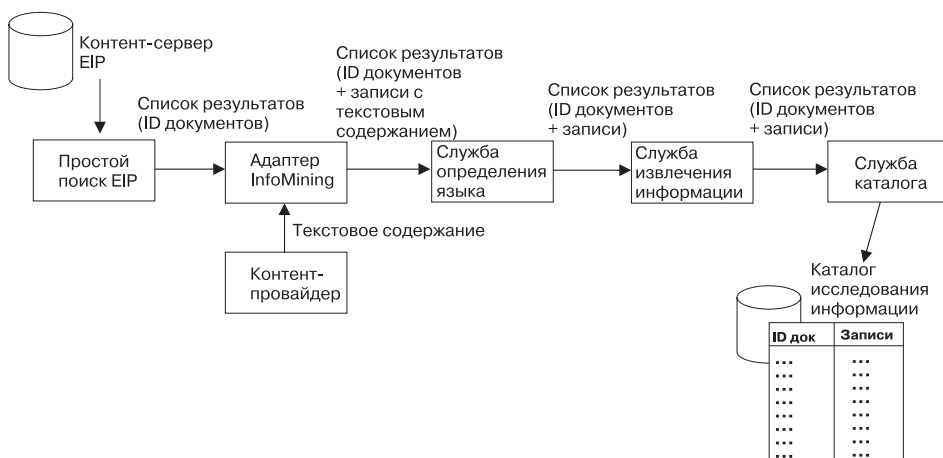


Рисунок 45. Пример извлечения информации

Этот пример показывает, как извлечь информацию, например, имена, термины и выражения, из документов, полученные в результате простого поиска.

Результаты анализа сохраняются и могут использоваться для получения связанных документов или извлечения важной информации из отдельных документов.

На рис. 45 показан простой поиск документов, хранимых на контент-сервере EIP. Функция bean службы определения языка определяет язык, на котором написаны документы. Эта функция bean использует ID документов для получения содержимого документов и затем анализирует это содержимое для определения языка. Функция bean службы извлечения информации использует документы на английском языке для извлечения информации из этих документов и сохраняет результаты на складе метаданных. Затем эта информация - ID документов и извлеченная информация - делается доступной для дальнейшей обработки.

В данном примере используются следующие функции bean:

- CMBCConnection
- CMBQueryService (для простого поиска EIP)
- CMBSearchResults (для простого поиска EIP)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService

- CMBInformationExtractionService
- CMBCatalogService

Программа этого примера:

1. Создает функции bean.
2. Настраивает функции bean.
3. Соединяет функции bean, чтобы служба идентификации языка могла анализировать документы. Результаты сохраняются в каталоге.
4. Служба извлечения информации анализирует документы. Результаты сохраняются в каталоге.
5. Выводит результаты поиска (список документов) и извлеченную информацию для проверки.

Ниже приводятся исходные коды Java. Поскольку способ действия функций bean извлечения информации аналогичен описанному для функций bean категоризации, подробности можно посмотреть в разделе “Каталогизация документов” на стр. 567.

### Полный исходный код InformationExtraction.java

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBInformationExtractionService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class InformationExtraction implements
 CMBResultListener, CMBExceptionListener
{
 String DEFAULT_CMDBNAME = "icmnlsdb";
 String DEFAULT_CMDBUSER = "icmadmin";
```

```

String DEFAULT_CMDBPASSWORD = "password";
String DEFAULT_SAMPDBNAME = "eipsampl";
String DEFAULT_SAMPDBUSER = "icmadmin";
String DEFAULT_SAMPDBPASSWORD = "password";

String CATALOG_NAME = "Sample";
String SEARCH_TEMPLATE = "SearchLongBySource";
String SEARCH_VALUE = "ibmpress";
String SEARCH_CRITERION = "source";

public InformationExtraction() throws Throwable
{
 // создание функций bean
 CMBConnection samplConnection = new CMBConnection();
 CMBConnection eipConnection = new CMBConnection();
 CMBQueryService queryService = new CMBQueryService();
 CMBSearchResults searchResults = new CMBSearchResults();
 CMBLanguageIdentificationService languageIdentificationService =
 new CMBLanguageIdentificationService();
 CMBInformationExtractionService informationExtractionService =
 new CMBInformationExtractionService();
 CMBCatalogService catalogService = new CMBCatalogService();
 CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
 CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

 // Чтение параметров
 BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

 System.out.print("Sample database [" + DEFAULT_SAMPDBNAME + "] : ");
 String sampDbName = din.readLine();
 if (sampDbName.trim().equals("")) sampDbName = DEFAULT_SAMPDBNAME;

 System.out.print("User ID for " + sampDbName +
 " [" + DEFAULT_SAMPDBUSER + "] : ");
 String sampUserName = din.readLine();
 if (sampUserName.trim().equals("")) sampUserName = DEFAULT_SAMPDBUSER;

 System.out.print("Password for " + sampDbName +
 " [" + DEFAULT_SAMPDBPASSWORD + "] : ");
 String sampPasswd = din.readLine();
 if (sampPasswd.trim().equals("")) sampPasswd = DEFAULT_SAMPDBPASSWORD;

 System.out.print("EIP database [" + DEFAULT_CMDBNAME + "] : ");
 String eipDbName = din.readLine();
 if (eipDbName.trim().equals("")) eipDbName = DEFAULT_CMDBNAME;

 System.out.print("User ID for " + eipDbName +
 " [" + DEFAULT_CMDBUSER + "] : ");
 String eipUserName = din.readLine();
 if (eipUserName.trim().equals("")) eipUserName = DEFAULT_CMDBUSER;

 System.out.print("Password for " + eipDbName +
 " [" + DEFAULT_CMDBPASSWORD + "] : ");
 String eipPasswd = din.readLine();
 if (eipPasswd.trim().equals("")) eipPasswd = DEFAULT_CMDBPASSWORD;
}

```

```

System.out.print("Catalog [" + CATALOG_NAME + "] : ");
String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG_NAME;

System.out.println("\n\nRunning InformationExtraction
 with the following settings:");
System.out.println("Sample database = " + sampDbName);
System.out.println("User ID for " + sampDbName + " = " + sampUserName);
System.out.println("Password for " + sampDbName + " = " + sampPasswd);
System.out.println("EIP Database = " + eipDbName);
System.out.println("User ID for " + eipDbName + " = " + eipUserName);
System.out.println("Password for " + eipDbName + " = " + eipPasswd);
System.out.println("Catalog = " + catalog + "\n");

int key;
do {
 System.out.print("Continue (y/n)? ");
 InputStreamReader inReader = new InputStreamReader(System.in);
 key = inReader.read();
 if (key == 'n') System.exit(0); }
while (key != 'y');

// Настройка функций bean
samlConnection.setServerName(sampDbName);
samlConnection.setUserid(sampUserName);
samlConnection.setPassword(sampPasswd);
samlConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samlConnection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter1.setContentProvider(new CMBDefaultContentProvider());
adapter1.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// соединение функций bean
samlConnection.addCMBConnectionReplyListener(queryService);
samlConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter1);
eipConnection.addCMBConnectionReplyListener
 (languageIdentificationService);
eipConnection.addCMBConnectionReplyListener
 (informationExtractionService);
eipConnection.addCMBConnectionReplyListener(catalogService);
eipConnection.addCMBConnectionReplyListener(adapter2);

```

```

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener
 (informationExtractionService);
informationExtractionService.addCMBStoreRecordRequestListener
 (catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);

adapter2.addCMBResultListener(this);

samlConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
informationExtractionService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// запуск запроса
samlConnection.connect();
eipConnection.connect();
catalogService.setDefaultCategoryPath(catalogService.getTaxonomy().
 getRootCategory().getPathAsString());

CMBSchemaManagement schema = samplConnection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION,
 CMBBaseConstant.CMB_OP_EQUAL, searchValues);

CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent(this,
 CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

samlConnection.disconnect();
eipConnection.disconnect();
}

// реализация com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
 if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
 return;

 Vector cmbItemVector = (Vector)e.getData();

 if(cmbItemVector == null)
 return;

 try {
 for(int i = 0; i < cmbItemVector.size(); i++)

```



```

 {
 CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

 CMBRecord currentRecord = currentItem.getInfoMiningRecord();

 System.out.println("\n\nPID : " + currentRecord.getPID());
 System.out.println("Features : " + currentRecord.getValue("IKF_FEATURES"));
 } /* for */
 }
 catch (CMBNoSuchKeyException nske)
 {
 nske.printStackTrace();
 }
}

// реализация com.ibm.mm.beans.CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
 ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
 try
 {
 new InformationExtraction();
 System.exit(0);
 }
 catch(Throwable t)
 {
 t.printStackTrace();
 }
}
}

```

На рис. 46 на стр. 590 показан поток событий по функциям bean.

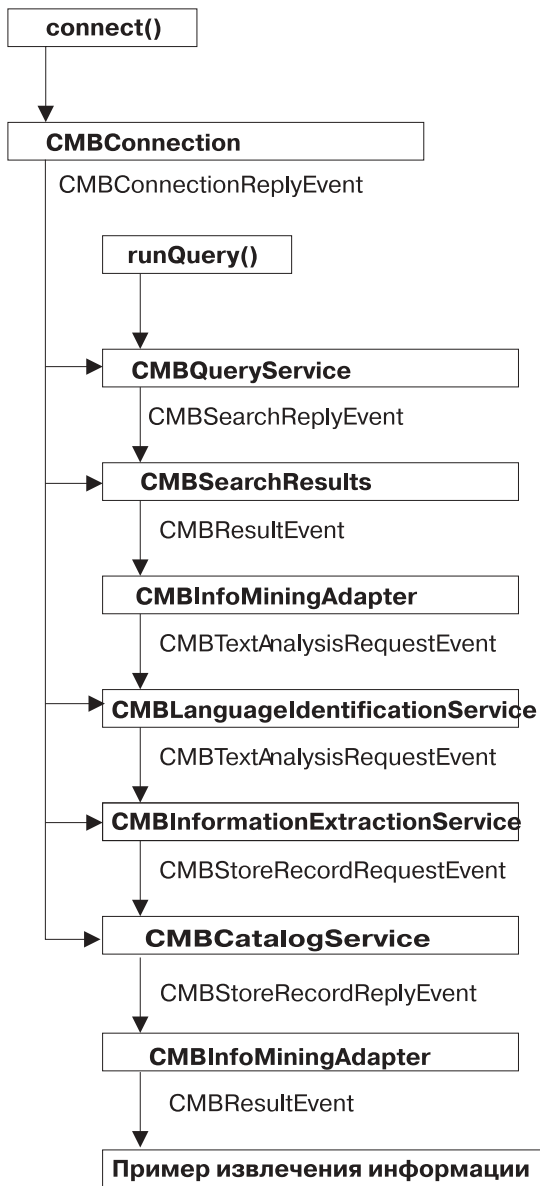


Рисунок 46. Пример извлечения информации: Поток событий

## Кластеризация

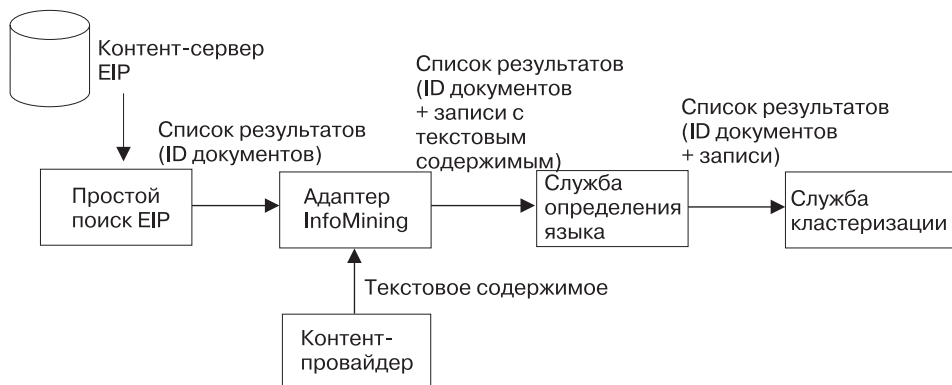


Рисунок 47. Пример кластеризации

В этом примере показано, как можно кластеризовать документы, полученные простым поиском EIP. Перед кластеризацией необходимо идентифицировать язык документов. Служба кластеризации только собирает автоматически документы, возвращенные поиском EIP. Сам процесс кластеризации надо запустить вручную, вызвав метод `cluster()`.

В данном примере используются следующие функции bean:

- `CMBConnection`
- `CMBQueryService` (для простого поиска EIP)
- `CMBSearchResults` (для простого поиска EIP)
- `CMBInfoMiningAdapter`
- `CMBLanguageIdentificationService`
- `CMBClusteringService`

Программа этого примера:

1. Создает функции bean.
2. Настраивает функции bean.
3. Соединяет функции bean, чтобы служба идентификации языка могла анализировать документы и идентифицировать язык документов, а служба кластеризации могла собирать документы, подготовленные для последующей кластеризации.

4. Запускает кластеризацию, вызывая метод `cluster()`. Затем результат (класса `CMBClusterResult`) выводится на экран в удобной для чтения форме.

Функции bean для службы кластеризации аналогичны другим функциям bean исследования информации за исключением того, что:

- Служба кластеризации только собирает документы, возвращенные поиском EIP. Сам процесс кластеризации надо запустить вручную.
- Каталог не предназначен для хранения результатов кластеризации, и поэтому служба каталога в этом сценарии не нужна.

Дополнительную информацию о работе функций bean смотрите в “Каталогизация документов” на стр. 567.

Ниже приводятся исходные коды Java.

### Полный исходный код `Clustering.java`

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBClusteringService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBClusterResult;
import com.ibm.mm.beans.infomining.CMBClusterNode;

public class Clustering implements CMBResultListener,
 CMBExceptionListener
{
 String DEFAULT_CMDBNAME = "icmnlbdb";
 String DEFAULT_CMDBUSER = "icmadmin";
 String DEFAULT_CMDBPASSWORD = "password";
 String DEFAULT_SAMPDBNAME = "eipsampl";
 String DEFAULT_SAMPDBUSER = "icmadmin";
 String DEFAULT_SAMPDBPASSWORD = "password";

 String CATALOG_NAME = "Sample";
 String SEARCH_TEMPLATE = "SearchLongBySource";
```

```

 String SEARCH_VALUE = "ibmpress";
 String SEARCH_CRITERION = "source";

public Clustering() throws Throwable
{
 // создание функций bean
 CMBCConnection samp1Connection = new CMBCConnection();
 CMBCConnection eipConnection = new CMBCConnection();
 CMBQueryService queryService = new CMBQueryService();
 CMBSearchResults searchResults = new CMBSearchResults();
 CMBLanguageIdentificationService languageIdentificationService =
 new CMBLanguageIdentificationService();
 CMBClusteringService clusteringService = new CMBClusteringService();
 CMBInfoMiningAdapter adapter = new CMBInfoMiningAdapter();

 // Чтение параметров
 BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

 System.out.print("Sample database [" + DEFAULT_SAMPDBNAME + "] : ");
 String sampDbName = din.readLine();
 if (sampDbName.trim().equals("")) sampDbName = DEFAULT_SAMPDBNAME;

 System.out.print("User ID for " + sampDbName +
 " [" + DEFAULT_SAMPDBUSER + "] : ");
 String sampUserName = din.readLine();
 if (sampUserName.trim().equals("")) sampUserName = DEFAULT_SAMPDBUSER;

 System.out.print("Password for " + sampDbName +
 " [" + DEFAULT_SAMPDBPASSWORD + "] : ");
 String sampPasswd = din.readLine();
 if (sampPasswd.trim().equals("")) sampPasswd = DEFAULT_SAMPDBPASSWORD;

 System.out.print("EIP database [" + DEFAULT_CMDBNAME + "] : ");
 String eipDbName = din.readLine();
 if (eipDbName.trim().equals("")) eipDbName = DEFAULT_CMDBNAME;

 System.out.print("User ID for " + eipDbName +
 " [" + DEFAULT_CMDBUSER + "] : ");
 String eipUserName = din.readLine();
 if (eipUserName.trim().equals("")) eipUserName = DEFAULT_CMDBUSER;

 System.out.print("Password for " + eipDbName +
 " [" + DEFAULT_CMDBPASSWORD + "] : ");
 String eipPasswd = din.readLine();
 if (eipPasswd.trim().equals("")) eipPasswd = DEFAULT_CMDBPASSWORD;

 System.out.print("Catalog [" + CATALOG_NAME + "] : ");
 String catalog = din.readLine();
 if (catalog.trim().equals("")) catalog = CATALOG_NAME;

 System.out.println("\n\nRunning Clustering with the following settings:");
 System.out.println("Sample database = " + sampDbName);
 System.out.println("User ID for " + sampDbName + " = " + sampUserName);
 System.out.println("Password for " + sampDbName + " = " + sampPasswd);
 System.out.println("EIP Database = " + eipDbName);

```

```

System.out.println("User ID for " + eipDbName + " = " + eipUserName);
System.out.println("Password for " + eipDbName + " = " + eipPasswd);
System.out.println("Catalog = " + catalog + "\n");

int key;
do {
 System.out.print("Continue (y/n)? ");
 InputStreamReader inReader = new InputStreamReader(System.in);
 key = inReader.read();
 if (key == 'n') System.exit(0); }
while (key != 'y');

// настройка функций bean
samp1Connection.setServerName(sampDbName);
samp1Connection.setUserid(sampUserName);
samp1Connection.setPassword(sampPasswd);
samp1Connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samp1Connection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

eipConnection.setServerName(eipDbName);
eipConnection.setUserid(eipUserName);
eipConnection.setPassword(eipPasswd);
eipConnection.setConnectToIKF(true);
eipConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
eipConnection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);

adapter.setContentProvider(new CMBDefaultContentProvider());
adapter.setCatalogName(catalog);
clusteringService.setMinClusterCount(2);
clusteringService.setMaxClusterCount(6);
clusteringService.setClusterFeatureCount(5);

// соединение функций bean
samp1Connection.addCMBConnectionReplyListener(queryService);
samp1Connection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter);
eipConnection.addCMBConnectionReplyListener(languageIdentificationService);
eipConnection.addCMBConnectionReplyListener(clusteringService);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter);
adapter.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener(clusteringService);

samp1Connection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
clusteringService.addCMBExceptionListener(this);
adapter.addCMBExceptionListener(this);

// запуск запроса
samp1Connection.connect();

```

```

eipConnection.connect();

CMBSchemaManagement schema = samplConnection.getSchemaManagement();
CMBSearchTemplate searchTemplate =
 schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION,
 CMBBaseConstant.CMB_OP_EQUAL, searchValues);

CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent
 (this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
queryService.onCMBSearchRequest(searchRequest);

 // запуск кластеризации и вывод результатов
 CMBClusterResult clusterResult = clusteringService.cluster();

 System.out.println(clusterResult.getClusterCount() +
 " clusters found for " +
 clusterResult.getDocumentCount() +
 " documents:");
 CMBClusterNode[] clusterNodes = clusterResult.getClusterNodes();
 for(int i = 0; i < clusterNodes.length; i++) {
 CMBClusterNode node = clusterNodes[i];
 String[] features = node.getClusterFeatures();
 String[] names = node.getDocumentNames();
 String label = node.getLabel();
 System.out.println("Cluster " + label);
 System.out.print(" Cluster Features : ");
 for(int j = 0; j < features.length; j++) System.out.print(features[j] + " ");
 System.out.println();
 System.out.println(" Documents in cluster :");
 for(int j = 0; j < names.length; j++) System.out.println(" " + names[j]);
 }

 // отсоединение
 samplConnection.disconnect();
 eipConnection.disconnect();
}

// реализация com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
 return;
}

// реализация com.ibm.mm.beans.CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
 ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
 try
 {

```

```

 new Clustering();
 System.exit(0);
 }
 catch(Throwable t)
 {
 t.printStackTrace();
 }
}

```

На рис. 48 показан поток событий по функциям bean.

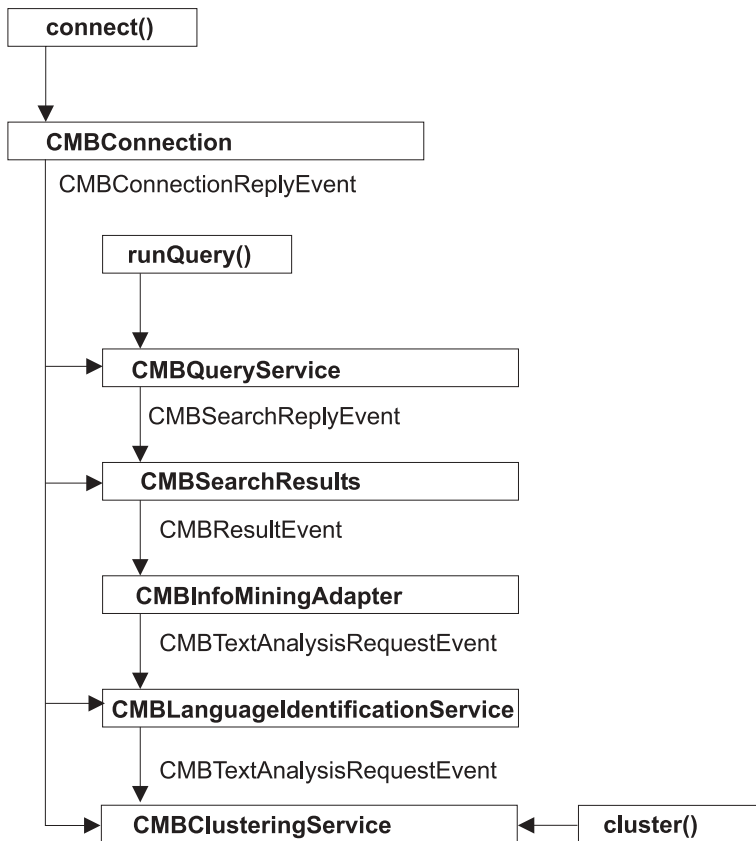


Рисунок 48. Пример кластеризации: Поток событий



## Импорт документов из пространства Web

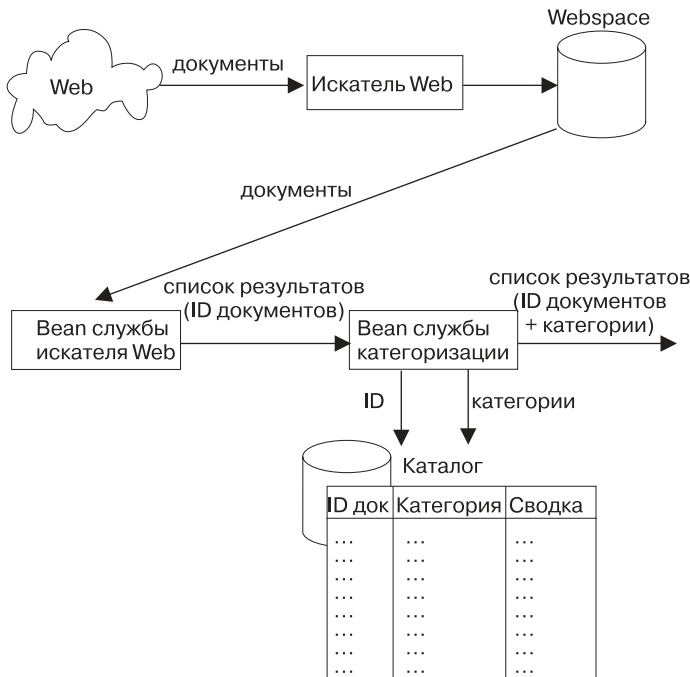


Рисунок 49. Пример Web Crawler

В этом примере показан импорт документов, полученных с помощью искателя, в компонент исследования информации и текстовый анализ (категоризация и составление сводок) этих документов. Эта операция возможна, только когда запущена или работает служба Web Crawler, и новые или измененные объекты сохраняются в конкретном каталоге пространства Web, заданном в конфигурации Web Crawler атрибутом `summaries-dir`.

При использовании Web Crawler для исследования информации убедитесь, что запускаете его с параметрами конфигурации исследования информации, описанными в разделе управления исследованием информации книги *Управление Enterprise Information Portal*. Эти параметры - часть примера файла конфигурации для компонента исследования информации `im-crawler-config-sample.xml`, расположенного в каталоге:

- В Windows:  
`<CMBROOT>\samples\java\beans\infomining\webcrawler\`
- В UNIX (AIX и Solaris):  
`<CMBROOT>/samples/java/beans/infomining/webcrawler/`

После того, как Web Crawler поместит копии документов страниц Web в указанный каталог пространства Web, вы можете импортировать их в компонент исследования информации Enterprise Information Portal с помощью функции bean CMBWebCrawlerService.

Web Crawler и функцию bean CMBWebCrawlerService можно запустить как постоянный процесс для отслеживания документов и их импорта по мере изменения. Событие CMBResultEvent может сигнализировать, когда число импортированных документов превысит определенное значение или когда будут импортированы все отслеженные файлы. Функция bean CMBWebCrawlerService уведомляет об этом все подключенные интерфейсы. Данное событие содержит импортированные файлы в качестве вектора для объектов CMBItem.

**Права доступа для CMBWebCrawlerService в AIX и Solaris.** При импорте документов с помощью CMBWebCrawlerService соответствующие файлы или удаляются из файловой системы или, если опция архивирования **archiveEnabled** включена, перемещаются из каталога disks в каталог архивов. Каталог архивов расположен на том же уровне, что и каталог disks (в .../webspaces/ikf), и у него такая же структура подкаталогов. Для использования CMBWebCrawlerService убедитесь, что у соответствующего ID пользователя есть следующие полномочия:

- Права записи для каталога disks и всех вложенных в него подкаталогов и файлов. Расположение каталога disks в .../webspaces/ikf/disks определяется атрибутом summaries-dir в файле конфигурации Web Crawler. Эти права доступа требуются для удаления или перемещения документов и файлов, полученных с помощью искателя.
- Права записи для каталога .../webspaces/ikf при включенном архивировании, так что каталог архивов может быть создан при первом же использовании архивирования.
- Права записи для каталога архивов, его подкаталогов и всех файлов при включенном архивировании, чтобы файлы, перемещаемые из каталога disks, могли быть созданы в каталоге архивов.

**Изменение операции импорта.** В функции bean CMBWebCrawlerService можно изменить некоторые параметры импорта. Исходно программа проверяет документы, полученные с помощью Crawler, каждые 30 минут, но это значение можно изменить. Кроме того, она выдает уведомление CMBResultEvent всем ожидающим его после каждых 100 импортированных документов. Это число документов можно изменить; можно также задать отправку уведомления после того, как все файлы, полученные искателем, будут импортированы в Enterprise Information Portal.

**Свойства функции bean CMBWebCrawlerService:**

**pollCycles**

Число циклов опроса.

**pollMinutes**

Время в минутах между опросами. Значение по умолчанию - 30.

**rootDir**

Каталог для %IMY\_WEBSPACE% на локальном хосте.

**archiveEnabled**

Сохраняет полученные с помощью искателя файлы в `.../webspaces/ikf/archives`. Значение по умолчанию - `false`; это значит, что `CMBWebCrawlerService` при обработке удаляет файлы из `.../webspaces/ikf/disks` без резервного копирования в другое место.

**pageSize**

Число импортированных элементов перед выдачей уведомления `CMBResultEvent` для интерфейсов `CMBResultListener`.

**webSpace**

Пространство Web, отслеживаемое искателем.

Как и в случае других функций bean текстового анализа, результаты поиска функции bean Web Crawler могут быть преобразованы в доступные для последующего сложного поиска.

В данном примере используются следующие функции bean:

- `CMBConnection`
- `CMBWebCrawlerService`
- `CMBInfoMiningAdapter`
- `CMBLanguageIdentification`
- `CMBCategorizationService`
- `CMBSummarizationService`
- `CMBCatalogService`

Программа этого примера:

1. Создает функции bean
2. Настраивает функции bean
3. Соединяет функции bean
4. Запускает службу Web Crawler
5. Выводит результаты, найденные искателем, и результаты текстового анализа

Объяснения всех этих шагов приведены после исходного кода `WebCrawler.java`.

**Полный исходный код WebCrawler.java**

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;
```

```

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBWebCrawlerService;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBCategorizationService;
import com.ibm.mm.beans.infomining.CMBSummarizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class WebCrawler implements CMBResultListener, CMBExceptionListener
{
 String CMDBNAME = "icmnlsdb";
 String CMDBUSER = "icmadmin";
 String CMDBPASSWORD = "password";
 String CATALOG = "Sample";

 String WEBSpace_DIR = ""; // укажите каталог, где находятся пространства Web
 String WEBSpace_NAME = ""; // укажите имя вашего пространства Web

 public WebCrawler() throws Throwable
 {
 // создание функций bean
 CMBConnection connection = new CMBConnection();
 CMBWebCrawlerService crawlerService = new CMBWebCrawlerService();
 CMBLanguageIdentificationService languageIdentificationService = new
 CMBLanguageIdentificationService();
 CMBCategorizationService categorizationService = new CMBCategorizationService();
 CMBSummarizationService summarizationService = new CMBSummarizationService();
 CMBCatalogService catalogService = new CMBCatalogService();
 CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
 CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

 // Чтение параметров
 BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

 System.out.print("Database [" + CMDBNAME + "] : ");
 String dbName = din.readLine();
 if (dbName.trim().equals("")) dbName = CMDBNAME;

 System.out.print("User name [" + CMDBUSER + "] : ");
 String userName = din.readLine();
 if (userName.trim().equals("")) userName = CMDBUSER;

 System.out.print("Password [" + CMDBPASSWORD + "] : ");
 String passwd = din.readLine();
 if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

 System.out.print("WebSpace directory [" + WEBSpace_DIR + "] : ");
 String webSpaceDir = din.readLine();
 if (webSpaceDir.trim().equals("")) webSpaceDir = WEBSpace_DIR;

 System.out.print("WebSpace name [" + WEBSpace_NAME + "] : ");
 String webSpaceName = din.readLine();
 if (webSpaceName.trim().equals("")) webSpaceName = WEBSpace_NAME;

 System.out.print("Catalog [" + CATALOG + "] : ");

```

```

String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG;

System.out.println("\n\nRunning Summarization with the following settings:");
System.out.println("Database = " + dbName);
System.out.println("User = " + userName);
System.out.println("Password = " + passwd);
System.out.println("Webpace directory = " + webspaceDir);
System.out.println("Webpace name = " + webspaceName);
System.out.println("Catalog = " + catalog + "\n");

int key;
do {
 System.out.print("Continue (y/n)? ");
 InputStreamReader inReader = new InputStreamReader(System.in);
 key = inReader.read();
 if (key == 'n') System.exit(0); }
while (key != 'y');

// Настройка функций bean
connection.setServerName(dbName);
connection.setUserId(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
connection.setServiceConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
crawlerService.setRootDirectory(webspaceDir);
crawlerService.setWebSpace(webspaceName);

adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);

// соединение функций bean
connection.addCMBConnectionReplyListener(crawlerService);
connection.addCMBConnectionReplyListener(adapter1);
connection.addCMBConnectionReplyListener(languageIdentificationService);
connection.addCMBConnectionReplyListener(categorizationService);
connection.addCMBConnectionReplyListener(summarizationService);
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(adapter2);

crawlerService.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener(categorizationService);
categorizationService.addCMBTextAnalysisRequestListener(summarizationService);
summarizationService.addCMBStoreRecordRequestListener(catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);
adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
crawlerService.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// запуск запроса
connection.connect();
catalogService.setDefaultCategoryPath

```

```

 (catalogService.getTaxonomy().getRootCategory().getPathAsString());

crawlerService.start();
connection.disconnect();
}

// реализация com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
 if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
 return;

 Vector cmbItemVector = (Vector)e.getData();

 if(cmbItemVector == null)
 return;

 try {
 for(int i = 0; i < cmbItemVector.size(); i++)
 {
 CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

 CMBRecord currentRecord = currentItem.getInfoMiningRecord();

 System.out.println("\n\nPID : " + currentRecord.getPID());
 System.out.println("Categories : " + currentRecord.getValue("IKF_CATEGORIES"));
 System.out.println("Summary : " + currentRecord.getValue("IKF_SUMMARY"));
 } /* for */
 }
 catch (CMBNoSuchKeyException nske)
 {
 nske.printStackTrace();
 }
}

//реализация com.ibm.mm.beans.CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
 ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
 try
 {
 new WebCrawler();
 System.exit(0);
 }
 catch(Throwable t)
 {
 t.printStackTrace();
 }
}
}

```

## Создание функций bean

Вы создаете соединение с контент-сервером, учитывая защиту; функции исследования информации нужно знать, кто работает с системой. Это дает гарантию что использовать систему смогут только зарегистрированные пользователи. Такое соединение может быть установлено с помощью функции bean CMBConnection. Функция bean CMBWebCrawlerService используется для

импорта документов, предварительно найденных с помощью Crawler, в компонент исследования информации и для выдачи уведомления CMBSResultEvent. Информация идентификации языка, сводок и категорий создается с помощью функций bean CMBLanguageIdentification, CMBSummarizationService и CMBCategorizationService. Два адаптера преобразуют события результатов в события требований текстового анализа, а затем сохраняют события ответов записи снова в событиях результатов поиска.

Код создания функций bean:

```
CMBConnection connection = new CMBConnection();
CMBWebCrawlerService crawlerService = new CMBWebCrawlerService();
CMBLanguageIdentificationService languageIdentificationService =
 new CMBLanguageIdentificationService();
CMBCategorizationService categorizationService = new CMBCategorizationService();
CMBSummarizationService summarizationService = new CMBSummarizationService();
CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();
```

### Настройка функций bean

Код, показанный в разделе настройки, нужно привести в соответствие с вашей установкой. Необходимо указать имя контент-сервера соединения, ID пользователя и соответствующий пароль.

После служебной функции bean Web Crawler нужно указать корневой каталог вашего локального хоста, где служба Web Crawler сохраняла или изменяла свои объекты в заданном пространстве Web, а также предварительно заданное имя пространства Web. Для возможности запуска функций bean текстового анализа и функции bean службы каталога их необходимо привязать к существующему каталогу.

Для настройки в этом примере используется следующий код:

```
connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
connection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);
crawlerService.setRootDirectory(webspaceDir);
crawlerService.setWebSpace(webspaceName);

adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);
```

### Соединение функций bean

На рис. 50 на стр. 604 показан поток событий для функций bean примера.

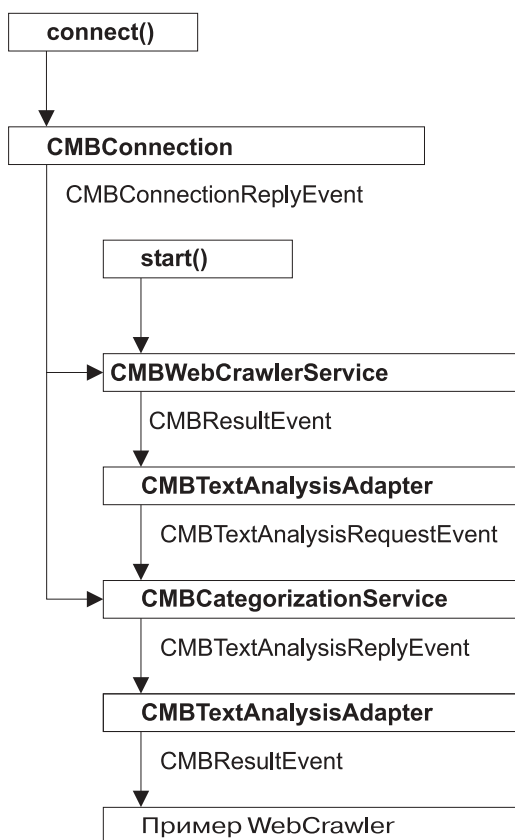


Рисунок 50. Пример Web Crawler: Поток событий

Пять функций bean, используемых в этом примере, ожидают хэндл соединения от CMBConnectionReplyEvent. Функция bean CMBWebCrawlerService инициирует службу Crawler, результатом работы которой служит событие, приводящее к запуску потока событий для других функций bean.

Код соединения функций bean:

```

connection.addCMBConnectionReplyListener(crawlerService);
connection.addCMBConnectionReplyListener(adapter1);
connection.addCMBConnectionReplyListener(languageIdentificationService);
connection.addCMBConnectionReplyListener(categorizationService);
connection.addCMBConnectionReplyListener(summarizationService);
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(adapter2);

crawlerService.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener(categorizationService);
categorizationService.addCMBTextAnalysisRequestListener(summarizationService);

```



```

summarizationService.addCMBStoreRecordRequestListener(catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);
adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
crawlerService.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

```

Поскольку исключительные ситуации тоже посылаются как события, класс `WebCrawler` должен обработать соответствующие события, используя интерфейс `CMBExceptionListener`, который соединяется с функциями bean для получения уведомления об исключительных ситуациях.

### Запуск службы Web Crawler

Для запуска службы `Web Crawler` нужно установить соединение с контент-сервером, вызвав метод соединения функции bean `CMBConnection`:

```
connection.connect();
```

Код запуска службы `Web Crawler`:

```

catalogService.setDefaultCategoryPath
 (catalogService.getTaxonomy().getRootCategory().getPathAsString());
crawlerService.start();

```

Чтобы закрыть текущее соединение, вызывается метод `disconnect()`:

```
connection.disconnect();
```

### Вывод результатов анализа текста

В классе `WebCrawler` используется интерфейс `CMBResultListener` для вывода списка документов, найденных во время поиска, и для вывода информации категорий и сводок, полученной для каждого документа. Событие `CMBResultEvent`, полученное как аргумент для метода `onCMBResult`, содержит вектор объектов `CMBItem`, где каждый объект `CMBItem` представляет документ:

```
Vector cmbItemVector = (Vector)e.getData();
```

В объект `CMBItem` встраивается постоянный ID документа:

```

CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

CMBRecord currentRecord = currentItem.getInfoMiningRecord();

System.out.println("\n\nPID : " + currentRecord.getPID());

```

и результаты анализа текста.

Затем вы получаете информацию сводок и категорий:

```

System.out.println("Categories : " +
 currentRecord.getValue("IKF_CATEGORIES"));
System.out.println("Summary : " +
 currentRecord.getValue("IKF_SUMMARY"));

```

## Поиск документов по категориям

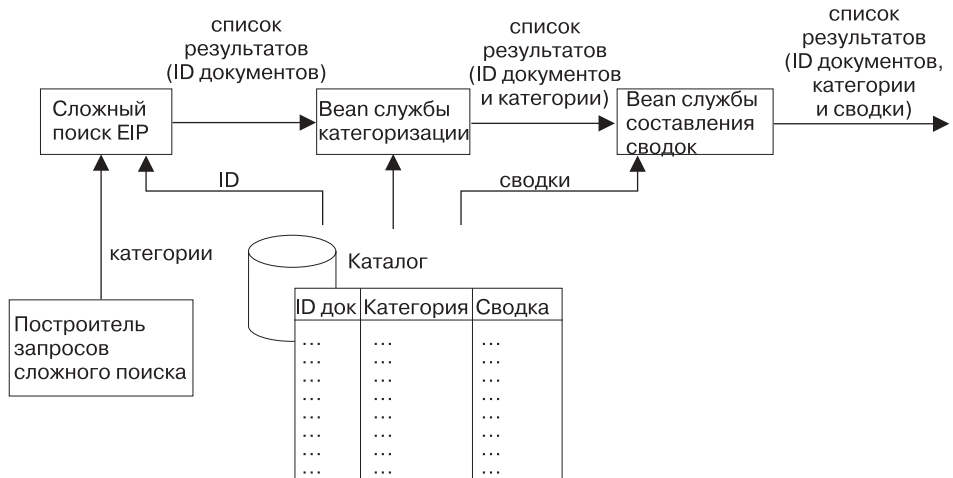


Рисунок 51. Пример сложного поиска

Этот пример показывает, как запустить сложный поиск и найти информацию категорий для найденных документов. Сложный поиск возможен только для документов, которые были сделаны доступными для этого вида поиска, как показано в примере категоризации. Смотрите “Положение файлов примеров” на стр. 566. В отличие от предыдущих примеров, где простой поиск EIP выполнялся по всему контент-серверу EIP, этот пример выполняет поиск метаданных на складе данных.

Если при передаче сложных запросов возникают проблемы с производительностью, смотрите дополнительную информацию в разделе “Настройка производительности” на стр. 625.

После создания списка результатов сложного поиска функция bean службы каталога может использовать ID найденных документов для получения метаданных, которые были сохранены ранее на складе данных.

В данном примере используются следующие функции bean:

- CMBCConnection
- CMBAdvancedSearchService
- CMBInfoMiningAdapter
- CMBCatalogService

Программа этого примера:

1. Создает функции bean
2. Настраивает функции bean
3. Соединяет функции bean
4. Запускает сложный поиск
5. Выводит результаты поиска и текстового анализа для проверки

Объяснения всех этих шагов приведены после исходного кода AdvancedSearch.java.

### Полный исходный код AdvancedSearch.java

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBException;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBAdvancedSearchService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBCategory;

public class AdvancedSearch implements CMBResultListener,
 CMBExceptionListener
{
 String CMDBNAME = "icmnlsdb";
 String CMDBUSER = "icmadmin";
 String CMDBPASSWORD = "password";

 String CATALOG = "Sample";
 String SEARCH_TERM = "Monitor";

 CMBCatalogService catalogService = null;

 public AdvancedSearch() throws Exception
 {
 // создание функций bean
 CMBConnection connection = new CMBConnection();
 CMBAdvancedSearchService advancedSearchService = new CMBAdvancedSearchService();
 catalogService = new CMBCatalogService();

 // Чтение параметров
 BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

 System.out.print("Database [" + CMDBNAME + "] : ");
```

```

String dbName = din.readLine();
if (dbName.trim().equals("")) dbName = CMDBNAME;

System.out.print("User name [" + CMDBUSER + "] : ");
String userName = din.readLine();
if (userName.trim().equals("")) userName = CMDBUSER;

System.out.print("Password [" + CMDBPASSWORD + "] : ");
String passwd = din.readLine();
if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

System.out.print("Catalog [" + CATALOG + "] : ");
String catalog = din.readLine();
if (catalog.trim().equals("")) catalog = CATALOG;

System.out.print("Search Term [" + SEARCH_TERM + "] : ");
String searchTerm = din.readLine();
if (searchTerm.trim().equals("")) searchTerm = SEARCH_TERM;

System.out.println("\n\nRunning AdvancedSearch with the following settings:");
System.out.println("Database = " + dbName);
System.out.println("User = " + userName);
System.out.println("Password = " + passwd);
System.out.println("Catalog = " + catalog);
System.out.println("Search Term = " + searchTerm + "\n");

int key;
do {
 System.out.print("Continue (y/n)? ");
 InputStreamReader inReader = new InputStreamReader(System.in);
 key = inReader.read();
 if (key == 'n') System.exit(0); }
while (key != 'y');

// Настройка функций bean
connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
connection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
connection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);

catalogService.setCatalogName(catalog);
advancedSearchService.setCatalogName(catalog);
String[] recordKeys = {"IKF_SUMMARY"};
advancedSearchService.setKeysToBeFetched(recordKeys);
advancedSearchService.setQueryString
 ("(\"IKF_CONTENT\" CONTAINS \"" + searchTerm + "\"");

// соединение функций bean
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(advancedSearchService);
advancedSearchService.addCMBResultListener(this);

```

```

connection.addCMBExceptionListener(this);
advancedSearchService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);

// запуск запроса
connection.connect();
advancedSearchService.runQuery();
connection.disconnect();
}

// реализация com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
 if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
 return;

 Vector cmbItemVector = (Vector)e.getData();

 if(cmbItemVector == null)
 return;

 try {
 for(int i = 0; i < cmbItemVector.size(); i++)
 {
 CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

 CMBRecord currentRecord = currentItem.getInfoMiningRecord();

 String summary = (String)currentRecord.getValue("IKF_SUMMARY");

 CMBCategory[] categories =
 catalogService.getCategoriesForRecord(currentItem);
 String categoryString = categories[0].getPathAsString();
 for(int j = 1; j < categories.length; j++)
 {
 categoryString += ", " + categories[j].getPathAsString();
 }

 System.out.println("\n\nPID : " + currentRecord.getPID());
 System.out.println("Categories : " + categoryString);
 System.out.println("Summary : " +
 ((summary == null)? "n/a": summary));
 } /* for */
 }
 catch (CMBException ex)
 {
 ex.printStackTrace();
 }
}

// реализация com.ibm.mm.beans.CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
 ((Exception)e.getData()).printStackTrace();
}

```

```

public static void main(String[] args)
{
 try
 {
 new AdvancedSearch();
 System.exit(0);
 }
 catch(Exception e)
 {
 e.printStackTrace();
 }
}
}

```

### Создание функций bean

Чтобы выполнить поиск, нужно установить соединение с контент-сервером. Такое соединение может быть установлено с помощью функции bean CMBConnection. Функция bean CMBAdvancedSearchService необходима для выполнения сложного поиска. Информацию категорий можно получить с помощью функции bean CMBCatalogService.

Код создания функций bean:

```

CMBConnection connection = new CMBConnection();
CMBAdvancedSearchService advancedSearchService = new CMBAdvancedSearchService();
catalogService = new CMBCatalogService();

```

### Настройка функций bean

Код, показанный в разделе настройки, нужно привести в соответствие с вашей установкой. Необходимо указать имя контент-сервера соединения, ID пользователя и соответствующий пароль.

Для возможности запуска функций bean службы сложного поиска и службы каталога их необходимо привязать к существующему каталогу.

Для настройки в этом примере используется следующий код:

```

connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
connection.setConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);
connection.setServiceConnectionType
 (CMBConnection.CMB_CONNTYPE_DYNAMIC);

catalogService.setCatalogName(catalog);
advancedSearchService.setCatalogName(catalog);
String[] recordKeys = {"IKF_SUMMARY"};
advancedSearchService.setKeysToBeFetched(recordKeys);
advancedSearchService.setQueryString
 ("(\"IKF_CONTENT\" CONTAINS '\"' + searchTerm + '\"')");

```

# Соединение функций bean

На рис. 52 показан поток событий для функций bean примера.

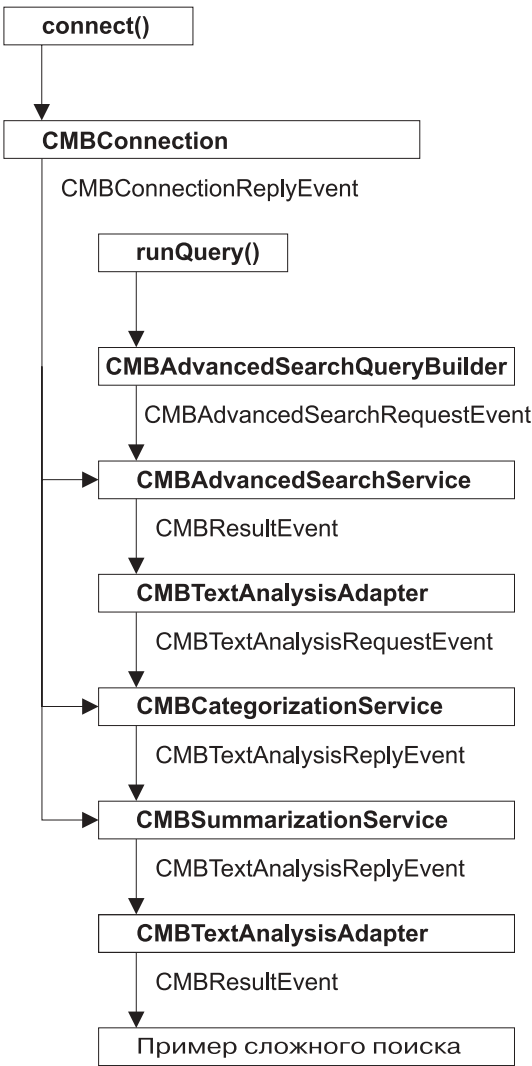


Рисунок 52. Пример сложного поиска: Поток событий

Две функции bean, используемые в этом примере, ожидают хэндл соединения от CMBConnectionReplyEvent. Функция bean CMBAAdvancedSearchService инициирует поиск, результатом которого служит событие, приводящее к запуску потока событий для других функций bean.

Код соединения функций bean:

```

connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(advancedSearchService);
advancedSearchService.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
advancedSearchService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);

```

Поскольку исключительные ситуации тоже посылаются как события, класс `AdvancedSearch` должен обработать соответствующие события, реализовав интерфейс `CMBExceptionListener`, который соединяется с функциями bean для получения уведомления об исключительных ситуациях.

### Запуск запроса

Для запуска запросов нужно установить соединение с контент-сервером, вызвав метод соединения функции bean `CMBConnection`:

```
connection.connect();
```

Для запуска сложного поиска нужно задать каталог, строку запроса поиска и метаданные для поиска.

Запуск сложного поиска:

```
advancedSearchService.runQuery();
```

Чтобы закрыть текущее соединение, вызывается метод `disconnect()`:

```
connection.disconnect();
```

### Вывод результатов анализа текста

В классе `AdvancedSearch` используется интерфейс `CMBResultListener` для вывода списка документов, найденных при поиске, и для вывода информации категорий для каждого документа. Событие `CMBResultEvent`, полученное как аргумент для метода `onCMBResult`, содержит вектор объектов `CMBItem`, где каждый объект `CMBItem` представляет документ:

```
Vector cmbItemVector = (Vector)e.getData();
```

В объект `CMBItem` встраивается постоянный ID документа:

```

CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

CMBRecord currentRecord = currentItem.getInfoMiningRecord();
String summary = (String)currentRecord.getValue("IKF_SUMMARY");

CMBCategory[] categories =
 catalogService.getCategoriesForRecord(currentItem);
String categoryString = categories[0].getPathAsString();
for(int j = 1; j < categories.length; j++)
{
 categoryString += ", " + categories[j].getPathAsString();
}

```



```

 }

 System.out.println("\n\nPID : " + currentRecord.getPID());

```

и результаты функций bean текстового анализа, если они были в потоке событий.

Затем вы получаете информацию категорий:

```

System.out.println("Categories :
 " + currentRecord.getValue("IKF_CATEGORIES"));

```

Вектор, возвращенный методом `getCategories()` класса `CMBItem`, содержит объекты класса `CMBCategory`, которые можно использовать для определения полного пути к текущей категории.

## Построение собственного контент-провайдера

Возможно, вы захотите построить собственный контент-провайдер, чтобы, применяя фильтры, извлекать текст документа в вашем собственном формате.

В этом разделе рассказано, как можно написать свою программу контент-провайдера для работы с пользовательской моделью объектов или с собственными форматами в частях объектов `CMBItem`. Для этой задачи в Исследование информации поддерживаются:

- Интерфейс **`CMBContentProvider`** - определяет интерфейс для классов и поддерживает поиск текста для текстового анализа.
- Метод **`setContentProvider(CMBContentProvider)`** в классе `CMBInfoMiningUtilities` - конфигурирует `ContentProvider`.

Интерфейс `CMBContentProvider` определяет один метод `getContent()`, возвращающий текст указанного элемента для текстового анализа. Например:

```

public CMBTextAnalysisDocument getContent(CMBConnection connection, CMBItem item)
throws CMBContentProviderException;

```

- Параметр `connection` - открытое соединение с сервером.
- Параметр `item` - текущий элемент обработки.
- Исключительная ситуация `CMBContentProviderException` - если происходит ошибка при обработке текущего элемента.
- Возврат текста как объекта класса `CMBTextAnalysisDocument`.

Чтобы сообщить системе, какой использовать `ContentProvider`, воспользуйтесь методом `setContentProvider(CMBContentProvider)` в классе `CMBInfoMiningUtilities` для задания объекта с этим интерфейсом. Например:

```

CMBInfoMiningUtilities.setContentProvider
 (new MyCompaniesLatestGreatestContentProvider());

```

Чтобы разработать свой собственный контент-провайдер, в качестве отправной точки можно воспользоваться примером контент-провайдера (SimpleContentProvider). Этот пример находится в каталоге <CMBROOT>\samples\java\beans\infomining\contentprovider

С Исследование информации поставляется контент-провайдер по умолчанию. Она дополняет интерфейс CMBContentProvider, позволяя выбрать для обработки какие-то отдельные части и предлагая механизм пропуска объектов, слишком больших для обработки в памяти, например, потоков видео.

Контент-провайдер по умолчанию регистрируется так:

```
CMBInfoMiningUtilities.setContentProvider(new CMBDefaultContentProvider());
```

---

## Использование API служб

API служб исследования информации - это API Java, которые интегрируют функции исследования информации в виде служб EIP. Используя API служб исследования информации, можно написать прикладные программы, которые смогут:

- Фильтровать текстовое содержимое из документов разных форматов, например, pdf, Microsoft Word и HTML
- Выполнять текстовый анализ документов для создания метаданных, включая сводки и категории
- Сохранять метаданные документа в постоянных записях
- Выполнять поиск записей

### Примечание

Перед использованием функций bean исследования информации надо понять различия между API служб и JavaBeans.

- JavaBeans предоставляет возможность быстрой разработки прикладных программ. Некоторые элементы кода 'привинчиваются друг к другу' и не могут изменяться пользователем. Дополнительную информацию смотрите в книге "Построение прикладной программы исследования информации при помощи функций bean" на стр. 561.
- API служб исследования информации обеспечивают дополнительную гибкость при построении прикладных программ исследования информации. Код может быть показан в большей мере в виде 'строительных блоков', которые можно складывать в соответствии с вашими специфическими требованиями.

## Соединение с API служб исследования информации

Чтобы пользоваться API служб исследования информации, необходимо создать объект службы при помощи класса `DKIKServiceFed`. В зависимости от нужного типа прикладной программы надо импортировать класс из одного из следующих трех пакетов:

- Если вы хотите создать программу на хосте сервера, используйте следующий оператор импорта: `import com.ibm.mm.sdk.server.DKIKServiceFed;`
- Если вы хотите создать программу на хосте клиента, используйте следующий оператор импорта: `import com.ibm.mm.sdk.client.DKIKServiceFed;`
- Если вы хотите сохранить гибкость и создать программу, которая может выполняться как на сервере, так и на клиенте, используйте следующий оператор импорта: `import com.ibm.mm.sdk.cs.DKIKServiceFed;`

Выполнив нужный оператор импорта, создайте объект `ikfService`, введя:  
`DKIKService ikfService = DKIKServiceFed.create();`

Создав объект службы, вы можете соединиться с базой данных, используя имя, ID пользователя и пароль.

```
ikfService.connect("databaseName", "userID", "password", null);
```

Теперь у вас есть соединение с API служб исследования информации и можете использовать классы из следующих двух пакетов:

- **`com.ibm.mm.sdk.common.infomining`** для управления задачами библиотек, каталогов и записей.
- **`com.ibm.mm.sdk.common.infomining.analysis`** для использования различных инструментов текстового анализа.

Объяснение этих главных функций смотрите в следующих разделах.

Чтобы отсоединиться, введите:

```
ikfService.disconnect();
```

## Управление библиотекой, таксономиями и каталогами

Классы для управления задачами библиотек, каталогов и таксономий находятся в пакете **`com.ibm.mm.sdk.common.infomining`**.

### Библиотека:

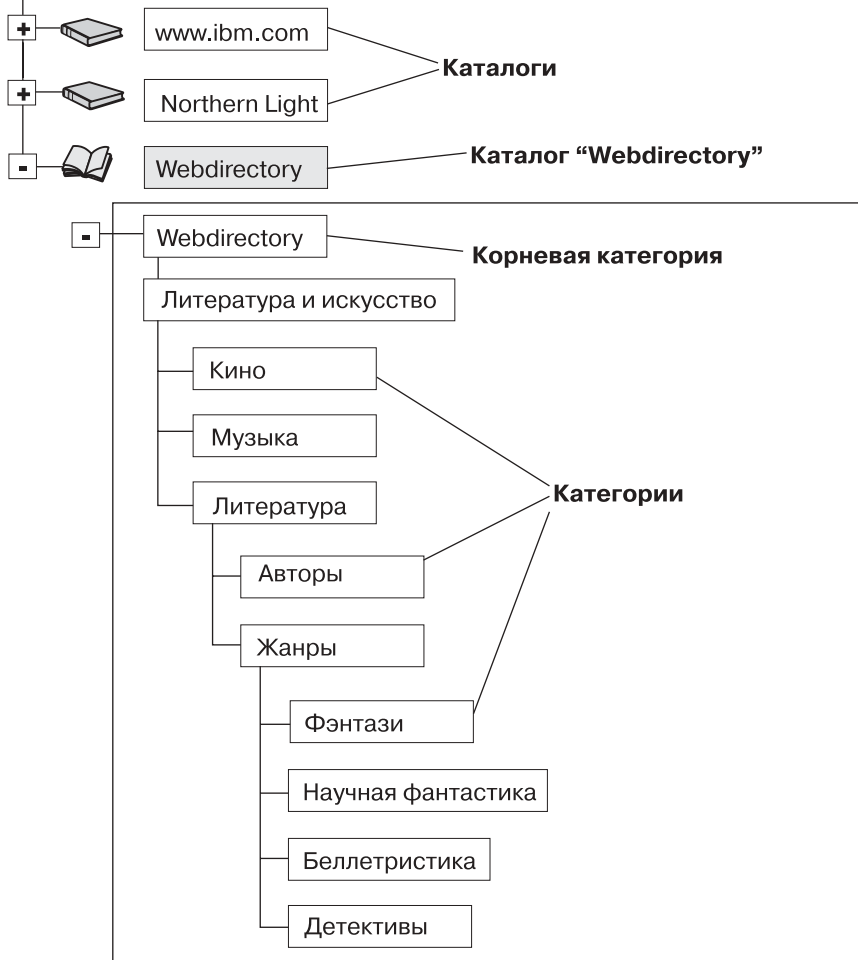


Рисунок 53. Пример каталога

В примере каталога Library содержит три каталога - `www.ibm.com`, `Northern Light` и `Webdirectory`. `Webdirectory` имеет структуру дерева категорий, показанную на диаграмме.

Каталоги могут создаваться только при помощи Information Structuring Tool (IST).

### Возврат библиотеки

Чтобы получить объект библиотеки, введите:

```
DKIKFLibrary library = ikfService.getLibrary();
```

### Получение списка всех каталогов библиотеки

Чтобы получить имена всех каталогов библиотеки, введите:

```
String[] catalogNames = library.getCatalogNames();
```

Будет возвращен массив со списком всех имен каталога. Пример возвращает следующие имена каталога без определенного порядка: Webdirectory, www.ibm.com и Northern Light.

### Возврат отдельного каталога

Чтобы получить определенный каталог, введите:

```
DKIKFCatalog catalog = library.getCatalog(catalogNames[0]);
```

Этот пример возвращает каталог с первым именем из списка, возвращенного на предыдущем шаге. В данном случае возвращается каталог Webdirectory.

### Возврат таксономии каталога

Чтобы получить таксономию каталога, введите:

```
DKIKFTaxonomy taxonomy = catalog.getTaxonomy();
```

У объекта таксономии есть разные методы для работы с категориями.

#### Примечание

Объект таксономии считывается из базы данных, и все дальнейшие действия выполняются над этой копией.

### Получение списка всех категорий таксономии

Чтобы получить все категории таксономии, введите:

```
DKIKFCategory[] categories = taxonomy.getCategories();
```

Будет возвращен массив со списком всех категорий, включая корневую. Пример возвращает следующие категории без определенного порядка: Horror, Author, Music, Literature, Webdirectory, Genres, Science Fiction, Fantasy, Movies, Romance и Arts.

#### Примечание

Объекты категорий считываются из базы данных, и все дальнейшие действия выполняются над копией.

### Возврат корневой категории

Чтобы получить корневую категорию, введите:

```
DKIKFCategory rootCategory = taxonomy.getRootCategory();
```

Этот пример возвращает корневую категорию из списка, возвращенного на предыдущем шаге. В данном случае возвращается корневая категория `Webdirectory`.

### Возврат отдельной категории

Чтобы вернуть определенную категорию, введите:

```
DKIKFCategory category = taxonomy.getCategory("Webdirectory/Literature/Genres");
```

В этом примере возвращается категория `Genres`.

Символ `"/"` - текущий разделитель в объекте таксономии.

#### Примечание

В объекте таксономии `getCategory` доступны три метода, требующие разные параметры:

- Путь как строку из имен категорий, разделенных заданным символом
- Путь как массив имен категорий
- Другой объект категорий

Дополнительную информацию смотрите в *электронном справочнике API*

### Проверка актуальности текущей таксономии

Чтобы убедиться, что текущая таксономия - самая последняя, введите:

```
if(taxonomy.getTimestamp().before(catalog.getTaxonomyLastModified()))
{
 taxonomy = catalog.getTaxonomy();
}
```

#### Примечание

Объект таксономии считывается из базы данных, и все дальнейшие действия выполняются над этой копией. Поэтому проверяйте актуальность объекта таксономии.

### Возврат дочерних категорий

Чтобы вернуть дочерние категории некоторой категории, введите:

```
DKIKFCategory[] children = category.getChildren();
```

Будет возвращен массив со списком всех дочерних категорий данной категории. В примере на основе категории `Genres` возвращаются следующие дочерние категории без определенного порядка: `Fantasy`, `Science Fiction`, `Romance`, и `Horror`.

#### Примечание

Для категории может быть возвращено несколько дочерних категорий, но не возвращаются дальнейшие подкатегории этих дочерних категорий.

### Возвращение родительской категории некоторой категории

Чтобы вернуть родительскую категорию, введите:

```
DKIKFCategory parent = category.getParent();
```

Пример, основанный на категории Genres, возвращает только категорию Literature.

### Возвращение заданной схемы каталога и получение списка всех атрибутов в схеме

Чтобы получить заданную схему каталога, введите:

```
DKIKFSchema schema = catalog.getSchema();
Iterator keys = schema.keySet().iterator();

while(keys.hasNext())
{
 String key = (String)keys.next();
 System.out.println("key: " + key + "type: " + schema.getType(key).getName());
}
```

Пример вернет схему каталога Webdirectory.

Затем можно вернуть ключи (имена атрибутов) схемы. Дополнительную информацию смотрите в *электронном справочнике API*.

#### Примечание

Объект схемы считывается из базы данных, и все дальнейшие действия выполняются над этой копией.

## Использование инструментов исследования информации

Классы, которые могут использовать инструменты исследования информации, находятся в пакете **com.ibm.mm.sdk.common.infomining.analysis**. Инструменты позволяют:

- Сгенерировать сводку документа
- Определить категорию документа
- Определить язык документа
- Извлекать из документов информацию, такую как имена, термины и выражения
- Кластеризовать наборы документов

Инструменты обрабатывают текстовые документы, являющиеся объектами класса `DKIKFTextDocument`. Есть два метода создания объекта текстового документа, а именно:

- Методы создания из класса `DKIKFTextDocument`, если содержание документа уже существует как строка Java
- Класс `DKIKFDocumentFilter`, если содержание документа представлено в формате

Оба класса находятся в пакете **`com.ibm.mm.sdk.common.infomining`**.

### Текстовый документ

Чтобы создать объект текстового документа, если содержание документа существует как строка Java, введите:

```
DKIKFTextDocument document = DKIKFTextDocument.create("the document content");
```

#### Примечание

Доступны три разных метода `create`, которые позволяют:

- Создать текстовый документ с заданным содержанием
- Создать текстовый документ с заданным содержанием и именем
- Создать текстовый документ с заданным содержанием, именем и языком

Дополнительную информацию смотрите в *электронном справочнике API*.

### Фильтрация документа

Чтобы создать объект текстового документа из форматированного текста, введите:

```
DKIKFDocumentFilter documentFilter = new DKIKFDocumentFilter(ikfService);
byte[] documentBytes = ... //задайте байты документа из источника данных
DKIKFTextDocument document2 = documentFilter.getTextDocument(documentBytes);
```

Обратите внимание на то, что оригинальный текстовый документ может иметь любой из поддерживаемых форматов, например, pdf или документ Microsoft Word.

Задайте кодировку фильтра при помощи метода объекта фильтра документа. Дополнительную информацию смотрите в *электронном справочнике API*.

### Определение языка документа

Чтобы определить язык документа, введите:



```
DKIKFLanguageIdentifier languageIdentifier =
 new DKIKFLanguageIdentifier(ikfService);
DKIKFLanguageIdentificationResult[] languageResults =
 languageIdentifier.analyze(document);
document.setLanguage(languageResults[0].getLanguage());
```

Будет возвращен массив со списком объектов результата, содержащих язык и значение достоверности, где наибольшее значение достоверности будет первым в списке.

#### Примечание

При использовании других инструментов исследования информации, включая службу составления сводок и категоризации, для текстового документа должен быть задан язык.

Используйте методы объектов результата языка для возвращения значений достоверности и языка.

```
string language = languageResults[0].getLanguage();
float confidence = languageResults[0].getConfidence();
```

Дополнительную информацию о службе распознавания языка и параметрах по умолчанию смотрите в *электронном справочнике API*.

### Генерация сводки документа

Чтобы создать сводку документа, введите:

```
DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);
DKIKFSummarizationResult summarizationResult = summarizer.analyze(document);
```

Чтобы вернуть сводку, введите:

```
string summary = summarizationResult.getSummary();
```

Дополнительную информацию о службе составления сводок и параметрах по умолчанию смотрите в *электронном справочнике API*.

### Извлечение информации из документа

Чтобы извлечь из документа информацию, введите:

```
DKIKFInformationExtractor extractor = new DKIKFInformationExtractor(ikfService);
DKIKFInformationExtractionResult information = extractor.analyze(document);
```

Используйте методы объектов результата извлечения информации, предоставляющие дополнительные возможности анализа информации.

```
DKIKFFeature[] features = information.getFeatures();
```

Дополнительную информацию о службе извлечения информации и параметрах по умолчанию смотрите в *электронном справочнике API*.

## Кластеризация

При помощи кластеризации можно сгруппировать сходные документы, для чего введите:

```
DKIKFClusterer clusterer = new DKIKFClusterer();
cluster.analyze(doc1);
cluster.analyze(doc2);
cluster.analyze(doc3);
DKIKFClusterResult clusterResult = clusterer.cluster();
```

Используйте методы объекта результата кластеризации, чтобы вернуть узлы кластеризации, число кластеров и общее число документов.

```
int clusterCount = clusterResult.getClusterCount();
DKIKFClusterNode[] nodes = clusterResult.getClusterNodes();
int documentCount = clusterResult.getDocumentCount();
```

Дополнительную информацию о службе кластеризации и параметрах по умолчанию смотрите в *электронном справочнике API*.

## Категоризация

Чтобы присвоить документам категории, введите:

```
DKIKFCategorizer categorizer = new DKIKFCategorizer(catalog);
DKIKFCategorizationResult[] categorizationResults = categorizer.analyze(document);
```

Будет возвращен массив со списком объектов результата категоризации, содержащих категории и значения достоверности, где наибольшее значение достоверности будет первым в списке.

Используйте методы объектов результата категоризации для возвращения значений достоверности и категорий.

```
DKIKFCategory bestCategory = categorizationResults[0].getCategory();
```

### Примечание

Чтобы использовать службу категоризации, необходимо создать и обучить каталог при помощи Information Structuring Tool (IST). Служба категоризации использует один каталог.

Заметьте также, что возвращенные объекты категорий не принадлежат объекту таксономии. Используйте методы таксономии для поиска родительских и дочерних категорий.

Дополнительную информацию о службе категоризации и параметрах по умолчанию смотрите в *электронном справочнике API*.

## Создание записей и сохранение метаданных в каталогах

Классы для создания записей и сохранения метаданных находятся в пакете **com.ibm.mm.sdk.common.infomining**. Они позволяют:

- Создать новую запись в каталоге
- Получить запись из каталога
- Вернуть категории записи
- Изменить значение записи
- Изменить категорию, присвоенную записи
- Удалить запись

Заметьте, что API служб исследования информации может управлять только каталогами, созданными при помощи Information Structuring Tool (IST). После создания каталога возможно добавление записей и метаданных.

### Создание новой записи в каталоге

При создании записи используется PID и схема каталога:

```
DKIKFRecord record = DKIKFRecord.create("PID", schema);
record.setValue("IKF_TITLE", "This is the title");
record.setValue("IKF_SUMMARY", "This is the document summary");
record.setValue("IKF_CONTENT", "This is the document content");
DKIKFCategory[] categoriesParam = {bestCategory};
catalog.createRecord(record, categoriesParam);
```

В этом примере задаются значения заголовка, сводки и содержания записи. Запись создается с заданными выше значениями и может быть отнесена к одной или нескольким категориям.

### Получение записи из каталога

Чтобы получить запись из каталога при помощи PID, введите:

```
DKIKFRecord retrievedRecord = catalog.getRecord("PID");
```

### Возвращение категорий записи

Чтобы вернуть категории записи, используйте:

```
DKIKFCategory[] recordCategories = catalog.getCategoriesForRecord("PID");
```

### Изменение значения записи в каталоге

Чтобы изменить значение записи, например, заголовок записи, введите:

```
record.setValue("IKF_TITLE", "This is the new title");
catalog.updateRecord(record);
```

### Примечание

Доступны три разных метода `update`, которые позволяют:

- Изменить только значения записи (смотрите приведенный выше пример для заголовка)
- Изменить значения и записи и заданные категории
- Изменить только категории записи (смотрите приведенный ниже пример, в котором запись одной категории добавляется еще к одной категории)

Дополнительную информацию смотрите в *электронном справочнике API*.

### Добавление записи одной категории в другую категорию

Чтобы добавить запись в другую категорию, введите:

```
DKIKFCategory oldCategories = catalog.getCategoriesForRecord("PID");
DKIKFCategory newCategory = taxonomy.getCategory("Webdirectory/arts/movies");
List categoriesList = Arrays.asList(oldCategories);
categoriesList.add(newCategory);
catalog.updateRecord("PID",
 categoriesList.toArray(new DKIKFCategory[categoriesList.size()]));
```

В этом примере запись добавляется в новую категорию `Movies`. Если вы хотите сохранить ранее присвоенные категории, их надо включить в вызов изменения записи, как показано в приведенном выше примере. Вызов изменения записи для новой категории только удалит все ранее присвоенные категории.

Есть три метода изменения записи; дополнительную информацию смотрите в “Изменение значения записи в каталоге” на стр. 623.

Кроме того, можно добавить запись в корневую категорию и в несколько категорий. Подробности смотрите в *электронном справочнике API*.

### Удаление сохраненной записи

Чтобы удалить запись, введите:

```
catalog.deleteRecord("PID");
```

## Поиск документов

Классы для поиска записей находятся в пакете `com.ibm.mm.sdk.common.infomining`.

### Поиск записей, содержащих определенное слово

Например, чтобы найти запись, содержащую слово `"Bach"` в категории `"Music"`, введите:

```
String queryString = "(" + IKF_CONTENT + " contains \"Bach\") and
 (DKIKF_CATEGORY = \"Webdirectory/Music\")";
DKIKFSearchConfiguration searchConfiguration = new DKIKFSearchConfiguration();
searchConfiguration.setTaxonomy(taxonomy);
DKIKFSearchResult searchResult =
 catalog.searchRecords(queryString, searchConfiguration);
Iterator resultPIDs = searchResult.iterator();
```

Чтобы запустить метод `searchRecords` для объекта каталога, нужны:

- Строка запроса и
- Объект конфигурации поиска

Объект конфигурации поиска задает свойства поиска, например, максимальное число результатов поиска. Если вы используете категории в строке запроса, нужен также объект таксономии.

Результаты поиска могут быть строками PID или записями, и могут быть заданы в вызове `DKIKFSearchConfiguration`.

### Настройка производительности

При передаче сложных запросов могут возникнуть проблемы с производительностью, особенно если строка запроса содержит несколько операций OR. Чтобы повысить производительность, избегайте сложных запросов, содержащих операции OR, применяя для преобразованию выражений OR правила де Моргана. Стройте сложные запросы текстового поиска при помощи унарных операторов + и -. Для прикладных программ, использующих API служб, преобразование этих выражений в язык запросов исследования информации выполняется при помощи классов из `com.ibm.mm.sdk.common.infomining.DKIKFWebQueryConverter`. Для программ, использующих функции bean, применяется метод `CMBAAdvancedSearchService.convertWebQuery`.

## Запуск задачи сервера

Если вы хотите выполнять определенную последовательность задач из прикладной программы клиента, вы можете собрать из этих вызовов задачу сервера, один раз передать ее на сервер, а затем сколько угодно часто вызывать эту задачу сервера с клиента. Это максимально снижает уровень сетевого трафика и существенно повышает производительность.

Задача сервера - это объект, реализующий интерфейс `com.ibm.mm.sdk.common.infomining.DKIKFServerTask`. Объект определяется из прикладной программы клиента, задается на объекте служб при помощи метода `setServerTask` и затем выполняется на сервере при помощи метода `runServerTask`.

В приведенном ниже примере объект класса `AnalysisTask`, который реализует интерфейс задачи сервера, задается в службе и затем выполняется:

```

...
ikfService.setServerTask(new AnalysisTask(secondCatalog.getName()));
HashMap documentMap = new HashMap();
documentMap.put("PID1", DKIKFTextDocument.create("content1"));
documentMap.put("PID2", DKIKFTextDocument.create("content2"));
documentMap.put("PID3", DKIKFTextDocument.create("content3"));
Map recordMap = (Map)ikfService.runServerTask(documentMap);

Iterator pids = recordMap.keySet().iterator();
while(pids.hasNext())
{
 DKIKFRecord record = (DKIKFRecord)recordMap.get(pids.next());
 System.out.println(record.getPID());
 System.out.println(record.getValue("IKF_LANGUAGE"));
 System.out.println(record.getValue("IKF_SUMMARY"));
}
...

```

карта документов передается задаче сервера для обработки. Задача сервера возвращает карту записей для заданных документов, содержащую созданные метаданные (в данном случае язык документа и сводку).

Исходный код для примера задачи сервера:

```

public class AnalysisTask implements DKIKFServerTask {
 private String catalogName;
 private DKIKFSchema catalogSchema;

 public AnalysisTask(String catalogName) {
 this.catalogName = catalogName;
 }

 public Serializable runServerTask(DKIKFService ikfService, Serializable argument)
 throws DKIKFServerTaskException {
 try {
 //the schema is retrieved only once
 if(catalogSchema == null) {
 catalogSchema = ikfService.getLibrary().getCatalog(catalogName).getSchema();
 }

 //подготовка аргумента, инструментов и карты для возврата
 Map documentMap = (Map)argument;
 DKIKFLanguageIdentifier languageIdentifier = new DKIKFLanguageIdentifier(ikfService);
 DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);
 HashMap recordMap = new HashMap();
 Iterator pids = documentMap.keySet().iterator();

 //создание записи для каждого pid
 while(pids.hasNext()) {
 String pid = (String)pids.next();
 DKIKFTextDocument document = (DKIKFTextDocument)documentMap.get(pid);
 DKIKFRecord record = DKIKFRecord.create(pid, catalogSchema);
 String language = languageIdentifier.analyze(document)[0].getLanguage();
 document.setLanguage(language);
 record.setValue("IKF_LANGUAGE", language);
 record.setValue("IKF_SUMMARY", summarizer.analyze(document).getSummary());
 recordMap.put(pid, record);
 }

 //возврат результатов
 return recordMap;
 }
 }
}

```

```

 }
 catch(Exception e) {
 throw new DKIKFServerTaskException(e);
 }
}
}

```

Задача сервера определяется с именем каталога, который содержит схему, необходимую для создания записи. Метод `runServerTask` получает схему только один раз. Для каждого из документов он создает запись, запускает инструменты для анализа документа и сохраняет созданные метаданные в записи. В заключение все созданные записи возвращаются вызвавшей программе (прикладной программе клиента).

#### Примечание

В приведенном выше примере кода возвращаются только метаданные, никаких записей в каталоге не создается.

## Пример прикладной программы Исследование информации на основе программ JSP

Программа исследования информации страниц сервера Java (JSP) выполняет в компоненте исследования информации поиск документов по категориям. Она выводит найденные документы в структуре категорий.

Пример программы JSP находится в каталоге:

|                |                                      |
|----------------|--------------------------------------|
| <b>Windows</b> | <CMBROOT>\samples\jsp\infomining\    |
| <b>AIX</b>     | /usr/lpp/cmb/samples/jsp/infomining/ |
| <b>Solaris</b> | /opt/IBMcmb/samples/jsp/infomining/  |

В этом каталоге находятся следующие файлы:

**advSearch.jsp**    Файл верхнего уровня примера.

В этой части приводится код обработки форм сложного поиска и инструкции форматирования форм (HTML). Кроме того, файл управляет выбором поднаборов данных. В нем содержатся инструкции инициализации специально для сложного поиска, в частности, сведения о доступности категорий, в которых можно выполнять поиск.

**catView.jsp**    В этом файле задается специальный код для поднаборов данных и инструкции форматирования (HTML) для поднаборов категорий возвращенных результатов. Файл содержит циклы для перебора возвращенных результатов, но основная часть - это инструкции форматирования.

|                    |                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>classes.jsp</b> | В этом файле задан код логических функций и код соединения функций bean. Он содержит только код Java. Реализуется простой класс структуры данных, используемый для просмотра возвращенных результатов. Кроме того, реализован обработчик событий для получения и обработки возвращенных результатов. Именно здесь задается основная обработка после возврата списка результатов сложного поиска. |
| <b>logon.html</b>  | Для выполнения примера требуется ввести учетную запись и каталог.<br><br>Вводимые учетная запись и каталог составлены из имени сервера, имени пользователя, пароля и имени каталога.                                                                                                                                                                                                             |

Исходный код - это пример использования функций bean исследования информации. В коде приводится описание его работы, например, определение функций beans, соединение их между собой, обработка возврата документов при помощи обработчика событий и тому подобное.

Для запуска программ JSP надо установить Enterprise Information Portal и компонент исследования информации. Кроме того, необходим сервер Web, поддерживающий программы JSP.

Более подробную информацию о программах JSP можно получить на странице: <http://java.sun.com/products/jsp/index.html>

## Установка JSP

Перед внедрением JSP убедитесь, что установлен и запущен сервер IBM WebSphere Application Server (WAS).

Права пользовательского доступа, необходимые для внедрения JSP:

- Для Windows: Полномочия администратора
- Для AIX: Привилегии пользователя root
- Для Solaris: Привилегии пользователя root

JSP внедряется как прикладная программа Web в каталог <WAS\_Home>\installedApps\JSP.ear\JSP.war. Если вы вносили изменения в JSP примера, замените страницы JSP в упомянутом каталоге на созданные вами. WAS перекомпилирует их автоматически.

Подробную информацию о конфигурировании сервера WebSphere Application Server для JSP смотрите в книге *Планирование и установка Enterprise Information Portal*.



---

## Замечания

Эта публикация разрабатывалась для продуктов и услуг, предлагаемых в США.

IBM может не предоставлять продукты, услуги или средства, описываемые в этом документе, в других странах. За информацией о продуктах и услугах, предоставляемых в вашей стране, обращайтесь к местному торговому представителю IBM. Ссылки на продукты, программы или услуги IBM не означают и не предполагают, что можно использовать только указанные продукты, программы или услуги IBM. Разрешается использовать любые функционально эквивалентные продукты, программы или услуги, если при этом не нарушаются права фирмы IBM на интеллектуальную собственность. Однако при этом пользователь сам несет ответственность за оценку и проверку работы с другими (не IBM) продуктами, программами и услугами.

IBM может располагать патентами или рассматриваемыми заявками на патенты, относящимися к предмету данной публикации. Получение этого документа не означает предоставления каких-либо лицензий на эти патенты. Запросы относительно лицензий направляйте по адресу:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

По поводу лицензий, связанных с использованием наборов двухбайтных символов (DBCS), обращайтесь в отдел интеллектуальной собственности IBM в вашей стране или направьте запрос в письменной форме по адресу:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

**Следующий абзац неприменим в Великобритании или в любой другой стране, где подобные оговорки противоречат местному законодательству: INTERNATIONAL BUSINESS MACHINES CORPORATION ПРЕДОСТАВЛЯЕТ ДАННУЮ ПУБЛИКАЦИЮ “КАК ЕСТЬ”, БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ, ВКЛЮЧАЯ (НО НЕ ОГРАНИЧИВАЯСЬ ТАКОВЫМИ) ПРЕДПОЛАГАЕМЫЕ ГАРАНТИИ СОБЛЮДЕНИЯ АВТОРСКИХ ПРАВ, РЫНОЧНОЙ ПРИГОДНОСТИ ИЛИ СООТВЕТСТВИЯ**

ОПРЕДЕЛЕННОЙ ЦЕЛИ. В некоторых странах для ряда сделок не допускается отказ от явных или предполагаемых гарантий; в таком случае данное положение к вам не относится.

В данной публикации могут встретиться технические неточности или типографские опечатки. В публикацию время от времени вносятся изменения, которые будут отражены в ее последующих изданиях. IBM оставляет за собой право в любое время вносить усовершенствования и/или изменения в описанные в этом замечании продукты и/или программы.

Ссылки на Web-сайты не-IBM приводятся только для вашего удобства и ни в коей мере не должны рассматриваться как рекомендации пользоваться этими Web-сайтами. Материалы на этих Web-сайтах не входят в число материалов по данному продукту IBM, и весь риск пользования этими Web-сайтами несете вы сами.

IBM может использовать или распространять информацию так, как сочтет нужным, без каких-либо обязательств с ее стороны.

Если обладателю лицензии на данную программу понадобятся сведения о возможности: (i) обмена данными между независимо разработанными программами и другими программами (включая данную) и (ii) совместного использования таких данных, он может обратиться по адресу:

IBM Corporation  
J46A/G4  
555 Bailey Avenue  
San Jose, CA 95141-1003  
U.S.A.

Такая информация может быть предоставлена на определенных условиях (в некоторых случаях к таким условиям может относиться оплата).

Лицензированная программа, описанная в данном документе, и все лицензированные материалы, доступные вместе с ней, предоставляются IBM на условиях Пользовательского соглашения IBM.

Все приводимые здесь данные о производительности были получены в контролируемой среде. Таким образом, результаты, полученные в других операционных средах, могут существенно отличаться от них. Часть измерений могла проводиться в системах на уровне разработки, и нет никаких гарантий, что на обычных компьютерах будут получены те же результаты. Более того, некоторые результаты могли быть получены путем экстраполяции. Реальные результаты могут быть другими. Пользователи должны проверить данные в своей собственной среде.

Информация о продуктах других фирм была получена от поставщиков этих продуктов, из их опубликованных объявлений или из других общедоступных источников. IBM не проверяла эти продукты и не может подтвердить точность характеристик, совместимость или иные заявления, связанные с продуктами других фирм. Вопросы, касающиеся характеристик продуктов других фирм (не IBM) следует адресовать поставщикам этих продуктов.

Все утверждения о будущих планах и намерениях IBM могут быть изменены или отменены без уведомлений, и описывают исключительно цели фирмы.

В этой публикации содержатся примеры данных и отчетов, используемых при выполнении текущих служебных задач. Чтобы проиллюстрировать эти задачи с максимальной наглядностью, в примерах используются имена физических лиц, названия компаний, фирм и продуктов. Все эти имена и названия являются вымышленными, и всякое сходство с именами, названиями и адресами, используемыми в реальной предпринимательской деятельности, являются не более чем совпадением.

#### ЛИЦЕНЗИЯ НА ПРАВО КОПИРОВАНИЯ:

Эта информация содержит примеры исходных текстов прикладных программ, которые иллюстрируют приемы программирования на различных платформах. Вы можете копировать, модифицировать и распространять эти программы примеров в любой форме без платы фирме IBM в целях разработки, использования, продажи или распространения прикладных программ, соответствующих программному интерфейсу платформы, для которой написаны примеры. Эти примеры не были тщательно протестированы при всех возможных условиях. Поэтому IBM не может гарантировать надежность, возможность обслуживания и работоспособность этих программ и не подразумевает таких гарантий. Разрешается копировать, изменять и распространять эти примеры программ в любой форме без оплаты фирме IBM для целей разработки, использования, сбыта или распространения прикладных программ, соответствующих интерфейсам прикладного программирования IBM.

---

### Торговые марки

Следующие термины являются товарными знаками корпорации International Business Machines в Соединенных Штатах и/или других странах:

|                                  |              |           |
|----------------------------------|--------------|-----------|
| IBM                              | DisplayWrite | PowerPC   |
| 400                              | e-business   | PTX       |
| Advanced Peer-to-Peer Networking | HotMedia     | QBIC      |
| AIX                              | Hummingbird  | RS/6000   |
| AIXwindows                       | ImagePlus    | SecureWay |
| APPN                             | IMS          | SP        |

|                        |               |                  |
|------------------------|---------------|------------------|
| AS/400                 | Micro Channel | VideoCharger     |
| C Set ++               | MQSeries      | Visual Warehouse |
| CICS                   | MVS/ESA       | VisualAge        |
| DATABASE 2             | NetView       | VisualInfo       |
| DataJoiner             | OS/2          | WebSphere        |
| DB2                    | OS/390        |                  |
| DB2 Universal Database | PAL           |                  |

Approach, Domino, Lotus, Lotus 1-2-3, Lotus Notes и SmartSuite - товарные знаки или зарегистрированные товарные знаки Lotus Development Corporation в Соединенных Штатах и в других странах.

Intel и Pentium - товарные знаки или зарегистрированные товарные знаки Intel Corporation Corporation в Соединенных Штатах и в других странах.

Microsoft, Windows и Windows NT - зарегистрированные товарные знаки Корпорации Microsoft в США и/или других странах.

Java и все основанные на Java товарные знаки и логотипы - товарные знаки или зарегистрированные товарные знаки Sun Microsystems, Inc. в США и/или других странах.

UNIX - зарегистрированный товарный знак The Open Group в США и в других странах.

Названия других компаний, продуктов и услуг могут быть товарными знаками или марками сервиса других фирм.

---

## Глоссарий

В данном глоссарии приводятся определения терминов и сокращений, используемые в этой системе. *Курсивом* выделены термины, определения которых представлены в других статьях данного глоссария.

### A

**ADSM.** Смотрите *Tivoli Storage Manager*.

**API.** Смотрите *интерфейс прикладного программирования*

**Audio/Video Interleaved (AVI).** Спецификация файла RIFF (*Resource Interchange File Format*), позволяющая чередовать в файле аудио- и видеоданные. Отдельные дорожки можно поместить в чередующиеся порции для воспроизведения или записи при поддержании последовательного доступа к файловому устройству.

**AVI.** Смотрите *Audio/Video Interleaved*.

### B

**BLOB.** Смотрите *двоичный большой объект*.

### C

**CGI.** Смотрите *Общий интерфейс шлюза*.

**CIF.** Смотрите *общий файл обмена*.

**CIU.** Смотрите *общий блок обмена*.

**Common Gateway Interface (CGI).** Стандарт для обмена информацией между Web-сервером и программами, которые по отношению к нему являются внешними. Такие внешние программы могут быть написаны на любом языке программирования, поддерживаемом операционной системой, в которой работает Web-сервер. Смотрите *сценарий CGI*.

### D

**DCA.** Смотрите *архитектура содержимого документа*.

**DDO.** Смотрите *динамический объект данных*.

**DTD.** Смотрите *определение типа документа*.

### E

**Extensible Markup Language (XML).**

Стандартный метаязык для определения языков разметки, основанный на SGML и являющийся его подмножеством. В XML исключены наиболее сложные и редко используемые части SGML, что упрощает написание программ и обработку типов документов, работу с структурированной информацией, ее передачу и совместное использование в различных компьютерных системах. Использование XML не требует программ высокой надежности для сложной обработки данных, что необходимо для SGML. XML разработан при содействии World Wide Web Consortium (W3C).

### H

**HTML.** Смотрите *язык гипертекстовой разметки*.

### I

**Image Object Content Architecture (IOCA).**

Набор структур, используемых для обмена изображениями и для их вывода.

**IOCA.** Смотрите *архитектура содержимого объектов изображений*.

### J

**JavaBeans.** Не зависящая от платформы технология программных компонентов, позволяющая строить многократно используемые

компоненты Java, называемые “компонентами bean”. После построения beans можно сделать доступными для использования другими разработчиками программного обеспечения или прикладными программами Java. При помощи JavaBeans разработчики программного обеспечения могут применять и компоновать beans в графической среде разработки с возможностями перетаскивания.

**Joint Photographic Experts Group (JPEG).** (1) Группа, разработавшая стандарт для сжатия оцифрованных естественных (с непрерывными переходами тонов) изображений. (2) Стандарт для неподвижных изображений, разработанный этой группой.

**JPEG.** Смотрите *Joint Photographic Experts Group*.

## L

**LAN.** Смотрите *локальная сеть*.

## M

**Machine-generated data structure (MGDS).** (1) Протокол формата структурированных данных IBM для передачи символьных данных между различными программами Content Manager ImagePlus for OS/390. (2) Данные, извлеченные из изображения и переведенные в формат обобщенного потока данных (GDS).

**MGDS.** Смотрите *компьютерная структура данных*.

**Mixed Object Document Content Architecture (MO:DCA).** Архитектура IBM, разработанная для обмена данными между программами как в единой среде, так и между средами.

**Mixed Object Document Content Architecture—Presentation (MO:DCA—P).** Поднабор архитектуры MO:DCA, которая используется, когда конверт содержит документы, посылаемые на рабочую станцию Content Manager ImagePlus for OS/390 для вывода или печати.

**MO:DCA.** *Mixed Object Document Content Architecture*

**MO:DCA—P.** *Mixed Object Document Content Architecture—Presentation*

**Multipurpose Internet Mail Extensions (MIME).** Смотрите *тип MIME*.

## O

**OLE.** Смотрите *связывание и встраивание объектов*.

## P

**PID.** Смотрите *постоянный идентификатор*.

## Q

**QBIC.** Смотрите *запрос по содержимому изображения*.

## R

**Resource Interchange File Format (RIFF).** Формат для хранения звука или графики при их воспроизведении на различных типах компьютерного оборудования.

**RIFF.** Смотрите *Resource Interchange File Format*.

**RMI-сервер (RMI server).** Сервер, обеспечивающий реализацию модели распределенных объектов RMI Java.

## S

**SMS.** Смотрите *управляемое системой хранение*.

## T

**Tivoli Storage Manager (TSM).** Продукт типа *клиент/сервер*, который дает возможность управлять хранением и предоставляет службы доступа к данным в гетерогенной среде. Он поддерживает различные способы взаимодействия, содержит средства управления, обеспечивающие резервное копирование и хранение файлов, а также позволяет производить планирование операций по резервному копированию.

**TOC.** Смотрите *содержание*.

**TSM.** Смотрите *Tivoli Storage Manager*.

## X

**XDO.** Смотрите *расширенный объект данных*.

**XML.** Смотрите *Extensible Markup Language*.

## A

**абстрактный класс (abstract class).** *Класс* объектно-ориентированного программирования, который представляет собой понятие; классы, полученные на его основе, представляют собой реализации этого понятия. Вы не можете сконструировать объект абстрактного класса; то есть, создать экземпляр абстрактного класса нельзя.

**анализ информации.** Автоматизированная процедура извлечения важной информации из текста (суммирование), поиска доминирующих тем в наборе документов (категоризация) и поиска нужных документов на основе мощного и гибкого механизма запросов.

**архив (archive).** Память для длительного хранения информации, обычно недорогая и медленная; часто находится территориально в удаленном месте для защиты от аварий оборудования и стихийных бедствий.

**архитектура содержимого документа (document content architecture, DCA).** Архитектура, которая гарантирует целостность информации для документов при обмене документами в корпоративной сети. DCA обеспечивает правила, определяющие форму и значение документов. В ней определяются редактируемая форма текста (изменяемая) и окончательная форма текста (неизменяемая).

**атрибут (attribute).** Единица данных, описывающая определенную характеристику или свойство (например, имя, адрес, возраст и т.п.) элемента; ее можно использовать для поиска этого элемента. У атрибута есть тип, которые описывает допустимые значения данных, хранящихся в этом атрибуте, и значение в допустимом диапазоне.

Пример атрибута - информация о файле в мультимедийной файловой системе, такая как заголовок, время воспроизведения или тип кодирования (MPEG1, H.263 и т.п.). Для Enterprise Information Portal смотрите также *атрибут объединения* и *собственный атрибут*.

**атрибут объединения (federated attribute).**

Категория метаданных Enterprise Information Portal, отображенная в *собственные атрибуты* на одном или нескольких *контент-серверах*. Например, атрибут объединения номер полиса может в Content Manager отображаться в *атрибут policy num*, а в Content Manager ImagePlus for OS/390 - в атрибут *policy ID*.

## B

**библиотечный клиент.** Компонент системы Content Manager, который поддерживает низкоуровневый программный интерфейс библиотечной системы. В библиотечный клиент входят API, составляющие часть комплекта разработчика программ.

**библиотечный объект (library object).** Смотрите *элемент*.

**библиотечный сервер.** Компонент системы Content Manager, который хранит и обрабатывает запросы об *элементах* и управляет этими запросами.

## B

**выдвижение (staging).** Процесс перемещения хранящегося *объекта* с неподключенного или низкоприоритетного устройства на подключенное или высокоприоритетное, обычно по требованию системы или по заказу пользователя. Когда пользовательские требования на объект сохраняются в постоянной памяти, рабочая копия объекта записывается в *кэш менеджера ресурсов* (на сцену).

**вызов удаленного метода (Remote Method Invocation - RMI).** Набор API, обеспечивающий распределенное программирование. Объект в одной системе Java Virtual Machine (JVM) может вызывать методы для объектов в других JVM.

**высвободить (release).** Отменить критерий приостановки для *элемента*. Высвобождение приостановленного элемента произойдет, если будет достигнуто соответствие критериям или если пользователь с соответствующими полномочиями перезапишет критерии и вручную высвободит элемент.

## Г

**Гбайт (GB).** Смотрите *гигабайт*.

**гигабайт (gigabyte, GB).** (1) Для памяти процессора, реальной и виртуальной памяти, а также для пропускной способности канала -  $2^{30}$  или 1073741824 бита. (2) Для объема дисковой памяти и объема передаваемой информации - 1 000 000 000 байт.

**группа атрибутов (attribute group).** Объединение из одного или нескольких *атрибутов* для удобства работы с ними. Например, группа атрибутов Адрес может содержать атрибуты Улица, Город, Область и Почтовый индекс.

**группа пользователей (user group).** Группа из одного или нескольких отдельных *пользователей*, имеющая единое групповое имя.

**группа хранения (storage group).** Связывает систему хранения с классом хранения.

## Д

**двоичный большой объект (BLOB, binary large object).** Последовательность байтов, объем которой находится в диапазоне от 0 байт до 2 Гбайт. С такой строкой не связана ни кодовая страница, ни набор символов. В виде BLOB хранятся объекты изображений, аудио- и видеообъекты.

**динамический объект данных (dynamic data object - DDO).** В прикладных программах: общий способ представления сохраненного объекта, который позволяет перемещать этот объект в место хранения и из него.

**документ (document).** *Элемент*, который хранится, вызывается и передается из системы Content Manager в другую систему или

пользователю как отдельная единица. Ожидается, что элемент с *семантическим типом* документ содержит информацию, которая образует документ, хотя и не обязательно реализует при этом модель документа Content Manager.

Элемент, созданный с документным типом элементов (конкретная реализация модели документа Content Manager), должен содержать части документа. Документные типы элементов можно использовать для создания элементов с семантическим типом документов или папок.

Части документа могут иметь разные типы содержимого, включая, например, текст, изображения и электронные таблицы.

**дочерний компонент (child component).**

Дополнительный второй или низший уровень иерархического *типа элементов*. Каждый дочерний компонент непосредственно связан с вышестоящим уровнем.

## З

**заказчик (patron).** Термин, используемый в API Content Manager для *пользователя*.

**запрос по содержимому изображения (query by image content - QBIC).** Технология запроса, позволяющая искать не текст, а визуальное содержание изображения, называемое характеристиками. Используя QBIC, можно искать объекты по таким визуальным характеристикам, как цвет и текстура.

## И

**изолированная система (stand-alone system).**

Сконфигурированная система Content Manager, где все компоненты Content Manager установлены на одном персональном компьютере.

**индексировать (index).** Добавлять или редактировать значения атрибутов, идентифицирующих определенный *элемент* или *объект*, чтобы иметь возможность получать его позже.

**индексный класс (index class).** Смотрите *тип элемента*.



### **интерфейс прикладного программирования (application programming interface - API).**

Программный интерфейс, обеспечивающий возможность взаимодействия приложений друг с другом. API - это набор конструкций или операторов языка программирования, которые могут добавляться в код прикладной программы, чтобы обеспечить выполнение специальных функций и служб, предоставляемых базовой лицензионной программой.

**итерация (iterator).** Класс конструкций, который позволяет перебирать объекты в наборе по одному.

## **К**

**категория (category).** Смотрите *тип элемента*.

**класс (class).** В объектно-ориентированной разработке или программировании: модель или шаблон, которые можно инициировать для создания объектов с общим определением и, следовательно, с общими свойствами, операциями и режимами. Объект является экземпляром класса.

**классификация типов элементов (item type classification).** Категоризация в пределах *типа элементов* для дальнейшей идентификации *элементов* данного типа. Все элементы одного типа имеют одну и ту же классификацию типов элементов.

В Content Manager задана следующая классификация типов элементов: *папка, документ, объект, видео, изображение и текст*; пользователи могут определять свои собственные классификации типов объектов.

**класс содержимого (content class).** Смотрите *тип MIME*.

**класс соединителя (connector class).** *Класс* объектно-ориентированного программирования, который обеспечивает стандартный доступ к собственным API определенных *серверов содержимого*.

**класс управления (management class).** Термин, используемый в API для *миграционных правил*.

**класс хранения (storage class).** Идентифицирует тип накопителя, на котором сохраняется объект. Он не связан непосредственно с физическим положением объекта; однако он прямо связан с *менеджером устройств*. Возможные классы хранения:

DASD  
Жесткий диск  
Оптический  
Поточный  
Лента  
TSM

**клиент/сервер (client/server).** Модель взаимодействия при распределенной обработке данных, при которой программа на одном узле посылает требования программе на другом узле и ждет ее ответа. Программу, посылающую требование, называют клиентом, а отвечающую программу - сервером.

**ключевое поле (key field).** Смотрите *атрибут*.

**комбинированный поиск (combined search).** Запрос, в котором сочетаются следующие типы поиска: *параметрический* поиск, текстовый поиск или поиск изображений.

**компонент (component).** Общий термин для *корневого компонента* и *дочернего компонента*.

**конструкция (constructor).** В языках программирования: метод, имя которого совпадает с именем класса и который используется для создания и инициализации объектов этого класса.

**контейнер (container).** Элемент пользовательского интерфейса, в котором содержатся объекты. В *менеджере папок* - *объект*, который может содержать другие папки или документы.

**координатор рабочих потоков (workflow).** В рабочем потоке ранних версий Content Manager - пользователь, которому направляется уведомление о том, что *рабочий элемент* в *рабочем потоке* не был обработан в течение некоторого заданного времени. Этот пользователь

выбирается для конкретной *группы пользователей* либо при создании рабочего потока.

**корневой компонент (root component).** Первый или единственный уровень иерархического *типа элементов*, состоящий из определенных системой и определенных пользователем *атрибутов*.

**критерий поиска (search criteria).** В Content Manager - значения *атрибутов*, используемые для получения хранящегося *элемента*. В Enterprise Information Portal - конкретные поля, заданные администратором в *шаблоне поиска* для ограничения или дальнейшего определения возможностей выбора у *пользователей*.

**курсор (cursor).** Именованная управляющая структура, которая в прикладной программе позволяет указать определенную строку в некотором упорядоченном наборе строк. Курсор позволяет получать строки из этого набора.

**кэш (cache).** Буфер специального назначения, меньше и быстрее основной памяти; используется для хранения копии часто требуемых данных. Использование кэша сокращает время доступа, но может увеличить требования к памяти. Смотрите также *кэш менеджера ресурсов* и *сетевой кэш*.

**кэш менеджера ресурсов (resource manager cache).** Область рабочей памяти для *менеджера ресурсов*. Другое ее название - *цена*.

**кэш менеджера ресурсов (staging area).** Область рабочей памяти для *менеджера ресурсов*. Другое название - *кэш менеджера ресурсов*.

**кэш сервера объектов (object server cache).** Смотрите *кэш менеджера ресурсов*.

## Л

**локальная сеть (local area network, LAN).** Сеть, в которой набор устройств соединен друг с другом для передачи информации; может быть соединена с сетью большего размера.

## М

**макет (overlay).** Набор предопределенных данных (линий, теней, текста, рамок или логотипов), объединяемых при печати с переменными данными на странице.

**менеджер папок.** Модель Content Manager для управления такими данными, как электронные документы и папки. API менеджера папок можно использовать как первичный интерфейс между вашими прикладными программами и контент-серверами Content Manager.

**менеджер ресурсов.** Компонент системы Content Manager, который управляет *объектами*. На эти объекты ссылаются *элементы*, которые хранятся на *библиотечном сервере*.

**менеджер соединений (connection manager).** Компонент Content Manager, который помогает поддерживать соединения с сервером библиотеки вместо того, чтобы устанавливать новое соединение для каждого запроса. У менеджера соединений есть свой интерфейс прикладного программирования.

**менеджер устройств (device manager).** В системе Content Manager - интерфейс между *менеджером ресурсов* и одной или несколькими физическими устройствами.

**метод (method).** В Java-разработках или в Java-программировании: программный компонент, который реализует режим, заданный операцией. Синоним этого термина в C++ - функция элемента.

**миграционные правила (migration policy).** Задаваемые пользователем правила перемещения *объектов* из одного *класса хранения* в другой. Описывают параметры хранения и перемещения по классам группы объектов в иерархии хранения.

**минимальный клиент (thin client).** Клиент с малым объемом установленных программных средств или вообще без них, но имеющий доступ к программным средствам, которыми управляют и которые предоставляют соединенные с ним сетевые серверы. Минимальные клиенты

представляют собой альтернативу полнофункциональным клиентам (например, рабочим станциям).

**монтировать (mount).** Переводить носитель данных в рабочее состояние.

**мощность (cardinality).** Число строк в таблице базы данных.

**мультимедиа (multimedia).** Объединение различных элементов (текста, графики, звука, неподвижных изображений, видео, анимации) для воспроизведения и управления ими при помощи компьютера.

**мультимедийная файловая система (multimedia file system).** *Файловая система*, оптимизированная для хранения и считывания видео- и аудиофайлов.

**мусорщик (purger).** Функция менеджера ресурсов, которая удаляет объекты из системы.

## N

**набор привилегий (privilege set).** Совокупность привилегий для работы с компонентами и функциями системы. Администратор дает наборы привилегий пользователям (задаваемым ID) и группам пользователей.

**надкласс (superclass).** *Класс*, производным которого является какой-то другой класс. Между классом и надклассом могут находиться один или несколько классов.

## O

**обмен (interchange).** Возможность импорта или экспорта изображения вместе с его индексом из одной системы Content Manager ImagePlus for OS/390 в другую систему ImagePlus с использованием *общего файла обмена* или *общего блока обмена*.

**обработчик пользователя (user exit).** Точка в поставляемой IBM программе, в которой управление передается подпрограмме обработчика пользователя.

**общий блок обмена (common interchange unit, CIU).** Независимый блок передачи для общего файла обмена (CIF). Это часть CIF, определяющая отношение с принимающей базой данных. CIF может содержать несколько CIU.

**общий файл обмена (common interchange file, CIF).** Файл, содержащий один поток данных ImagePlus Interchange Architecture (IPIA).

**объединенный поиск (federated search).** Сгенерированный в Enterprise Information Portal запрос, обеспечивающий одновременный поиск данных на одном или нескольких *контент-серверах*, которые могут быть разнородными.

**объединенный склад данных (federated datastore).** Виртуальное представление для обозначения любого числа указанных *контент-серверов*, например, серверов Content Manager.

**объединенный текстовый индекс (federated text index).** Объект метаданных Enterprise Information Portal, отображенный на один или несколько *собственных текстовых индексов* на одном или нескольких *контент-серверах*.

**объект (object).** Любое цифровое содержимое, которое пользователь может сохранять, получать и использовать как единое целое, например, изображения JPEG, аудиофайлы MP3, видеофайлы AVI и фрагменты текста из книг.

**объект объединения (federated entity).** Объект метаданных Enterprise Information Portal, состоящий из *атрибутов объединения* и (необязательно) связанный с одним или несколькими *текстовыми индексами объединения*.

**определение сервера (server definition).** Характеристики конкретного *контент-сервера*, которые позволяют однозначно его идентифицировать в Enterprise Information Portal.

**определение типа документа (document type definition, DTD).** Правила, определяющие структуру для определенного класса документов XML. Определение типа документов определяет структуру с элементами, атрибутами и обозначениями и задает ограничения, как каждый элемент, атрибут и обозначение могут

использоваться с определенным классом документов. DTD аналогично схеме базы данных и полностью описывает структуру определенного языка разметки.

**определение типа сервера (server type definition).** Список заданных администратором характеристик, которые позволяют однозначно идентифицировать настроенный сервер определенного типа в Enterprise Information Portal.

**основные атрибуты (base attributes).** Набор индексов, связанных с каждым *объектом*. У любого объекта Content Manager есть базовые атрибуты.

**отображение пользователя (user mapping).** Связь между ID пользователей и паролями Enterprise Information Portal и соответствующими ID пользователей и паролями на одном или нескольких контент-серверах. Отображения пользователей обеспечивают единую регистрацию в Enterprise Information Portal и на нескольких контент-серверах.

## П

**пакет (package).** Собрание родственных *классов* и интерфейсов, обеспечивающих защиту доступа и управление пространством имен.

**папка (folder).** *Элемент* любого типа *элементов* (независимо от классификации), с *семантическим типом* папка. Любой элемент с семантическим типом папка содержит особые функциональные возможности папки, обеспечиваемые Content Manager, в дополнение ко всем возможностям нересурсного элемента и дополнительным возможностям классификации типа элементов (например, *документ* или ресурсный элемент). Папки могут содержать любое число элементов любого типа, в том числе документы и подпапки. Папки индексируются по *атрибутам*.

**параметрический поиск (parametric search).** Запрос информации об *объектах*, созданный на основе *свойств* этих объектов.

**перенос (migration).** (1) Процесс перемещения данных и источников из одной компьютерной системы в другую без их преобразования

(подобного, например, преобразованию при перемещении в новую операционную среду). (2) Установка новой версии или выпуска программы для замены прежней версии или выпуска.

**перечень сервера (server inventory).** Полный список *собственных объектов* и *собственных атрибутов* на указанных контент-серверах.

**подкласс (subclass).** *Класс*, который является производным от другого класса. Между классом и подклассом могут находиться один или несколько классов.

**подключенный (inline).** В Content Manager: объект на включенном, но в данный момент не смонтированном диске. Сравните со *смонтированным*.

**поднабор индексного класса (index class subset).** В ранних версиях Content Manager - представление *индексного класса*, используемое прикладной программой для хранения, вызова и вывода папок и объектов.

**подпрограмма обработчика пользователя (user exit routine).** Написанная пользователем подпрограмма, которой передается управление в заранее заданных точках вызова *обработчика пользователя*.

**пользователь (user).** Лицо, которому требуются службы Content Manager. Обычно этот термин применяется к пользователям клиентских программ, а не к разработчикам программ, применяющим API Content Manager. В Enterprise Information Portal - тот, кто идентифицируется программой управления Enterprise Information Portal.

**постоянный идентификатор (persistent identifier - PID).** Идентификатор, обеспечивающий уникальную идентификацию *объекта* независимо от того, где он хранится. PID состоит из ID элемента и его местонахождения.

**построить изображение (render).** Преобразовать данные, содержание которых обычно не является визуальным, для вывода в виде изображения. В Content Manager документы текстового процессора для вывода можно преобразовать в изображение.

**потокковые данные (streamed data).** Любые данные, пересылаемые через сетевое соединение с определенной скоростью. Поток может состоять из данных одного типа или представлять собой комбинацию типов. Скорости данных, измеряемые в битах в секунду, различны для различных типов потоков и сетей.

**представление индексного класса (index class view).** В ранних версиях Content Manager - термин, используемый в API для *поднаборов индексных классов*.

**привилегия (privilege).** Право получать доступ к указанному *объекту* указанным способом. К привилегиям относятся права на создание, удаление и выбор объектов, хранящихся в системе. Привилегии назначаются администратором.

**приостановить (suspend).** Удалить *объект* из *рабочего потока* и задать критерий приостановки для его последующей активации. Последующая активация объекта позволяет продолжить его обработку.

**программа клиента.** Программа, написанная с использованием API Content Manager с целью пользовательской настройки интерфейса. Программа, написанная с использованием Интернет- или объектно-ориентированных API для обращения к *контент-серверам* с системы Enterprise Information Portal.

**Программа клиента для Windows (Client Application for Windows).** Полная система управления объектами, входящая в состав Content Manager и написанная с использованием API Content Manager. Поддерживает создание документов и папок, хранение, вывод, обработку и управление доступом. Ее можно настраивать при помощи подпрограмм обработчиков пользователя и вызывать ее части через API.

**процесс маршрутизации документов (document routing process).** В Content Manager - последовательность *рабочих шагов* и правила, управляющие этими шагами, определяющие последовательность передачи *документа* или *папки* при обработке.

## Р

**рабочее состояние (work state).** Состояние отдельного *рабочего элемента*, документа или папки.

**рабочий комплект (workbasket).** Собрание документов или папок, обрабатываемых или ожидающих обработки. В определение рабочего комплекта входят правила, управляющие выводом, состоянием и защитой его содержимого.

**рабочий пакет (work packet).** В Enterprise Information Portal Версии 7.1 - собрание документов, направляемое из одной точки в другую. Пользователи получают доступ к рабочим пакетам и работают с ними посредством *рабочих списков*.

**рабочий поток (workflow).** В ранних версиях Content Manager - последовательность *рабочих комплектов*, которые *документ* или *папка* проходят в процессе обработки. В Enterprise Information Portal - последовательность *рабочих шагов* и правила, управляющие этими шагами, определяющие последовательность передачи *рабочего пакета*, документа или папки при обработке.

Например, принятие страхового иска описывает процесс действий, которые надо выполнить с отдельным страховым иском, чтобы принять его.

**рабочий список (worklist).** Собрание *рабочих элементов*, документов или папок, назначаемых пользователю.

**рабочий элемент (work item).** В рабочем потоке ранних версий Content Manager и расширенном рабочем потоке Enterprise Information Portal - рабочая операция, выполняемая в пределах *рабочего потока*.

**рабочий этап (work step).** Отдельная точка в *рабочем потоке* или в *процессе маршрутизации документов*, через которую должны проходить отдельные *рабочие элементы*, документы или папки.

**ранг (rank).** Целочисленное значение, обозначающее релевантность данной части по

отношению к результатам запроса. Чем выше ранг, тем ближе соответствие.

**расширенный объект данных (extended data object - XDO).** В прикладных программах: общий способ представления сохраненного комплексного мультимедийного *объекта*, который позволяет перемещать этот объект в место хранения и из него. XDO обычно содержатся в *DDO*.

## С

**свойство (property).** Характеристика *объекта*, которая описывает его. Свойство можно изменить или модифицировать. Режим ввода - пример свойства объекта.

**связывание и встраивание объектов (Object Linking and Embedding -OLE).** Спецификация Microsoft для связывания и встраивания программ с целью их активации из других программ.

**связь (link).** Направленное взаимоотношение между двумя *элементами*: исходным элементом и элементом назначения. Набор связей можно использовать для моделирования ассоциаций "один со многими". Сравните со *ссылкой*.

**семантический тип (semantic type).** Использование или правила для *элемента*. В Content Manager заданы три семантических типа: основной, комментарий и примечание; пользователи могут также определять свои собственные семантические типы.

**сервер мультимедиа (media server).** Компонент системы Content Manager на платформе AIX, используемый для хранения видеофайлов и доступа к ним.

**сервер объектов.** Смотрите *менеджер ресурсов*.

**сервер содержимого (content server).** Программная система, в которой хранятся мультимедийные данные и бизнес-данные, а также соответствующие им метаданные, которые необходимы пользователям для работы с этими данными. Примеры контент-серверов: Content Manager и Content Manager ImagePlus for OS/390.

**сервер утилит (utility server).** Компонент Content Manager, используемый утилитами баз данных для целей планирования. Сервер утилит конфигурируется при конфигурировании *менеджера ресурсов* или *библиотечного сервера*. У каждого менеджера ресурсов и у каждого библиотечного сервера есть свой сервер утилит.

**сетевой кэш (LAN cache).** Область временного хранения на локальном *менеджере ресурсов*, содержащая копии объектов, которые хранятся на удаленном менеджере ресурсов.

**сетевой табличный файл (network table file).** Текстовый файл, содержащий специфичную для системы информацию конфигурации для каждого узла системы Content Manager. У каждого узла системы должен быть свой сетевой табличный файл, где указаны узлы и списки, с которыми он должен соединяться.

Этот файл носит имя FRNOLINT.TBL.

**символ подстановки (wildcard character).** Специальный символ, такой как звездочка (\*) или знак вопроса (?), который можно использовать для представления одного или нескольких символов. Символ подстановки может заменять любой символ или группу символов.

**система хранения (storage system).** Общий термин для части Content Manager, служащей для хранения информации. Смотрите *том TSM, устройство хранения мультимедиа* и *том*.

**склад данных (datastore).** (1) Общий термин для обозначения места (например, системы базы данных, файла или каталога) хранения данных. (2) В прикладной программе - виртуальное представление *контент-сервера*.

**смонтированный (mounted).** В Content Manager - объект на включенном и *смонтированном* в данный момент диске. Сравните с *подключенным*.

**собрание (collection).** Группа объектов со сходным набором правил управления, помещенная в хранилище.

**собрание объединения (federated collection).** Группа объектов, полученная в результате *объединенного поиска*.



**собственные атрибуты (native attributes).**

Характеристики объекта, управление которыми осуществляется на определенном *контент-сервере* и которые присущи только этому контент-серверу. Например, на контент-сервере Content Manager собственным атрибутом может являться *ключевое поле* `policy num`, в то время как на контент-сервере Content Manager OnDemand собственным атрибутом может являться поле `policy ID`.

**собственный объект (native entity).** *Объект*, управление которым осуществляется на определенном *контент-сервере* и который состоит из *собственных атрибутов*. Например индексные классы Content Manager - это собственные объекты, составленные из *ключевых полей* Content Manager.

**собственный текстовый индекс (native text index).**

Индекс текстовых *элементов*, управление которыми осуществляется на определенном *контент-сервере*. Например, один текстовый индекс поиска на контент-сервере Content Manager.

**состояние рабочего потока (workflow state).**

Состояние *рабочего потока* в целом.

**список действий (action list).** Одобренный список действий, заданный системным администратором или другим *координатором рабочего потока*, которые пользователю разрешено выполнять в *рабочем потоке* или в процессе маршрутизации документа.

**список управления доступом (access control list).**

Список, включающий в себя один или несколько ID пользователей или групп пользователей и присвоенных им *привилегий*. Списки управления доступом применяются для управления доступом пользователей к *элементам* и *объектам* системы Content Manager. Списки управления доступом применяются для управления доступом пользователей к *шаблонам поиска* в системе Enterprise Information Portal.

**ссылка (reference).** Однонаправленная одиночная ассоциация между *корневым* или *дочерним компонентом* и другим *корневым компонентом*. Сравните со *связью*.

**строка запроса (query string).** Символьная строка, которая задает свойства и значения свойств для запроса. Можно создать строку запроса в приложении, а затем передать ее запросу.

**сценарий CGI (CGI script).** Компьютерная программа, выполняющаяся на Web-сервере и использующая стандарт *Common Gateway Interface (CGI)* для выполнения задач, которые Web-сервер обычно не выполняет (например, получение доступа к базе данных и обработка форм). Сценарий CGI представляет собой программу CGI, написанную на таком языке сценариев, как Perl.

## Т

**таблица содержимого (table of contents, TOC).**

Список *документов* и *папок*, содержащихся в папке или в *рабочем комплекте*. Результаты поиска выводятся в виде оглавления папки.

**Тип MIME (MIME type).** Стандарт Интернета для идентификации типа объекта, передаваемого по Интернету. Типы MIME включают в себя несколько вариантов аудио-, графических и видеообъектов. У каждого объекта есть тип MIME.

**тип элементов (item type).** Шаблон для определения и последующего поиска *элементов*, состоящий из *корневого компонента*, нескольких возможных *дочерних компонентов* и классификации.

**том TSM (TSM volume).** Логическая область хранения данных, которой управляет *Tivoli Storage Manager*.

**том (volume).** Понятие, соответствующее реальному физическому устройству или носителю, где хранятся объекты системы.

## У

**уборщик (destager).** Функция *менеджера ресурсов* Content Manager, которая убирает объекты со *сцены* в следующий класс хранения, определяемый *миграционными правилами* для объекта.

**унифицированный указатель ресурсов (uniform resource locator, URL).** Последовательность символов, представляющая информационные ресурсы на компьютере или в сети, например, в Интернете. Эта последовательность символов включает в себя сокращенное имя протокола, используемого для доступа к информационному ресурсу, а также информацию, используемую этим протоколом для поиска информационного ресурса. Например, в Интернете используются такие сокращенные имена протоколов доступа к различным информационным ресурсам: http, ftp, gopher, telnet и news.

**управление доступом (access control).** Функция, благодаря которой доступ к тем или иным функциям и сохраненным *объектам* предоставляется только авторизованным пользователям и только разрешенными способами.

**управляемое системой хранение (system-managed storage - SMS).** Подход, используемый в Content Manager для управления хранением. Система определяет размещение объекта и автоматически управляет его резервным копированием, перемещением, защитой и отводимым ему пространством.

**устройство хранения мультимедиа (media archiver).** Физическое устройство, используемое для хранения данных потокового типа (аудио и видео). Пример устройства хранения мультимедиа - VideoCharger.

## Ф

**файл README (README file).** Файл, который необходимо просмотреть перед установкой или запуском программы, к которой прилагается этот файл. Обычно файл README содержит последнюю информацию о продукте, информацию по установке или советы по использованию программного продукта.

**файловая система (file system).** В AIX - способ разбиения жесткого диска на разделы для хранения данных.

**формат данных (data format).** Смотрите *тип MIME*.

**функция миграции (migrator).** Функция *менеджера ресурсов*, которая в соответствии с *миграционными правилами* перемещает объекты в следующий *класс хранения*, когда это предписывается правилами.

## Х

**характеристика (feature).** Информация о содержании изображения, которая хранится на сервере поиска изображений. Кроме того, свойство изображения, которое программы поиска изображений используют для определения соответствий. Используется четыре характеристики *QBIC* - усредненный цвет, гистограмма цветов, позиционный цвет и текстура.

**хронологический журнал (history log).** Файл, где хранится запись действий для *рабочего потока*.

**хэндл (handle).** Символьная строка, соответствующая объекту и используемая для вызова этого объекта.

## Ч

**часть (part).** Смотрите *объект*.

## Ш

**шаблон поиска (search template).** Форма, состоящая из *критериев поиска*, разработанных администратором для определенного типа объединенного поиска. Администратор также указывает *пользователей* и *группы пользователей*, которые могут получать доступ к данному шаблону поиска.

**шлюз (gateway).** Функциональное устройство, связывающее две компьютерные сети с разными архитектурами. Шлюз соединяет сети или системы с разными архитектурами. Мост соединяет сети или системы с одинаковыми или похожими архитектурами.



## Э

**элемент (element).** *Объект, который менеджер списков размещает для приложения.*

**элемент данных (item).** В Content Manager - общий термин для экземпляра *типа элементов*. Например, элементом может быть *папка, документ*, видеофайл или изображение. Общий термин для обозначения наименьшей единицы информации, которой управляет сервер Enterprise Information Portal. У каждого элемента есть идентификатор. Например, элементом данных может быть *папка* или *документ*.

## Я

**язык гипертекстовой разметки (Hypertext Markup Language - HTML).** Язык разметки, соответствующий стандарту SGML, который в первую очередь предназначен для поддержки вывода на экран текстовой или графической информации, содержащей гипертекстовые связи.



# Индекс

## A

addObject 469, 475  
addToFolder 480  
AIX  
    запуск сервера RMI 35  
    совместно используемые объекты  
        для C++ 36  
AllPrivSet 168, 171  
anyA, DKAny 117  
API открытого управления  
    документами (open document  
    management - ODMA) 419  
API служб  
    генерация сводки документа 621  
    запуск задачи сервера 625  
    извлечение информации из  
        документа 621  
    использование инструментов  
        исследования информации 619  
Исследование информации 614  
категоризация 622  
кластеризация 622  
определение языка  
    документа 620  
поиск 624  
соединение 615  
создание записей 622  
создание текстового  
    документа 620  
сохранение метаданных 622  
управление библиотекой 615  
управление каталогами 615  
управление таксономиями 615  
фильтрация документа 620  
ASCENDING 253  
ATTRONLY, ImagePlus for  
    OS/390 409  
AUTOCOMMIT, реляционные базы  
    данных 456

## B

BasicExpression 255  
Batch Entry System, FileNET 449  
BeanInfo 509  
BETWEEN 247, 253

BLOB,  
    См. двоичный большой объект  
    (BLOB)

## C

C++  
    DKAny 114  
    DKConstant.h 31  
    библиотеки 36  
    измененные классы 11  
    константы 43  
    настройка среды 35  
    новые классы 10  
    поддержка XML 31  
    совместно используемые объекты  
        для AIX 36  
    соединители 6  
    строки конфигурации для  
        реляционных баз данных 63  
    файлы DLL 36  
    файлы свойств сообщений об  
        ошибках 482  
    эскейп-последовательности 252  
cc2mime.ini 456  
CC2MIMEFILE, реляционные базы  
    данных 456  
CCSID 324  
changePassword 179, 468  
checkedOutUserid 481  
checkIn, dkDatastoreExt 481  
checkOut, dkDatastoreExt 481  
CLOB,  
    См. символьный большой объект  
    (CLOB)  
cmb81.jar 501  
CMBAnnotationPropertiesInterface 541  
CMBAnnotationServices 532  
CMBAnnotationServicesCallbacks 532  
CMBAnnotationSet 533  
cmbcc2mime.ini 553  
CMBCC2MIME.INI 446  
cmbcc2mime.ini.samp 446  
cmbclient.ini 5, 553  
cmbcm81.jar 465  
cmbcm81.lib 36  
cmbcm817.dll 465  
cmbcm817d.dll 465  
cmbcm81d.lib 36  
CMBConnection 398, 502, 544  
cmbcs.ini 503, 553  
CMBDataManagement 502, 544  
cmbdes.ini 428  
CMBDocumentServices 536, 544  
CMBDocumentViewer 513, 521  
    прекращение 521  
    спецификации 521  
CMBFolderViewer 513, 519  
CMBGenericDocViewre 533  
CMBItem 544  
CMBItemAttributesEditor 522  
CMBLogonPanel 513, 514  
CMBObject 547  
CMBPage 549  
CMBPageAnnotation 540  
CMBQueryService 502, 544  
cmbregist81.bat 35  
cmbregist81.sh 35  
CMBSchemaManagement 544  
CMBSearchPanel 517  
CMBSearchResults 544  
CMBSearchResultsView 398  
CMBSearchResultsViewer 513, 517  
CMBSearchTemplate 544  
CMBSearchTemplateList 398, 513, 515  
CMBSearchTemplateViewer 398, 513,  
    516  
cmbservlet.properties 550  
CMBServletAction 551  
cmbservletjsp.properties 550  
CMBSTCriterion 545  
CMBStreamingDocServices 532  
CMBStreamingDocServicesCallbacks 532  
cmbsvclient.ini 553  
cmbsvcs.ini 553  
CMBTraceLog 544  
CMBUserManagement 544  
cmbview81.jar 532  
CMBViewerConfiguration.properties 532  
CMCOMMON 5  
CMCOMMON\_URL 5  
cmvcmenv.properties 5  
COBRA  
    Persistent Data Service (PDS) 14  
    Persistent Object Service 14  
com.ibm.mm.sdk.client 32  
com.ibm.mm.sdk.server 32  
CompareOperator 255  
Comparison 254  
concatReplace 477

contains-text, текстовый поиск 233  
contains-text-basic, текстовый  
поиск 232

## CONTENT

FileNET 454

ImagePlus for OS/390 409

## Content Manager for AS/400

введение 410

запуск запроса 411

индексные классы 410

отображение объединения 20

получение списков объектов и  
атрибутов 410

## Content Manager Версия 8

eClient 6

XML 100

атрибуты 161

библиотечный сервер 159

введение 159

диаграмма компонентов 160

документные части 162

документы 162, 165

дочерние компоненты 162

задание и получение атрибутов  
элементов 193

запрос 227

изменение атрибутов  
элементов 193

изменение документов 264

корневые компоненты 162

маршрутизация документов 272

менеджер ресурсов 159

набор привилегий 168

нересурсные элементы 162

объекты 164

определение ресурсного типа  
элементов 195

основные понятия 160

отображение объединения 20  
папки 165

планирование прикладной  
программы 172

подсоединение 177

поиск элементов 199

получение документа 266

получение списка атрибутов для  
типа элементов 187

получение списка типов  
элементов 182

получение элементов 199

понятие запроса поиска 227

понятие языка запросов 227

пример сценария страховой  
компании 175

примеры 174

## Content Manager Версия 8

*(продолжение)*

примеры запросов 236

работа с менеджером

ресурсов 255

работа с папками 206

работа с управлением

доступом 216

резервирование и активирование  
элементов 202

ресурсные элементы 162

связи 164

связывание элементов 214

сервер SMS (System Managed

Storage - хранение, управляемое  
системой) 160

согласованность библиотечного  
сервера и менеджера

ресурсов 268

создание атрибутов 183

создание документа 261

создание контент-сервера 177

создание пароля 179

создание прикладной  
программы 175

создание типов элементов 179

ссылки 164

типы элементов 162

транзакции 268

удаление документа 266

управление версиями 165

управление группами

атрибутов 185

управление документами 257

управление доступом 166

группы пользователей 170

списки доступа 170

управление привилегиями

доступа 168

элементы 161

## Content Manager, ранние версии

eClient 6

введение 302

изменение папок 312

изменение частей 310

обработка больших  
объектов 302

отображение объединения 20

поиск изображений 7, 346

постоянный идентификатор  
(PID) 303

представление документов 303

представление папок 304

представление частей 303

рабочий поток 373

## Content Manager, ранние версии

*(продолжение)*

хранение XML 104

createChildDDO 189

createDDO 52

## D

data\_id 59

databaseNameStr 178

DataJoiner,

См. DB2 DataJoiner

DATASOURCE 428

## DB2

ассистент конфигурирования

клиента 5, 35

поддержка клиента 5, 35

строки конфигурации 63

DB2 DataJoiner 455

Ограничения версии 8.2 7

строки конфигурации 63

DB2 Net Search Engine (NSE) 40

db2cliws\_dj.bnd 7

db2clprj\_dj.bnd 7

DDO (dynamic data object -

динамический объект данных)

добавление свойств 53

импорт XML 105

обращение к атрибутам 56

обращение к содержимому  
папки 112

получение свойств атрибутов 59

сортировка собрания 123

экспорт XML 106

DDO (dynamic data object - объект  
динамических данных)

C++

удаление 61

dkDDO 465

Java 50

PID 53

атрибут, DKPARTS 107, 111

вывод 59

добавление 54

значения элемента данных 54

информация, Digital

Library 176, 302

Информация, Механизм  
текстового поиска 322

свойства 58

создание 51

representing in FileNET 448

retrieveObject 201

введение 14

группы атрибутов в CM 8 185

- DDO (dynamic data object - объект динамических данных) *(продолжение)*
  - заполнение при помощи setData 190
  - идентификация в Extended Search 435
  - объединения 20
  - поиск изображений, постоянный ID 360
  - получение всех содержащих папок 212
  - постоянный идентификатор 17
  - представление в поиске изображений 359
  - представление мультимедийного содержимого 16
  - результат поиска изображений 354
  - свойства 302
  - связь с контент-серверами 16
  - создание ссылок 164
  - спецификация COBRA 14
  - сравнение с атрибутами 17
- ddo.dtd 102
- ddoDocument 69
- DEFINED, FileNET 452
- deleteDKAttributeDef 122
- deleteObject 469
- DemoSimpleAppl.java 512
- DESCENDING 253
- DfltACLCode 171
- DIGIT 253
- DIV 253
- DK\_CM\_DOCUMENT 303
- DK\_CM\_FOLDER 304
- DK\_CM\_OPT\_ACCESS\_MODE 301
- DK\_CM\_OPT\_CONTENT 454
- DK\_CM\_PROPERTY\_ITEM\_TYPE 303, 304
- DK\_CM\_READWRITE 301
- DK\_CM\_VERSION\_LATEST 201
- DK\_DES\_GQL\_QL\_TYPE 433
- DK\_DL\_OPT\_ACCESS\_MODE 476
- DK\_OPT\_TS\_CCSID 324
- DK\_OPT\_TS\_LANG 324
- DK\_READWRITE 476
- DK\_SS\_CONFIG 376, 378
- DK\_SS\_NORMAL 376, 378
- DK\_TS\_DOCFMT\_HTML 334
- dkAbstractWorkFlowUserExit 496
- dkAnnotationExt 480
- DKAny
  - Typecode 114
  - вывод 116
- DKAny *(продолжение)*
  - изменения в версии 8.2 12
  - использование в собрании 118
  - использование конструкторов типов 114
  - код типа, получение 115
  - назначение 115
  - назначение из 115
  - проверка типа 116
  - советы по программированию 117
  - удаление собраний 122
  - уничтожение 117
  - управление памятью 114
- dkAttrDef 300
  - классы 472
- DKAttrFieldDefDD 421
- DKAttrGroupDefICM 186
- DKAttrKeywordDefDD 421
- DKAttrProfileDefDD 421
- DKBinderDefDD 421
- dkBlob 301
  - классы 476
  - методы 476
- DKBlobDL
  - setToBeIndexed 366
- DKBlobFN 449
- DKCabinetDefDD 421
- dkClob
  - классы 478
- dkCollection 467
- DKCollectionResumeListEntryICM 273
- DKConstant
  - константы
  - общие 482
- DKConstant.h 31
- DKConstantFN 451
- DKConstants 43
- dkCQExpr 467
- DKCQExpr 451
- dkDatastore
  - addObject 469
  - changePassword 468
  - deleteObject 469
  - executeWithCallback 468
  - listDataSourceNames 468
  - listDataSources 468
  - listMappingNames 469
  - moveObject 469
  - registerMapping 469
  - registerServices 468
  - retrieveObject 469
  - unRegisterMapping 469
  - unregisterServices 468
  - updateObject 469
- dkDatastore *(продолжение)*
  - введение 468
  - метод evaluate 468
  - метод execute 468
  - откат 468
  - отсоединение 468
  - принятие 468
  - соединение 468
- DKDatastore 299
  - пользовательские соединители 465
- DKDatastorexx 299
- DKDatastoreAccessError 454
- dkDatastoreDef 299
  - классы 470
- DKDatastoreDef
  - методы 470
- DKDatastoreDefDL
  - дополнительные функции 470
- DKDatastoreDefOD
  - дополнительные функции 471
- DKDatastoreDES
  - listEntities 429
  - listEntityAttrs 429
  - запрос 433
- DKDatastoreDL
  - Java 44
  - вывод схемы и атрибутов схемы 47
  - опции DKDatastoreDL 46
  - получение списка серверов 46
  - соединение 44
- dkDatastoreExt
  - классы 480
  - функции 480
- DKDatastoreFN 447
- DKDatastoreICM 175
- DKDatastoreIP 401
- DKDatastoreOD 385
  - listEntities 386
- DKDatastoreQBIC 348
- DKDatastoreTS
  - Java 319
  - опции DKDatastoreTS 324
  - получение списка для схемы 326
  - получить список серверов 325
  - соединение 323
  - атрибуты 322
- DKDatastoreV4 410
- DKDatastoreIP
  - listEntityAttrs 405
- DKDatastoreOD
  - listSearchTemplates 389
- dkDDO 465

- DKDDO,
  - См. динамический объект данных (DDO)
- DKDLITEMID 322
- DKDocRoutingServiceICM 273
- DKDocRoutingServiceMgmtICM 273
- DKDocumentDefDD 421
- DKDSIZE 322
- dkEntityDef 300
  - классы 471
  - функции 471
- DkEntityDefIP
  - getAttr 403
- DKEntityDefIP
  - listAttrNames 403
- DKException 41, 173
- DKFederatedCollection 123
- dkFederatedIterator 123
- dkFederatedQuery 123
- DKFederatedQuery 23
- DKFixedView 385
- DKFixedViewDataOD 385
- DKFolder 304
- dkIterator 123
  - изменения в версии 8.1 10
- DKLink 214
- DKLinkCollection 215
- DKLITEMID 359
- DKMessage\_en.properties 482
- DKMessage\_en\_US.properties 482
- DKMessage\_es.properties 482
- DKMessage\_es\_ES.properties 482
- DKMessageId 482
- DKMessageId, FileNET 454
- DKParametricQuery 451
- DKPARTNO 322, 359
- DKParts
  - Extended Search 436
  - ранние версии CM 303
- DKPidXDO 481
- DKPrivilegeSetICM 218
- DKProcessICM 273
- dkQuery 467
- dkQueryableCollection 467
- DKRANK 303, 322, 359, 369
- DKRCNT 322
- DKREPTYPE 322, 359
- DKResource 385
- DKResourceGrpOD 385
- DKResults 467
  - позиционирование итератора 124
- dkResultSetCursor 301
  - функции 474
- DKResumeListEntryICM 274

- DKRMConfiguration 178
- DKRoomDefDD 421
- DKRouteListEntryICM 273
- dkSchemaMapping 469
- DKSequentialCollection 310, 312, 467
- dkSequentialIterator 467
- dkServerDef 300
  - классы 473
  - функции 473
- dkSort 123
- DKStorageManageInfo 98
- DKString
  - изменения в версии 8.2 12
- DKTimestamp
  - приостановка рабочего потока 490
- dkUserManagement 482
- DkViewOD 385
- DKViews 385
- DKWorkBasketDL 373
- DKWorkFlowFed
  - возобновить 490
  - приостановить (suspend) 490
- DKWorkFlowServiceDL 373
- DKWorkFlowServiceFed
  - svWf 489
- DKWorkFlowServicesFed 485
- dkWorkFlowUserExit 496
- DKWorkItemFed
  - checkIn 494
  - checkOut 494
- DKWorkListFed 492
- DKWorkListICM 273, 283
- DKWorkNodeICM 273, 276
- DKWorkPackageICM 274, 292
- dkXDO 478
  - изменения в версии 8.1 10
- dkXDOBase 466
  - открытие 478
- DLSEARCH\_DocType 126
- doAction 496
- Domino.Doc
  - API открытого управления документами (open document management - ODMA) 419
  - введение 419
  - вывод списка атрибутов шкафов 425
  - запрос 426
  - запросы для объекта DKResults 126
  - модель объекта 420
  - отображение объединения 20
  - получение списков объектов и подобъектов 422

- Domino.Doc (*продолжение*)
  - синтаксис запросов 427
- DSNAME, реляционные базы данных 456
- dsType 398

## E

- eClient
  - введение 6
  - созданные с помощью JavaBeans 499
- Enterprise Information Portal
  - базы данных 14
  - введение 1
  - диаграмма структуры 13
  - динамические объекты данных (dynamic data objects - DDO) 14
  - инфраструктура базы данных 465
  - классы соединителей 6
  - компоненты 4
  - механизмы документов 531
  - многоуровневая архитектура 3
  - общие классы 465
  - отображение схем 14
  - перенастройка 4
  - поддерживаемые контент-серверы 13
  - понятия 13
  - постоянный идентификатор (PID) 17
  - расширенные объекты данных (XDO) 15
  - сценарий страхования 1
  - характеристика рабочего потока 6
  - что нового 8
- Enterprise Java Bean (EJB) 503
- ENTITY\_TYPE 385, 397
- ErrorCode 173
- ErrorState 173
- ESCAPE,KEYWORD, запрос 253
- ESCAPE\_LITERAL 254
- EXCEPT 246, 253
- executeWithCallback 468
- exponent, запрос 253
- ExpressionList 255
- ExpressionWithOptionalSortBy 254
- Extended Search
  - введение 428
  - запрос при помощи GQL 433
  - идентификация объекта DDO 435
  - обработка полей 436
  - обработка содержимого 436

Extended Search *(продолжение)*  
объединенный поиск 446  
отображение объединения 20  
получение BLOB 444  
получение документа 444  
получение списка серверов 428  
получение списков объектов и  
атрибутов 429  
связывание типов MIME 446  
создание PID 436  
Extenders 15

## F

F\_DOCFORMAT 448  
F\_DOCTYPE 448  
FeServerDefBase 483  
fetchNext 474  
fetchObject 474  
FileNET,  
    *См.* Panagon Image Services  
findObject 474  
FLoadSampleTSQBICDL 329  
FLOAT\_LITERAL 253  
fromXML 101  
FTSearch 426  
FunctionName 255

## G

getCommonPrivilege 481  
getContent 476  
getContentToClientFile 476, 478  
getDatastore 480  
getOpenHandler 477, 479  
getPosition 474

## H

HSM (Hierarchical Storage Management  
- иерархическое управление  
хранением) 255

## I

IBM Enterprise Information Portal for  
Multiplatforms  
    компоненты 4  
        клиент администратора 4  
        пример клиентской  
        прикладной программы 6  
        соединители 5  
        управляющая база данных 4  
ICMLogon 169  
ID пользователя  
    отображение в объединении 22  
IDENTIFIER 254  
ImagePlus for OS/390  
    eClient 6  
    введение 401

ImagePlus for OS/390 *(продолжение)*  
    отображение объединения 20  
    параметры запроса 408  
    получение списка атрибутов 402  
    получение списка объектов 402  
    синтаксис запросов 408  
imldiag.log 345  
IN RANGE 452  
INBOUNDLINK 230  
indexOf 477, 478  
INTERGER\_LITERAL 253  
INTERSECT 246, 253  
IS 253  
isBegin 474  
isCheckedOut 481  
isEnd 474  
IsFTIndexed 426  
isInBetween 474  
isOpen 475  
isOpenSynchronous 477, 479  
isScrollable 474  
isSupported 480  
isUpdatable 474  
isValid 474  
ItemAdd 169  
ItemAdminPrivSet 169  
ItemQuery 169  
ItemReadPrivSet 171  
ItemSetSysAttr 169  
ItemSetUserAttr 169  
ItemSQLSelect 169  
ItemTypeQuery 169

## J

j2ee.jar 502  
Java  
    DKConstant.h 31  
    FeServerDefBase 483  
    измененные классы 10  
    комплект программы просмотра  
        документов 529  
    константы 43  
    настройка среды 33  
    новые классы 9  
    пакеты 32  
    работа с XML 100  
    сборщик мусора 31  
    соединители 6  
    создание действий рабочего  
        потока 496  
    увеличение размера стека  
        JVM 398  
    файлы свойств сообщений об  
        ошибках 482  
    функции XML 31

Java *(продолжение)*  
    эскейп-последовательности 252  
Java Server Pages (JSP)  
    сервлет 550  
java.lang.exception 174  
java.lang.objects 193  
JavaBeans,  
    *См.* функции bean  
JDBC (Java Database  
    Connectivity) 455  
JDBC\_DRIVER, реляционные базы  
    данных 456  
JDBC\_SERVERS\_FILE, реляционные  
    базы данных 456  
JDBC\_SERVERS\_URL, реляционные  
    базы данных 456  
JSP,  
    *См.* Java Server Pages (JSP)

## K

KEY 451

## L

LANG 324  
LETTER 254  
LIKE 248, 253  
LIKE, FileNET 452  
LinkTypeName 214  
ListConstructor 255  
ListContent 255  
listDataSourceNames 468  
listDataSources 178, 468  
listEntityNames 182  
listFunctions  
    dkDatastoreExt 480  
listMappingNames 469  
listResourceMgrs 178  
listWorkflowTemplates 495  
listWorkLists 491  
Literal, запрос 255  
LoadFolderTSQBICDL 329  
LocalPart 255  
LogicalOrSetExpression 254  
LogicalOrSetPrimitive 254  
LorgicalOrSetTerm 254

## M

MATCH\_DICT 140  
MATCH\_INFO 140  
MAX\_RESULTS, FileNET 453  
Microsoft Visual Studio .NET 38  
MOD 253  
Model View Controller (MVC) 504,  
    540  
moveObject 469  
moveObject, dkDatastoreExt 481

## N

nameOfAttrStr 193  
NameText 255  
NATIVECONNECTSTRING,  
    реляционные базы данных 455  
NoAccessACL 171  
NodeGenerator 255  
NONZERO 253  
NoPrivSet 168  
NOT 253  
NULL, запрос 253

## O

Object Management Group (OMG) 14  
OnDemand  
    асинхронный поиск 398  
    введение 384  
    включение режима папок 397  
    вывод атрибутов 393  
    вывод документов 393  
    запрос группы программ 391  
    имена свойств 385  
    использование папок в качестве  
        шаблона поиска 398  
    комментарии 399  
    отображение объединения 20  
    отсоединение от 385  
    подсоединение 385  
    поиск документа 390  
    получение документов 390  
    получение списка групп  
        программ 386  
    получение списка информации для  
        сервера 386  
    получение списка папок 389  
    представление серверов и  
        документов 385  
    трассировка 399  
Open Database Connectivity  
    (ODBC) 455  
    строки конфигурации 63  
OptionalExpressionList 255  
OptionalPredicateList 255

## P

Panagon Capture Software 449  
Panagon Image Services  
    введение 447  
    запрос 451  
    классы 447  
    классы документов 449  
    необязательные параметры  
        поиска 453  
    операции 451

Panagon Image Services (*продолжение*)  
    поддерживаемые типы  
        данных 448  
    получение списков объектов и  
        атрибутов 449  
    представление документов и  
        страниц 448  
    примеры 451  
    синтаксис запросов 451  
    устранение неисправностей 454  
PENDING, ImagePlus for OS/390 409  
pEnt, C++ 50  
Persistent Data Service (PDS),  
    COBRA 14  
Persistent Object Service, COBRA 14  
Predicate 255  
PrivSetCode 168  
PublicReadACL 171

## Q

QbColor 352  
QbColorFeatureClass 349, 352  
QbColorHistogramFeatureClass 349,  
    352, 353  
QbDraw 353  
QbDrawFeatureClass 349, 353, 354  
QbHistogram 352  
QbTexture 353  
QbTextureFeatureClass 349, 353  
QName 255

## R

REFERENCEDBY 230  
REFERENCER 230  
registerMapping 469  
registerServices 468  
removeAllElement 122  
removeFromFolder 480  
removeMember 311  
replaceElementAt 122  
ReplayAction, сервлет 551  
retrieveFromOverlay 481  
retrieveObject 469

## S

SAttributeDefinitionCreationICM 161  
SConnectDisconnectICM 174  
score-basic, текстовый поиск 232  
    текстовый поиск  
        contains-text 233  
SDocModelItemICM 165  
SDocRoutingDefinitionCreationICM 275,  
    276, 281, 283, 288  
SDocRoutingListingICM 278, 284,  
    292, 294

SDocRoutingProcessingICM 289, 290,  
    291, 293  
SequencedValue 254  
setClassOpenHandler 477, 479  
setContent 476  
setContentFromClientFile 476, 478  
setData 190  
setDatastore 480  
setInstanceOpenHandler 477, 479  
setPosition 474  
setToBeIndexed 366  
setToFirstCollection 124  
setToLastCollection 124  
setToNext 474  
setToNextCollection 124  
setToPreviousCollection 124  
SFolderICM 165, 206, 212  
SItemCreationICM 165  
SItemDeletionICM 202  
SItemRetrievalICM 202  
SItemTypeCreationICM 162  
SItemTypeRetrievalICM 205  
SItemUpdateICM 199, 203  
SLinksICM 164, 214, 215  
SLinkTypeDefinitionCreationICM 215  
SMS (System Managed Storage -  
    хранение, управляемое системой),  
    Content Manager Версии 8 160  
SORTBY 253, 254  
SortSpec 254  
SortSpecList 254  
SOURCEITEMREF 230  
SReferenceAttrDefCreationICM 164,  
    230  
SResourceItemMimeTypesICM 198  
SSearchICM 199  
Step, запрос 255  
STRING\_LITERAL 253  
subString 477, 478  
SuperUserACL 171  
svWF 489  
SYSREFERENCEATTRS 230  
SystemAdmin 169  
SystemAdminPrivSet 169  
SystemDefineItemType 169

## T

taglib.tld 502  
TARGETITEMREF 241  
TCallbackOD 398  
TCheckStatusTS 345  
TCP/IP  
    текстовый поиск 323  
TExportICM 100  
TExportPackageICM 100



Text Information Extender (TIE)  
  Net Search Engine (NSE) 40  
TImageAnnotation 541  
TImportICM 100  
toXML 101  
TRetrieveFolderWithCallbackOD 398  
TRetrieveWithCallbackOD 398  
TxdoAsyncRetDL 302

## U

UNICODE\_CHARACTER 253  
UNION 246, 253  
unlockCheckedOut 481  
unRegisterMapping 469  
unregisterServices 468  
UpdateFTIndex 426  
updateObject 195, 469

## V

valueObj 193  
VideoCharger 159  
Visual Studio.NET 38

## W

W3C XML Query 227  
wakeUpService 333  
WAL PRS\_Parse 451  
web.xml 502  
WebSphere Studio Application  
  Developer  
    построение функций bean 501  
Windows  
  задание подсистемы консоли 39  
  запуск сервера RMI 35  
  файлы DLL C++ 36

## X

XDO  
  dkBlob 476  
  dkXDO 466  
  FileNET 449  
  Java 61  
    DDO, часть 65  
    PID 62  
    индексирование 328  
    отдельный 67  
    свойства данных 62  
    советы по  
      программированию 64  
  большой объект 177  
  введение 15  
  вызов функции 73  
  добавление в собрание  
    хранения 98  
  добавление из буфера 67  
  добавление из файла 68

### XDO (продолжение)

  добавление объекта комментария  
    в 69  
  добавление объекта  
    мультимедиа 79  
  задание типа MIME 198  
  иерархия типов классов 197  
  изменение 71  
  изменение собрания хранения 99  
  индексация в предыдущих версиях  
    CM 63  
  индексирование в поиске  
    изображений 366  
  класс 466  
  конкретные классы DK 481  
  объекты 164  
  получение 71  
  получение объекта  
    мультимедиа 92  
  представление мультимедийного  
    содержимого 16  
  примеры 71  
  создание ресурсного  
    элемента 197  
  сравнение с атрибутами 17  
  текстовый поиск 243  
  удаление 71, 86  
  экспорт XML 106  
  элементы 478

### XML

  библиотека тегов функций  
    bean 502  
  введение 100  
  запрос W3C 227  
  извлечение  
    Web-адрес 105  
    буфер 105  
    файл 104  
  импорт 101  
  импорт в CM 105  
  определение типа документа  
    (DTD) для импорта 102  
  поддержка в Java 31  
  представление XML модели  
    данных запросов примеров 237  
  примеры 104  
  примеры инструментов 100  
  сохранение в CM 103  
  экспорт 106

XQuery Path Expressions (XQPE) 227

## A

арифметический запрос  
  операции 241  
  синтаксис 254

ассистент конфигурирования  
  клиента 5  
атрибуты

  Content Manager Версия 8 161  
  Extended Search 429  
  вывод в функциях bean 513, 522  
  группировка 161  
  задание и получение в CM 8 193  
  запрос 240  
  изменение групп в CM 8 186  
  изменение для элемента 193  
  компоненты bean 505  
  конкретные классы DK для  
    определения 472  
  многозначные 161  
  обращение 230  
  обращение в DDO 56  
  определение 161, 300  
  поиск изображений 359  
  получение документного типа  
    элементов 260  
  получение свойств из 59  
  получение списка в CM for  
    AS/400 410  
  получение списка в FileNET 449  
  получение списка в IP OS/390 402  
  получение списка для  
    OnDemand 387  
  получение списка для типа  
    элементов 187  
  получение списков в реляционных  
    базах данных 457  
  пользовательские 229  
    как разрешить текстовый  
      поиск 232  
  представление через DDO 177  
  создание в CM 8 183  
  сравнение с DDO и XDO 17  
  удаление для DKAny 122  
  управление в сервисе  
    контроллера 556  
  управление версиями 201  
  управление группами в CM 8 185  
  функции bean 546, 547  
атрибуты ссылок 230

## B

библиотека  
  C++, список 36  
  Java 33, 35  
  Microsoft Visual Studio .NET 38  
библиотека тегов  
  связанные с документами 549  
  связанные с папками 549  
  связанные с поиском 547

библиотека тегов (*продолжение*)  
связанные с соединениями 545  
связанные с элементом 547  
связанные со схемой 545

библиотечный сервер, Content  
Manager Версия 8 159  
получение списка 178  
управление версиями 165

большой объект 177  
обработка в ранних версиях  
СМ 302

буфер, добавление XDO 67

## В

версия 8.1, что нового 8  
версия 8.2, что нового в API 7  
версия, запрос 244  
возобновление рабочего потока 490  
вставка, BLOB 477  
входящие связи 215  
выделение соответствий,

См. выделение, текстовый поиск  
выделение, текстовый поиск 140

получение информации для  
каждого результата 140  
получение информации для  
отдельного результата 146

вызов удаленного метода (RMI)

запуск в Java 35  
использование с API Java 35  
конфигурирование 5  
соединение с функциями  
bean 504

соединители контент-серверов 6

выполнение (execute) запросов 125

выполнение поиска (метод  
evaluate) 40

выполнение поиска (метод  
execute) 40

выражение, запрос 254

выражения условий, FileNET 453

выражения, запрос 245

выходная\_опция 46

## Г

гистограмма цветов

допустимые значения 352

определение 349

пример запроса 353

графический пользовательский  
интерфейс (GUI),  
См. функции bean, визуальные

## Д

двоичные данные,  
См. XDO

двоичный большой объект (BLOB,  
binary large object) 477

асинхронное открытие 477

возврат длины 477

вставка данных аргумента 477

добавление 476

идентификация обработчика

файлов 477

изменение 476

классы 301

конкатенация 477

конкретные классы DK 476

конкретные классы

DKPidXDO 481

копирование данных 476

методы 476

поиск 477

получение 476

получение в Extended Search 444

проверка с помощью функции

XDO 73

создание XDO для 62

управление содержимым 476

действия

обращение к спискам 496

создание 496

динамическая конфигурация, функции

bean 503

длина BLOB 477

добавление, BLOB 476

документная модель,

См. документные части

документные части

Content Manager Версия 8 162

константа 179

тип элемента 179

документы 191

Content Manager Версия 8 162,  
165

Domino.Doc 421

вывод в функциях bean 513, 521

добавление 469

запуск процесса

маршрутизации 288

изменение 469

изменение в СМ 8 264

кластеризация с помощью

функций bean 508

документы (*продолжение*)

комментирование 529

компоненты bean 508

кэширование в сервлете

контроллера 553

маршрутизация через

процесс 272

отличия в управлении в ранних

версиях СМ 311

перемещение 469

поддержка версий частей в модели

данных управления 267

получение 469

получение в СМ 8 266

представление в модели

данных 230

просмотр,

См. комплект программы

просмотра документов

рабочий поток ранних версий

СМ 373

резервирование и

активирование 481

семантический тип 191

создание в СМ 8 261

создание документного типа

элементов 259

создание модели данных

управления в СМ 8 259

текстовый поиск 232

удаление 469

удаление в СМ 8

SDocModelItemICM 266

управление в СМ 8 257

функции bean 544, 549

документы с разметкой

текстовый поиск 333

дочерние компоненты

Content Manager Версия 8 162

диаграмма 162

представление в модели

данных 229

представление как атрибутов

DDO 177

создание в СМ 8 190

создание ссылок 164

управление версиями 166

## З

заккрытие, указатель набора  
результатов 475

залы, Domino.Doc 421

запись в журнал,  
*См.* трассировка  
 запрашиваемое собрание  
   Java 155  
     оценка 155  
     получение результатов 155  
     советы по  
       программированию 157  
   сравнение запрашиваемое  
     собрания и  
       комбинированного  
       запроса 157  
 запрос  
   Java 125  
     параметрический тип 126  
     сравнение dkResultSetCursor и  
       DKResults 126  
     текстовый тип 134  
     типы объектов запроса 125  
 запрос по содержанию изображения  
 (Query by Image Content - QBIC),  
   *См.* поиск изображений  
 запросы  
   CM for AS/400 411  
   Content Manager Версия 8 227  
   Domino.Doc 426  
   Extended Search 433  
   Panagon Image Services 451  
   арифметические операции 241  
   версия 244  
   грамматика языка 253  
   класс dkQuery 467  
   комбинированные  
     выражения 467  
   метод evaluate 125  
   метод execute 125  
   модель данных 237  
   общие выражения 467  
   объединения  
     обработка 23  
   поиск OnDemand 390  
   поиск изображений 351, 360  
   поиск по среднему цвету 353  
   поиск структурированных  
     документов 333  
   понятие языка 227  
   пример с символами  
     подстановки 244  
   примеры 236, 240  
   примеры маршрутизации 295  
   просмотр результатов 245  
   реляционные базы данных 460  
   синтаксис 240  
   синтаксис Domino.Doc 427  
   синтаксис ImagePlus 408

запросы (*продолжение*)  
   синтаксис запроса поиска  
     изображений 351  
   собрание объединения 124  
   текстовый поиск 232  
   эскейп-последовательности 247  
   язык 247  
   язык GQL (Generalized Query  
     Language - обобщенный язык  
     запросов) Generalized Query  
     Language (GQL) 433

## И

идентификаторы  
   поиск изображений 348, 350  
 изменение содержимого 478  
   XDO 71  
 изменение, BLOB 476  
 ИЛИ 253  
 импорт XML 101  
 индексирование  
   поиск изображений 366  
 индексы  
   текстовый поиск по 135  
 инструменты  
   Content Manager Версия 8 174  
 интерфейсы прикладного  
   программирования (API)  
     возможности 31  
     понятия 13  
     программные компоненты 175  
     электронная справка 174  
 интерфейсы прикладного  
   программирования (application  
   programming interfaces, API)  
   Java 31  
     архитектура 32  
     множественный поиск 39  
     отличия от C++ 31  
     пакеты 32  
 информационный центр 174  
 информация диагностики,  
   *См.* трассировка  
 искатель Web  
   введение 6  
   компоненты bean 508  
 исключительные ситуации,  
   обработка 41  
 исследование информации  
   компоненты bean 508  
   описание 6  
   поддержка функций bean 504  
   построение прикладной  
     программы 561  
   примеры 561

исследование информации  
 (*продолжение*)  
   функции bean 561  
 Исследование информации  
   использование API служб 614  
   поиск по категориям 606  
   положение файлов примеров 566  
   пример Web Crawler 597  
   пример извлечения  
     информации 584  
   пример импорта документов 597  
   пример категоризации 566  
   пример кластеризации 591  
   пример сложного поиска 606  
   пример составления сводок 577  
   программы JSP 627  
   собственный  
     контент-провайдер 613  
 исходящие связи 211, 215  
 итераторы  
   объединение 123  
   позиционирование в  
     DKResults 124

## К

каталог данных  
   ограничения версии 8.2 8  
 Каталог данных  
   отображение объединения 20  
 каталог данных DB2 Data Warehouse  
   Manager,  
   *См.* Информационный каталог  
 каталоги, поиск изображений 354  
 класс выражений 467  
 классификация элементов,  
   *См.* типы элементов  
 классы определения данных 301  
 клиент  
   построение для работы с  
     исследованием информации 6  
 клиент администратора  
   введение 4  
   описание 4  
   установка на дополнительных  
     рабочих станциях 5  
 клиент администратора системы  
   настройка 28  
   обработчики пользователя 28  
   создание рабочих списков 492  
 Ключевые слова, Domino.Doc 421  
 команда msg, FileNET 454  
 Комбинированные запросы  
   C++  
     ранжирование 372

- Комбинированные запросы *(продолжение)*
  - C++ *(продолжение)*
    - советы по программированию 373
  - Java 234, 369
    - использование сферы 371
    - параметрический с текстовым 370
- комбинированный запрос
  - определение 40
- комментарии
  - FileNET 449
  - OnDemand 399
  - добавление объекта в XDO 69
  - документы 529
  - класс dkAnnotationExt 480
  - классы служб 530
  - компоненты bean 506, 509
  - константа 191
  - механизм 531
  - настройка 541
  - поддержка редактирования 540
  - семантический тип 191
  - службы 538
- компилятор Microsoft Visual C++,  
См. компилятор Visual C++
- компилятор Visual C++ 36
- комплект программы просмотра документов
  - автономная программа просмотра 534
  - апплет или сервер 536
  - введение 529
  - всплывающие меню 532
  - двойной режим, апплет или сервер 537
  - механизмы
    - документ AFP2Web 531
    - документ INSO 531
    - документ Java 531
    - документ MS-Tech 531
  - минимальный клиент 536
  - настройка общей программы просмотра документов 532
  - Прикладная программа Java 535
  - примеры программ 534
  - службы аннотирования 538
  - создание общей программы просмотра 531
  - структура 530
  - функции 529
- компоненты bean 507
  - CMBSchemaManagement 502
  - JAR-файлы 501
- компоненты bean *(продолжение)*
  - Model View Controller (MVC) 504
  - web.xml 502
  - введение 499
  - визуальные
    - CollationStrength 523
    - введение 512
    - внешние программы просмотра 522
    - всплывающие меню 519
    - замена 524
    - имена 512
    - использование в окнах 527
    - ключевые события 524
    - области текстового поиска 517
    - общее поведение 523
    - определение 499
    - особое поведение 524
    - панель дерева 518
    - панель поиска 517
    - панель регистрации 514
    - переопределение всплывающих меню 519
    - построение прикладных программ 525
    - программа просмотра документов 521
    - программы просмотра по умолчанию 522
    - просмотр версий 523
    - редактор атрибутов 522
    - скрытие и вывод кнопок 523
    - события справки 524
    - соединение 523, 525
    - сохранение/восстановление конфигурации 524
    - спецификации программы просмотра 521
    - список шаблонов поиска 515
    - функция просмотра папок 519
    - функция просмотра результатов поиска 517
    - функция просмотра шаблонов поиска 516
  - вспомогательные 506
  - другие строители 501
  - использование в строителях 501
  - исследование информации 508
  - классы BeanInfo 509
  - классы исключительных ситуаций 509
  - классы событий и ожидания 510
- компоненты bean *(продолжение)*
  - комментарии 509
  - комментарий 506
  - невизуальные
    - введение 503
    - возможности 504
    - категории 504
    - конфигурации 503
    - определение 499
    - особенности 510
    - построение прикладных программ 512
    - свойства 511
    - события 511
  - одиночные 510
  - ожидание событий сеанса 510
  - основные понятия 499
  - поддержка пакетного режима 503
  - поддержки 505
  - построение в WebSphere Studio Application Developer 501
  - работа с потоками 510
  - рабочий поток 502, 506
  - рабочий список 502
  - связанные с программой просмотра Java
    - определение 499
  - службы документов 508
  - соединение 502
  - схема 502
  - трассировка 511
  - требование поиска 502
  - требования 501
  - управление версиями 513, 523
  - управление данными 502
  - характеристики
    - восстановимость 501
    - интроспекция 500
    - методы 501
    - настройка 500
    - свойства 500
    - события 500
- константы 43
- маршрутизация 298
- контейнеры
  - Content Manager Версия 8 164
  - константа 191
  - семантический тип 191
- контент-провайдер в исследовании информации 613
- контент-серверы
  - Content Manager for AS/400 410
  - Content Manager Версия 8 159

контент-серверы *(продолжение)*  
 Content Manager, ранние версии 302  
 DDO (dynamic data object - объект динамических данных) 14  
 Domino.Doc 419  
 Extended Search 428  
 ImagePlus for OS/390 401  
 OnDemand 384  
 Panagon Image Services 447  
 XDO (extended data object - расширенный объект данных) 15  
 введение 1  
 возврат имен 468  
 возврат объектов ID пользователя ID 468  
 выполнение параметрического запроса с 130  
 выполнение текстового запроса с 137  
 выполнение транзакций 468  
 добавление документа или папки 469  
 добавление элементов в папки 480  
 доступ к информации на запрос 468  
 запуск запроса изображений с 362  
 идентификация, чему принадлежит указатель 475  
 иерархия определений данных 301  
 изменение документа или папки 469  
 изменение пароля 468  
 имена классов 299  
 использование с RMI 35  
 классы XDO 481  
 классы расширения 480  
 компоненты bean 504  
 конкретные классы DK для определения 473  
 настройка 464  
 объединения 19  
 определение 299  
 отсоединение 468  
 оценка (evaluate) параметрического запроса с 132  
 оценка (evaluate) текстового запроса с 139  
 оценка запроса изображений с 364

контент-серверы *(продолжение)*  
 параметры сервлета контроллера 551  
 перемещение документа или папки 469  
 поддерживаемые соединители 5  
 поиск изображений 346  
 получение документа или папки 469  
 получение общей привилегии 481  
 получение объекта оверлея формы 481  
 получение объекта управления 260  
 получение списка имен функций 480  
 получение ссылки на 480  
 пользовательские классы управления 482  
 постоянные идентификаторы (persistent identifiers - PID) 17  
 расширение feServerDefBase для пользовательских соединителей 483  
 расширение объектов 468  
 регистрация в объединении 22  
 регистрация информации отображения 469  
 резервирование документов или папок 481  
 реляционные базы данных 455  
 связь с DDO 16  
 серверы, поддерживаемые EIP 13  
 соединение 468  
 создание 299  
 создание объекта DDO в 468  
 удаление документа или папки 469  
 удаление элементов из папок 480  
 указатель набора результатов 301  
 управление данными 13  
 управление двоичными данными 476  
 управление содержимым CLOB 478  
 контраст 349  
 корневые компоненты Content Manager Версия 8 162  
 представление в модели данных 229  
 связь 164  
 создание в CM 8 189  
 создание ссылок 164  
 управление версиями 166

критерий поиска, bean 545  
 кэширование, сервлет контроллера 553

## Л

литералы  
 ImagePlus for OS/390 408  
 литералы, запрос 245

## М

маршрутизация 507  
 введение 272  
 возобновление процесса 291  
 вывод списка рабочих узлов 276  
 запуск 288  
 компоненты bean 505, 507  
 константы 298  
 конфигурирование 273  
 окончание процесса 289  
 определение нового обычного рабочего узла 275  
 определение нового процесса и связанного с ним маршрута 284  
 определение новой точки сбора 278  
 определение рабочего списка 281  
 получение информации о рабочем пакете 292  
 получение списка процессов маршрутизации документа 293  
 получение списка рабочих списков 283  
 предоставление привилегий 296  
 примеры 295  
 приостановка процесса 290  
 продолжение процесса 289  
 произвольная 295  
 просмотр PID пакетов в рабочем списке 291  
 рабочие узлы 272  
 рабочий пакет 274  
 рабочий список 274  
 создание объектов служб 274  
 списки управления доступом (ACL) 298  
 список возобновления 274  
 точки сбора 272  
 маршрутизация документов, См. маршрутизация  
 менеджер ресурсов, Content Manager Версия 8 159  
 получение списка 178  
 работа с 255  
 работа с объектами 256

- менеджер ресурсов, Content Manager
  - Версия 8 *(продолжение)*
    - управление версиями 165
- метаданные
  - исследование информации 6
- Механизм текстового поиска
  - Java
    - гибридный запрос 320
    - загрузить и индексировать
      - данные 333
    - задание размера кучи 302
    - запрос GTR 321
    - контекстный запрос 320
    - логический запрос 319
    - свободный текстовый
      - запрос 320
    - советы по
      - программированию 324
  - механизмы поиска, регистрация 468
  - минимальный клиент 536
  - многопоточность 31
  - модель данных, Content Manager
    - Версии 8
      - применение языка запросов 228
  - модель данных, запрос
    - XML-представление 237
  - модель документов объединения 19
  - модули расширения DB2 15
  - мультимедийное содержимое 16

## Н

- направленность 349
- национальная версия
  - сервлет 550
- нересурсные элементы
  - Content Manager Версия 8 162
- несколько символов (MC) 325

## О

- область
  - библиотека тегов 544
  - транзакции 268
- обработчики пользователя
  - рабочий поток 496
  - управление системой 28
- обратный вызов, выполнение 40, 228
- общая привилегия,
  - dkDatastoreExt 481
- общая программа просмотра
  - документов,
    - См. комплект программы просмотра документов
- общие классы EIP 465

- объединение
  - базовый класс определения
    - сервера 483
  - итераторы 123
  - поиск
    - Extended Search 446
    - введение 4
    - класс выражений 467
  - собрания 123
  - уровень для функций bean 504
  - функции bean 546
- объединения 28
  - атрибуты 20
  - документная модель данных 19
- запрос
  - иллюстрация обработки 24
  - примеры 25
  - результатов 23
  - синтаксис 25
  - создание 23
- объекты 20
  - отображение 20
- отображение ID
  - пользователей 22
- отображение схем 22
- папки 28
- поиск
  - введение 19
  - иллюстрация 21
  - иллюстрация отношений 24
  - процесс 23
- поиск изображений 25
- регистрация контент-серверов 22
- собрание 24
- функции управления системой 28
- элементы 19
- объект мультимедиа
  - добавление в ранних версиях
    - CM 79
  - имена примеров кода 99
  - получение 92
  - удаление 86
- объект оверлея 481
- объект оверлея формы 481
- объекты
  - Content Manager Версия 8 164
  - Extended Search 429
  - ImagePlus for OS/390 409
  - выполнение запроса в
    - FileNET 453
  - идентификация 17
  - иерархия классов 300
  - изменение в CM 8 195
  - компоненты bean 505

## объекты *(продолжение)*

- конкретные имена классов DK для
  - определения 471
- отображение в Domino.Doc 421
- отображение в объединении 20
- отображение схемы 468
- папки OnDemand как
  - собственные 398
- перемещение объектов 481
- поиск в Domino.Doc 425
- получение списка в CM 8 182
- получение списка в CM for
  - AS/400 410
- получение списка в
  - Domino.Doc 422
- получение списка в FileNET 449
- получение списка в IP OS/390 402
- получение списков в реляционных
  - базах данных 457
- привилегии 168
- собственные 398, 468
- текстовый поиск
  - содержимого 232
- управляемые, Content Manager
  - Версии 8 167
- функции bean 546, 547
- хранение значений
  - атрибутов 193
- хранение папок объединения в 28
- операции
  - Domino.Doc 427
  - FileNET 451
- запрос ImagePlus for OS/390 408
- параметрический поиск 230
- язык GQL (Generalized Query
  - Language - обобщенный язык запросов) Generalized Query
    - Language (GQL) 446
- определение типа документа (DTD),
  - импорт XML 102
- откат 468
- открытие, BLOB 477
- открытие, CLOB 478
- открытие, указатель набора
  - результатов 475
- относительные позиции, указатель
  - набора результатов 153
- отображение схем
  - введение 14
- связывание в объединении 21
- отображение схемы
  - класс dkSchemaMapping 468
- методы dkDatastore 469
- пользовательские
  - соединители 465



- отображение схемы *(продолжение)*
  - реляционные базы данных 455
- оценка (evaluate) запросов 125
- очередь обработки изображений 366
- П**
  - пакет com.ibm.mm.sdk.common 465
  - пакет cs 32
  - пакеты
    - cs (клиент и сервер) 32
    - иерархия в Java 32
    - клиент Java 32
    - сервер Java 32
  - папки
    - Content Manager Версия 8 165
    - включение режима в
      - OnDemand 397
    - вывод в функциях bean 513, 519
    - добавление 469
    - добавление содержимого 207
    - добавление элементов в 480
    - доступ 112
    - изменение 469
    - изменение в ранних версиях
      - CM 312
    - папки OnDemand как собственные
      - объекты 398
    - перемещение 469
    - поведение Content Manager 112
    - получение 469
    - получение всех папок с
      - конкретным DDO 212
    - получение содержимого 211
    - получение списка в
      - OnDemand 389
    - представление в ранних версиях
      - CM 304
    - рабочий поток ранних версий
      - CM 373
    - резервирование и
      - активирование 481
    - семантический тип 191
    - создание в CM 8 206
    - удаление 469
    - удаление содержимого 209
    - удаление элементов из 480
    - управление в сервлете
      - контроллера 557
      - функции bean 549
    - папки, Domino.Doc 421
    - параметрический поиск
      - Panagon Image Services 451
      - выполнение запроса 128
      - выполнение запроса с
        - контент-сервера 130
  - параметрический поиск *(продолжение)*
    - запрос в CM for AS/400 418
    - запросы с несколькими
      - критериями 127
    - объединения 25
    - операции 230
    - определение 40
    - основные понятия 230
    - оценка запроса с
      - контент-сервера 132
    - трассировка 41
    - формулирование нескольких
      - критериев 127
    - формулирование строки
      - запроса 126
  - параметры
    - C++
      - построение в NT 38
    - Java
      - в NT 34, 37, 38
      - клиент/сервер 32
      - советы по
        - программированию 33
      - соединение и отсоединение
        - клиента 45
    - Использование примеров
      - апплетов и сервлета Java 543
      - локальный доступ 544
      - получение сервлета 543
      - программа Java на
        - клиенте 543
      - удаленный доступ 545
    - среда C++ 35
    - среда Java 33
  - пароль
    - изменение 468
    - изменение в функциях bean 524
    - отображение в объединении 22
    - создание в Content Manager Версии
      - 8 179
  - подобъекты
    - получение списка в
      - Domino.Doc 422
  - подсистема консоли, задание в
    - Windows 39
  - поиск
    - класс DKResults 467
    - компоненты bean 505
    - модули API 176
    - понятие языка запросов 227
    - функции bean 544, 547
  - поиск изображений 346
    - closeCatalog 354
    - listDatabases 354
- поиск изображений *(продолжение)*
  - openCatalog 354
  - атрибуты 359
  - базы данных 347, 348
  - введение 7
  - выполнение запроса с
    - контент-сервера 362
  - гистограмма цветов 349, 352, 353
  - диаграмма 348
  - загрузка данных для
    - индексирования 366
  - запрос 360
  - заявления 350
  - идентификаторы 350
  - индексирование объекта
    - XDO 366
  - каталоги 347, 348
  - критерий поиска 351
  - максимальное число
    - результатов 351
  - объединения 25
  - определение 40
  - отсоединение от 354
  - оценка запроса с
    - контент-сервера 364
  - подсоединение 354
  - получение списка баз данных 357
  - получение списка каталогов 357
  - получение списка серверов 355
  - получение списка
    - характеристик 357
  - представление информации при
    - помощи объекта DDO 359
  - размещение цветов 349, 353, 354
  - синтаксис запросов 351
  - создание запросов 351
  - средний цвет 349, 352, 353
  - текстура 349, 353
  - характеристики 347, 351
- поиск по цвету
  - гистограмма 349, 352, 353
  - размещение 349, 353, 354
  - средний 349, 352, 353
  - текстура 349, 353
- поиск, выполнение с обратным
  - вызовом 40, 228
- поле дополнительных параметров,
  - OnDemand 385
- поле параметров инициализации,
  - OnDemand 385
- получение
  - C++
    - частей 316
  - Java 314

- получение *(продолжение)*
    - папки 317
    - частей 315
  - получение информации 292
  - получение, BLOB 476
  - пользователи
    - управление доступом в CM 8 170
  - пользовательские атрибуты 229
    - как разрешить текстовый поиск 232
  - пользовательские соединители
    - разработка 464
    - расширение класса FeServerDefBase 483
    - управление системой 465
  - поля, Domino.Doc 421
  - поля, Extended Search 436
  - постоянный идентификатор (PID)
    - DDO (dynamic data object - объект динамических данных) 14
    - XDO (extended data object - расширенный объект данных) 15
    - введение 17
    - маршрутизация 291
    - поиск изображений 360
    - управление версиями 166
  - построение прикладной программы
    - визуальные функции bean 525
    - использование служб аннотирования 540
    - невизуальные функции bean 512
    - планирование 172
    - создание в Content Manager Версия 8 175
  - поточковые службы документов 530
  - права,
    - См. привилегии
  - преобразование кодовых страниц
    - задание подсистемы консоли в Windows 39
  - привилегии
    - Content Manager Версия 8 167, 168
    - вывод свойств набора 220
    - доступ к данным 168
    - задание 168
    - заранее сконфигурированные списки ACL 171
    - коды 169
    - компоненты bean 506
    - пользователи и группы пользователей в CM 8 170
    - предопределенные наборы Content Manager Версии 8 168
  - привилегии *(продолжение)*
    - предоставление привилегий для маршрутизации 296
    - системные 168
    - создание в CM 8 216
    - создание набора 218
    - списки управления доступом (ACL) 170
    - типы наборов 169
    - функции bean 548
  - привилегии пользователя
    - Content Manager Версия 8 167, 168
  - Пример Web Crawler при исследовании информации 597
  - пример извлечения информации исследования информации 584
  - пример категоризации для исследования информации 566
  - пример кластеризации для исследования информации 591
  - пример клиента 346
    - прикладная программа 6
  - пример правил 176
  - пример сложного поиска при исследовании информации 606
  - пример составления сводок при исследовании информации 577
  - примеры TxdoAdd 99
  - примеры кодов, обновления 7
  - примечание
    - константа 191
    - семантический тип 191
  - приостановка процесса 290
  - приостановка рабочего потока 490
  - программы JSP при исследовании информации
    - внедрение WAS 628
    - примеры 627
  - программы просмотра документов Java,
    - См. комплект программы просмотра документов
  - произвольная маршрутизация 295
  - прокрутка указателя набора результатов 474
  - профили, Domino.Doc 421
  - процессы
    - возобновление 291
    - завершение 289
    - маршрутизация 273
    - определение связанного с ним маршрута 284
    - получение списка 293
    - приостановка 290
  - процессы *(продолжение)*
    - работа продолжается 289
  - пул соединений
    - параметры сервлета 551
    - сервлет 550
- ## Р
- рабочие узлы
    - получение списка 276
  - рабочие элементы
    - доступ 493
  - рабочий комплект
    - идентификация в ранней версии CM 382
    - определение в ранних версиях CM 373
    - получение списка в ранней версии CM 379
    - правила 373
    - создание в ранних версиях CM 378
  - рабочий пакет 274, 292
    - введение 485
    - просмотр строк PID в рабочем списке 291
  - рабочий поток
    - введение 6, 485
    - выполнение в ранней версии CM 382
    - доступ к рабочим спискам 492
    - доступ к рабочим элементам 493
    - запуск 486
    - компоненты bean 502
    - модель 373
    - определение в ранних версиях CM 373
    - отправка результатов из папки объединения 28
    - отсоединение от 486
    - перемещение элементов 494
    - поддержка функций bean 504
    - подсоединение 485
    - получение списка 489
    - получение списка в ранней версии CM 376
    - получение списка всех рабочих списков 491
    - получение списка всех шаблонов 495
    - получение списка элементов в ранней версии CM 380
    - прекращение 488
    - приостановка 490
    - ранняя версия службы Content Manager 373



- рабочий поток *(продолжение)*
    - создание в ранних версиях
      - СМ 374
    - создание действий Java 496
    - удаление 488
  - рабочий список 274
    - определение 281
    - получение списка 283
    - просмотр PID пакета 291
  - рабочий узел
    - маршрутизация 272
    - определение 275
  - рабочих списков
    - доступ 492
    - доступ к рабочим элементам 493
    - компоненты bean 502
    - перемещение элементов 494
    - получение списка 491
  - размещение цветов
    - допустимые значения 353
    - определение 349
    - пример запроса 354
  - расширенные объекты данных,
    - См. XDO
  - расширенный объект данных,
    - См. XDO
  - реляционные базы данных
    - введение 455
    - запрос 460
    - отображение объединения 20
    - подсоединение 455
    - получение списка атрибутов 457
    - получение списка объектов 457
    - строки конфигурации 456
    - строки соединения 455
  - ресурсное содержимое,
    - См. XDO
  - ресурсные элементы
    - Content Manager Версия 8 162
    - константа 179
    - определение типа элементов 195
    - тип элемента 179
  - ровно один символ (SC) 325
- С**
- сборщик мусора, Java 31
  - свойство Mime2App 521
  - связи
    - Content Manager Версия 8 164
    - входящие и исходящие 215
    - имена типов 215
    - источник 214
    - константы имен типов 215
    - назначение 214
  - связи *(продолжение)*
    - определение связей между
      - элементами 214
    - переходы по запросу 241
    - получение связанных
      - элементов 215
    - представление в модели
      - данных 229
    - тип атрибутов 177
    - элемент описания 214
  - связывание, DataJoiner 7
  - сеансы
    - сервлет 550
  - сеансы с превышенным сроком
    - ожидания, сервлет 550
  - семантический тип
    - определение 191
    - пользовательский 191
    - предопределенные типы 191
  - сервер RMI,
    - См. вызов удаленного метода (RMI)
  - сервлет контроллера
    - Replay 551
    - действия 549, 551
    - использование 550
    - матрица функций комплекта
      - инструментов 558
    - модули времени выполнения
      - служб 553
    - наборы JSP 550
    - национальная версия 550
    - обращение 551
    - очистка 550
    - параметр разделителя имен 554
    - параметры 551, 555
    - параметры по умолчанию 551
    - параметры преобразования 554
    - параметры пула соединений 551
    - параметры требования
      - действие 556
      - общие 555
      - ответ 555
      - папки 557
      - поиск 555
      - связанные с документами 557
      - связанные с
        - соединениями 555
      - управление содержимым
        - элементы 556
    - пул соединений 550
    - расширение 550
    - соглашения 551
    - страницы с сообщениями об
      - ошибках 552
  - сервлет контроллера *(продолжение)*
    - трассировка 553
    - управление сеансами 550
    - файл свойств 550, 554
  - сервлет,
    - См. сервлет контроллера
  - символы подстановки
    - Domino.Doc 427
    - запрос 248
    - текстовый поиск 233
  - символьный большой объект (CLOB) 478
    - добавление содержимого 478
    - классы 478
    - обработчик файла 479
    - получение содержимого 478
    - удаление содержимого 478
    - удаление фрагмента 479
  - склад данных,
    - См. контент-серверы
  - службы рабочего потока EIP,
    - См. рабочий поток
  - собрание хранения
    - добавление XDO в 98
    - изменение для XDO 99
  - собрания
    - dkCollection class 467
    - Extended Search 436
    - объединение 123
    - сортировка 123
    - удаление 122
  - собрания и итераторы
    - C++
      - управление памятью 121
    - Java 117
      - последовательное
        - собрание 118
      - последовательный
        - итератор 118
  - события
    - компоненты bean 510
  - соединители
    - локальные в сравнении с
      - удаленными 6
    - настройка 464
    - описание 5
    - параметры сервлета
      - контроллера 551
  - соединитель ICM,
    - См. Content Manager Версии 8
  - создание 191
  - сообщения
    - Информация о DKException 173
    - файлы свойств 482

- сообщения об ошибках
  - файлы свойств 482
- список возобновления 274
- справочник по прикладному программированию (application programming reference, APR) 174
- среда, настройка
  - C++ 35
  - Java 33
- средний цвет
  - допустимые значения 352
  - определение 349
  - пример запроса 353
- ссылки
  - Content Manager Версия 8 164
  - переходы по запросу 243
  - тип атрибутов 177
- страницы
  - FileNET 448
  - функции bean 549
- строки конфигурации, реляционные базы данных 63
- схема, beans 544
- сценарий, пример со страховой компанией для Content Manager Версии 8 175

## T

- таблица FASERVERTYPES 483
- текстовый поиск
  - OnDemand 385
  - score-basic 233
  - выделение 140
  - выполнение запроса с
    - контент-сервера 137
  - задание правил индексации 338
  - запрашиваемые собрания 156
  - запросы поиска по нескольким индексам 135
  - запуск запроса 135
  - запуск процесса
    - индексирования 342
  - как разрешить текстовый поиск
    - для пользовательских атрибутов 232
  - компоненты bean 517
  - операции 233
  - определение 40
  - основные понятия 231
  - оценка запроса с
    - контент-сервера 139
  - поиск документов 232
  - поиск содержимого объектов 232
  - поиск структурированных документов 333, 345

- текстовый поиск (*продолжение*)
  - получение списка моделей документов 334
  - постоянный идентификатор (PID) 322
  - пример запроса 242
  - простой запрос 232
  - расширенный поиск 233
  - символы подстановки 233
  - синтаксис запросов 232
  - создание модели документа 336
  - трассировка 40
  - формулирование строки запроса 134
  - функции управления 319
- текстура
  - детальность 349
  - допустимые значения 353
  - контраст 349
  - направленность 349
  - определение 349
  - пример запроса 353
- терминология имен классов 299
- Тип представления 329
- Типы MIME
  - задание для ресурсного элемента 198
  - связывание в Extended Search 446
- типы элементов
  - Content Manager Версия 8 162
  - запрос нескольких 240
  - классификации 179
  - компоненты корневые и дочерние 162
  - получение для них списка атрибутов 187
  - получение списка 182
  - создание в Content Manager Версии 8 179
  - создание определения 196
  - управление версиями 166, 205
- точка сбора
  - описание 272
  - определение 278
- транзакции, Content Manager Версии 8 268
  - активирование и резервирование 270
  - меры предосторожности при использовании явных транзакций 269
  - обработка 270
  - особенности 269
  - список (list) 271

- трассировка
  - FileNET 454
  - JNI, сервлет 553
  - OnDemand 399
  - компоненты bean 511
  - контейнер кода возврата 173
  - обработка ошибок при помощи DKEException 173
  - параметрический поиск 41
  - примеры инструментов 174
  - сервлет контроллера 553
  - состояние ошибки 173
  - стек 173
  - текстовый поиск 40
  - функции bean 544
- трассировка JNI, сервлет 553
- трассировка стека 173

## У

- удаление 477
- удаление, BLOB 477
  - двоичный большой объект (BLOB, binary large object)
    - удаление фрагмента 477
- удаление, CLOB 478
- указатель набора результатов
  - Java 151
    - задание и получение 151
    - открытие и закрытие 151
    - создание собрания 154
  - функции 474
- указатель, набор результатов
  - добавление элементов 475
  - закрытие 475
  - изменение 474
  - изменение элементов
    - updateObject 475
  - открытие 475
  - позиционирование 474
  - получение информации
    - контент-сервера 475
  - получение хэнгла 475
  - получение элементов 474
  - проверка доступности 474
  - прокрутка 474
  - удаление элементов
    - deleteObject 475
  - указание на элемент 474
  - уничтожение 475
- унифицированный идентификатор ресурсов (URI) 257
- уничтожение, указатель набора результатов 475

- управление
  - объект
    - получение 260
    - привилегии 169
- управление версиями
  - Content Manager Версия 8 165
  - атрибуты 201
  - задание для элемента 203
  - компоненты bean 513, 523
  - постоянный идентификатор (PID) 166
  - правила 166
  - правила управления версиями 204
  - типы элементов 205
  - управление прикладной программой 166
  - части в модели данных
    - управления документами 267
  - элементы 201
- управление доступом
  - Content Manager Версия 8 166
  - NoAccessACL 171
  - PublicReadACL 171
  - SuperUserACL 171
  - диаграмма 167
  - назначение списка элементу 226
  - определение списков (ACL) 221
  - получение и вывод списков (ACL) 223
  - правила для списков (ACL) 170
  - работа с 216
  - работа со списками для маршрутизации документов 298
  - списки (ACL) 167, 170
- управление объектами
  - Java 304
    - изменение 194, 309
    - создание 188, 304
    - удаление 202, 314
- управляемые объекты 167
- управляющая база данных
  - введение 4
  - описание 4
  - отображение схем 14
- условия фильтра, FileNET 453
- устранение неисправностей
  - трассировка 40
  - файлы свойств сообщений об ошибках 482

## Ф

- файл, добавление XDO из 68

- файлы DLL
  - C++ 36
- файлы примеров
  - Content Manager Версия 8 174
- файлы примеров, положение
  - Исследование информации 566
- функции bean
  - вызов 502
  - выполнение поиска
    - OnDemand 398
  - документы 549
  - журнал трассировки 544
  - источник данных
    - функция bean источника данных 545
  - критерий поиска 545
  - папки 549
  - пул соединений
    - CMDBConnectionPool 544
  - результаты поиска 544
  - служба запросов 544
  - службы документов 544
  - соединение 544, 545
  - управление данными 544
  - управление пользователями 544
  - управление схемами 544
  - шаблон поиска 544, 545
  - элементы 544
- функции bean журнала
  - замечаний 547
- функции bean ожидания сеанса 510
- функции bean поддержки 505
- функция bean searchTemplate 545
- функция bean соединения 545

## Х

- характеристики
  - вес, поиск изображений 351
  - значение, поиск изображений 351
  - имя, поиск изображений 351
  - максимальное число результатов, поиск изображений 351
  - поиск изображений 347
- хронология
  - константа 191
  - семантический тип 191
- хэндл, указатель набора результатов 475

## Ч

- частей
  - Extended Search 436
  - изменение в ранних версиях
    - CM 310
  - ранние версии CM 303

- частей (*продолжение*)
  - управление версиями 267
- части элементов
  - сравнение с DDO и XDO 17
- что нового в API Версии 8.2 7
- что нового в версии 8.1 8

## Ш

- шаблон 349
- шаблон поиска, OnDemand
  - использование папок в качестве 398
  - программа просмотра 398
- шаблоны
  - рабочий поток 495
- шкафы, Domino.Doc 421
  - получение списка атрибутов 425

## Э

- элемент
  - добавление 310
  - удаление 310
- элементы
  - Content Manager Версия 8 161
  - активирование и резервирование 202
  - добавление в папку 207
  - задание и получение атрибутов элементов в CM 8 193
  - изменение атрибутов 193
  - компоненты bean 505
  - константа дерева 211
  - объединения 19
  - поиск в CM 8 199
  - получение в CM 8 199
  - получение связанных элементов 215
  - построение связей через папки 165
  - представление в модели данных 229
  - связь 214
  - создание документного типа элементов 259
  - создание ссылок 177
  - удаление из папок 209
  - управление в сервлете контроллера 556
  - управление версиями 166, 203
  - функции bean 544, 547
- элементы данных, DDO 54
- эскейп-последовательности, примеры запросов 247

## Я

язык GQL (Generalized Query Language

- обобщенный язык запросов)

Generalized Query Language (GQL)

выражения 433

отличия от SQL 446

язык XML,

См. XML

язык запросов 247





Номер программы: 5724-B43

Напечатано в Дании

SH43-0217-01



Spine information:

IBM Content  
Manager for  
Multiplatforms/IBM  
Information  
Integrator for Content



Руководство по прикладному  
программированию для рабочих станций

*Версия 8 Выпуск 2*