IBM Content
Manager for Multiplatforms/IBM Information
Integrator for Content

Workstation Application Programming Guide

Version 8 Release 2

SC27-1347-01

IBM Content
Manager for Multiplatforms/IBM Information
Integrator for Content

Workstation Application Programming Guide

Version 8 Release 2

SC27-1347-01

Note

Before using this information and the product it supports, read the information in “Notices” on page 495.

Second Edition (March 2003)

This edition applies to Version 8 Release 2 of IBM Enterprise Information Portal for Multiplatforms, IBM Content Manager for Multiplatforms (product numbers 5724-B43, 5724-B19), and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Working with Content Manager Version 8.2 123

Working with other content servers 235

Building EIP workflow applications 381

Building applications with non-visual and visual JavaBeans. 393

Working with the Java document viewer toolkit 417

About this guide

This guide describes how to use the Java™, JavaBeans™, and C++ application programming interfaces (APIs) provided with Enterprise Information Portal Version 8 Release 2 (EIP) and Content Manager (CM) Version 8 Release 2. The APIs and beans provide building blocks for creating applications to access content stored in heterogeneous content servers.

In earlier versions, CM and EIP kept separate application programming guides. In Version 8 Release 2, the two products share many of the APIs and are based on many of the same programming concepts. Also, because much of the new functionality in Enterprise Information Portal Version 8 Release 2 involves the new connector to CM Version 8 Release 2, the two guides merged into one.

This guide includes:

- An introduction to Enterprise Information Portal and CM application programming concepts, including dynamic data object concepts in the context of Java and C++
- A description of the function accessible through the CM Version 8 Release 2 connector
- Documentation on all other Enterprise Information Portal connectors to content servers
- Updates to visual and non-visual JavaBeans
- Updates to programming information for Information Mining, IBM® Web Crawler, and workflow

Illustrations referring to Content Manager imply both pre-Version 8.1 and Version 8 Release 2 of the product.

Who should use this guide

This guide is intended for application programmers with some or all of the following skills:

- Experience with either C++, Java, JavaBeans, or HTML
- Familiarity with relational database concepts
- Knowledge of the DDO/XDO protocol

Where to find more information

Your product package includes a complete set of information to help you plan for, install, administer, and use your system. Product documentation and support are also available on the Web.

Information included in your product package

The product package contains an information center and each publication in portable document format (.PDF).

The information center

The product package contains an information center that you can install when you install the product. For information about installing the information center see *Planning and Installing Your Content Management System*.

The information center includes the documentation for Content Manager, Enterprise Information Portal, and Content Manager VideoCharger. Topic-based information is organized by product and by task (for example, Administration). In addition to the provided navigation mechanism and indexes, a search facility also aids retrievability.

PDF publications

You can view the PDF files online using the Adobe Acrobat Reader for your operating system. If you do not have the Acrobat Reader installed, you can download it from the Adobe Web site at www.adobe.com.

Table 1 shows the Content Manager publications included with IBM Content Manager for Multiplatforms.

Table 1. Content Manager publications

File name	Title	Publication number
install	*Planning and Installing Your Content Management System*[1]	GC27-1332-01
migrate	*Migrating to Content Manager Version 8*	SC27-1343-01
sysadmin	*System Administration Guide*	SC27-1335-01

When you order IBM Content Manager for Multiplatforms, you also receive IBM Enterprise Information Portal for Multiplatforms. Or, you can separately order IBM Enterprise Information Portal for Multiplatforms. Table 2 shows the Enterprise Information Portal publications that are included with the product.

Table 2. Enterprise Information Portal publications

File name	Title	Publication number
apgwork	*Workstation Application Programming Guide*[1]	SC27-1347-01
ecliinst	*Installing, Configuring, and Managing the eClient*	SC27-1350-02
eipinst	*Planning and Installing Enterprise Information Portal*	GC27-1345-01
eipmanag	*Managing Enterprise Information Portal*	SC27-1346-01
messcode	*Messages and Codes*[2]	SC27-1349-01

Notes:

1. The *Workstation Application Programming Guide* contains information about programming applications for both Content Manager and Enterprise Information Portal.
2. *Messages and Codes* contains the messages and codes for Content Manager and Enterprise Information Portal.

Support available on the Web

Product support is available on the Web. Click **Support** from the product Web sites at:

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

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

The documentation is included in softcopy with the product. To access product documentation on the Web, click **Library** on the product Web site.

An HTML-based documentation interface, called Enterprise Documentation Online (EDO), is also available from the Web. It currently contains the API reference information. Go to the Enterprise Information Portal Library Web page for information about accessing EDO.

How to send your comments

Your feedback helps IBM to provide quality information. Please send any comments that you have about this publication or other Content Manager or Enterprise Information Portal documentation. You can use either of the following methods to provide comments:

- Send your comments from the Web. Visit the IBM Data Management Online Reader's Comment Form (RCF) page at:

  www.ibm.com/software/data/rcf

  You can use the page to enter and send comments.
- Send your comments by e-mail to comments@vnet.ibm.com. Be sure to include the name of the product, the version number of the product, and the name and part number of the book (if applicable). If you are commenting on specific text, include the location of the text (for example, a chapter and section title, a table number, a page number, or a help topic title).

What's new in Enterprise Information Portal Version 8

Version 8.2: Version 8.2 includes a variety of enhancements. Version 8.2 adds more functionality to system administration workflow and supports the latest in database technology, DB2 Universal Database Version 8.1. These highlights, and other enhancements to the Version 8.2 product, are summarized below:

Enterprise Information Portal name change to IBM Information Integrator for Content

Enterprise Information Portal has been renamed to Information Integrator for Content. Although the names of the books have changed for Version 8.2, the text within the books continues to show the product name Enterprise Information Portal. When searching the Web for more information, you can continue to use Enterprise Information Portal, or EIP, until the transition to the new name is complete.

Support for DB2 UDB V8.1

Enterprise Information Portal V8.2 supports. The connection concentration feature of DB2 V8.1 provides increased scalability for two-tier applications and clients.

Federated folders

eClient now has the ability to organize documents and native folders from multiple repositories into a single federated folder and start that folder on a workflow. Federated folders also allows users to persistently store search results in the EIP federated database where users can retrieve them at any time. Full CRUD (create, retrieve, update, and delete) operations are available against these federated folders without re-indexing.

Advanced workflow collection points

Workflow is now fully supported on AIX and Solaris. The workflow builder, APIs, Collection Points Monitor, and JavaBeans provide improved workflow function and usability.

Microsoft Visual Studio .NET for building applications

The Enterprise Information Portal 8.1 and later APIs now support Microsoft Visual Studio .NET for writing content management applications or to integrate applications built using Microsoft Visual Studio .NET.

Version 8.1: Version 8.1 begins a legacy of integration and versatility. One of the many highlights and improvements from previous Content Manager products is the new data model structure which allows for more document customization. The changes to the Content Manager product in Version 8.1 are summarized below:

Support for Sun Solaris

You can install Java connectors, features, and databases on Solaris systems.

Common system administration

A single client application provides separate access to Content Manager and Enterprise Information Portal administration.

New connectors

- The ICM connector for Content Manager Version 8 Release 1 allows you to take advantage of Content Manager Version 8's powerful document storage features.
- The new C++ Extended Search Version 3.7 connector runs on AIX®.

Improved connectors

- Parametric text searches are supported from the federated layer and through a direct Extended Search connection.
- Functional enhancements and performance improvements to the OnDemand connector, including:
  - Modifications to the structure of an OnDemand DDO.
  - Asynchronous search is now supported

IBM Web Crawler

IBM Web Crawler is a feature that allows users to search for and summarize information on the Web and in Lotus Notes® databases.

Workflow enhancements

Workflow is now fully supported on AIX and Solaris. The workflow builder, APIs, and JavaBeans provide improved workflow function and usability.

Information center

The browser-based information center includes the documentation for Content Manager, Enterprise Information Portal, and Content Manager VideoCharger™. Topic-based information is organized by product and by task (for example, Administration). In addition to the provided navigation mechanism and indexes, a search facility also aids retrievability.

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features for this product include:

- The ability to operate all features using the keyboard instead of the mouse.

- Support for enhanced display properties.
- Options for video and audio alert cues.
- Compatibility with assistive technologies
- Compatibility with operating system accessibility features
- Accessible documentation formats

What's new in Content Manager Version 8

Version 8.2: Version 8.2 includes a variety of enhancements from Version 8.1. Version 8.2 adds more workflow features to the eClient, increases resource management function, and supports the latest in database and client technology, including DB2 Universal Database Version 8.1, Oracle Version 8.1.7.4 and Version 9.2.0.1, and WebSphere Version 5. These highlights, and other enhancements to the Version 8.2 product, are summarized below:

Enterprise Information Portal name change to IBM Information Integrator for Content

Enterprise Information Portal has been renamed to Information Integrator for Content. Although the names of the books have changed for Version 8.2, the text within the books continues to show the product name Enterprise Information Portal. When searching the Web for more information, you can continue to use Enterprise Information Portal, or EIP, until the transition to the new name is complete.

Support for Oracle Version 8.1.7.4 or Version 9.2.0.1 or later

Content Manager V8.2 adds support for Oracle databases managing the metadata stored in both library server and resource manager. Migration tools are included for Oracle users of Content Manager Version 7. **Note:** Oracle does not manage Enterprise Information Portal database server contents.

Replication

Content Manager V8.2 includes resource manager replication, which is the ability to store objects in multiple locations, managed by replication resource managers. Object replicas will behave as LAN cache objects for improved load balancing.

LAN cache

LAN cache support in Content Manager V8.2 provides application-transparent caching, using local servers as defined by the system administrator.

Support for DB2 UDB V8.1

Content Manager V8.2 and Enterprise Information Portal V8.2 supports DB2/UDB V8.1. The connection concentration feature of DB2 V8.1 provides increased scalability for two-tier applications and clients (such as the Content Manager V8 Client for Windows). DB2/UDB V8.1 has replaced the DB2 Universal Database Text Information Extender (TIE) with Net Search Extender (NSE).

Support for WebSphere Application Server Version 4 and Version 5

WebSphere Application Server Version 5 introduces server deployment and data access and management from any web browser.

Federated folders

eClient now has the ability to organize documents and native folders from multiple repositories into a single federated folder and start that folder on a workflow. Federated folders also allows users to persistently store search results in the EIP federated database where users can retrieve them at any time. Full CRUD (create, retrieve, update, and delete) operations are available against these federated folders without re-indexing.

Advanced workflow collection points
Workflow is now fully supported on AIX and Solaris. The workflow builder, APIs, Collection Points Monitor, and JavaBeans provide improved workflow function and usability.

Microsoft Visual Studio .NET for building applications
The Content Manager and Enterprise Information Portal 8.1 and later APIs now support Microsoft Visual Studio .NET for writing content management applications or to integrate applications built using Microsoft Visual Studio .NET.

Version 8.1: Version 8.1 begins a legacy of integration and versatility. One of the many highlights and improvements from previous Content Manager products is the new data model structure which allows for more document customization. The changes to the Content Manager product in Version 8.1 are summarized below:

Improved performance
The library server and resource manager use DB2 stored procedures and leverage DB2 technology to significantly reduce network traffic and improve performance and scalability.

Support for Sun Solaris
Both the library server and resource manager can be installed on Sun Solaris.

Enhanced data model
The new hierarchical data model provides the basis for customized compound document management solutions.

Improved workflow
Through integrated document routing, workflow capabilities have been improved with sequential routing, dynamic routing, and collection points.

Integrated text search
In addition to attribute-based searching, client users can now perform full-text searching on text-based document information. The text search function now uses the DB2 Universal Database Text Information Extender, which contributes to a streamlined process for setting up text searching.

Common system administration
A single client application provides separate access to Content Manager and Enterprise Information Portal. Within Content Manager, administrative domains provide a way to limit administrative access to subsections of the library server.

Full-function desktop client and enhanced eClient
Client enhancements provide users with an out-of-the-box application for rapid deployment or line of business application integration. The Client for Windows supports integrated text search, document routing, the hierarchical data model (to a single child component level),

versioning, and index during import. The eClient includes integrated text search, EIP advanced workflow, version control, and multi-valued attributes.

Easier installation

Installation is consistent across supported operating systems and customized installation information is provided by the Start Here CD's Planning Assistant. Silent and console installations are also provided.

Information center

The browser-based information center includes the documentation for Content Manager, Enterprise Information Portal, and Content Manager VideoCharger. Topic-based information is organized by product and by task (for example, Administration). In addition to the provided navigation mechanism and indexes, a search facility also aids retrievability.

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features for this product include:

- The ability to operate all features using the keyboard instead of the mouse.
- Support for enhanced display properties.
- Options for video and audio alert cues.
- Compatibility with assistive technologies
- Compatibility with operating system accessibility features
- Accessible documentation formats

PeopleSoft and Siebel integrations

Users of PeopleSoft and Siebel applications can now configure these applications to access content stored in a variety of content servers using the eClient.

Introducing Enterprise Information Portal

Many paper-intensive enterprises, such as insurance companies and financial institutions, administer large volumes of business-related content. The need for an enterprise solution for managing and accessing business information spans many industries.

A *content server* is a software system that stores multimedia, business forms, documents, and related data, along with metadata that allows employees to process and work with the content. When there is no way to effectively connect disparate content servers, a business can waste time and money by duplicating information or training employees to perform multiple searches.

Enterprise Information Portal provides leading-edge technology to bring all of your enterprise resources to your workstation desktop. IBM Enterprise Information Portal for Multiplatforms can help you maximize the value of your information and multimedia assets by connecting disparate content servers through a single client. With an IBM Enterprise Information Portal for Multiplatforms client users can quickly and concurrently access all connected content servers. Users can also do information mining or "intelligent" searches across content servers, including the Web or an intranet, and they can perform workflow tasks within your business processes.

With IBM Enterprise Information Portal for Multiplatforms, you can customize applications for your enterprise. Using the IBM Enterprise Information Portal for Multiplatforms toolkits, application programmers can write both desktop and Web-based applications.

This chapter provides an overview of IBM Enterprise Information Portal for Multiplatforms. A scenario about a fictitious insurance company, XYZ Insurance, demonstrates IBM Enterprise Information Portal for Multiplatforms's features and functionality.

Searching for customer information

XYZ Insurance (XYZ), a large property and casualty insurance company, has an extensive collection of photographs, claims, policies, adjuster's notes, reports from experts, and other business documents.

XYZ keeps all memos that are sent to policy holders, along with medical and appraisal electronic forms in Lotus® Domino™.Doc file cabinets. XYZ archives all policy declarations, notices, and invoices in a Content Manager OnDemand server for long-term storage and quick access. XYZ stores all claim forms, photographs, and letters received from policy holders in a Content Manager for AS/400® system folder. XYZ keeps reports from experts in a DB2 Universal Database™ (DB2® UDB) Data Warehouse Center Information Catalog Manager. XYZ also stores corporate media assets such as high-resolution graphics in a Content Manager system for the advertising, public relations, and new business departments to share. In addition, XYZ keeps information, such as company procedures, on its company intranet.

The need

Claims, customer calls, and general policy holder servicing cannot be handled with the content from one server because employees need to access all customer information. To provide customer service, employees require simultaneous access to a variety of content servers. XYZ Insurance needs a solution that connects their content servers and their company intranet for searching and retrieving information. They also want to expand their use of workflow processing.

Many different employees need to access documents, from clerks to claim adjusters to agents. XYZ must restrict access to certain items, while providing unlimited access to others. XYZ also wants an easy-to-use interface to reduce the need for training.

The solution

XYZ Insurance deploys IBM Enterprise Information Portal for Multiplatforms because the comprehensive search technologies allow them to connect and search all of their content servers for the retrieval of data. Now, when an XYZ Call Center representative receives a call, a single federated search retrieves all of the necessary policy holder information.

XYZ Insurance also uses Enterprise Information Portal Information Mining feature to search for and retrieve information from the company's intranet. They also want to expand their use of workflow processes.

The Enterprise Information Portal solution

IBM Enterprise Information Portal for Multiplatforms is a comprehensive product; its components work together to provide a solution uniquely suited to your enterprise. Centered on a multiple-tier architecture, IBM Enterprise Information Portal for Multiplatforms provides an administration client for managing searches, clients (as samples) for running searches, and connectors for connecting to disparate content servers such as Content Manager, Content Manager ImagePlus® for OS/390®, Content Manager OnDemand, Lotus Domino.Doc, DB2 UDB, DB2 DataJoiner®, and DB2 Data Warehouse Center Information Catalog Manager. You can write additional connectors for additional content servers.

Figure 1 on page 3 shows the concept of the multiple-tier architecture of IBM Enterprise Information Portal for Multiplatforms.

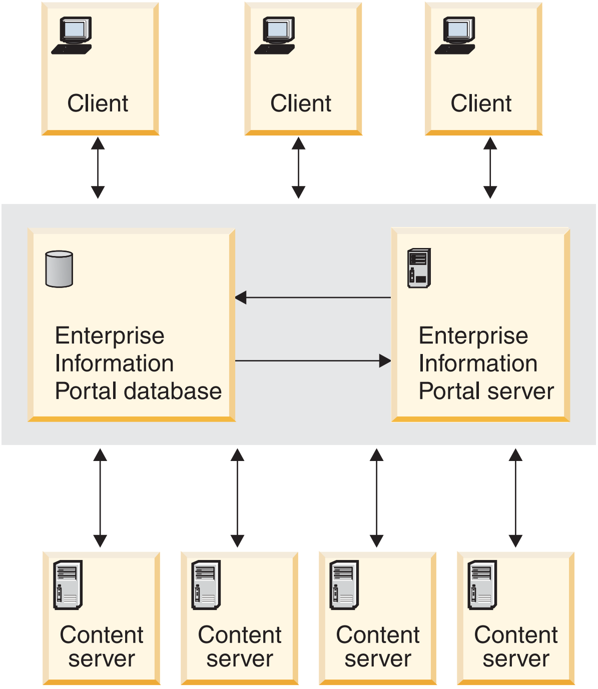

Client
Client
Client
Enterprise Information Portal database
Enterprise Information Portal server
Content server
Content server
Content server
Content server

Figure 1. Multiple-tier architecture

The IBM Enterprise Information Portal for Multiplatforms architecture allows your client applications to run single searches on one or more content servers. To perform searches, a client uses search templates that were defined by the IBM Enterprise Information Portal for Multiplatforms administrator.

Using a search template, the client runs a federated search. A *federated search* is one that runs simultaneously across multiple content servers whose native attributes have previously been mapped to the federated attributes used in the search template. The IBM Enterprise Information Portal for Multiplatforms search templates contain search criteria, which reference federated attributes that are mapped to native attributes on each of the content servers. The IBM Enterprise Information Portal for Multiplatforms administrator creates the search templates. IBM Enterprise Information Portal for Multiplatforms provides connectors as a common interface to heterogeneous interfaces of content servers. The content servers then return data objects to the client.

The IBM Enterprise Information Portal for Multiplatforms architecture provides the following advantages:

- Access using a single query to multiple and varying content servers that support e-business transactions and customer service applications.
- Information mining capability across multiple content servers, including the Web.
- Workflow process access to data across multiple, heterogeneous content servers.
- Support for the development of client applications that are independent of data's location on any content server, because of the separation of client applications, indexes, and data.

IBM Enterprise Information Portal for Multiplatforms components

This section describes each IBM Enterprise Information Portal for Multiplatforms component. These components are shipped as part of the IBM Enterprise Information Portal for Multiplatforms product.

Administration database

The IBM Enterprise Information Portal for Mulitplatforms database is a DB2 UDB database. It stores all of the information you need to manage EIP and its components.

Migrating your Enterprise Information Portal Version 7.1 database: You must migrate your data from Enterprise Information Portal Version 7.1 *before* using your Enterprise Information Portal Version 8 Release 2 administration database.

Administration client

The system administrator uses the IBM Enterprise Information Portal for Multiplatforms administration client to:

- Define each content server for federated searching.
- Identify native entities and attributes on content servers and map them to federated entities.
- Create search templates.
- Identify and manage users who can access search templates, the information mining feature, and workflow processes.
- Define business workflow processes.

This information is stored in the IBM Enterprise Information Portal for Multiplatforms database.

For convenience, it is recommended that you install the administration client on the same workstation, or server, as the IBM Enterprise Information Portal for Multiplatforms database.

You can also have as many administration clients as you want on other workstations. To do this configuration, you need to do one of the following:

- Have DB2 Client support installed and use the DB2 Client Configuration Assistant to configure access to the system administration database on each workstation on which the administration client is installed.
- Use a Remote Method Invocation (RMI) configuration by starting the RMI server on which the IBM Enterprise Information Portal for Multiplatforms database is installed. You need to make sure that your `\CMBROOT\cmbclient.ini` file points to this server. The location of this INI file is where the directory is specified by CMCOMMON keyword in the cmbcmenv.properties file. This file can also be on a URL specified by the CMCOMMON_URL keyword in the cmbcmenv.properties file.

Connectors

Connector classes permit client applications to access content servers and the IBM Enterprise Information Portal for Multiplatforms database. IBM Enterprise Information Portal for Multiplatforms ships the following connectors:

- Relational database connectors (DB2, JDBC, ODBC)
- Federated connector (to the IBM Enterprise Information Portal for Multiplatforms database)
- Content Manager Version 8 Release 2

- Content Manager Version 7 Release 1 connector
- Content Manager OnDemand connector
- Content Manager ImagePlus for OS/390 connector
- Content Manager for AS/400 connector
- Lotus Domino.Doc connector
- Extended search connector

The federated connector contains the connector class for the IBM Enterprise Information Portal for Multiplatforms database. Each content server connector contains the appropriate connector classes.

In a Java environment, the Java versions of the connectors are either local or remote. In C++, there are only local connectors. Local connectors are a set of connector classes you use for directly connecting to various content servers. Local connectors can reside on an IBM Enterprise Information Portal for Multiplatforms desktop client or on an IBM Enterprise Information Portal for Multiplatforms RMI server. Remote connectors are used to connect to a content server through an RMI server or RMI server pool member. Using the remote connector eliminates the need to connect directly to the content server.

IBM eClient

eClient is a browser-based user interface for access to documents stored in Content Manager (all platforms), Content Manager OnDemand (all platforms), and Content ManagerImagePlus for OS/390.

Information Mining

Information Mining provides linguistic services to find hidden information in text documents on content servers. During processing of the text documents, metadata (information about data) is created that can be summarized, categorized, and searched. IBM Enterprise Information Portal for Multiplatforms provides samples that show how to use Information Mining capabilities in a thin client. You can build your own desktop client or thin client to work with Information Mining.

IBM Web Crawler

IBM Web Crawler allows you to search and import content from the web. Search results and metadata can then be analyzed and categorized using Information Mining tools. IBM Web Crawler can crawl http, ftp, ntp, lotus, and domino servers.

Workflow

With Enterprise Information Portal, you can control the flow of work in your business. By using Enterprise Information Portal workflow feature, you can define and run the workflow process of a work group, department, or enterprise. Using a graphical builder, you can construct a comprehensive, easy-to-understand graphical representation of a workflow process in the Enterprise Information Portal workflow builder. Your users can then use the defined workflow process to perform their tasks, using a client that you develop or the Enterprise Information Portal thin client samples.

Content Manager Version 7 Text Search Engine

You can use this feature to automatically index, search, and retrieve documents stored in Content Manager Version 7. Users can locate documents by searching for words or phrases.

Restriction: The text search server and client is an optional Content Manager feature that you can configure and run with pre-Version 8.1 Content Manager servers only. If you do not use the pre-Version 8.1 Content Manager servers, do not install this feature.

Content Manager Version 7 image search server and client

This feature uses IBM QBIC® (query by image content) technology with which you can search for objects by certain visual properties, such as color and texture.

Restriction: Image search server’s functionality is not supported in both Enterprise Information Portal Version 8 and Content Manager Version 8. Its functionality is still available using pre-Version 8.1 Content Manager.

What’s new in Version 8.2 APIs

Comprehensive code samples

Code samples have been updated for Content Manager Version 8.2 features. In addition, all code samples now appear inside boxes (labelled by language) for better readability.

DB2 DataJoiner restrictions

If your content server requires DB2 Universal Database Version 8.1, then the EIP DataJoiner connector will not work due to binding problems. This is because DataJoiner 2.1 does not recognize SQL statements within the DB2 Version 7 bind files.

To solve this problem, read the FAQ question at the DataJoiner Web site (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?″. The answer will instruct you on the following steps to connect to the DataJoiner Version 2.1.1 server:

1. Download the two bind files and rename them as db2cliws_dj.bnd and db2clprr_dj.bnd.
2. Enter the following DB2 commands:

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

DB2 Data Warehouse Manager Information Catalog Manager restrictions

The IC connector requires DB2 Universal Database Version 7.2, and will not work with DB2 UDB Version 8.1.

Federated folders (Java only)

Enterprise Information Portal Version 8 Release 2 now provides special federated entities that can hold *federated folders*. These federated folders can store the combined results from a federated query, such as a document from Content Manager and a related document from OnDemand. You can then send the results directly into a workflow.

Microsoft Visual Studio .NET support

Both Enterprise Information Portal and Content Manager versions 8.2 (and later) APIs now support Microsoft Visual Studio .NET.

What’s new in Version 8.1 APIs

Enterprise Information Portal Version 8 Release 2 provides unprecedented access to disparate content servers. The new features and components include:

- XML import capabilities:

  You can now use XML to import and export content into Content Manager through DDOs and XDOs (using Java APIs).
- Improved installation procedures
- Additional connectors for relational databases:

  Enterprise Information Portal provides relational database connectors for DB2 UDB, DB2 DataJoiner, DB2 Data Warehouse Manager Information Catalog Manager, and other databases through JDBC or ODBC drivers.
- Advanced information mining and search capabilities:

  Information Mining offers advanced text searching using a flexible query that you can restrict to documents of certain categories.
- Workflow capabilities:

  By using Enterprise Information Portal workflow feature, you can define and run the workflow process of a work group, department, or enterprise.
- Federated level access control:

  You can control access to Enterprise Information Portal information mining and workflow processes through the use of privilege sets and access control lists. Additional access control to data can be managed by the access control features of each content server.
- Additional support for Content Manager:
  - List, add, retrieve, update, and delete of content class
  - Asynchronous retrieval of object content

New and changed Java classes

The Java common classes are used by all of the connectors. This package contains interfaces (such as dkDatastore) and abstract classes (such as dkAbstractDatastore and dkXDO) as well as concrete classes (such as DKDDO) used by the connectors.

New classes:

- dkAbstractAccessControlList
- dkAbstractAttrGroupDef
- dkAbstractAuthorizationMgmt
- dkAbstractConfigurationMgmt
- dkAbstractDatastoreAdmin
- dkAbstractDataObjectBase
- dkAbstractPrivilege
- dkAbstractPrivilegeGroup
- dkAbstractPrivilegeSet
- dkAbstactResultSetCursor
- dkAbstractUserDef
- dkAbstractUserMgmt
- dkAttrGroupDef
- dkAuthorizationMgmt
- dkCheckableObject

- DKChildCollection
- dkConfigurationMgmt
- DKLinkCollection
- dkPersistentCheckableObject
- dkPrivilege
- dkPrivilegeGroup
- dkUserDef
- dkUserGroupDef

Changed classes:

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

Changes in class behavior for Enterprise Information Portal Version 8

Between EIP 7.1 and EIP 8.2, some classes or class components have changed their behavior. This section describes those changes. For detailed information, see the *online API reference*.

- dkIterator: next() advances the iterator to the next item and gets that item and previous() moves to the previous item and gets that item
- dkXDO: getPidObject() and setPidObject(DKPid pid)

New and changed C++ classes

The C++ common classes package is used by all of the connectors. This package contains interfaces (such as dkDatastore) and abstract classes (such as dkAbstractDatastore and dkXDO) as well as concrete classes (such as DKDDO) used by the connectors.

New classes:

- dkAbstractAccessControlList
- dkAbstractAttrGroupDef
- dkAbstractAuthorizationMgmt
- dkAbstractConfigurationMgmt

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

Changed classes:

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

Changes in class behavior for Enterprise Information Portal 8.2

Between EIP 7.1 and EIP 8.2, some classes or class components have changed their behavior. This section describes those changes. For detailed information, see the *online API reference*.

- The use of DKString with DKAny on AIX has changed. To retrieve a DKString from a DKAny, you must now get the DKAny into a DKAny variable and assign it to a DKString variable.

Example: This example assumes that the DDO has been retrieved from a dkResultSetCursor fetchNext() API call from a content server and the DDO variable has been set, as shown below. This example loops through the data items in the DDO to find the string values.

```
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(); // This is how to get a DKString from a DKAny
  }
}
```

- Interfaces that implement dkXDO have changed with regards to the getPidObject and setPidObject methods.

Enterprise Information Portal application programming concepts

Enterprise Information Portal offers object-oriented (OO) application programming interfaces (APIs) that you can use to create query applications that access and display relational data as well as multimedia data. This chapter provides a brief overview of how these APIs fit into the Enterprise Information Portal architecture, and describes the object-oriented programming concepts on which the APIs are based.

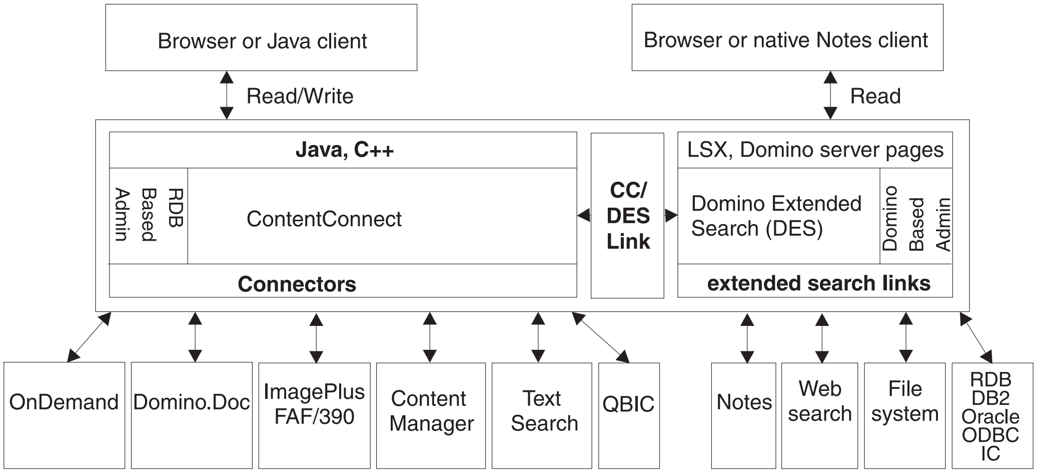


Figure 2. Enterprise Information Portal organization

Understanding data access through content servers

A content server is a data repository that is compatible with the DDO/XDO protocol. A content server supports user sessions, connections, transactions, cursors, and queries. Applications using the application programming interfaces (APIs) and class libraries described in this book can perform functions supported by the content servers, such as add, retrieve, update, and delete DDOs. Enterprise Information Portal supports the following content servers:

- Content Manager Version 8 Release 2
- Content Manager Version 7 Release 1
- Domino.Doc
- Extended Search
- ImagePlus for OS/390
- Content Manager OnDemand
- VisualInfo™ for AS/400
- DB2
- DB2 DataJoiner
- Information Catalog
- DB2 Warehouse Manager Information Catalog Manager
- JDBC/ODBC servers

Applications that use Enterprise Information Portal can create a federated content server, which acts as a common server. Enterprise Information Portal federated classes enable federated searching, retrieval, and updating across several content servers.

The Enterprise Information Portal federated content server and each of the content servers have different schemas. Integrating multiple heterogeneous content servers into a federated system requires conversion and mapping.

Schema mapping functions provide the schema information for each content server. The information provided by schema mapping is used during federated searching, federated collection, and Enterprise Information Portal system administration. Enterprise Information Portal keeps the schema and mappings, as well as other administration information in its administration database.

Understanding dynamic data object concepts

In compliance with Object Management Groups' (OMG) CORBA Persistent Object Service and Object Query Service Specification, Enterprise Information Portal provides an implementation of the dynamic data object (DDO) and its extension, the extended data object (XDO), which are part of the CORBA Persistent Data Service (PDS) protocols. The concepts of DDO and XDO are not specific to any one content server, and can be used to represent data objects in any database management system supported by Enterprise Information Portal.

The dynamic data object is an interface used to move data in and out of a content server. DDOs exist in the application and do not exist after an application terminates.

Dynamic data objects (DDO)

DDO is a content server-neutral representation of an object's persistent data. Its purpose is to contain all of the data for a single persistent object. It's also an interface to retrieve persistent data from, or load persistent data into, a content server.

A DDO has a single persistent ID (PID), an object type, and a set of data items whose cardinality is called the data count. Each data item can have a name, a value, an ID, one or more data properties, and data property count. Each data property can have an ID, a name, and a value.

For example, a DDO can represent a row of a database table whose columns are represented by DDO's data items and their properties. A DDO can contain one or more extended data objects (XDOs) that represent non-traditional data types. Figure 3 on page 13 shows dynamic data objects and data items.

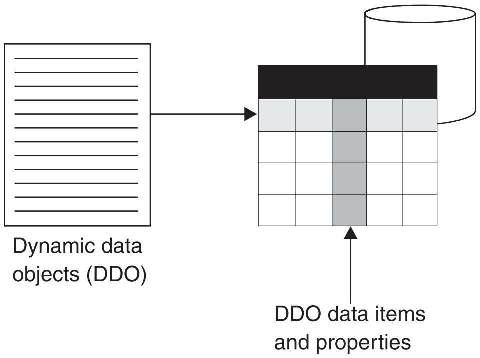


Figure 3. Dynamic data objects and items

Extended data objects (XDO)

An XDO is a representation of complex multimedia data, for example a resource item storing an image or document in Content Manager or a new data type introduced by a relational database's object-relational facilities, such as IBM DB2 Extenders.

XDOs complement DDOs by storing multimedia data of complex types and offering functions that implement the data type's behaviors in the application. XDOs can be contained in, or owned by, a DDO to represent a complex multimedia data object.

XDOs have a set of properties to represent such information as data types and IDs. XDOs can also be stand-alone dynamic objects. Figure 4 shows an example of XDOs.

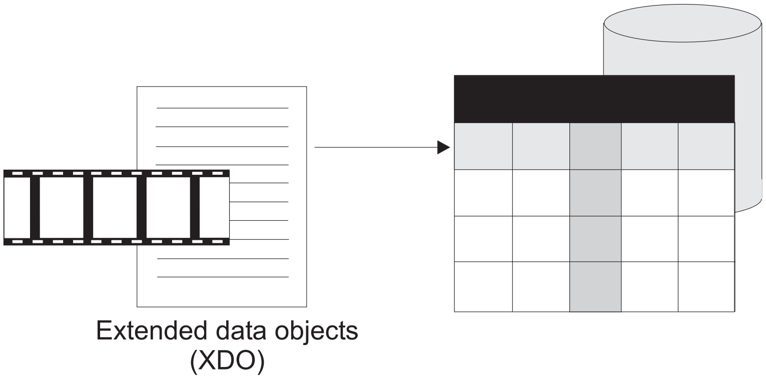


Figure 4. Extended data objects (XDOs)

Representing multimedia content

DDOs and XDOs can represent data objects of any type and structure. For example, a movie can be represented by a DDO. This DDO contains multiple data items, which represent attributes of the movie such as `Director_Name` or `Movie_Title`, and multimedia XDOs, which represent the movie's multimedia data such as video clips or still images.

In Content Manager 8.2, a DDO consists of all the metadata that describes the object, such as a document or image. An XDO extends the DDO functionality further to support resource content. Resource content is any type of content ranging from binary data or text to video and audio streams. An item that implements (and should be) an XDO is also a DDO and supports all of the functionality provided by a DDO plus the additional functionality that is provided by (and should be) an XDO.

Understanding content servers and DDOs

DDOs are created and dynamically associated with a content server. The association between a DDO and a content server is established with the DDOs PID.

In general, an Enterprise Information Portal application goes through the five steps listed below to move data in and out of a content server:

1. Create a content server.
2. Establish a connection to the content server.
3. Create the DDOs to be operated on, and associate the content server with the DDOs.
4. Add, retrieve, update, and delete the DDOs using appropriate methods.
5. Close the connection and destroy the content server.

Comparing DDO/XDOs with attribute values and item parts

A DDO corresponds to an item in Enterprise Information Portal. The DDO’s object type corresponds to the item’s associated item type. The data items of a DDO correspond to an item’s attributes. For example, in Content Manager an item type is created using a set of attributes, and an item is always indexed by an item type.

A DDO can hold one or more XDOs that correspond to item parts in Enterprise Information Portal.

Understanding persistent identifiers (PID)

The persistent identifier (PID) uniquely identifies a persistent object in any content server. A DDO’s PID consists of an item ID, a content server name, and other related information. When a DDO is added to a content server, the system assigns a unique PID to the DDO.

Because a DDO is a dynamic interface to persistent data that is moved in or out of content servers, different DDOs can represent the same persistent data entities, and therefore the DDOs can have the same PID. For example, a DDO can be created to move a data entity into a content server to store data persistently, and another DDO can be created to hold the same data entity that is checked out from the same content server for modification. In this case, these two DDOs share the same PID value.

Working with a federated content server and federated searching

Federated searching is the process of searching for data in one or more content servers. You use a DKDatastoreFed object for a federated search. Federated search works with classes that are specific implementations of dkDatastore, dkDatastoreDef, and other related classes that support federated searches. The specific federated classes work together with other common classes, such as those for queries, collections, and data objects and are part of the Enterprise Information Portal framework.

Federated classes work across different content servers, such as Content Manager ImagePlus for OS/390 or Domino.Doc. The classes provide a set of generic functions for federated search and access across the content servers. This common view, called *federated document model*, is illustrated in Figure 5.

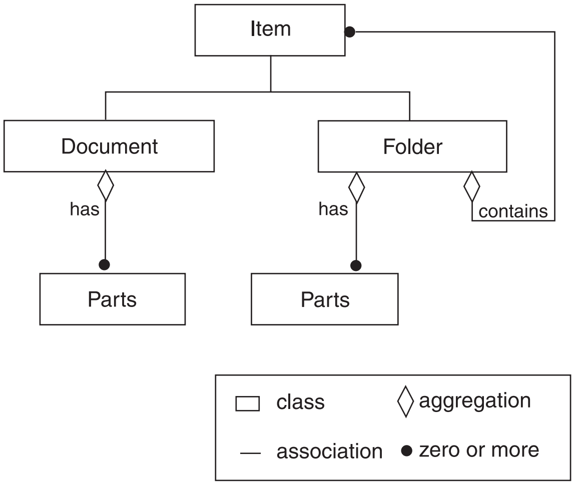


Figure 5. Federated document view

An item can be a document or a folder. A document can contain zero or more parts. A folder can have zero or more items which can be documents or other folders.

Not all content servers can support the federated document model. For example, a DB2 database does not have folders or parts. An item maps to a row in a DB2 or other relational database table, and is used if a content server does not support documents or folders.

In general, a document is represented in your program by a dynamic data object (DDO), which is a self-describing data object for transferring data into and out of a content server. The DDO itself has a general structure and supports a variety of models. It is not limited to the federated document model. This flexibility allows a DDO to represent data in different content servers, each with its own data model.

An *entity* is a content server object comprised of attributes. An *attribute* is a label used for metadata in content servers, for example, profiles, fields, or keywords are attributes in Domino.Doc content servers.

Each content server has its own terminology to explain the model it is supporting. Table 3 relates the terminology used for various content servers to the federated model:

Table 3. Mapping terminology for each content server

Content server	Data source	Entity	Attribute	View
Content Manager 8.2	Library server	item type	attribute	item type view or item type subset
Content Manager 7.1	Library server	index class	• attribute • key attribute	index class view
OnDemand	OnDemand server	• application group • folder	• field • criteria	N/A
ImagePlus	ImagePlus for OS/390 server	entity	attribute	N/A
Content Manager for AS/400	Content Manager for AS/400 server	index class	attribute	index class view
Domino.Doc	Domino server	library room cabinet binder	profile field keyword	N/A
Extended Search	Extended Search server	database name	database name	N/A
Relational database	IBM DB2 UDB, JDBC, ODBC, IBM DB2 DataJoiner	table	column	view
Information Catalog	DB2 Warehouse Manager Information Catalog Manager	index class	property	
Federated content server	mapping server	mapped federated entity	mapped federated attribute	search template
Federated content server that can hold federated folders	server	federated entity	federated attribute	federated folder

Figure 6 on page 17 illustrates a federated search. The federated search uses the Enterprise Information Portal federated content server, working through search templates. The federated content server then calls the searches for the individual content servers to perform the actual search on the content servers. This association is established by schema mapping.

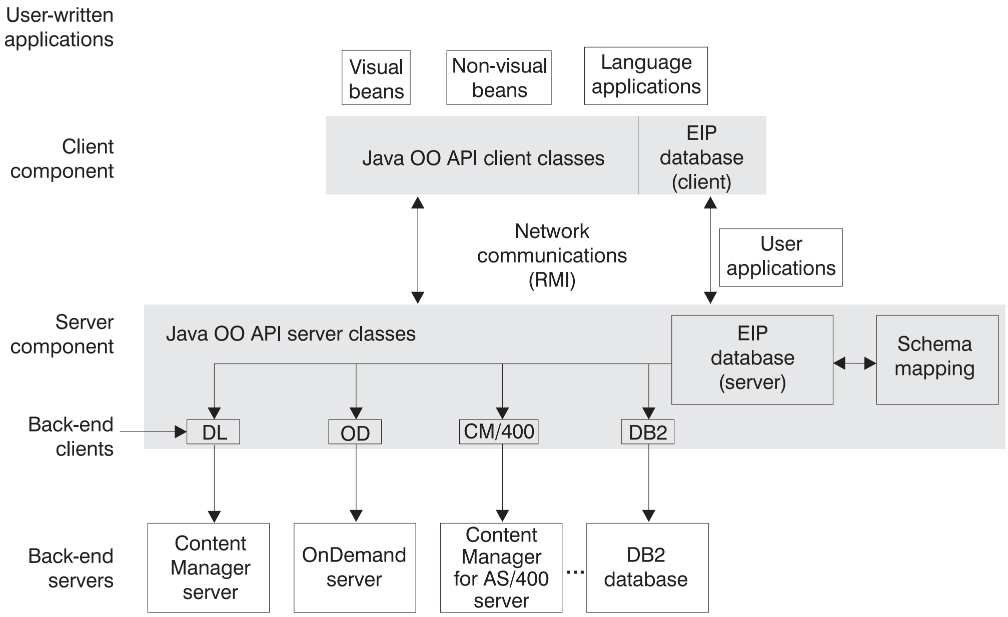


Figure 6. Structure of federated searches

The federated content server can use either local or remote connectors to connect to the content servers, and can use RMI for this communication. You can also develop applications on top of the API classes.

Federated schema mapping

A *schema mapping* represents a mapping between the schema in the content server and the structure of the items the user wants to process in the application. A *federated schema* is the conceptual schema of a Enterprise Information Portal federated content server and defines a mapping between the concepts in the federated content server and concepts in each participating content server. The schema mapping handles the difference between how the data is physically stored and how the user wants to process the data in an application.

The mapping information is represented in memory in schema mapping classes.

Using federated content server mapping components

In addition to schema mapping information for mapping the entities and attributes, a federated content server must also have access to the following information:

User ID and password mapping
To support a single logon feature, each user ID in the Enterprise Information Portal can be mapped to the corresponding user ID on each content server.

Content server registration
Each content server must be registered so that it can be located and logged on to by the Enterprise Information Portal.

The user ID and content server information is maintained in the Enterprise Information Portal administration database.

Running federated queries

To run a federated search using the APIs, start by creating a federated query string. You can then create and run the query in several ways:

- You can create a federated query object, DKFederatedQuery, passing it the query string; then invoke the execute or evaluate method on the object to process the query.
- You can pass the query string to the execute or evaluate method of the federated content server to process the query directly.

The query string is parsed into a federated query form, which is essentially a content server neutral representation of the query. The federated query form is the input for a federated search.

If the query comes from a graphical user interface (GUI) based application, the query does not need to be parsed and the corresponding federated query form can be directly constructed.

As a federated search is processed, Enterprise Information Portal performs the following steps:

- Translate the query canonical form into several native queries that run on each content server. The translation information is obtained from the schema mapping.
- Convert federated entities and attributes into native entities and attributes for each of the content servers. This process uses the mapping and conversion mechanisms described in the schema mapping.
- Filter only the relevant data during the construction of native queries.
- Form native queries and submit them to the individual content servers.

Each content server runs the submitted query. The results are returned to the federated query, which can process them as following:

- Convert native entities and attributes into federated entities and attributes according to the mapping information.
- Filter the results to include only the requested data.
- Merge the results from several content servers into a federated collection.

The result of a federated search is returned as a federated collection. You can create an iterator to access the individual collection members. Each call to the next method in the iterator returns a DKDDO object, which is a content server neutral dynamic data object.

The federated collection provides the facility to separate the query results according to the content server. Create a sequential iterator by invoking the createMemberIterator method in the federated collection. Using this sequential iterator, you can access each member collection, which is a DKResults object, and process it separately.

The components of a federated search and their relationships are illustrated in Figure 7 on page 19.

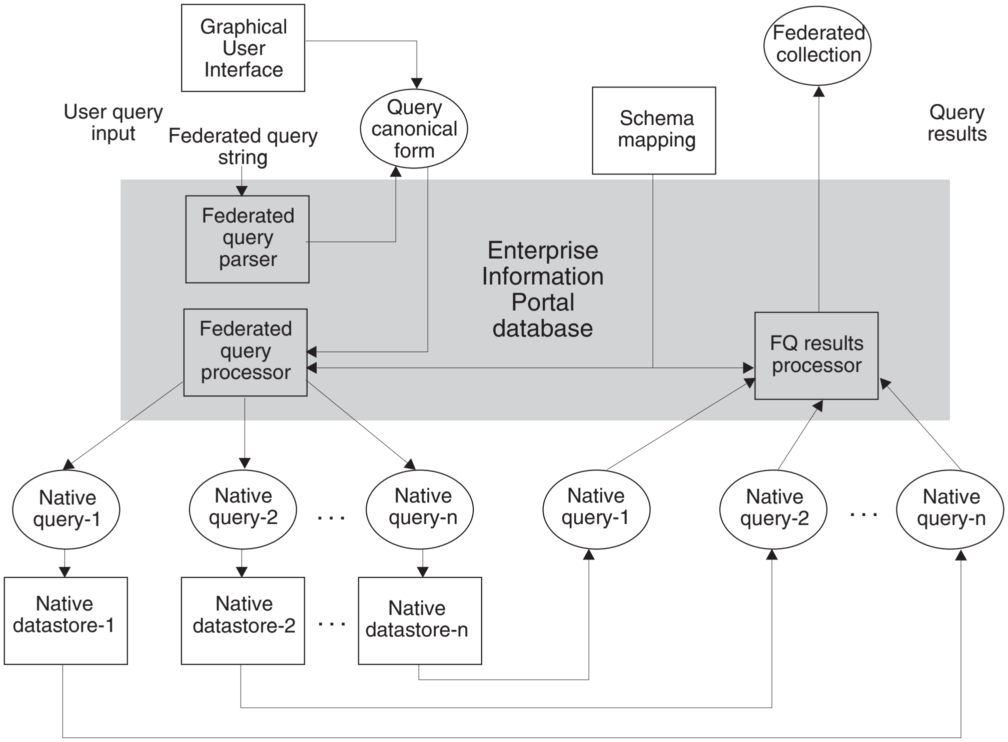


Figure 7. Federated query processing

Federated query syntax

When you create a federated query, it must be in the proper syntax, as shown below. The federated content server does not support image query.

```
PARAMETRIC_SEARCH=([ENTITY=entity_name,]
                      [MAX_RESULTS=maximum_results,]
                      [COND=(conditional_expression)]
                      [; ...]
                         );
             [OPTION=([CONTENT=yes_no_attronly]
                             )]

      [and

   TEXT_SEARCH=(COND=(text_search_expression)
                     );
            [OPTION=([SEARCH_INDEX={search_index_name | (index_list) };]
                      [ASSOCIATED_ENTITY={associated_entity_name)};]
                      [MAX_RESULTS=maximum_results;]
                      [TIME_LIMIT=time_limit]
                       )]
      ]
```

The NOT operator is not supported in federated searches.

Examples of federated query strings

Federated parametric query using the LIKE operator

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

Federated parametric query using the LIKE and > operator

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

Federated parametric query using the LIKE and < operator

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

Federated parametric query using the BETWEEN operator

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

MAX_RESULTS returns all results when set to zero.

Federated parametric query using the NOTBETWEEN operator

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

Federated parametric query using the IN operator

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

Federated parametric query using the NOTIN operator

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

Federated parametric query using the == operator

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

Federated parametric query using the <> operator

```
"PARAMETRIC_SEARCH = ( ENTITY = F_ICMNLSDB_Journal, MAX_RESULTS = 0,
COND = (fJSectionTitle <> 'not available') )"
```

Federated parametric query using the AND and OR operators

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

Federated parametric query using the CONTAINS_TEXT_IN_CONTENT operator

This example searches for text in the content. The content can be a word or a phrase. This is only valid when the text-searchable federated entity (FedTextResource) is mapped to a Content Manager Version 8 text-searchable item type or an Extended Search text-searchable entity.

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

Federated parametric query using the CONTAINS_TEXT operator

Searches for text in attribute values. The text can be a word or a phrase. This is only valid when the text-searchable federated attribute (fJTitle) is mapped to a Content Manager Version 8 text-searchable attribute or a Extended Search text-searchable attribute.

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

Federated text query

Searches for text in content. The text can be a word or a phrase. The ASSOCIATED_ENTITY keyword is only applicable when a federated entity is text-searchable. This is only valid when the text-searchable federated entity (FedEntity) is mapped to a Content Manager Version 8 text-searchable item type or an Extended Search text-searchable entity.

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

Federated text query

Searches for text in the content. This can be a word or a phrase. The federated text index, FedTMINDEX, is mapped to a Content Manager Version 7 Text Miner search index. The SEARCH_INDEX keyword is only applicable for this type of mapping. To specify a word or a phrase in the condition you must set the configuration string to GENFEDTEXTQRY=YES when defining a Content Manager Version 7 server that supports text search.

```
"TEXT_SEARCH = ( COND = ('operating system') );
OPTION = ( SEARCH_INDEX = FedTMINDEX)"
```

Federated text query

Searches for text in content across Content Manager Version 7, Content Manager Version 8, and Extended Search. This can be a word or a phrase. The federated text index, FedTMINDEX, is mapped to a Content Manager Version 7 Text Miner search index. The SEARCH_INDEX keyword is only applicable for this type of mapping. To specify a word or a phrase in the condition you must set the configuration string to GENFEDTEXTQRY=YES when defining a Content Manager Version 7 server that supports text search. The ASSOCIATED_ENTITY keyword is only applicable when a federated entity is text-searchable. This is only valid when the text-searchable federated entity (FedTextResource) is mapped to a Content Manager Version 8 text-searchable item type or an Extended Search text-searchable entity.

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

Federated parametric and text query using the OR operator

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

Federated parametric and text query using the AND operator

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

Federated parametric and text query on attributes using the OR operator

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

Federated parametric and text query on attributes using the OR operator

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

Storing query results in federated folders (Java only)

Enterprise Information Portal Version 8 Release 2 now provides special federated entities that can hold *federated folders*. These federated folders can store the combined results from a federated query, such as a document from Content Manager and a related document from OnDemand. You can then send the results directly into a workflow.

EIP stores the folders as DDOs, which you can add to a DKFolder collection. You can also store the folders as XDOs in a DKParts collection.

The special federated entity uses additional tables to hold the folders. All other functionality (such as queries, APIs, and attributes) behaves identically to a normal federated entity. The special entity only stores DDO PIDs in the folders.

If you choose not to map a special federated entity, then the federated query only searches special federated entitites. If you choose to map a special federated entity to other native entities, then the federated query additionally searches those entitites.

For code samples, refer to the `CMBROOT\dk\samples` directory.

Working with system administration

Enterprise Information Portal provides the classes and APIs for you to access system administration functions. Refer to the *online API reference* for information on the specific classes.

Customizing the EIP system administration client

The EIP system administration client supports extending the system administration application to include custom functions:

- You can replace the user and user group dialogs in the EIP system administration client with your own dialogs.
- You can add new nodes to the hierarchy in the EIP system administration client.
- You can add new menu items to the Tools menu in the system administration client.

You can call user exits before and after you log on to the EIP system administration.

Programming with the application programming interfaces (APIs)

The application programming interfaces (APIs) are a set of classes that access and manipulate either local or remote data. This section describes the APIs, the implementation of multiple search functions, and Internet connectivity.

The APIs support:

- A common object model for data access.
- Multiple search and update across a heterogeneous combination of content servers.
- A flexible mechanism for using a combination of search engines; for example, the Content Manager text search feature.
- Workflow capability.
- Administration functions.
- **Java only:** Client/server implementation for Java application.

Multistream support for the Java APIs is fully enabled for Windows® servers only. AIX servers, clients, and Windows clients cannot support multistreaming.

Understanding differences between the Java and C++ APIs

The list below describes differences between the IBM Enterprise Information Portal for Multiplatforms Java and C++ API sets:

- The operators defined in the C++ API are not defined in the Java API. They are supported as Java functions.
- The Java class object (java.lang.Object) is used in place of the C++ class DKAny to represent a generic object.
- Common and global constants are defined in the interface DKConstant in the Java API; in C++ they are in `DKConstant.h`.
- The Java APIs use Java's garbage collector.
- The Java functions DKDDO.toXML() and DKDDO.fromXML() are not available in C++.

Understanding client/server architecture (Java only)

The APIs provide a convenient programming interface for application writers. APIs can reside on both the EIP server and the client (both provide the same interface). The client API communicates with the server to access data through the network via Java RMI (Remote Method Invocation). Communication between the client and the server is performed by classes; it is not necessary to add any additional programs.

API classes consist of the following packages: server, client, cs, and common. The client and server classes provide the same APIs, but have different implementations.

- The server package is `com.ibm.mm.sdk.server`. The classes in the server package communicate directly with the federated or backend content server.
- The client package is `com.ibm.mm.sdk.client`. The classes in the client package communicate with the classes in the server package via RMI.

- The common classes are shared by both the client and server. Sometimes an application does not know where the content resides. For example, an application can have content residing on the client at one time and the server at another time. The `cs` package connects the client and server dynamically.

The client application must import the client package, the dynamic application must import the `cs` package, and server application must import the server package.

Although the same API is provided for the client and server, the client package has an additional exception item because it communicates with the server package. Note, however, that the client/server interface is not supported for all the connectors. For example, this is not supported by CM V8.

Packaging for the Java environment

The Enterprise Information Portal APIs are contained in four packages as part of `com.ibm.mm.sdk`: `common`, `server`, `client`, and `cs`.

`server (com.ibm.mm.sdk.server)`
Access and manipulate content server information

`client (com.ibm.mm.sdk.client)`
Communicate with the `server` package using Remote Method Invocation (RMI)

`common (com.ibm.mm.sdk.common)`
Common classes for both the server package, client package, and the cs package

`cs (com.ibm.mm.sdk.cs)`
Connect the client or server dynamically

Your application must use the `common` with either the `server` package for local applications, or the `client` package for applications that access the remote server, or the `cs` package.

Programming tips

Do not import client and server packages in the same program. If you are developing a client application, import the client package. Otherwise, import the server package. If you do not know where the content resides, then use the cs package (with the server or client packages). Importing multiple packages can result in compile errors.

Some connectors contain calls to C code in the implementation. For these connectors, use the client package for Web applications that require pure Java interfaces. The client package is created with pure Java programs; the server package can include JNI calls.

Because a client requires the exception, `java.rmi.RemoteException`, always attach this exception in the application whether the application runs on a server or client.

Setting up the Java environment (Java only)

When you set up your Windows, AIX, or Solaris environment, you must have imported the following packages:

server package

Import when a content server and application are on the server side

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

client package

Import when a content server and application are on the client side.

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

cs package

Import when a content server location is different from the application location.

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

Setting the Java environment variables for Windows

You can open a Windows command prompt with the environment set up for developing Enterprise Information Portal applications by selecting **Start ⟶ Programs ⟶ IBM Enterprise Information Portal for Multiplatforms 8.2 ⟶ Development Window.** As an alternative, you can run `cmbenv81.bat` in a Windows command prompt to set up the environment.

If you want to modify your environment variables, change the following:

PATH Make sure your PATH contains `*X*:\CMBROOT\DLL;` where `X` is the drive on which you installed Enterprise Information Portal.

CLASSPATH

Make sure your CLASSPATH contains `*X*:\CMBROOT\LIB\*xxx*` where `X` is the drive on which you installed Enterprise Information Portal and *xxx* are the .jar files, (for example, cmbfed81.jar).

Setting the Java environment variables for AIX

In the AIX environment, you can use a shell script, `cmbenv81.sh`, to set up your development environment for developing EIP applications.

If you do not use the script, you must set the following environment variables:

PATH Make sure your PATH contains `/usr/lpp/cmb/lib`

LIBPATH

Make sure your LIBPATH contains `/usr/lpp/cmb/lib`

LD_LIBRARY_PATH

Make sure your LD_LIBRARY_PATH contains `/usr/lpp/cmb/lib`

CLASSPATH

Make sure your CLASSPATH contains `/usr/lpp/cmb/lib/*xxx*` where `xxx` are the .jar files, (for example, cmbfed81.jar)

Use the `-qalign=packed` compiler option so that the objects align properly.

Setting the Java environment variables for Solaris

In the Solaris environment, you can use a shell script, `cmbenv81.sh`, to set up your development environment for developing EIP applications.

If you do not use the script, you must set the following environment variables:

PATH Make sure your PATH contains `/opt/cmb/lib`

LIBPATH
Make sure your LIBPATH contains `/opt/cmb/lib`

LD_LIBRARY_PATH
Make sure your LD_LIBRARY_PATH contains `/opt/cmb/lib`

CLASSPATH
Make sure your CLASSPATH contains `/opt/cmb/lib/`*`xxx`* where `xxx` are the .jar files, (for example, cmbfed81.jar)

Use the `-qalign=packed` compiler option so that the objects align properly.

Using Remote Method Invocation (RMI) with content servers

Because the client classes in the Java APIs need to communicate with the server classes to access data through the network, both the server and client must be prepared for client/server execution. On the server machine, the RMI server must be running to receive the request from the client using a specified port number. The client program requires the server name and port number. For communications between client and server, the client must know the port number of the server it needs to connect to.

An RMI server can connect to an infinite number (limited only by system resources) of content servers, but each server must be connected to at least one content server. A master RMI server can reference other RMI servers in the server pool. When an RMI client first searches for a content server, it starts an RMI server. If the content server is not found there, the RMI pool servers are searched next.

If the same RMI client searches for the content server again, the client searches the RMI server where it found the content server the first time.

To start the RMI server, use `cmbregist81.bat` on Windows or `cmbregist81.sh` on AIX or Solaris. Before starting the RMI server, define the correct port number and server type. For information on configuring and administering RMI servers, refer to *Planning and Installing Enterprise Information Portal* and *Managing Enterprise Information Portal*.

Setting up the C++ environment (C++ only)

When you set up your Windows or AIX environment, you must establish the settings described in this section. Table 4 on page 27 lists Library, AIX shared and DLL requirements.

Requirement: To use C++, you must install DB2 Client support and the Client Configuration Assistant on all remote servers accessing the Enterprise Information Portal database. The user ID and password used to connect to the database must be the same user ID and password you use with the Enterprise Information Portal database. For details, see the *Managing Enterprise Information Portal*.

Attention: `cmbcm81`*`x`*`.lib` is for release build and `cmbcm81`*`x`*`d.lib` is for debug build, where *x* represents either Version 6 or 7 (.Net) of the Microsoft Visual C++ compiler.

Table 4. Shared objects and DLL environment information

Connector	Library	Windows DLLs	Shared objects for AIX
Common Note: This is not a connector. It is a common set of APIs used by all of the connectors.	cmbcm81*x*.lib cmbcm81*x*d.lib	cmbcm81*x*.dll cmbcm81*x*d.dll	libcmbcm815.a
Content Manager Version 8	cmbicm81*x*.lib cmbicm81*x*d.lib	cmbicm81*x*.dll cmbicm81*x*d.dll cmbicmfac81*x*.dll cmbicmfac81*x*d.dll	libcmbicm815.a libcmbicmfac815.so
Earlier Content Manager	cmbdl81*x*.lib cmbdl81*x*d.lib	cmbdl81*x*.dll cmbdl81*x*d.dll cmbdlfac81*x*.dll cmbdlfac81*x*d.dll de_db2.dll de_db2_d.dll de_ora.dll de_ora_d.dll	libcmbdl815.a libcmbdlfac815.so
Federated	cmbfed81*x*.lib cmbfed81*x*d.lib	cmbfed81*x*.dll cmbfed81*x*d.dll cmbfedfac81*x*.dll cmbfedfac81*x*d.dll	libcmbfed815.a libcmbfedfac815.so
DB2 Universal Database Version 8.1	cmbdb281*x*.lib cmbdb281*x*d.lib	cmbdb281*x*.dll cmbdb281*x*d.dll cmbdb2fac81*x*.dll cmbdb2fac81*x*d.dll	libcmbdb2815.a libcmbdb2fac815.so
ODBC	cmbodbc81*x*.lib cmbodbc81*x*d.lib	cmbodbc81*x*.dll cmbodbc81*x*d.dll cmbodbcfac81*x*.dll cmbodbcfac81*x*d.dll	Not supported
OnDemand	cmbod81*x*.lib cmbod81*x*d.lib	cmbod81*x*.dll cmbod816*x*d.dll cmbodfac81*x*.dll cmbodfac81*x*d.dll	libcmbod815.a libcmbodfac815.so
ImagePlus for OS/390	cmbip81*x*.lib cmbip81*x*d.lib	cmbip81*x*.dll cmbip81*x*d.dll cmbipfac81*x*.dll cmbipfac81*x*d.dll	Not supported
VisualInfo	cmbv481*x*.lib cmbv481*x*d.lib	cmbv481*x*.dll cmbv481*x*d.dll cmbv4fac81*x*.dll cmbv4fac81*x*d.dl	Not supported
Domino.Doc	cmbdd81*x*.lib cmbdd81*x*d.lib	cmbdd81*x*.dll cmbdd81*x*d.dll cmbddfac81*x*.dll cmbddfac81*x*d.dll	Not supported
Domino Extended Search	cmbdes81*x*.lib cmbdes81*x*d.lib	cmbdes81*x*.dll cmbdes81*x*d.dll cmbdesfac81*x*.dll cmbdesfac81*x*d.dll	libcmbdes815.a libcmbodfac815.so

Setting the C++ environment variables for Windows

You can open a DOS command prompt with the environment configured for developing Enterprise Information Portal applications by selecting **Start → Programs → IBM Enterprise Information Portal for Multiplatforms 8.1 → Development Window.** As an alternative, you can run `CMBenv81.bat` in a DOS command prompt to set up the environment.

If you want to modify your environment variables, change the following:

PATH

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

where *x* is the drive that EIP is installed on.

INCLUDE

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

where *x* is the drive that EIP is installed on.

Setting the C++ environment variables for AIX

Refer to the sample MAK files in the `samples` directory for more information.

Set the following environment variables:

In the AIX environment, you can use one of three batch files to set up your development environment.

1. For a Bourne shell, use `cmbenv81.sh`
2. For a C shell, use `cmbenv81.csh`
3. For a Korn shell, use `cmbenv81.ksh`

Set the following environment variables:

NLS path

```
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
```

Building C++ programs

Follow the procedures for your compiler and development environment to create the MAK files and build your application.

When building an application using the C++ APIs that you will use for debugging, link your application with the debug version of the API libraries, that is, the ***d.lib** libraries. When building your final production application, link with the non-debugging libraries (***.lib**).

Working with Microsoft Visual Studio .NET

The Enterprise Information Portal and Content Manager versions 8.2 and later APIs support Microsoft Visual Studio .NET. When using Microsoft Visual Studio

.NET to build your applications, however, you must link to the appropriate library, which is determined by the connector and Visual Studio C++ version you are using, at compile time.

Example MAK files (supporting both Visual Studio Version 6 and Visual Studio .NET) are provided with the ICM API samples.

To determine the library that you need to connect to, at compile time, use the format *connector name*816.lib when using Microsoft Visual Studio C++ 6 and *connector name*817.lib when using Microsoft Visual Studio .NET, as demonstrated in the tables below.

Table 5. Example Microsoft Visual Studio C++ 6 library names

Connector	Library
Common	cmbcm816.lib
Content Manager 8.1 and later	cmbicm816.lib
Federated	cmbfed816.lib

Table 6. Microsoft Visual Studio .NET library names

Connector	Library
Common	cmbcm817.lib
Content Manager 8.1 and later	cmbicm817.lib
Federated	cmbfed817.lib

Setting the console subsystem for code page conversion on Windows

C++

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

void main(int argc, char *argv[])
{
   // set sub system to console at the beginning of program this
   // will cause the code page that the error messages are returned
   // in by DKExceptions to be converted from the Windows Graphical
   // User Interface (ANSI format) to the Console (OEM format)
   // If this is not specified the default is DK_SS_WINDOWS
   DKEnvironment::setSubSystem(DK_SS_CONSOLE);
   ...
}
```

Understanding multiple search options

Searches may be performed based on virtually any of piece of an item or component, text within an item or component, or text within resource content.

Content Manager Version 8 offers an integrated text feature, which no longer requires a separate text search facility (earlier Content Manager still does). See “Understanding text search” on page 180.

Use the multiple search options to search within a given content server, using one or a combination of supported queries, listed below, or search on the results of a previous search. Each search type is supported by one or more search engines. Not all content servers support multiple search options.

Parametric search
Searches text such as item and component properties, attributes, references, links, folder contents. For example, You use a parametric query to search for documents using a customer’s name. The query requires an exact match on the condition specified in the query predicate and the data values stored in the content server.

Text search

Searches any text marked as `textSearchable(true)` using a Text Search engine such as DB2 Net Search Engine (NSE). The query is based on the content of text fields for approximate match with the given text search expression; for example, the existence (or nonexistence) of certain phrases or word-stems.

Note: The DB2 Net Search Engine (NSE) was named DB2 Text Information Extender (TIE) in earlier versions of DB2.

Image search

Searches characteristics within images. The query is based on the content of images for approximate match with the given image search expression; for example, the presence of a certain color in the images.

Combined search

Searches using both parametric and text search.

Content Manager only: CM has one search engine and three choices for how and when searches should be executed (and results returned): Execute, Evaluate, and Execute with Callback. Refer to the `SSearchICM` sample for table and explanations.

Tracing

To handle problems that arise in your API applications, you can use tracing and exception handling.

Tracing text queries using Text Search Engine

The Text Search Engine (TSE) and all of its functions can only be used with earlier Content Manager. Content Manager Version 8 offers an integrated text feature, which does not require a separate text search facility. See "Understanding text search" on page 180.

The following environment variable setting writes the trace for a Text Search Engine query, in binary format, to a specified file:

`CMBTMDSTREAMTRACE=`*`fileName`*

(for example, `.\tm.out` for Windows or `./tm.out` for AIX)

The following environment variable settings writes the trace for the Text Search Engine API calls used during a text query to a specified file:

`CMBTMTRACE=`*`fileName`*

The following environment setting writes the text search terms to a specified file: `CMBTMTERM=`*`fileName`* (for example,`.\tmterm.out`)

Note: Content Manager Version 8 uses an integrated text search. If you are using earlier Content Manager, you can still use the Text Search Engine (TSE).

Tracing parametric queries

For earlier Content Manager using the Text Search Engine, use the following environment variable setting to write the parametric query passed to the folder manager:

`CMBDLQRYTRACE=`*`fileName`*

(for example, `<.\dlqry.out>` for Windows or `<./dlqry.out>` for AIX)

Handling exceptions

When the APIs encounter a problem, they throw an exception. Throwing an exception creates an exception object of DKException class or one of its subclasses.

When a DKException is created, the connector layer logs diagnostic information into a log file, assuming the default logging configuration is used. See *Messages and Codes* for more information on the log and configuration files used by the EIP APIs.

When a DKException is caught, it allows you to see any error messages, error codes, and error states that occurred while running. When an error is caught, an error message is issued along with the location of where the exception was thrown. Additional information such as error ID are also given. The code below shows an example of the throw and catch process for EIP and CM:

Java

```
try{
    ... EIP API Operations ...
}
catch (DKException exc){
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    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) {
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    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{
    ... EIP API Operations ...
}
catch (DKException &exc){
    // NOTE: Print Function Provided in SConnectDisconnectICM API Sample.
    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;
    // Print API Location(s) Detecting Error
for(unsigned int j=0; j < exc.locationCount(); j++){ //Print all locations
        const DKExceptionLocation* p = exc.locationAtIndex(j);
        cout << " Location " << j << ": " << p->fileName()   << "::"
             << p->functionName() << '['  << p->lineNumber() << ']' << endl;
    }
    if(exc.textCount()<=0) // Write statement if no locations.
        cout << "  Locations: <none> "                   << endl;
    // Error Message(s)
    for(unsigned int i=0; i < exc.textCount(); i++) // Print All Messages
        cout << "  Message " << i << ": " << exc.text(i) << endl;
    if(exc.textCount()<=0) // Notify user if no messages.
        cout << "   Messages: <none> "                   << endl;
    cout << "----------------------------------------"   << endl;
}
```

| For more information on error detection and handling, refer to the SConnectDisconnectICM API Education Sample available in CMBROOT\Samples\java\icm or CMBROOT\Samples\cpp\icm.

Constants

The constants specified are in the form of DK_CM_ (Common constants) or DK_*XX*_ (where the *XX* indicates different content servers). Refer to Table 7 on page 34 for a list of the extensions (extensions appended to each DKDatastore).

When you specify DDO constants, use DK_CM_DATAITEM_TYPE_ ... (for example, DK_CM_DATAITEM_TYPE_STRING) for property types. For attribute types, use the DK_CM_...*type* constants (for example, DK_CM_INTEGER).

Java

Common constants are defined in DKConstant.java. You can also review a text version of these constants in DKConstant.txt. Constants for specific content servers are defined in DKConstant*XX*.java; for example, constants unique to Content Manager are in DKConstantICM.java.

C++

Common constants are defined in DKConstant.h. For a list of constants and corresponding values, view DKConstant2.h (but do not include this in your program). Constants for specific content servers are defined in header files of the form DKConstant*XX*.h; for example, constants unique to Content Manager are in DKConstantICM.h.

Connecting to content servers

An object of the class DKDatastore*xx* (where *xx* indicates a specific content server) represents and manages a connection to a content server, provides transaction support, and runs server commands. Refer to Table 7 for the exact extensions.

Table 7. Server type and class name terminology

Content server	Class name
Content Manager Version 8.2	DKDatastoreICM
Earlier 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
Relational databases	DKDatastoreDB2, DKDatastoreJDBC (for Java) DKDatastoreODBC (for C++)

Establishing a connection

Each DKDatastore*xx* class provides methods for connecting to it and disconnecting from it. The following example uses a Content Manager library server named ICMNLSDB, the user ID ICMADMIN and password PASSWORD. For information on Content Manager, see Connecting to the Content Manager system; for other content servers, see Working with other content servers. The example creates a DKDatastoreICM object for the CM content server, connects to it, works with it, then disconnects from it.

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); //Create datastore object
dsICM.connect("ICMNLSDB","ICMADMIN","PASSWORD","");  //Connect to datastore

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

dsICM.disconnect(); // Disconnect from datastore
dsICM.destroy(); // Destroy reference
```

For the complete sample application, refer to the SConnectDisconnectICM.java in the CMBROOT\Samples\java\icm directory.

C++

```
DKDatastoreICM* dsICM = new DKDatastoreICM(); //Create datastore object
dsICM->connect("ICMNLSDB","ICMADMIN","PASSWORD",""); //Connect to datastore

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

dsICM->disconnect();  //Disconnect from datastore
delete(dsICM);        //Destroy reference
```

For the complete sample application, refer to the `SConnectDisconnectICM.cpp` in the `CMBROOT\Samples\cpp\icm` directory.

When connecting to a content server you must be aware of the requirements for each content server; for example, the password for ImagePlus for OS/390 can be no more than eight characters in length.

Connecting and disconnecting from a content server in a client

You use the same code in “Establishing a connection” on page 34 to access a content server from a client application. To do so, simply replace `import com.ibm.mm.sdk.server.*;` with `import com.ibm.mm.sdk.client.*;`. Your client application must handle any communications errors incurred.

Setting and getting content server options

You can access or set the processing options on a content server using the methods in DKDatastore*xx*. The following example shows how to set and get the option for establishing an administrative session on a Content Manager library server. See the *online API reference* for the list of options and their descriptions.

In this example for setting and getting a content server option in Content Manager, caching is turned off. For content servers supporting this option, it is recommended that the default (ON) be used. **Requirement:** A valid DKDatastoreICM object is already created in a variable called `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);
```

When getting a content server option, `output_option` usually is an integer, but you can cast it to be an object.

Listing content servers

DKDatastore*xx* provides a method to list the servers that it can connect to. The list of servers are returned in a DKSequentialCollection of DKServerDef*xx* objects (where *xx* identifies the specific content server).

Restriction: The Domino.Doc content server does not provide a method that lists the servers.

After you obtain a DKServerDef*xx* object you can retrieve the server name and server type, and use the server name to establish a connection to it.

The following example lists the servers that you are configured to connect to.

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM();    // Create a datastore object
dkCollection    coll = dsICM.listDataSources(); // Obtain data source list
dkIterator      iter = coll.createIterator();   // Create an iterator
while(iter.more()){                             // While there are more
    DKServerDefICM srvrDef = (DKServerDefICM) iter.next();
    System.out.println("Found server '"+srvrDef.getName()+"'");
}
```

C++

```
DKDatastoreICM* dsICM = new DKDatastoreICM();   // Create a datastore object
dkCollection* coll = (dkCollection*)dsICM->listDataSources(); // Obtain list
dkIterator*      iter = coll->createIterator(); // Create an iterator
while(iter->more()){                            // While there are more
    DKServerDefICM* srvrDef = (DKServerDefICM*) iter->next()->value();
    cout << "Found server '" << srvrDef.getName() << "'" << endl;
    delete(srvrDef);                            // Free memory
}
delete(iter);                                   // Free memory
delete(coll);
delete(dsICM);
```

Listing the entities and attributes for a content server

DKDatastore*xx* provides methods for listing the entities and their attributes, for a content server. Each attribute name is part of a name space. The default name space is used for all attributes where a name space is not specified.

The list of entities is returned in a DKSequentialCollection object of dkEntityDef objects. The attributes for an entity are returned in a DKSequentialCollection object of dkAttrDef objects. After you obtain a dkAttrDef object, you can retrieve information about the attribute, such as its name and type, and use the information to form a query.

For further details about these two methods, see the *online API reference*.

The following example shows how to retrieve the list of item types as well as the list of attributes from a Content Manager server.

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;
  // ------ Create the datastore and connect (assumes the
  //     parameters for the connection are previously set)
  DKDatastoreICM dsICM = new DKDatastoreICM();
  dsICM.connect(db,userid,pw,"");
  // ----- List the item types
  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());
    //continued . . .
```

Java (continued)

```
// ----- List the attributes
    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("      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("      sequenceNo " + attrDef.getSequenceNo());
      System.out.println("      userFlag " + attrDef.getUserFlag());
    }
  }
  dsICM.disconnect();
  }
  catch(DKException exc)
  {
// ----- Handle the exceptions
```

A complete sample, `SItemTypeRetrievalICM`, includes how to list item type definitions. Another complete sample, `SAttributeDefinitionRetrievalICM`, includes how to list attribute definitions. Both samples are available in the `CMBROOT\Samples\java\icm` directory.

The following C++ example shows how to retrieve the list of index classes and attributes from a Content Manager server:

C++

```
// Get a collection containing all Item Type Definitions.
DKSequentialCollection* itemTypeColl = (DKSequentialCollection*)
    dsICM->listEntities();
// Accessing each and printing the name & description.
cout << "\nItem Type Names in System:
     (" << itemTypeColl->cardinality() << ')' << endl;
// Create an iterator to iterate through the collection
dkIterator* iter = itemTypeColl->createIterator();
// while there are still items in the list, continue
while(iter->more()){
    DKItemTypeDefICM* itemType = (DKItemTypeDefICM*) iter->next()->value();
    cout << " - " << itemType->getName() << ": " <<
        itemType->getDescription() << endl;
    delete(itemType); // Free Memory

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

For more information, see the SItemTypeRetrievalICM sample, which includes how to list item type definitions. The SAttributeDefinitionRetrievalICM, includes how to list attribute definitions. The samples are available in the CMBROOT\Samples\cpp\icm directory.

Tip: Instead of deleting pEnt immediately, you can defer it and use the apply function to delete the attribute definition inside of the collection. See “Managing memory in collections (C++ only)” on page 95 for more information.

C++

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

Working with dynamic data objects (DDOs)

This section describes how to use a DDO and contains examples that help you learn how to:

1. Associate a DKDDO with a content server.
2. Create a DKDDO.
3. Create Persistent Identifiers (PIDs) for DKDDO attributes.
4. Add attributes and define attribute properties.
5. Define the DKDDO as a folder or as a document.
6. Set and view values for the attribute properties.
7. Check the DKDDO properties.
8. Check the attribute properties.
9. Display the DKDDO content.
10. Delete the DKDDO.

You use the DKDDO class for dynamic data objects (DDOs) in your IBM Enterprise Information Portal for Multiplatforms applications. A DKDDO object represents an item, which, for example, could be a Content Manager document or a folder or a user-defined object. A DKDDO object contains attributes. Each attribute has a name, a value, and properties. Each attribute is identified by a data ID. Attributes are numbered consecutively starting with 1; the attribute number is the data ID.

Because the name, value, and property of an attribute can vary, DKDDO provides flexible mechanisms to represent data originating from a variety of content servers and formats. For example, items from different item types in Content Manager, or rows from different tables in a relational database. The DKDDO itself can have properties that apply to the whole DKDDO, instead of to only one particular attribute.

You associate a DKDDO with a content server before calling the add, retrieve, update and delete methods to put its attributes into the content server or retrieve them. You set the content server either as a parameter when you create the DKDDO object or by calling setDatastore method.

Every DKDDO has a persistent object identifier (PID), which contains information for locating the attributes in the content server.

Creating a DKDDO

DKDDO has several constructors. You can create a DKDDO by calling its constructor without any parameters.

Java

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

C++

```
DKDDO ddo;
```

This DDO ddo must grow dynamically to accommodate more attributes. For a more efficient constructor, pass in the exact number of attributes you want (for example, ten):

Java

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

C++

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

Important: In APIs such as DKDatastoreICM and DKDatastoreOD,create a DKDDO by using the createDDO() methods in the DKDatastore XX class. In CM V8 DDOs must be created using these methods. The following example creates a DKDDO by passing in both content server and object type for Content Manager Version 8:

Java

```
DKDatastoreICM dsICM = new DKDatastoreICM(); //create a CM datastore
DKDDO ddo=dsICM.createDDO("ICMSAMPLE", //create a DDO to hold an object type
DKConstant.DK_CM_DOCUMENT);
```

For more information about DDO creation, refer to the SItemCreationICM sample available in CMBROOT\Samples\java\icm.

C++

```
// create a Content Manager datastore
DKDatastoreICM* dsICM = new DKDatastoreICM();
// create a DDO to hold an object type
DKDDO* ddo = dsICM->createDDO("ICMSAMPLE",
DK_CM_DOCUMENT);
```

For more information about DDO creation, refer to the SItemCreationICM sample available in CMBROOT\Samples\cpp\icm.

For other connectors (such as earlier Content Manager), you can create a DKDDO by supplying content server and object type with the following example:

Java

```
DKDatastoreDL dsDL=new DKDatastoreDL(); //create a CM datastore
DKDDO ddo=new DKDDO(dsDL, "DLSAMPLE");  //create a DDO to hold an object type
                                        //DLSAMPLE in dsDL
```

C++

```
// create an earlier Content Manager datastore
DKDatastoreDL dsDL;
// create a DDO to hold an object type DLSAMPLE in dsDL
DKDDO* cddo = new DKDDO(&dsDL, "DLSAMPLE");
```

Which constructor you use depends on your application; refer to the *online API reference* for information on the constructors.

Adding properties to a DDO

When creating a DKDDO object to represent a DDO, you have to specify its item type property (a document, folder, or item).

You can pass this property as one of the options in the DKDatastore*XX*.createDDO(*itemTypeName*,*itemPropertyType*/*SemanticType*) method.
`DKDatastoreICM.createDDO(itemTypeName,itemPropertyType/SemanticType)`

Or, if you already created the DKDDO without setting its item type property, you can use the following example to set the type of DDO to a ″document″.

Java

```
//----- Add the property that it is a document
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, new Short(DK_CM_DOCUMENT));
```

C++

```
any = DK_CM_DOCUMENT;                              // it is a document
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);
```

Creating a persistent identifier (PID)

Each DDO must have a persistent identifier (PID). The PID contains information about the content server's name, type, ID, and object type. The PID identifies the DDO's persistent data location. For example, in earlier Content Manager content server, the PID is the item ID. The item ID is one of the most important parameters for the `retrieve`, `update`, and `delete` functions. In Content Manager V8, the PID has five parts (see *Working with Content Manager 8.2* for more information).

For the `add` function, the content server creates and returns the item ID. For example, connectors that provide the `DKDatastoreXX.createDDO()` method automatically create the PID object in the `DKDDO.add()` operation.

The following example creates a DDO for retrieving a known item:

Java

```
// Given a connected DKDatastoreICM object named "dsICM"
// Create a new DDO
DKDDO ddo = dsICM.createDDO("book",DKConstant.DK_CM_DOCUMENT);
// PID automatically created by the function above.
ddo.add(); // Add the new item to the datastore.
// PID Completed by the System
DKPidICM pid = (DKPidICM) ddo.getPidObject();
```

C++

```
// Given a connected DKDatastoreICM object named "dsICM"
// Create a new DDO
DKDDO* ddo = dsICM->createDDO("book",DK_CM_DOCUMENT);
// PID automatically created by the function above.
ddo->add(); // Add the new item to the datastore.
// PID Completed by the System
DKPidICM* pid = (DKPidICM*) ddo->getPidObject();
```

Connect to the content server and call the `retrieve` function to retrieve the DDO created in the example.

Content Manager 8.2 connector has a PID class that is a subclass of DKPid. It is called DKPidICM which is the pid used by the DKDDOs and dkResources.

Working with data items and properties

DKDDO provides methods to add attributes and attribute properties to a DKDDO object.

Suppose an item type `NameOfItemType` has the attributes `Name` of type `integer` and is defined to be nullable in the CM V8 repository. You create a DKDDO object to handle an item of that entity, and you want to add two data items to the DKDDO.

The following example creates an item, sets attribute properties, and saves to the persistent content server in Content Manager. **Requirement:** Assumes that you have a connected DKDatstoreICM in variable `dsICM`. The user-defined item type `S_withChild` must be defined with `varchar S_varchar`, `long integer S_integer`, `short integer S_short`, and `time S_time`, as defined in the `SItemTypeCreationICM` API Education Sample.

Java

```
// Create a new item in memory
DKDDO ddo = dsICM.createDDO("S_withChild", DKConstant.DK_CM_DOCUMENT);

// Set attributes  (Additional attributes set in SItemCreationICM sample)
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"));

// Add to datastore
ddo.add();
```

The complete sample application from which this example was taken (SItemCreationICM) is available in the CMBROOT\Samples\java\icm directory.

C++

```
// Create a new item in memory
DKDDO* ddo = dsICM->createDDO("S_withChild", DK_CM_DOCUMENT);

// Set attributes  (Additional attributes set in SItemCreationICM sample)
// NOTE: Values below are automatically converted to type 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"));

// Add to datastore
ddo->add();
```

The complete sample application from which this example was taken (SItemCreationICM) is available in the CMBROOT\Samples\cpp\icm directory.

You must set the property type for an attribute; nullable and other properties are optional.

The following example accesses attribute values of a DDO.

Java

```
// Cast operatoins will enable access as subclass of Object type returned.
// NOTE: Additional attributes accessed in SItemRetrievalICM sample.

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: "+attrValr);
```

The complete sample application from which this example was taken (SItemRetrievalICM) is available in the CMBROOT\Samples\java\icm directory.

C++

```
// Assignment and cast operations coverts values from DKAny to each type.
// NOTE: Additional attributes accessed in SItemRetrievalICM sample.

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;
```

The complete sample application from which this example was taken (SItemRetrievalICM) is available in the CMBROOT\Samples\cpp\icm directory.

Getting the DKDDO and attribute properties

When processing a DKDDO, you must first know its type: document, folder, or item. The following sample code demonstrates how to determine the DDO type:

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:
         // --- process a document
         ....
         break;
       case DK_CM_FOLDER:
         // --- process a folder
       case DK_CM_ITEM:
         // --- Process an item in Content Manager
         ....
         break;
    }
}
```

For more information on accessing item type properties, refer to the SItemRetrievalICM API Education Sample, available in the CMBROOT\Samples\java\icm directory.

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:
        // process document
        ...
        break;
        case DK_CM_FOLDER:
        // process folder
        ...
        break;
    }
}
```

For more information on accessing item type properties, refer to the SItemRetrievalICM API Education Sample, available in the CMBROOT\Samples\cpp\icm directory.

To retrieve properties of an attribute, you must get the `data_id` of the attribute; then you can retrieve the properties.

Both the `data_id` and `property_id` start from 1. If you specify 0, then you receive an exception.

Java

```
data_id = ddo.dataId("Title");  // get data_id of Title
// ----- Get the number of properties for the attribute
short number_of_data_prop = ddo.dataPropertyCount(data_id);
// -----  Display all data properties belonging to this attribute
//        using a loop; the index starts at 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));
   }
```

For the complete sample application, refer to the `SItemRetrievalICM` in the `CMBROOT\Samples\java\icm` directory.

C++

```
// get data_id of Title
data_id = ddo->dataId("Title");
// how many props does it have?
unsigned short number_of_data_prop = ddo->dataPropertyCount(data_id);
// displays all data properties belonging to this attribute
// notice that the loop index starts from 1, where
// 1 <= i <= number_of_data_prop
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;
}
```

For the complete sample application, refer to the `SItemRetrievalICM` in the `CMBROOT\Samples\cpp\icm` directory.

Displaying the whole DDO

During application development, you might need to display the contents of a DKDDO for debugging purposes.

Java

```
short number_of_attribute = ddo.dataCount();
short number_of_prop      = ddo.propertyCount();
short number_of_data_prop;
// list DDO properties
for (short k = 1; k <= number_of_prop; k++) {
     System.out.println( k + " Property Name = " + ddo.getPropertyName(k) +
                         ",\t   value = " + ddo.getProperty(k));
}
// list data-items and their properties
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));
     }
}
```

For a complete example of accessing and printing a DDO (and all subcomponents), refer to `SItemRetrievalICM` and its `printDDO()` static function in the `CMBROOT\Samples\java\icm` directory.

C++

```
unsigned short number_of_attribute = ddo->dataCount();
unsigned short number_of_prop;
unsigned short number_of_data_prop;
// list DDO properties
for (short k = 1; k <= number_of_prop; k++) {
     cout << k << " Property Name = " << ddo->getPropertyName(k) <<
     ",\t   value = " << ddo->getProperty(k)  << endl;
}
// list data-items and their properties
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;
     }
}
```

For a complete example of accessing and printing a DDO (and all subcomponents), refer to `SItemRetrievalICM` and its `printDDO()` static function in the `CMBROOT\Samples\cpp\icm` directory.

Deleting a DDO (C++ only)

A DKDDO has two representations: the one in memory, and the persistent copy. To delete the DKDDO from memory, call its destructor. Note that this still leaves the persistent copy unchanged in the content server.

You delete the persistent copy in the content server with the `dkddo:del()` function. This does not affect the DKDDO representation in memory (the attribute values are in a DKAny object). The destructor deletes object references to dkCollection and dkDataObjectBase, including references to DKParts, DKFolder, DKDDO, and DKBlob.

Working with extended data objects (XDOs)

An XDO represents a component that can store resource content, such as a resource item or document part.

Resource items (XDOs) extend non-resource items (DDOs). You create resource items much like the regular ones. Resource Items area created just as regular Items are. Depending on the type of resource Item, the XDO can be extended further.

Java

```
Class Hierarchy
    Type     DDO       XDO        Extension
    -----   -----    --------    -----------
    Lob     DKDDO -> DKLobICM
    Text    DKDDO -> DKLobICM -> DKTextICM
    Image   DKDDO -> DKLobICM -> DKImageICM
    Stream  DKDDO -> DKLobICM -> DKStreamICM
```

Content Manager only: Since CM 8 requires XDOs to be of the correct subclass, DKDDOs should always be created using the DKDatastoreICM’s `createDDO()` methods. This allows the system to automatically set up important information in the DKDDO structure, and provides greater functionality such as resources, CM document model, and folders. Items of type resource (returned from DKDatastoreICM.createDDO) can be cast to the correct XDO or subclass depending on the XDO classification. For more information on creating items in general and the `DKDatastoreICM.createDDO()` function, refer to the `SItemCreationICM` sample.

To create an XDO for binary objects use DKBlob*xx*, where *xx* is the suffix representing the specific server. For example, use DKBlobICM for Content Manager, DKBlobOD for OnDemand, or DKBlobIP for ImagePlus for OS/390. When you create a DKBlob*xx* object, you must pass it the content server DKDatastore*xx*. For Content Manager, you use DKLobICM to create the XDO.

Using an XDO persistent identifier (PID)

An XDO needs a PID to store its data persistently. To use an XDO to locate and store data, you must supply a PID for the DKBlob*xx*, using a DKPidXDO*xx*. Relational Databases require the table, column and datapredicate string to locate the persistent data in a content server. For relational databases (RDB), the table name, column name and data predicate are required for DKPidXDO*xx*.

In the ICM Connector, use DKPidICM to represent the pid of a dkResource object which is an XDO.

Understanding XDO properties

Use the methods of the DKBlob*xx* to set the properties of an XDO where they apply; all properties are not available for all content servers. When loading, default values for the properties are set if specific values are not specified. For example, the following defaults are use with earlier Content Manager:

RepType (representation type)
The default is FRN$NULL. For VisualInfo for AS/400, you must use "   ", eight blank spaces surrounded by leading and trailing quotation marks.

ContentClass
The default is DK_CM_CC_UNKNOWN. For the valid values, see `DKConstant2DL.h` in the `\cmbroot\include` directory for Enterprise Information Portal.

AffiliatedType
The default is: DK_DL_BASE.

AffiliatedData
The default is: NULL.

To index object content with earlier Content Manager correctly, you must set SearchEngine, SearchIndex, and SearchInfo in the extension object DKSearchEngineInfoDL.

For working with XDOs in Content Manager, see “Working with items” on page 138.

C++ Tip: For the valid values of ContentClass, See the file `INCLUDE/DKConstant2DL.h` provided with Content Manager.

DB2, ODBC and DataJoiner configuration strings (C++ only)

This section defines the C++ DB2, ODBC and DataJoiner configuration strings.

CC2MIMEFILE=(filename)
Specify the cmbcc2mime.ini file (optional).

DSNAME=(content server name)
Specify the content server name (optional). When this content server is used by Federated, this option is set automatically.

AUTOCOMMIT=ON | OFF
Specify autocommit is on or off. Default is off (optional). When this content server is used by Fed autocommit is always on. This is set automatically.

This section defines the C++ DB2, ODBC and DataJoiner connect strings.

NATIVECONNECTSTRING=(native connect string)
Specify a native connect string to be passed to the native connect call (optional).

SCHEMA=name
Specify schema to be used for `listEntities`, `listEntityAttrs`, `listPrimaryKeyNames`, `listForeignKeyNames` functions (optional).

Java programming tips

For Content Manager V8 and later, an XDO is a dkResource object. You use DKPidICM to represent the PID of the resource object.

For earlier Content Manager, Content Manager for AS/400, and IP 390, you identify an XDO by the combination of item ID, part ID and the RepType. For RDB, the key to identify an XDO is combination of table, column and data predicate string. To handle a stand-alone XDO, you provide the item ID and part ID. The RepType is optional since the system provides a default value for it.

Use the add method of DKBlob*xx* to add the current content to a content server. You can retrieve the part ID value after `add` if you want to do some other operation with that object later.

Use the getPidObject() method on dkXDO to get the DKPid object.

You can use the following statement after `add` to obtain the system assigned part ID:

Java

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

Attention: In earlier Content Manager, you need a valid part ID to add a part to be indexed by the search manager (you cannot set the part ID to 0).

In this release, two methods in dkXDO have been modified: DKPid dkXDO.getPid() is deprecated and replaced by getPidObject. DKPid dkXDO.getPidObject() These methods use to return a DKPidXDO now they return a DKPid object.

C++ programming tips

For Content Manager, VI400® and IP390, you identify an XDO by the combination of item ID, part ID, and RepType. For Relational Databases, the combination of table name, column name and datapredicate is the key to identify an XDO. For a standalone XDO, you must provide the item ID and part ID. RepType is optional, because the system provides a default value (FRN$NULL).

For the `add` function, you must provide a part ID. You can retrieve the part ID value after `add` if you want to do some other operation with that object later.

Important: When adding a part for the search manager to index on a Content Manager content server, you must have a valid part ID and cannot set the part ID to 0.

Programming an XDO as a part of DDO

An XDO represents a single part object, if you have a DDO representing a document, which is a collection of resource content objects. You can manipulate the XDO as a component of the DDO or as a stand-alone object. When you access the XDO as a part of the DDO, the DDO provides the item ID. When using the XDO as a stand-alone object, you use the existing item ID for the XDO.

The following example creates a document and adds document parts in Content Manager. **Requirement:** The user-defined item type, `S_docModel`, must be defined in the system, classified as Document Model, and supports ICMBASE and ICMBASETEXT part types, as defined in the `SItemTypeCreationICM` API Education Sample.

Java

```
// Create a document
DKDDO ddoDocument = dsICM.createDDO("S_docModel", DKConstant.DK_CM_DOCUMENT);

// Create parts
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);

// Set parts' MIME type     (SResourceItemMimeTypesICM.txt sample)
base.setMimeType("application/msword");
baseText1.setMimeType("text/plain");
baseText2.setMimeType("text/plain");

// Load content into parts  (SResourceItemCreationICM sample)
base.setContentFromClientFile("SResourceItemICM_Document1.doc");
// Load file
baseText1.setContentFromClientFile("SResourceItemICM_Text1.txt");
// into memory
baseText2.setContentFromClientFile("SResourceItemICM_Text2.txt");

// Access the DKParts attribute
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
        DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS));

// Add parts to document
dkParts.addElement(base);
dkParts.addElement(baseText1);
dkParts.addElement(baseText2);

// Add new document to persistent datastore
ddoDocument.add();
```

For the complete sample application, refer to the SDocModelItemICM.java in the CMBROOT\Samples\java\icm directory. SResourceItemCreationICM shows more examples of XDO use.

C++

```
// Create a document
DKDDO* ddoDocument = dsICM->createDDO("S_docModel", DK_CM_DOCUMENT);
// Create Parts
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);

// Set parts' MIME type     (SResourceItemMimeTypesICM.txt sample)
base->setMimeType("application/msword");
baseText1->setMimeType("text/plain");
baseText2->setMimeType("text/plain");
// Load content into parts  (SResourceItemCreationICM sample)
 // Load the file into memory.
base->setContentFromClientFile("SResourceItemICM_Document1.doc");
baseText1->setContentFromClientFile("SResourceItemICM_Text1.txt");
baseText2->setContentFromClientFile("SResourceItemICM_Text2.txt");
// Access the DKParts attribute
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
         ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS));

// Add parts to document
dkParts->addElement((dkDataObjectBase*)(DKDDO*)base);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText1);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText2);
// Add new document to persistent datastore
ddoDocument->add();
```

For the complete sample application, refer to the SDocModelItemICM.java in the CMBROOT\Samples\cpp\icm directory. SResourceItemCreationICM shows more examples of XDO use.

Programming a stand-alone XDO

All of the following examples are specific to Content Manager 8.2. For examples for earlier Content Manager and other content servers, see “Representing items using DDOs” on page 136, “Working with other content servers” on page 235, and refer to the sample programs in the CMBROOT\Samples directory.

Adding an XDO from the buffer

This example shows how to add an XDO from a buffer in Content Manager. It creates an XDO, loads content into memory, and stores persistently content from memory. **Requirement:** The user-defined item type S_lob of classification resource must be defined in the system, as defined by the SItemTypeCreationICM API Education Sample. Additionally, the resource manager and SMS collection definitions must be set up and set as default for the item type or for the user, as performed in the SResourceMgrDefCreationICM, SSMSCollectionDefCreationICM, SResourceMgrDefSetDefaultICM, and SSMSCollectionDefSetDefaultICM samples.

Java

```
// Create an empty resource object
DKLobICM lob = (DKLobICM) dsICM.createDDO("S_lob", DKConstant.DK_CM_DOCUMENT);

// Set the MIME type (SResourceItemMimeTyesICM.txt sample)
lob.setMimeType("application/msword");

// Load content into item's local memory
lob.setContentFromClientFile("SResourceItemICM_Document1.doc");

// Add to datastore with content already in memory
lob.add();
```

For the complete sample application, refer to the `SResourceItemCreationICM` in the `CMBROOT\Samples\java\icm` directory.

C++

```
// Create an empty resource object
DKLobICM* lob = (DKLobICM*) dsICM->createDDO("S_lob", DK_CM_DOCUMENT);

// Set the MIME type (SResourceItemMimeTyesICM.txt sample)
lob->setMimeType("application/msword");

// Load content into item's local memory
lob->setContentFromClientFile("SResourceItemICM_Document1.doc");

// Add to datastore With content already in memory
lob->add();
```

For the complete sample application, refer to the `SResourceItemCreationICM` in the `CMBROOT\Samples\cpp\icm` directory.

Adding an XDO from a file

The following example adds an XDO to the content server (storing the content directly from file) in Content Manager. **Requirement:** The user-defined item type `S_lob` of classification resource must be defined in the system, as defined by the `SItemTypeCreationICM` API Education Sample. Additionally, the resource manager and SMS collection definitions must be setup and set as default for the item type or for the user, as performed in `SResourceMgrDefCreationICM`, `SSMSCollectionDefCreationICM`, `SResourceMgrDefSetDefaultICM`, and `SSMSCollectionDefSetDefaultICM` samples.

Java

```
// Create an empty resource object
DKTextICM text = (DKTextICM) dsICM.createDDO("S_text", DKConstant.DK_CM_ITEM);

// Set the MIME type (SResourceItemMimeTyesICM.txt sample)
text.setMimeType("text/plain");

// Store content directly from a file
text.add("SResourceItemICM_Text1.txt");
```

For the complete sample application, refer to the `SResourceItemCreationICM` in the `CMBROOT\Samples\java\icm` directory.

C++

```
// Create an empty resource object
DKTextICM* text = (DKTextICM*) dsICM->createDDO("S_text", DK_CM_ITEM);

// Set the MIME type (SResourceItemMimeTyesICM.txt sample)
text->setMimeType("text/plain");

// Store content directly from a file
text->add("SResourceItemICM_Text1.txt");
```

For the complete sample application, refer to the `SResourceItemCreationICM` in the `CMBROOT\Samples\cpp\icm` directory.

Adding an annotation object to an XDO

The following example adds an annotation part to a document in Content Manager. **Requirement:** The user-defined item type, `S_docModel`, must be defined in the system, classified as Document Model, and support the ICMANNOTATION part type, as defined in the `SItemTypeCreationICM` sample. Assume that you are given an instance of a document already stored persistently in the `ddoDocument` variable. Also, assume that you are given a connected DKDatastoreICM object in variable `dsICM`.

Java

```
// Check out / lock the item for update
dsICM.checkOut(ddoDocument);

// Create annotation part
DKLobICM  annot = (DKLobICM) dsICM.createDDO("ICMANNOATATION",
                             DKConstantICM.DK_ICM_SEMANTIC_TYPE_ANNOTATION);

// Set annotatoin MIME type (SResourceItemMimeTypesICM.txt sample)
annot.setMimeType("image/bmp");

// Load content into parts  (SResourceItemCreationICM sample)
annot.setContentFromClientFile("myAnnotation.bmp");

// Access the DKParts attribute
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
        DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS));

// Add parts to document
dkParts.addElement(annot);

// Save the changes to the persistent datastore
ddoDocument.update();

// Check in / unlock the item after update
dsICM.checkIn(ddoDocument);
```

For the complete sample application, refer to the `SDocModelItemICM` in the `CMBROOT\Samples\java\icm` directory.

C++

```
// Check out / lock the item for update
dsICM->checkOut(ddoDocument);

// Create annotation part
DKLobICM* annot = (DKLobICM*) dsICM->createDDO("ICMANNOTATION",
                              DK_ICM_SEMANTIC_TYPE_ANNOTATION);

// Set annotatoin MIME type (SResourceItemMimeTypesICM.txt sample)
annot->setMimeType("image/bmp");

// Load content into parts  (SResourceItemCreationICM sample)
annot->setContentFromClientFile("myAnnotation.bmp");

// Access the DKParts attribute
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
         ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS));

// Add parts to document
dkParts->addElement((dkDataObjectBase*)(DKDDO*)annot);

// Save the changes to the prsistent datastore
ddoDocument->update();

// Check in / unlock the item after update
dsICM->checkIn(ddoDocument);
```

For the complete sample application, refer to the `SDocModelItemICM` in the `CMBROOT\Samples\cpp\icm` directory.

Examples of working with an XDO

The following examples illustrate using a stand-alone XDO.

Retrieving, updating, and deleting an XDO

To retrieve, update or delete an object in a content server, you provide the correct item ID, part ID and RepType for the XDO that represents the object.

The following example retrieves, updates, and deletes an XDO in Content Manager. **Requirement:** The user-defined item type `S_text` of classification resource must be defined in the system, as defined by the `SItemTypeCreationICM` API Education Sample. Additionally, the resource manager and SMS collection definitions must be setup and set as default for the item type or for the user, as performed in `SResourceMgrDefCreationICM`, `SSMSCollectionDefCreationICM`, `SResourceMgrDefSetDefaultICM`, and `SSMSCollectionDefSetDefaultICM` samples. Additionally, assume that you are given a PID string in variable `pidString` for a resource item that already exists in the content server.

Java

```
// Given: String pidString

// Re-create Blank DDOs for Existing Item (SItemRetrievalICM sample)

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

// Retrieve the item with the resource content
lob.retrieve(DKConstant.DK_CM_CONTENT_YES);

// Check out / lock the item for update  (SItemUpdateICM sample)
dsICM.checkOut(lob);

// Set the new MIME type (SResourceItemMimeTypesICM.txt sample)
lob.setMimeType("application/msword");

// Update datastore with new content
lob.update("SResourceItemICM_Document2.doc");

// Check in / unlock the item after update
dsICM.checkIn(lob);

// Delete item
lob.del();
```

This code sample comes from `SResourceItemRetrievalICM`, `SResourceItemUpdateICM`, and `SResourceItemDeletionICM` in the `CMBROOT\Samples\java\icm` directory.

C++

```
// Given: DKString pidString

// Re-create Blank DDOs for Existing Item (SItemRetrievalICM sample)
DKLobICM* lob = (DKLobICM*) dsICM->createDDO(pidString);

// Retrieve the item with the resource content
lob->retrieve(DK_CM_CONTENT_YES);

// Check out / lock the item for update  (SItemUpdateICM sample)
dsICM->checkOut(lob);

// Set the new MIME type (SResourceItemMimeTypesICM.txt sample)
lob->setMimeType("application/msword");

// Update datastore with new content
lob->update("SResourceItemICM_Document2.doc");

// Check in / unlock the item after update
dsICM->checkIn(lob);

// Delete item
lob->del();

// Free Memory
delete(lob);
```

This code sample comes from `SResourceItemRetrievalICM`, `SResourceItemUpdateICM`, and `SResourceItemDeletionICM` in the `CMBROOT\Samples\cpp\icm` directory.

Invoking an XDO function

This example demonstrates how to test the DKBlob class using an earlier Content Manager server. For this example, you must know the item ID and part ID of the XDO.

Java

```
public class txdomiscDL implements DKConstantDL
{
  public static void main(String args[])
  {
    int    partId = 5;
    String itemId = "GAWCVGGVFUG428UJ";
    String repType = "";
    // Check the number of arguments for main and determine what to do
    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);
       apid.setRepType(repType);
       axdo.setPidObject(apid);
       System.out.println("repType=" + apid.getRepType());
       System.out.println("itemid=" + apid.getItemId());
       System.out.println("partId=" + apid.getPartId());
// continued...
```

Java (continued)

```
       // ----- Before retrieve
 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();

 // ----- After retrieve
 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)
    {
  // ------ Handle the exceptions
}
```

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;
// continued...
```

C++ (continued)

```
       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;

       //== before retrieve
       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();

       //== after retrieve
       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;
        }
        //== open content
        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);
        else if (concls == DK_DL_CC_AVI)
           axdo->setInstanceOpenHandler("mplay32", TRUE);
        axdo->open();
// continued...
```

C++ (continued)

```
       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;
}
```

Adding an XDO media object in earlier Content Manager

For every media object added, an entry is created in the FRN$MEDIA table. This entry contains the information about the media user data. The physical media object is stored in the VideoCharger content server specified in the network table. For the following example, you must know the item ID of the XDO.

Java

```
public class txdoAddVSDL implements DKConstantDL
{
// ----- Main method
public static void main(String[] args)
{
    String  fileName = "/icing1.mpg1";        //a media object
    String itemId = "K1A04EWBVHJAV1D7";       //a known itemId
    int partId = 45;
    // ----- Check the arguments for 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);
    }
    // ----- Processing
    try
    {
      // ----- connect to datastore
      DKDatastoreDL dsDL = new DKDatastoreDL();
      // replace following with your library server, userid, password
      System.out.println("connecting to datastore...");
      dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
      System.out.println("datastore connected");
      // ----- create xdo and pid
      DKBlobDL axdo = new DKBlobDL(dsDL);
      DKPidXDODL  apid = new DKPidXDODL();
      apid.setPartId(partId);
      apid.setPrimaryId(itemId);
      axdo.setPidObject(apid);
      // you must use the content class DK_DL_CC_IBMVSS for a media object
      axdo.setContentClass(DK_DL_CC_IBMVSS);
      System.out.println("contentClass=" + axdo.getContentClass());
      System.out.println("partId = " + axdo.getPartId());
// continued...
```

Java (continued)

```
      // ----- set up DKMediaStreamInfoDL
      DKMediaStreamInfoDL aVS = new DKMediaStreamInfoDL();
      aVS.setMediaFullFileName(fileName);
      // if fileName contain a list of media segments then use following
      //       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>");
      // following are optional, if not set default value will be provided
      aVS.setMediaNumberOfUsers(2);
      aVS.setMediaAssetGroup("AG");
      // ----- same as defined in VideoCharger server
      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();
    }
    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
 {
  // connect to datastore
  cout << "Connecting datastore ..." << endl;
  DKDatastoreDL dsDL;
  // replace following with your library server, user ID, password
  dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
  cout << "datastore connected" << endl;
  // *** create xdo and pid
  DKBlobDL* axdo = new DKBlobDL(&dsDL);
  DKPidXDODL*  apid = new DKPidXDODL;
  apid ->setPartId(partId);
  apid ->setPrimaryId(itemId);
  axdo ->setPidObject(apid);
  // you must use the content class DK_DL_CC_IBMVSS for a media object
  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;
// continued...
```

C++ (continued)

```
  // *** setup 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>");

  //following are optional, if not set then default value will be provided
  aVS.setMediaNumberOfUsers(1);
  aVS.setMediaAssetGroup("AG");
  // *** same as defined in VideoCharger server
  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;
   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;
}
```

Deleting an XDO media object

The following example shows how to delete an XDO media object. For this example you must know the item ID, part ID, and RepType (representation type) of the 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);
    }
    // ----- Check the arguments for 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);
    }

    // ----- Processing
    try
    {
      DKDatastoreDL dsDL = new DKDatastoreDL();
      System.out.println("connecting to datastore...");
      // replace following with your library server, userid, password
      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);
      boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
      System.out.println("isMediaObject?=" + flag2);
// continued...
```

Java (continued)

```
      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));
        // ----- set delete option for media object
        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();
    }
    // ------ Handle exceptions
    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[])
{
  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;
    // replace following with your library server, user ID, password
    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);
    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;
// continued...
```

C++ (continued)

```
    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;
   }
   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;
}
```

Retrieving an XDO media object

The following example shows how to retrieve an XDO media object. The retrieved object contains only the media metadata, not the media object itself. For this example you must know the item ID and part ID of the 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...");
      // ----- replace following with your library server, userid, password
      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));
      }
```

Java (continued)

```
      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));
      }
      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());
      // -----  Perform the retrieve call
      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);  //use lviewpro
      else if (cc == DK_DL_CC_AVI)
        axdo.setInstanceOpenHandler("mplay32 ", true);   //use mplay32
      else if (cc == DK_DL_CC_IBMVSS)
        axdo.setInstanceOpenHandler("iscoview ", true);  //use iscoview
      axdo.open();

      dsDL.disconnect();
      dsDL.destroy();
    }
    catch (DKException exc)
    {
       ... \\ handle exceptions and destroy the datastore+
    }
  }
}
```

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;
    // replace following with your library server, userid, password
    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);
    axdo ->setPidObject(apid);
    cout <<"itemId= "<<axdo->getItemId()<<endl;
    cout <<"partId= "
      <<((DKPidXDODL*)(axdo->getPidObject()))->getPartId()<<endl;
// continued...
```

C++ (continued)

```
    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;
// continued...
```

C++ (continued)

```
     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);
                              //default use Notepad in Windows
     int concls = axdo->getContentClass();
     if (concls == DK_DL_CC_GIF)
      axdo->setInstanceOpenHandler("lviewpro", TRUE);
                              //use lviewpro in Windows
      else if (concls == DK_DL_CC_AVI)
       axdo->setInstanceOpenHandler("mplay32", TRUE);
                              //use mplay32 in Windows
       else if (concls == DK_DL_CC_IBMVSS)
        axdo->setInstanceOpenHandler("iscoview", TRUE);
                             //use iscoview in 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;
}
```

Adding an XDO to a storage collection

To add an XDO object associated with user defined storage collection names, use the extension object DKStorageManageInfo*xx*, where *xx* is the suffix representing the specific server.

The following example uses DKStorageManageInfoDL, for an earlier Content Manager server; for Content Manager Version 8 and later, see “Working with Content Manager Version 8.2” on page 123.

Java

```
String fileName = "e:\\test\\notepart.txt"; //file for add
int    partId = 0;                       //let system decide the partId
String itemId = "V5SPB$WBLOHIQ4YI";           //an existing itemId
DKDatastoreDL dsDL = new DKDatastoreDL();     //required datastore
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD",""); //connect to dstore
DKBlobDL axdo = new DKBlobDL(dsDL);           //create XDO
DKPidXDODL  apid = new DKPidXDODL();          //create PID
apid.setPartId(partId);                       //set partId
apid.setPrimaryId(itemId);                    //set itemId
axdo.setPidObject(apid);                      //set PID object
axdo.setContentClass(DK_DL_CC_ASCII);         //set ContentClass

// ----- Create the DKStorageManageInfoDL
StorageManageInfoDL aSMS = new DKStorageManageInfoDL();
aSMS.setRetention(888);                       //optional
aSMS.setCollectionName("TESTCOLLECT1");       //already defined in DL SMS
aSMS.setManagementClass("TESTMGT1");          //optional
aSMS.setStorageClass("FIXED");                //optional
axdo.setExtension("DKStorageManageInfoDL", (dkExtension)aSMS);
axdo.add(fileName);                           //add from file
System.out.println("after add partId = " + axdo.getPartId());
                                           //display the partId after add
dsDL.disconnect();             // disconnect from  and destroy datastore
dsDL.destroy();
//  ------  Handle the exceptions
```

C++

```
DKString fileName="e:\\test\\notepart.txt"; //file for add
int    partId = 0;                     //let system decide the partId
DKString itemId = "V5SPB$WBLOHIQ4YI";         //an existing itemId
DKString rtype = "FRN$NULL";                  //optional
DKDatastoreDL dsDL;                           //required datastore
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD"); //connect to dstore
DKBlobDL* axdo = new DKBlobDL(&dsDL);         //create XDO
DKPidXDODL*  apid = new DKPidXDODL;           //create Pid
apid->setPartId(partId);                      //set partId
apid->setPrimaryId(itemId);                   //set itemId
apid->setRepType(rtype);                      //set repType
axdo->setPidObject(apid);                     //set pid object
axdo->setContentClass(DK_DL_CC_ASCII);        //set ContentClass

//---set DKStorageManageInfoDL-----
DKStorageManageInfoDL aSMS = new DKStorageManageInfoDL();
aSMS.setRetention(888);                       //optional
aSMS.setCollectionName("TESTCOLLECT1"); //already defined in DL SMS
aSMS.setManagementClass("TESTMGT1");          //optional
aSMS.setStorageClass("FIXED");                //optional
axdo->setExtension("DKStorageManageInfoDL", (dkExtension)aSMS);
axdo->add(fileName);                          //add from file
System.out.println("after add partId = " + axdo->getPartId());
                                      //display the partId after add
dsDL.disconnect();                    //disconnect from datastore
System.out.println("datastore disconnected");
```

Refer to the following code samples in the `CMBROOT\Samples` directory for examples of adding search indexed objects and media objects to Content Manager.

- `TxdoAddBsmsDL`
- `TxdosAddBsmsDL`

- TxdoAddFsmsDL
- TxdosAddFsmsDL
- TxdomAddsmsDL

Changing the storage collection of an XDO

You can change the storage collection of an existing XDO. After setting up the extension object DKStorageManageInfoICM, call the changeStorage method.

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......");
```

The complete sample application from which this example was taken (TxdoChgSmsICM) is located in the CMBROOT\Samples\java\dl or CMBROOT\Samples\cpp\dl directory.

Working with XML (Java only)

Enterprise Information Portal supports importing and exporting content from XML documents into and out of Content Manager as DDOs and XDOs using the Java APIs. This feature makes it possible to import, store, and retrieve a wide variety of objects in Content Manager—such as data or multimedia content—from disparate information systems without developing separate interfaces for each system. For example, if you have an object stored in one data system, you can convert it into an XML file and then import it into Content Manager using Enterprise Information Portal's Java APIs. Once in Content Manager, you can do anything with the object that you could with any other Content Manager object.

Extending the capability to import and export XML

Currently, the import and export XML functions, DKDDO.toXML() and DKDDO.fromXML() , do not support items with reference attribute values, links, or folder contents. The export operation succeeds, but uses the PID as a reference to the item in the particular content server. If then imported into a different content server, those items will not exist in the target system because the PID information is unique to the content server it came from. These two interfaces only create new items and do not support multiple versions of the same item.

As a solution, you must build a separate tool to handle referenced items, detect and overwrite existing items, or any other feature not supported by this interface.

The Version 8.1 Fixpack 1 and Version 8.2 of EIP provides new sample tools (TImportICM and TExportICM) and sample tool API (TExportPackageICM) that perform full-featured import and export functionality. The new sample functionality significantly extends the DKDDO.toXML() and DKDDO.fromXML() interfaces that are demonstrated in SItemXMLImportExportICM.

The new tools are packaged with Version 8.1 Fixpack 1 and Version 8.2 as Java Samples. For more information, refer to the Sample Import / Export Tools and API document and TExportPackageICM.doc, located in the CMBROOT\Samples\java\icm directory.

Importing XML documents

You can import XML from different sources, including standard input, files, buffers, and Web addresses (URLs). It is also possible to import an XML file in its entirety. These constructors extract content from an XML document, create a corresponding DKDDO and any dkXDO associated with it. You can then call the add method on the DDO to add the object into Content Manager. The new DDO belongs to a Content Manager Version 8 item type or an earlier Content Manager index class and can only be stored in Content Manager. Importing a self-referencing XML file allows you to store the original XML file as an XDO; that is, you do not lose the XML in the import process, making the XML itself available for possible future use.

When importing content from XML, use these methods in DKDDO:

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

The use of the DKDDO constructors with earlierContent Manager to import XML is deprecated.

As you import XML content, keep these parameters in mind:

1. Remember you can only import into Content Manager or earlier Content Manager.
2. XML files containing content for import must conform to the XML document type definition, shown below.
3. XML import and XML export are supported only by the Java APIs.

The following sections describe the prerequisites and methods for importing XML content:

- The XML document type definition (DTD)
- Storing content in XML documents
- Extracting content from different XML sources
- Importing XML content into Content Manager.

The XML Document Type Definition (DTD)

In order to import content to Content Manager to store as XML, you must store the content in XML documents that conform to ddo.dtd, which is located at 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 deprecated it is replaced by xdoIdString on an xdoRef   -->
<!-- repType deprecated it is replaced by xdoIdString on an 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 deprecated it is replaced by specificInfo   -->
<!ELEMENT searchEngineInfo EMPTY>
<!ATTLIST searchEngineInfo       searchEngine     CDATA #REQUIRED
                                 searchIndex      CDATA #REQUIRED
                                 searchInfo       CDATA #REQUIRED>
<!-- smsInfo deprecated it is replaced by specificInfo   -->
<!ELEMENT smsInfo EMPTY>
<!ATTLIST smsInfo         smsRetention          CDATA #IMPLIED
                          smsCollection         CDATA #IMPLIED
                          smsMgmtClass          CDATA #IMPLIED
                          smsStorageClass       CDATA #IMPLIED
                          smsObjServer          CDATA #IMPLIED>
<!ELEMENT xdoContent (#PCDATA)>
```

Storing content in XML documents

XML files can represent in different ways documents or folders for import into Content Manager. These documents and folders can also contain parts. The sample below shows first a typical XML data item, dataItem dataId="1", whose value is Basuki. DataItem 13, however, uses the dataName DKParts, which relates to a self-referencing XDO.

Sample for 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>
```

For some examples of how XML stores objects with Content Manager, see the samples in the DK\Samples\java\icm\ directory.

Sample for earlier 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>
```

For some examples of how XML stores objects with earlier Content Manager, see the following code samples in the samples directory:

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

- `dltypes01.xml`

Extracting content from different XML sources

The DKDDO methods can extract content from a variety of XML sources, including standard input, files, buffers, and Web addresses (URLs). Call the DKDDO methods to extract content from your XML source and to initiate the import process.

Here are examples of each XML source:

XML from a file:

Java

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

XML from a buffer:

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 from a Web address (URL):

Java

```
xmlSource = new DKNVPair("URL", "file:////d://myxml//dlsamp01.xml");
// replace file:////d:// with http://www.webaddress.com/ for URL
Int  importOptions=0;
```

Importing XML content into Content Manager

The following example follows these basic steps:

1. Create a DKDDO.
2. Create and connect to a content server, in this case Content Manager or earlier Content Manager.
3. Add the new DKDDO to the content server, again, in this case Content Manager.
4. Use either `dkDataObjectBase fromXML(DKNVPair xmlSource)`or `dkDataObjectBase fromXML(DKNVPair xmlSource, int options)` to import the XML source.

The resulting DKDDO conforms to the ddo.dtd specifications and belongs to a Content Manager item type or earlier Content Manager index class.

Java

```
// ----- Construct a DDO by importing the XML document
xmlSource = new DKNVPair("FILE", "icmsamp01.xml");
int importOptions = DK_CM_XML_VALIDATION;
DKDDO ddo = new DKDDO();
ds = new  DKDatastoreICM();
//  ...... connect to the datastore
ddo.setDatastore (ds);
// ----- Add the DDO to the datastore
ddo.add()
// ----- Import the XML
ddo.fromXML(xmlSource, importOptions);
```

For the complete sample application, refer to `SItemXMLImportExportICM.java` in the `CMBROOT\Samples\java\icm` directory.

Exporting XML

You can export a DDO and XDO data as an XML document from Content Manager or earlier Content Manager. Use the method `toXML(DKNVPair xmlDestination, String path, int options)` of DKDDO to export.

Restriction: The `DKDDO.toXML()` and `DKDDO.fromXML()` interfaces demonstrated in this sample cannot be used to export an item from one content server and import it into another if it references any other items through links, folder contents, or reference attributes. The import operation will create new items and will not update or overwrite existing items or support multiple versions of an item. For information on tools that extend this capability, refer to “Extending the capability to import and export XML” on page 77.

The following example shows how to export XML:

Java

```
DKNVPair xmlDestination = null;
        //----- Use this line to export to STDOUT
         //xmlDestination = new DKNVPair("STDOUT", null);

         //----- Use this line to export to a buffer
         //xmlDestination = new DKNVPair("BUFFER", new Object());

         //----- Use this line to export to a file
         String xmlFile = "export.xml";
         if(!fileName.equals("")){
            xmlFile = fileName;
         }

         String strOS = System.getProperty("os.name");
         if (strOS.indexOf("Win") != -1) { // Windows OS
            xmlFile = outputPath + "\\" + xmlFile;
         } else { // other than Win OS
            xmlFile = outputPath + "/" + xmlFile;
         }

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

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

For the complete sample application, refer to `SItemXMLImportExportICM.java` in the `CMBROOT\Samples\java\icm` directory.

Creating documents and using the DKPARTS attribute

The DKPARTS attribute in a DDO represents the collection of parts in a document. The value of this attribute is a DKParts object, which is a collection of DKPart objects. DKPart objects are items from *document part* classified item types, and contain resource content.

Content Manager only: A document is an item that can be stored, retrieved, and exchanged among Content Manager systems and users as a separate unit. An item given the *document* semantic type is expected to contain information that forms a document, but does not rigidly mean an implementation of a specific document model. An item created from a document (also known as a document model) classified item type means that the item will contain document parts, a specific implementation of a document model provided by Content Manager. Document classified item types can create items given either the document or folder semantic type. The document parts can include varied types of content, including for example, text, images, and spreadsheets.

The following example creates a document and adds document parts in Content Manager. **Requirement:** The user-defined item type, `S_docModel`, must be defined in the system, classified as Document Model. It must also support ICMBASE and ICMBASETEXT part types, as defined in the `SItemTypeCreationICM` API Education Sample.

Java

```
// Create a document
DKDDO ddoDocument = dsICM.createDDO("S_docModel", DKConstant.DK_CM_DOCUMENT);

// Create parts
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);

// Set parts' MIME type     (SResourceItemMimeTypesICM.txt sample)
base.setMimeType("application/msword");
baseText1.setMimeType("text/plain");
baseText2.setMimeType("text/plain");

// Load content into parts  (SResourceItemCreationICM sample)
base.setContentFromClientFile("SResourceItemICM_Document1.doc");
// Load file
baseText1.setContentFromClientFile("SResourceItemICM_Text1.txt");
// into memory
baseText2.setContentFromClientFile("SResourceItemICM_Text2.txt");

// Access the DKParts attribute
DKParts dkParts = (DKParts) ddoDocument.getData(ddoDocument.dataId(
        DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS));

// Add parts to document
dkParts.addElement(base);
dkParts.addElement(baseText1);
dkParts.addElement(baseText2);

// Add new document to persistent datastore
ddoDocument.add();
```

For information on creating documents with parts, refer to the `SDocModelItemICM` API Education Sample in the `CMBROOT\Samples\java\icm` directory.

C++

```
// Create a document
DKDDO* ddoDocument = dsICM->createDDO("S_docModel", DK_CM_DOCUMENT);

// Create Parts
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);

// Set parts' MIME type     (SResourceItemMimeTypesICM.txt sample)
base->setMimeType("application/msword");
baseText1->setMimeType("text/plain");
baseText2->setMimeType("text/plain");

// Load content into parts  (SResourceItemCreationICM sample)
base->setContentFromClientFile("SResourceItemICM_Document1.doc");
// Load the file into memory.
baseText1->setContentFromClientFile("SResourceItemICM_Text1.txt");
baseText2->setContentFromClientFile("SResourceItemICM_Text2.txt");

// Access the DKParts attribute
DKParts* dkParts = (DKParts*)(dkCollection*) ddoDocument->getData(
          ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS));

// Add parts to document
dkParts->addElement((dkDataObjectBase*)(DKDDO*)base);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText1);
dkParts->addElement((dkDataObjectBase*)(DKDDO*)baseText2);

// Add new document to persistent datastore
ddoDocument->add();
```

For information on creating documents with parts, refer to the SDocModelItemICM API Education Sample in the CMBROOT\Samples\cpp\icm directory.

The DDO owns all parts in the parts collection. Update and delete parts through the document DDO.

The following example shows how to retrieve and access parts from a DDO.

Java

```
// NOTE: Print function provided in SDocModelItemICM sample

// Get the DKParts object.
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);

// Go through part list
dkIterator iter = dkParts.createIterator();  // Create an Iterator
while(iter.more()){                          // While there are items left
    DKDDO part = (DKDDO) iter.next();        // Move pointer & return next
System.out.println("Item Id: "+((DKPidICM)part.getPidObject()).getItemId()");
}
```

For the complete sample application, refer to `SDocModelItemICM` in the `CMBROOT\Samples\java\icm` directory.

C++

```
// NOTE: Print function provided in SDocModelItemICM sample

// Get the DKParts object.
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);

// Go through part list
dkIterator* iter = dkParts->createIterator();  // Create an Iterator
while(iter->more()){                           // While there are items
 DKDDO* part = (DKDDO*) iter->next()->value(); // Move pointer & return next
 cout << "Item Id:" << ((DKPidICM*)part->getPidObject())->getItemId()<< endl;
}
delete(iter);                                     // Free Memory
```

For information on creating documents with parts, refer to the `SDocModelItemICM` API Education Sample in the `CMBROOT\Samples\cpp\icm` directory.

Creating folders and using the DKFOLDER attribute

In a folder DDO, you use the DKFOLDER attribute to represent the collection of documents and other folders that belong to the folder. The value of this attribute is a DKFolder object, which is a collection of DDOs. As shown below, the DKFolder attribute is set as the DKParts attribute is set.

Content Manager only: A folder is an item of any item type, regardless of classification, with the *folder* semantic type. Any item with the folder semantic type will contain specific folder functionality provided by Content Manager, in addition to all non-resource item capabilities and any additional functionality available from an item type classification such as document model or resource. Folders may contain any number of items of any type, including documents and subfolders. A folder is indexed by attributes.

The following example creates a folder and adds contents in Content Manager. **Requirement:** The user-defined item type, `S_simple`, must be defined in the system, as defined in the `SItemTypeCreationICM` API Education Sample.

Java

```
// Create new folder in memory
DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);

// Create and save contents to place in folder
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();

// Access the DKFolder attribute
DKFolder dkFolder = (DKFolder)
ddoFolder.getData(ddoFolder.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
                                   DKConstant.DK_CM_DKFOLDER));

// Add contents to folder
dkFolder.addElement(ddoDocument);
dkFolder.addElement(ddoItem);
dkFolder.addElement(ddoFolder2);  // Note, Folders can contain sub-folders.

// Save the folder in the persistent datastore.
ddoFolder.add();
```

For more information on creating Folders, refer to the `SFolderICM` API Education Sample in the `CMBROOT\Samples\java\icm` directory.

C++

```
// Create new folder in memory
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);

// Create and save contents to place in folder
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();

// Access the DKFolder attribute
DKFolder* dkFolder = (DKFolder*)(dkCollection*) ddoFolder->getData(
                     ddoFolder->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKFOLDER));

// Add contents to folder
dkFolder->addElement(ddoDocument);
dkFolder->addElement(ddoItem);
dkFolder->addElement(ddoFolder2);  // Note, Folders can contain sub-folders.

// Save the folder in the persistent datastore.
ddoFolder->add();
```

For more information on creating Folders, refer to the `SFolderICM` API Education Sample in the `CMBROOT\Samples\cpp\icm` directory.

In Content Manager Version 8, the folder item does not own the folder contents. An item may be added to multiple folders. Removing an item from a folder simply breaks the folder-content relationship managed by the system. Items in the folder must be updated and deleted independently. In earlier versions of Content Manager, the DDO owns the contents in the collection.

The following example shows how to retrieve and access folder contents from a DDO.

Java

```
// NOTE: Print function provided in SFolderICM API Education Sample

// Get the DKFolder object.
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);

// Access contents
dkIterator iter = dkFolder.createIterator(); // Create an Iterator
while(iter.more()){                          // While there are items left
 DKDDO ddo = (DKDDO) iter.next();         // Move to & return next element
 System.out.println("Item Id: "+((DKPidICM)ddo.getPidObject()).getItemId()");
}
```

For the complete sample application, refer to the `SFolderICM` education sample in the `CMBROOT\Samples\java\icm` directory.

C++

```
// NOTE: Print function provided in SFolderICM API Education Sample

// Get the DKFolder object.
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);

// Access contents
dkIterator* iter = dkFolder->createIterator(); //Create an Iterator
while(iter->more()){                           //While there are items left
 DKDDO* ddo = (DKDDO*) iter->next()->value(); //Move to & return next element
 cout << "Item Id: " << ((DKPidICM*)ddo->getPidObject())->getItemId()<< endl;
}
```

For the complete sample application, refer to the `SFolderICM` education sample in the `CMBROOT\Samples\cpp\icm` directory.

Using DKAny (C++ only)

DKAny contains any object whose type can vary at run time. A DKAny object can be any of the following types:

- null
- (unsigned) short

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

A DKAny can only be NULL if it has not been assigned a value (`DKAny any`). After it has been assigned a value, then `DKAny::isNull()` will return FALSE.

In addition to the above types, a DKAny object can also contain the following object reference types:

- dkDataObjectBase*
- dkCollection*
- void*

Using type code

You can determine the current type of a DKAny object by calling the typeCode function, which returns a `TypeCode` object, that is, `tc_null` for null, `tc_short` for short, and so forth. Refer to the *online API reference* for a complete listing of type codes.

Managing memory in DKAny

DKAny manages the memory for the object it contains, unless the contained object is an object reference type. Copy related operations involving object references will create a copy of the pointer only. You need to keep track of object reference types during copying and deletion.

Using constructors

DKAny provides a constructor for each type it supports. The following example shows how to create a DKAny object that contains some of the types listed in the previous section.

C++

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

Getting the type code

Use the `typeCode` function to find the type code of the object inside DKAny.

C++

```
DKAny::TypeCode type_code;
type_code = any1.typeCode(); // type_code is tc_ushort
type_code = any4.typeCode(); // type_code is tc_time
type_code = any5.typeCode(); // type_code is tc_dobase (object ref)
type_code = any6.typeCode(); // type_code is tc_voidptr since
                             // MyObject is not recognized by DKAny
```

Assigning a new value to DKAny

To assign a new value to an existing DKAny object, use the equal sign (=) assignment operator. DKAny provides an assignment for each type code.

C++

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

Assigning a value from DKAny

Assigning a DKAny back to a regular type requires a cast operator. For example:

C++

```
vlong      = (long) any2;                         // sets vlong to 200
DKTime at  = (DKTime) any4;                       // sets at to (10,20,30)
DKDDO* ddo = (DKDDO*) ((dkDataObjectBase*) any5); // extract the ddo
dkDataObjectBase* dobase = any7;                  // extract the DDO
```

You will get an invalid type conversion exception if the type does not match. Therefore, you must check the type code before converting DKAny to a regular type:

C++

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

You can create a case statement to check the type of DKAny, as follows:

C++

```
switch(any.typeCode()) {
   case DKAny::tc_short:
        // operation for short
        ...
        break;
   case DKAny::tc_ushort:
        // operation for unsigned short
        ...
        break;
   ...  etc.
}
```

If the DKAny object contains an object reference, you can get the DKAny content as a void pointer, then cast it to the proper type. However, use this operation only if you know the type code that is used inside DKAny:

C++

```
// knows exactly any5 contains DKDDO
ddo = (DKDDO*) any5.value();
```

Displaying DKAny

You can use cout to display the content of a DKAny object:

C++

```
cout << any3 << endl;   // displays "any string"
cout << any4 << endl;   // displays "10:20:30"
cout << any5 << endl;   // displays "(dkDataObjectBase*) <address>",
                        // where address is the memory location of the ddo
```

Destroying DKAny

Because DKAny can hold an object reference but does not manage memory for object reference types, you must manage the memory for these types. The following example manages the memory for a DKAny object:

C++

```
DKDDO* ddo = new DKDDO;              // creates a DKDDO in the heap
DKAny anyA((dkDataObjectBase*)ddo);
DKAny* anyB = new DKAny(anyA);       // creates anyB in the heap
                                     // anyA and anyB contains a
                                     // reference to the same ddo
...
delete anyB;                         // delete anyB, does not delete ddo
if (anyA.typeCode() == DKAny::tc_dobase)
    delete ((dkDataObjectBase*) anyA.value()); // deletes the ddo
```

The last delete statement must be performed before exiting the scope, otherwise anyA is deleted, leaving the DDO as a memory leak.

Programming tips

Recommendation: When converting an integer literal to DKAny, it is advisable to state the type explicitly to avoid an undesirable type conversion. Turn to

C++

```
any = 10;                       // ambiguous
any = (unsigned long) 10;       // unambiguous
any = (short) 4;                // unambiguous
```

Using collections and iterators

dkCollection is an abstract class providing the methods for working with a collection. DKSequentialCollection is the concrete implementation of dkCollection. Other collections are implemented as subclasses of DKSequentialCollection. These collections contain the data objects as members.

Collection members are usually objects of the same type; however, a collection can contain members of different types.

C++ only: When a new member is added, the collection owns it. When the member is retrieved, you get a pointer to a DKAny object inside the collection. This object belongs to the collection, meaning that the collection manages the memory for its DKAny members. A DKAny object can hold an object reference but cannot manage memory for object reference types, you must manage the memory for those.

Using sequential collection methods

DKSequentialCollection provides methods for adding, retrieving, removing, and replacing its members. In addition, it also has a sort method. The following example illustrates how to add a new member to a collection (the addElement method takes an object as the parameter).

Java

```
DKSequentialCollection sq = new DKSequentialCollection();
String str = " first member ";
sq.addElement(str);         // add a new element at the last position
```

C++

```
DKSequentialCollection sq;
DKAny any = DKString(" first member ");
sq.addElement(any);         // add a new element at last position
                            // any will be copied into the collection
                            // you own the original any, the collection
                            // owns the copy
```

Using the sequential iterator

You iterate over collection members using iterators. The APIs have two types of iterators: dkIterator and DKSequentialIterator.

Java

dkIterator, the base iterator, supports the next, more, and reset methods. The subclass DKSequentialIterator contains more methods. You create an iterator by calling the createIterator method on the collection. After creating the iterator, use the setToFirst() method to point to the first item. The following example shows using an iterator:

```
dkIterator iter = sq.createIterator();   // create an iterator for sq
Object member;
iter.setToFirst();
member = iter.at();
while(iter.more()) {              // While there are more members
      member = iter.next();       // move to the next member and get it

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

C++

Iterators are provided to let you iterate over collection members. There are two types of iterators: the base iterator dkIterator, which supports the `next`, `more`, and `reset` functions; and its subclass DKSequentialIterator, which contains more functions. An iterator is created by calling the `createIterator` function on the collection. This function creates a new iterator and returns it to you. Use the following code to iterate over a collection:

```
dkIterator* iter = sq.createIterator();   // create an iterator for sq
DKAny* member;
                                          // while there are more members
                                          // get the current member and
                                          // advance iter to the next member
while(iter->more()) {
    member = iter->next();

    cout << *member << endl;              // display it, if you want to
    ...                                   // do other processing
    }
delete iter;                              // do not forget to delete iter
```

DKSequentialIterator provides additional methods to move the iterator in either direction. The previous example could be rewritten as follows:

Java

```
// ----- Create a sequential iterator for sq
DKSequentialIterator iter =
             (DKSequentialIterator) sq.createIterator();
Object member;
iter.setToFirst();
while(iter.more()) {
      member = iter.at();                 // get the current member
      ....                                // do other processing
      iter.setToNext();                   // advance to the next position
     }
```

C++

```
DKSequentialIterator* iter =                  // create an iterator for sq
(DKSequentialIterator*) sq.createIterator();
DKAny* member;
      while(iter->more()) {
         member = iter->at();                 // get the current member
         ...                                  // do other processing
         iter->setToNext();                   // advance to the next position
      }
      delete iter;
```

This code allows you to perform some operations on the current member before moving to the next member. Such an operation could be replacing a member with a new one, or removing it.

Java

```
String st1 = "the new first member";
sq.replaceElementAt(st1, iter);   // replace current member with a new one
....                              // or
sq.removeElementAt(iter);         // remove the current member
....
```

C++

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

sq.replaceElementAt(any, *iter);   // replace current member with a new one
...                                // or
sq.removeElementAt();              // remove the current member
...
```

Tip: When you remove the current member, the iterator is advanced to the next member. When removing a member inside a loop, check it as in the following example. You should check the removal condition to avoid skipping the next member after removing the current one.

Java

```
   ....
   if (removeCondition == true)
     sq.removeElementAt(iter); // remove current member, do not advance the
                               //   iterator since it is advanced to the next
                               //   after the removal operation
   else
        iter.setToNext();      // if no removal, advance the iterator to the
   ....                        //   next position
```

C++

```
...
if (removeCondition == TRUE)
    sq.removeElementAt(*iter);   // remove current member, do not advance iter
                                 // since it is advanced to the next after
                                 // the removal operation
else
    iter->setToNext();           // no removal, advance the iterator
...                              // to the next position
```

Managing memory in collections (C++ only)

The collection manages the memory for its members, which are DKAny objects. The same rules governing DKAny objects apply here, if the object inside DKAny is an object reference type then you are responsible for managing the memory when you are:

- Destroying the collection.
- Replacing a member.
- Removing a member.

This example shows how to manage the memory in these situations:

C++

```
// retrieve the member and hang-on to it
member = iter->at();

// code to handle this member as to prevent memory leaks
if (member->typeCode() == DKAny::tc_dobase) {
    // delete it if no longer needed
    delete ((dkDataObjectBase*) member->value());
 }

sq.removeElementAt(*iter);           // remove it from the collection
```

Instead of deleting the member you can add it into another collection. You should take similar steps before using replaceElementAt and removeAllElement functions.

Before destroying a collection, delete its members. You can write a function to perform this task and pass this function to the apply function for the collection. Suppose you have a collection of DKAny objects containing DKAttributeDef objects. The following example deletes the collection:

C++

```
DKDatastoreICM dsICM;
...
DKAny any = dsICM.listSchemaAttributes("GRANDPA");
dkCollection* acoll = (dkCollection*) any;
...                                         // use the attributes
acoll->apply(deleteDKAttributeDef);                // deletes all members
delete acoll;
```

In this example, deleteDKAttributeDef is a function that takes the DKAny object as a parameter. It is defined as follows:

C++

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

You could write your own delete function to delete your collection or remove some members before deleting the collection.

The destructors for some known collections, like DKParts, DKFolder, and DKResults, perform these necessary clean-up steps. However, they do not manage storage when running replaceElementAt, removeElementAt, or removeAllElement functions.

Sorting the collection

Use the sort function to sort collection members in either ascending or descending order based on a specified key. You must pass a sort object and the desired order. The interface for sort objects is defined in dkSort.java (Java) or dkSort.hpp (C++). You can write your own sort function for sorting your specific collection. The following example illustrates how to sort a collection of DDOs:

Java

```
DKResults rs;
....                  // Execute a query to fill DKResults with DDOs
....
DKSortDDOId sortId;   // Declare the sort function object; sort on item-id
rs.sort(sortId);      //    by default, sort in ascending order
....
```

C++

```
DKResults* rs;
....
// Execute a query to fill DKResults with DDOs
....
DKSortDDOId* sortId; // Declare the sort function object; sort on item-id
rs->sort(sortId); // by default, sort in ascending order
....
```

Tip: The sort object is created in the stack, so it does not have to be explicitly deleted. The function is reentrant, meaning that a single copy can be shared, reused, or passed to another function.

Understanding federated collection and iterator

Use a federated collection in your application to process data objects resulting from a query as a collection. The federated collection preserves the sub-grouping relationships that exist between the data objects.

A federated collection is a collection of DKResults objects. It is created to hold the results of DKFederatedQuery, which can come from several heterogeneous content servers. Each DKResults object contains the search results from a specific content server. A federated collection can contain an infinite number of nested collections.

To step through a federated collection, create and use a dkIterator or DKSequentialIterator. Then create another dkIterator to step through each DKResults object to iterate over it and process it according to its originating content server.

You can also create a federated iterator, dkFederatedIterator, and use it to step through all collection members, regardless of which content server the result came from.

Restriction: You cannot query a federated collection.

Figure 8 shows the structure and behavior of DKFederatedCollection.

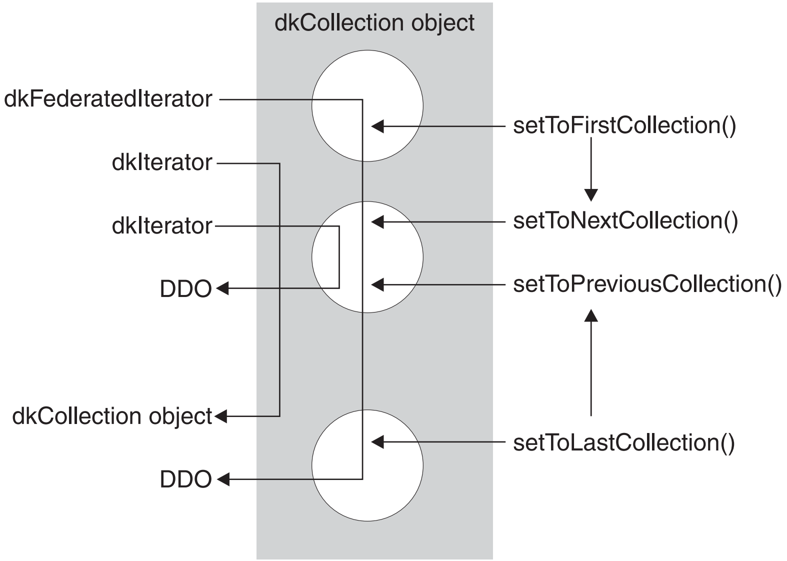


Figure 8. Structure and behavior of DKFederatedCollection

In Figure 8, the oval represents the DKFederatedCollection containing several smaller circles which are DKResults objects. The dkFederatedIterator traverses collection boundaries and returns a DDO each time.

The first dkIterator is an iterator for the DKFederatedCollection and returns a DKResults object each time. The second dkIterator is an iterator for the second DKResults object; it returns a DDO for each member of the DKResults collection.

The `setToFirstCollection` function in dkFederatedIterator sets the position to the first DDO of DKFederatedCollection. In this case, it is the first element of the first DKResults collection object. At this point, if the `setToNextCollection` function is invoked, it sets the iterator position to the first DDO of the second DKResults collection.

The `setToLastCollection` function in dkFederatedIterator sets the iterator position to the last DDO of DKFederatedCollection. In this case, it is the last element of the last DKResults collection object. If the `setToPreviousCollection` function is invoked, it sets the iterator position to the last DDO of the previous DKResults collection.

Querying a content server

You can search a content server and receive results in a dkResultSetCursor or DKResults object. For some servers, you can create a query object to represent your query, then invoke the `execute` function or `evaluate` function of the query object. With the help of its content servers, the query object performs query processing tasks, such as preparing and executing a query, monitoring the status of a query execution, and storing the results. Some content servers support using a query object as an alternative; earlier Content Manager and the federated content server are two of these.

For the content servers that support query objects, there are four types of query objects: parametric, text, image and combined. The combined query is composed of both text and parametric queries. Not all content servers can perform combined queries. Earlier Content Manager supports image query.

For Content Manager parametric and text queries are integrated. You should not use query objects; for information on how to query in Content Manager, see "Querying the Content Manager server" on page 178. For information about querying an earlier Content Manager server, see *Working with other content servers.*

A content server uses two methods for running a query: `execute` and `evaluate`. The `execute` function returns a dkResultSetCursor object, the `evaluate` function returns a DKResults object. The dkResultSetCursor object is used to handle large result sets and perform `delete` and `update` functions on the current position of the result set cursor. You can use the `fetchNextN` function to retrieve a group of objects into a collection.

dkResultSetCursor can also be used to rerun a query by calling the close and open methods. This is described in "Using the result set cursor" on page 116.

DKResults contains all of the results from the query. You can iterate over the items in the collection either forward or backward and can query the collection or use it as a scope for another query.

See "Opening and closing the result set cursor to rerun the query" on page 117 for more information.

Restriction: When you query a Domino.Doc content server, a DKResults object is returned. However, you cannot query it nor use it as a scope for another query.

Differences between dkResultSetCursor and DKResults

A dkResultSetCursor and a DKResults collection have the following differences:

- The dkResultSetCursor works like a content server cursor; it can be used for large result sets because the DKDDOs it contain are fetched one at a time. It can also be used to run a query again to get the latest results.

  Restriction: You cannot rerun a query on a Domino.Doc content server even when using a dkResultSetCursor.
- The DKResults contains the entire result set and supports a bi-directional iterator.

- Leaving a dkResultSetCursor open for long periods of time may degrade performance of concurrent users for some content servers.

Using parametric queries

A parametric query is a query requiring an exact match on the condition specified in the query predicate and the data values stored in the content server.

Note: The query examples in the following sections apply to earlier Content Manager. For query information about Content Manager V8, see "Understanding the query language" on page 177.

Formulating a parametric query string

To create a query you first formulate a query string. In the following example, the query string is defined to represent a query on the index class named GP2DLS2 in earlier Content Manager. For examples of query string in Content Manager, see "Query examples" on page 185. The condition of the query is to search for all documents or folders where the attribute DLSEARCH_DocType is not `null`. The maximum number of results returned is limited to five, and the content is set to `YES` so that contents of the document or folder are returned.

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)";
```

The example specifies that an earlier Content Manager server uses dynamic SQL for this query and that all folders and documents be searched. Different content servers use different query string syntax; federated has its own query string syntax. Refer to the information on the content server you want to search or the *online API reference* for more inforamtion.If the attribute name has more than one word or is in a DBCS language, it should be enclosed in apostrophes (′). If the attribute value is in DBCS, it should be enclosed in double quotation marks (″).

Formulating a parametric query on multiple criteria

You can specify more than one search criteria for a parametric query. The following example shows how to specify a query on two index classes for earlier 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)";
```

Executing a parametric query

After you have a query string you create the query object. The DKDatastore*xx* that represents a content server contains a method for creating a query object. You use the query object to execute the query and obtain the results. The following example shows how to create a parametric query object and execute the query on an earlier Content Manager server; you should not use a query object with Content Manager Version 8 or later. Once the query is executed, the results are returned in a DKResults collection.

Attention: When you delete a DKResults object, all of its members are also deleted. Make sure that you do not delete the element twice.

Java

```
// ----- Create the datastore, the query object, and the results set
DKDatastoreDL dsDL = new DKDatastoreDL();
dkQuery pQry = null;
DKResults pResults = null;
DKNVPair parms[] = null;
// ----- Connect to the datastore
dsDL.connect(libSrv,userid,pw,"");
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
                      "MAX_RESULTS=5," +
                      "COND=(DLSEARCH_DocType <> NULL));" +
                      "OPTION=(CONTENT=YES;"       +
                              "TYPE_QUERY=STATIC;" +
                              "TYPE_FILTER=FOLDERDOC)";
// ----- Create the query using the query string
pQry = dsDL.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Execute the query
pQry.execute(parms);
// ----- Process the results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
// ----- Disconnect when you are through
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (`TSamplePQryDL.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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();
```

The complete sample application from which this example was taken (`TSamplePQryDL.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

Executing a parametric query from a content server

The DKDatastore*xx* that represents a content server has a method to execute a query. The following example shows how to execute a parametric query on an earlier Content Manager content server. After the query is executed, the results are returned in a dkResultSetCursor object.

Java

```
// ----- Create the datastore and cursor
DKDatastoreDL dsDL = new DKDatastoreDL();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect to the content server
dsDL.connect(libSrv,userid,pw,"");
// ----- Formulate the query string
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)";
...
// ----- Execute the query using the query string
pCur = dsDL.execute(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Process query results as you want
...
// ----- When finished with the cursor, delete it, and disconnect
pCur.destroy();
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TExecuteDL.java) is available in the CMBROOT\Samples\java\dl directory.

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();
...
```

The complete sample application from which this example was taken (TExecuteDL.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Evaluating a parametric query from a content server

The DKDatastore*xx* that represents a content server has a method to evaluate a query. The results are returned in a DKResults collection. The following example shows how to evaluate a parametric query on an earlier Content Manager content server.

Java

```
// ------ Create the query string
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;
// ----- Create the datastore and connect
//  replace following with your library server, user ID,
// password DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Call evaluate, get the results, and create an
iterator to process them
DKResults pResults =
  (DKResults)dsDL.evaluate(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
       item = (DKDDO)pIter.next();
        ... // ------ Process the DKDDO as appropriate
      }
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();
       // Process the DKDDO
}
delete pIter;
delete pResults;
dsDL.disconnect();
```

Using text query

In Content Manager Version 8 and later, text and parametric queries are integrated; see “Creating combined parametric and text search” on page 182.

In earlier Content Manager, you can perform text and parametric searches. Text searches query the text indexes created by the Text Search Engine to search the actual document text.

Formulating a text query string

You start a text search by formulating a query string. In the following example, a query string is created representing a query against the TMINDEX text index. The query string contains criteria to search for all text documents with the word UNIX® or member. The maximum number of results returned is five.

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)";
```

Formulating a text query on multiple indexes

You can use text query to search more than one index. The following example shows how to specify a query for two indexes.

Important: If you specify more than one text search index in the query, the indexes must be the same type. For example, you can specify two precise indexes in the query, but you cannot specify a precise index and a linguistic index within the query.

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)";
```

Executing a text query

After you have a text query string you create the query object. The DKDatastore*xx* that represents a content server contains a method for creating a query object. The results are returned in a DKResults collection. You use the query object to execute the query and obtain the results. The following example shows how to create a text query object and execute a query.

Java

```
// ----- Create the datastore; declare query and the results
DKDatastoreTS dsTS = new DKDatastoreTS();
dkQuery pQry = null;
DKResults pResults = null;
DKNVPair parms[] = null;
// ----- Connect to the datastore
//      for example, dsTS.connect("zebra","7502",DK_CTYP_TCPIP);
dsTS.connect(srchSrv,"","","");
// ----- Formulate the query string
String cmd = "SEARCH=(COND=(member AND UNIX));" +
                     "OPTION=(SEARCH_INDEX=TMINDEX)";
// ----- Create and execute the query
pQry = dsTS.createQuery(cmd, DK_CM_TEXT_QL_TYPE, parms);
pQry.execute(parms);
// ---- Process the results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
// ------ When finished, disconnect
dsTS.disconnect();
dsTS.destroy();
```

The complete sample application from which this example was taken (`TSampleTQryTS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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();
```

The complete sample application from which this example was taken (`TSampleTQryTS.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

Running a text query from a content server

The DKDatastore*xx* used to represent a content server provides a method to run a query. The results are returned in a dkResultSetCursor object. The following example shows how to run a text query against an earlier Content Manager server:

Java

```
// ----- Create the datastore; declare query and the results
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect to the datastore
//       for example, dsTS.connect
("zebra","7502",DK_TS_CTYP_TCPIP);
dsTS.connect(srchSrv,"","","");

// ----- Formulate the query string
String cmd = "SEARCH=(COND=(internet OR UNIX));" +
               "OPTION=(SEARCH_INDEX=TMINDEX;" +
                "MAX_RESULTS=5)";
...
// ----- Execute the query and process the results
as appropriate
pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);
...
// ----- When finished, delete the cursor and disconnect
pCur.destroy();
dsTS.disconnect();
dsTS.destroy();
```

The complete sample application from which this example was taken (`TExecuteTS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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) {
           // Process the DKDDO
          ...
          delete item;
       }
}
delete pCur;
dsTS.disconnect();
```

The complete sample application from which this example was taken (`TExecuteTS.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

Evaluating a text query from a content server

The DKDatastore*xx* that you use to represent a content server provides an evaluate method to run a query and return a DKResults collection. The following example shows how to evaluate a text query against an earlier Content Manager content server.

Java

```
// ----- Create the datastore and the query string
DKDatastoreTS dsTS = new DKDatastoreTS();
String cmd = "SEARCH=(COND=($MC=*$ UN*));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreTS dsTS;
// ----- Connect to the datastore
dsTS.connect("TM","", ' ');
...
// ----- Call evaluate, get the results, and process
as appropriate
DKResults pResults = (DKResults)dsTS.evaluate(cmd,DK_CM_TEXT_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more()) {
       item = (DKDDO)pIter.next();
       // ----- Process the individual DKDDO objects
      }
// ----- Disconnect
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();
       // Process the DKDDO
       ...
}
delete pIter;
delete pResults;
dsTS.disconnect();
```

Getting match highlighting information

The match information contains the text of the document and the highlighting information for every match of the corresponding query.

When formulating the query string you set MATCH_INFO and MATCH_DICT options. Set MATCH_INFO to YES to return the match highlighting information. The MATCH_DICT option specifies whether the highlighting information will be obtained using a dictionary. The match information is returned in the DKMATCHESINFO attribute in the DKDDO returned from a text query. The value of the DKMATCHESINFO attribute will be a `DKMatchesInfoTS` object.

Getting match highlight information is time consuming because the document is retrieved from the content server and analyzed linguistically to determine potential matches. Running this process impacts the performance of a text query.

Getting match highlighting information for each text query result item: The following example retrieves match highlighting information for each text query result item during a text query. Because the `MATCH_DICT` option is set to `NO`, the dictionary is not used.

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect to the content server
//     replace following with your library server,
user ID, password
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,
FRNADMIN, PASSWORD)");
// ----- Formulate the query string
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())
{
   // ----- Get the next DKDDO
   item = pCur.fetchNext();
   if (item != null)
   {
// continued...
```

Java (continued)

```
     // ----- Process the 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;
            // ----- process the Match Hightlighting information
            if (pMInfo != null)
            {
               strDoc = pMInfo.getDocumentName();
             numberSections = pMInfo.numberOfSections();
               // ----- loop thru document sections
               for (j = 1; j <= numberSections; j++)
               {
                  pMSect = pMInfo.getSection(j);
                  strSection = pMSect.getSectionName();
                  numberParagraphs = pMSect.numberOfParagraphs();
                  // ----- loop thru section paragraphs
                for (k = 1; k <= numberParagraphs; k++)
                {
                     pMPara = pMSect.getParagraph(k);
                     lCCSID = pMPara.getCCSID();
                     lLang = pMPara.getLanguageId();
                     numberTextItems = pMPara.numberOfTextItems();
                     // ----- loop thru paragraph text items
                     for (m = 1; m <= numberTextItems; m++)
                     {
                        pMText = pMPara.getTextItem(m);
                        strText = pMText.getText();
                        // -----  if match found in text item get offset
                        //      and  length of match in text item
                        if (pMText.isMatch() == true)
                      {
                           lOffset = pMText.getOffset();
                           lLen = pMText.getLength();
                        }
                        numberNewLines = pMText.numberOfNewLines();
                     }
                   }
                 }
               }
            }
          }
       }
}
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 *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;
while (pCur->isValid())
 {
  item = pCur->fetchNext();
  if (item != 0)
  {
   // Process the 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;
        }
        case DKAny::tc_short :
        {
         ...
         break;
        }
        case DKAny::tc_dobase :
        {
// continued...
```

C++ (continued)

```
         // process the Match Hightlighting information
         pDOBase = a;
         pMInfo = (DKMatchesInfoTS*)pDOBase;
   if (pMInfo != 0)
   {
     strDoc = pMInfo->getDocumentName();
     numberSections = pMInfo->numberOfSections();
          // loop thru document sections
    for (j = 1; j <= numberSections; j++)
    {
     pMSect = pMInfo->getSection(j);
     strSection = pMSect->getSectionName();
     numberParagraphs = pMSect->numberOfParagraphs();
            // loop thru section paragraphs
     for (k = 1; k <= numberParagraphs; k++)
     {
      pMPara = pMSect->getParagraph(k);
      lCCSID = pMPara->getCCSID();
      lLang = pMPara->getLanguageId();
      numberTextItems = pMPara->numberOfTextItems();
            // loop thru paragraph text items
        for (m = 1; m <= numberTextItems; m++)
        {
         pMText = pMPara->getTextItem(m);
         strText = pMText->getText();
             // if match found in text item get offset and
             // length of match in text item
         if (pMText->isMatch() == TRUE)
         {
          lOffset = pMText->getOffset();
          lLen = pMText->getLength();
              }
             numberNewLines = pMText->numberOfNewLines();
            }
           }
          }
         }
         break;
        }
        default :
        {
         break;
        }
       }
      }
      ...
      delete item;
    }
}
delete pCur;
dsTS.disconnect();
```

Getting match highlighting information for a particular text query result item: The following example retrieves the match highlighting information for a specific item returned from a text query. The dkResultSetCursor passed to this routine must be in an open state.

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();
  // Process the 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(""))
   {
    strXNAME = p.getObjectType();
   }
   ...
  }
// continued...
```

Java (continued)

```
  // Get Match Highlighting Information
  pMInfo = dsTS.getMatches(pCur,strDID,strXNAME,false);
  strDID = "";
  strXNAME = "";
  if (pMInfo != null)
  {
   strDoc = pMInfo.getDocumentName();
   numberSections = pMInfo.numberOfSections();
   // loop thru document sections
   for (j = 1; j <= numberSections; j++)
   {
    pMSect = pMInfo.getSection(j);
    strSection = pMSect.getSectionName();
    numberParagraphs = pMSect.numberOfParagraphs();
    // loop thru section paragraphs
    for (k = 1; k <= numberParagraphs; k++)
    {
     pMPara = pMSect.getParagraph(k);
     lCCSID = pMPara.getCCSID();
     lLang = pMPara.getLanguageId();
     numberTextItems = pMPara.numberOfTextItems();
     // loop thru paragraph text items
     for (m = 1; m <= numberTextItems; m++)
     {
      pMText = pMPara.getTextItem(m);
      strText = pMText.getText();
      // if match found in text item get offset and
      // length of match in text item
      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();
    // Process the DKDDO
    for (i = 1; i <= item->dataCount(); i++)
    {
     anyObj = item->getData(i);
     strDataName = item->getDataName(i);
     if (strDataName == "")
     {
      strDID = pid.getId();
     }
     if (strXNAME == "")
     {
      strXNAME = p->getObjectType();
     }
     switch (anyObj.typeCode())
     {
      ...
     }
    }
// continued...
```

C++ (continued)

```
    // Get Match Highlighting Information
    pMInfo = dsTS.getMatches(pCur,strDID,strXNAME,FALSE);
    strDID = "";
    strXNAME = "";
    if (pMInfo != 0)
    {
     strDoc = pMInfo->getDocumentName();
     numberSections = pMInfo->numberOfSections();
     // loop thru document sections
     for (j = 1; j <= numberSections; j++)
     {
      pMSect = pMInfo->getSection(j);
      strSection = pMSect->getSectionName();
      numberParagraphs = pMSect->numberOfParagraphs();
      // loop thru section paragraphs
      for (k = 1; k <= numberParagraphs; k++)
      {
       pMPara = pMSect->getParagraph(k);
       lCCSID = pMPara->getCCSID();
       lLang = pMPara->getLanguageId();
       numberTextItems = pMPara->numberOfTextItems();
       // loop thru paragraph text items
       for (m = 1; m <= numberTextItems; m++)
       {
        pMText = pMPara->getTextItem(m);
        strText = pMText->getText();
        // if match found in text item get offset and
        // length of match in text item
        if (pMText->isMatch() == TRUE)
        {
         lOffset = pMText->getOffset();
         lLen = pMText->getLength();
        }
        numberNewLines = pMText->numberOfNewLines();
       }
      }
     }
     delete pMInfo;
    }
    ...
   delete item;
   }
}
delete pCur;
dsTS.disconnect();
```

Using the result set cursor

The dkResultSetCursor is a content server cursor that manages a virtual collection of DDO objects. This means that the collection does not materialize until you fetch an element from it. The collection is the resulting set of a items that met the criteria of the query. When you are finished using the cursor, call the `destroy` method to free the memory it used.

Important: The information in this section does not apply to Content Manager 8.2. See “Understanding the query language” on page 177 for details.

Opening and closing the result set cursor to rerun the query

When you create a result set cursor, it is in an open state. To rerun the query, close and reopen the cursor.

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();          //re-execute the query
```

C++

```
DKString cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);";
cmd +=   "OPTION=(CONTENT=YES;";
cmd +=   "TYPE_QUERY=DYNAMIC;" ;
cmd +=   "TYPE_FILTER=FOLDERDOC)";
...

dkResultSetCursor* pCur = dsDL.execute(cmd);
// re-execute the query
pCur->close();
pCur->open();
```

Setting and getting positions in a result set cursor

You can use the result set cursor to set and get the current position. The following example creates and executes a query. Inside a while loop, the cursor position is set to the first (or next) valid position. Then a DDO is fetched from that position. A null is returned from the `fetchObject` method if the cursor is past the last item.

Java

```
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);"  +
                      "OPTION=(CONTENT=YES;"   +
                       "TYPE_QUERY=DYNAMIC;"   +
                       "TYPE_FILTER=FOLDERDOC)";
DKNVPair parms[] = null;
DKDDO item = null;
int i = 0;
...
// ----- Execute the query; the result cursor is returned
dkResultSetCursor pCur = dsDL.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
// ----- Use a while loop to iterate thru the collection
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;
```

Another way to do this is:

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;
```

You can use relative positioning when iterating through the items. The following example skips every other item in the result set cursor.

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);   //  move cursor 2 positions forward
   item = pCur.fetchObject();         //    from the current podition
   if (item != null) {                //    (relative)
      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;
```

Creating a collection from a result set cursor

You can use a result set cursor to populate a collection with a specified number of items from the result set cursor. The first parameter of the `fetchNextN` method specifies how many items to put into the collection. Passing a zero in the first parameter to indicates that all items will be put into the collection.

In the following example, all items from the result set cursor are fetched into the sequential collection. If `fItems` is `TRUE`, at least one item was returned.

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);
```

Querying collections

A *queryable collection* is a collection that can be queried further, thus providing a smaller set and more refined results. A concrete implementation of a queryable collection is a DKResults object, returned as the results of a query evaluation. DKResults is a subclass of dkQueryableCollection and is a collection of DDOs.

Getting the result of a query

The following example illustrates how to submit a parametric query and get results. The results are in `rs`, which is a DKResults object. You can use previous code examples to process the collection and get the DDO.

Java

```
// ----- Create a datastore and establish a connection
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Create and execute a query object
String query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> null));";
DKParametricQuery pq =
 (DKParametricQuery) dsDL.createQuery(query1,DK_CM_PARAMETRIC_QL_TYPE, null);
pq.execute();
// ----- Get the reuslt
DKResult rs = (DKResults) pq.result();
```

C++

```
// establish a connection
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
// create a query object
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();
```

Evaluating a new query

You can query the result from a query to further refine it. The following code, based on the previous example, shows re-evaluating a query:

Java

```
String query2 = "SEARCH=(INDEX_CLASS=GRANDPA, COND=(Subject == 'Mystery'));";
Object obj = rs.evaluate(query2,DK_CM_PARAMETRIC_QL_TYPE, null);
....
```

C++

```
DKString query2="SEARCH=(INDEX_CLASS=GRANDPA, COND=(Subject=='Mystery'));";
         any = rs->evaluate(query2,DK_PARAMETRIC_QL_TYPE, NULL);
         ...
```

The second query returns `obj`, a DKResults object containing the refined results. The combined results of both queries would be equivalent to:

```
"SEARCH=(INDEX_CLASS=GRANDPA, COND=(Title <> NULL AND Subject == 'Mystery'));"
```

You can repeat the query until you get satisfactory results. After you start with one type of query, the subsequent queries must be of the same type. If you mix query types, the result might be null.

The following example shows sequential text queries:

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("TM","","","");

// ----- The first query
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();
// ----- The second query
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);
```

The second query returns `obj`, a DKResults object containing the refined results. The combined results of both queries would be equivalent to:

```
"SEARCH=(COND=(IBM AND Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
```

Using queryable collection instead of combined query

A combined query provides the flexibility to submit a combination of parametric and text queries, with or without scopes. However, all of these queries must be submitted at once, not one at a time as you would when evaluating a queryable collection.

A combined query returns a DKResults object; however, you cannot evaluate another parametric query against it. You cannot use combined queries on all content servers.

Evaluating a queryable collection with subsequent queries provides the flexibility to refine the results of a previous query, step by step, until you get a satisfactory final result. Subsequent queries are useful for browsing a content server dynamically and formulating the next query based on the previous results. However, if you know the total query in advance, it is more efficient to submit the complete query once or use a combined query.

Working with Content Manager Version 8.2

This section describes the Content Manager Version 8 Release 2 connector (ICM connector) application programming interfaces (APIs). The ICM connector is an extension of the Enterprise Information Portal (EIP) framework, so it is essential that you understand the EIP framework concepts described in "Enterprise Information Portal application programming concepts" on page 11 before continuing with this information.

You can use the ICM connector APIs to build and deploy custom applications that access a Content Manager content server. You can also use the APIs to integrate your existing applications into a Content Manager content server.

This section contains the following information:

- Understanding the Content Manager system
- Understanding Content Manager concepts
- Planning a Content Manager application
- Creating a Content Manager application
- Controlling access to information
- Processing transactions
- Searching for items

Understanding the Content Manager system

The main components of the Content Manager system include a library server, one or more resource managers, and a set of object-oriented application programming interfaces (APIs). To administer your Content Manager system, you are also provided with a Java-based system administration client.

The library server provides you with flexible data modeling capabilities, secure access to your system, efficient managing of content, and other features. The library server manages the relationships between items in the system and controls access to all of the system information, including the information stored in the resource managers configured in the system.

The resource manager is the component that stores the actual content of any binary object, like a scanned image, an office document, or video. You can integrate other resource managers, like Content Manager VideoCharger or other non-IBM products, into your Content Manager system. With the resource manager you can complete the following tasks:

- Automatically move content from costly high-speed media to slower less expensive media using System Managed Storage (SMS).
- Access the resource manager directly from a Web browser.
- Retrieve all or part of an object.
- Synchronize your data with the library server.

The APIs provide applications with access to the Content Manager system. The APIs are available for Java and C++. Using the APIs, your applications can take advantage of all of the Content Manager functionality, such as data modeling, integrated parametric and text search, third-party data access and delivery, and so forth.

The diagram in Figure 9 illustrates how the system components fit together. Keep in mind that this is only one implementation of a Content Manager system. In another system configuration you might have four resource managers, for example.

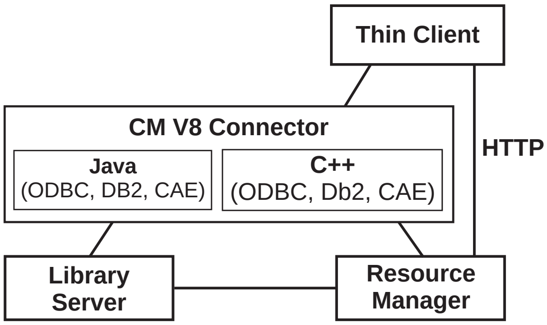


Figure 9. System configuration

Understanding Content Manager concepts

This section describes important Content Manager concepts. It is imperative that you understand the Content Manager concepts before you proceed to the programming tasks. The information described in this section includes:

- Items.
- Attributes.
- Item types.
- Root and child components.
- Objects.
- Links and References.
- Documents.
- Folders.
- Versioning.
- Access control.
- Document management data model.

Items

An item is the basic entity managed by the library server. Examples of items include a policy, claim, phone number, and so forth. An *item* is a generic term for an instance of an item type. If an object is a discrete piece of digital content, then an item is a representation of that object. The item is not the object, but it thoroughly identifies it and how to find it. In the system, items represent objects including documents and folders. To define business objects, like a document, you work with item definitions.

When an application creates an item, Content Manager assigns the item several system-defined attributes and allows you to define your own attributes.

The system-defined attributes include a creation time stamp and an item identifier (item ID). The item ID is unique for every item. The itemID is stored by Content

Manager and used to locate the item within the library server. When writing your application, you use the itemID to access all of the data associated with the item.

Attributes

An *attribute* is a unit of data that describes a certain characteristic or property (for example, name, address, age, and so forth) of an item, and it can be searched on to locate the item.

You can group attributes to make attribute groups. For example, the address attribute can be made up of a group of attributes including street, city, state, and zip code.

You can also define attributes that have multiple values. Such attributes are called multi-valued attributes, which are implemented as child components. For example, you can store multiple addresses, home address, work address, and so forth for a policy owner.

For additional information, see the `SAttributeDefinitionCreationICM` sample.

Item types

An *item type* (index class in earlier Content Manager versions) is essentially a template for defining and later locating like items. An item type consists of a root component, zero or more child components, and a classification. An item type is the overall structure containing all the components and associated data. For example, in an insurance scenario a policy item type contains items with attributes like policy number, name, claim, and so forth.

In Content Manager, there are four classifications of item types: non-resource, resource, document (also known as document model), and document part. A non-resource item type represents entities that are not stored in a resource manager. A resource item type represents objects stored in a resource manager, like files in a file system, video clips in a video server, LOBs (large objects) in database tables, and so forth. A document item type represents entities that contain document parts that contain resource content just as a single resource item type does. A document part item type represents objects stored in a resource manager, but are parts of a document, contained and owned by a document item type. Content Manager provides a base set of resource item types: LOB, text, image, stream, and video objects.

For more information, see the `SItemTypeCreationICM` sample.

Root and child components

An item type is composed of components: a root component and any number of child components, which are optional.

A *root component* is the first or only level of a hierarchical item type. An item type consists of both system- and user-defined attributes. Internally, the most basic item type contains only one component.

A *child component* is an optional second or lower level of a hierarchical item type. Each child component is directly associated with the level above it. Figure 10 on page 126 shows the diagram of the Content Manager meta-model. It shows root and child components and their relationships in forming an item hierarchy. The diagram also shows links, references, resource items, and resource objects.

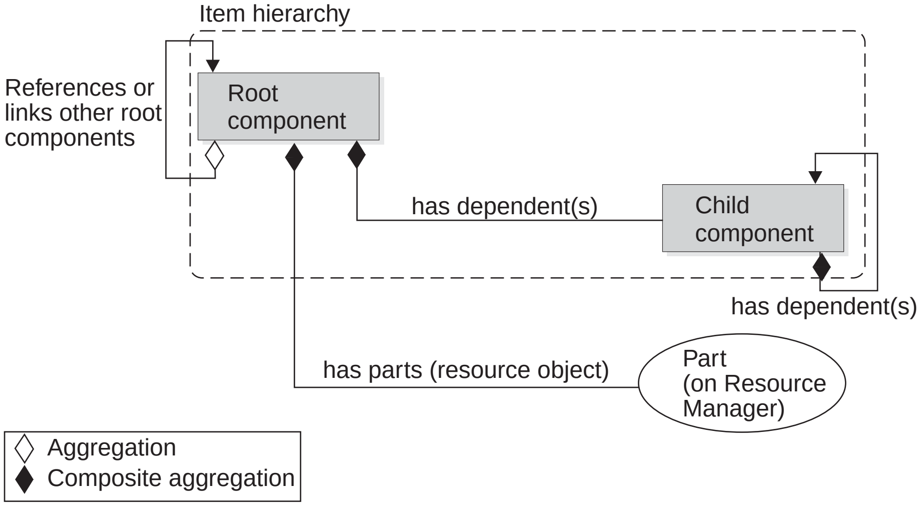


Figure 10. The Content Manager meta-model: A logical view.

Figure 11 shows an example of a data model for an insurance application with policy as the root component having claim, police report, and damage estimate as child components.

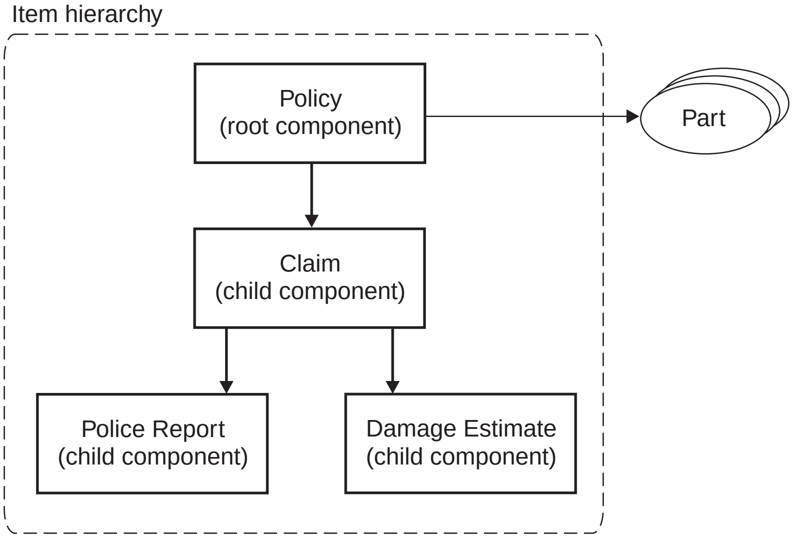

Item hierarchy
Policy (root component)
Part
Claim (child component)
Police Report (child component)
Damage Estimate (child component)

Figure 11. Components hierarchy

For more information about Content Manager data modeling concepts, the SItemTypeCreationICM sample.

Objects

An *object* (also known as resource content) is any digital content that a user can store, retrieve, and manipulate as a single unit (for example, JPEG images, MP3 audio, AVI video, and a text block from a book). An object is always stored in a resource manager. Access to an object is controlled by the library server.

For more information, see the `SResourceItemCreationICM` sample.

Links and references

You can use a *link* to model one to many associations between items. As shown in Figure 10, links can only be used to associate root components of items. Links do not refer to a particular version of an item.You can define any number of links. Links are a flexible way to link a source to a target. A link simply associates two items and provides the means to access the items it links or to other items that might link to those two items. Usage of links is determined at runtime by the user application. You can use any number of links in your application.

Links are open and non-restrictive, flexible building blocks that you can use in your applications. As such, you have the option of placing any further restrictions on links within your application.

One of the ways that you can use a link is to represent the foldering or container-containee relationship. If you choose to implement foldering using links, remember that the container does not *own* the containee, which means that the items in the container are not deleted when the container is deleted. For more information about links, see the `SLinksICM` sample.

A *reference* is between a component (either root or child) and another root component. A reference is represented as a reference attribute in a component defined at design-time. A component definition can have any number, specified during component type definition, of reference attributes that refer to other root components. A reference usually does not indicate ownership, but you can implement an ownership relationship if necessary.

When you add a reference to a component type, items of that component type can refer to another item. In terms of the DDO, the DDO has an attribute that is identified by the name of the reference. The attribute's value can be set to another DDO. The attribute value for the DDO is the DDO that is referred to by the reference.

References can be defined in both root and child component types referring to another root component type. See Figure 10 on page 126. References also refer to a specific version of an item, whereas links refer to all versions. For more information about reference attributes, see the `SReferenceAttrDefCreationICM` sample.

Documents

There are two types of documents that you might have in your system. The first type of document is an item of the semantic type ″document,″ which is expected to contain information that forms a document. This type of ″document″ can stand alone or contain parts if you have implemented the document model in your data model. For more information on this type of document, refer to the `SItemCreationICM` API Education Sample.

The other type of document is an item created from a ″document″ classified item type (also known as the ″document model″). This type of document contains document parts, a specific implementation of the Content Manager document model. The document parts can include various types of content, including text, images, and spreadsheets for example. For more information about the document model, see the `SDocModelItemICM` sample.

Folders

A *folder* is an item that may contain other items of any type. This may also include other folders. In Content Manager Version 8, the concept of folders is implemented by using link relationships between items. Items can contain other items to form a containment hierarchy, called a folder hierarchy. For example, a policy item belongs to the policy item type, and potentially has many claims, making policy a folder that holds other items such as a photo, a social security number, and so forth. Folders are very flexible because any item can be a folder and can contain any number of other items. For more information, see the `SFolderICM` sample.

Versioning

Versioning is the ability to store and maintain multiple versions of an item, including versions of the item's child components. You specify versioning rules when you define an item type. If an item type is enabled for versioning, all items in that item type will be versioned.

There are two types of versioning, always or by application. When an item is enabled for versioning always, a new version of the item is created automatically every time the item is updated and stored into the content server. When an item is enabled for versioning by application, the system only creates a new version when specified by the user application.

Versioning is handled by the Content Manager library server. Each version of an item, whose content is stored in the resource manager, will have its own copy of the RM content. Important characteristics of versioning support include the following:

- Versioning involves a root component and its entire hierarchy.
- Item types can have one of three possible versioning policies: versioned-always, versioned- never (the default) and application-controlled versioning.
- All the versions of an item in the CM system are searchable and retrievable.
- Any version of an item can be updated and deleted.
- For item types with application-controlled versioning, when the item is updated, the user has the option of applying the updates to the existing version or creating a new version based on the updates.
- Each version of an item has its own persistent identifier (PID). The PID has several parts of which two are relevant in the current context. The first relevant part is the ItemID which is the same across all different versions of the item. The other is the version number. Each version of the item has a different version number that can be retrieved and set as a string. Below is a sample that demonstrates how to work with version numbers.

  ```
  DKPidICM pid = (DKPidICM)ddo.getPidObject();
  String version = pid.getVersionNumber();
  ....
  pid.setVersionNumber(version);
  ```
- An item type can be configured to keep only a limited number of versions for each item. If an update to an item exceeds the maximum number of allowed, the oldest saved version is dropped and a new version is created by the system.
- If a version-enabled item is re-indexed, all previous versions of the item are automatically deleted.
- Child components of an item inherit the version of their parent component.
- The version of a child component type cannot be changed, since it follows the versioning of its parent type.

- Part-level versioning rules can be obtained from the item type relation object that represents the types.

For detailed information about versioning, see the `SItemUpdateICM` and the `SItemTypeCreationICM` sample.

Access control

The Content Manager access control model is comprised of the following fundamental elements:

- Privileges and privilege sets.
- Controlled entities.
- Users and user groups.
- Access control lists.

The various access control elements work as follows. Each Content Manager user is granted a set of user privileges. These privileges define the operation a user can perform. A user's effective access rights will never exceed the user's defined privileges.

The access control model of Content Manager is applied to the controlled entity. A *controlled entity* is a unit of protected user data. In Content Manager, the controlled entity can be at the level of item, item-type, or at the level of the entire library. For example, you can bind an ACL to an item type to enforce access control at the item type level. Operations on controlled entities are regulated by one or more control rules, called access control lists (ACLs). Every controlled entity in Content Manager system must be bound to an ACL.

When a user initiates an operation on an item, the system checks the user's privilege and the ACL bound to the item to determine if the user has the right to do such an operation on the item. Logically, the right to access an item also requires the right to access the item type, where the item is defined. Figure 12 shows an example of how the system determines user's access rights to an item based on privileges and ACLs.

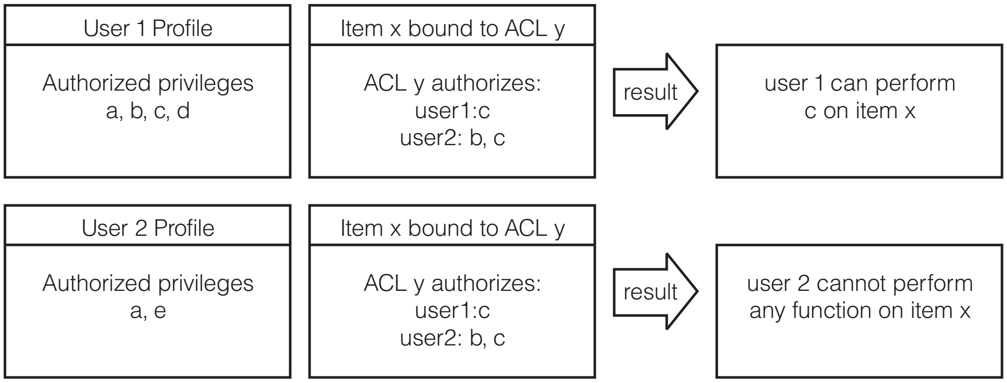


Figure 12. Access control diagram

Privileges and privilege sets

Privileges allow a user to perform a specific action on an item in the system, such as create or delete it. Every Content Manager user is granted a set of user privileges. The privileges define the maximum operations a user can perform on information in the Content Manager system. A user's access rights do not exceed the defined user privileges for the user.

Content Manager provides a number of pre-defined privileges that you cannot change, called system-defined privileges. You can also define your own privileges, called user-defined privileges. You enforce user-defined privileges in your application using user exit routines.

Every privilege has a system-generated, unique code called a privilege definition code. The privilege definition codes 0 to 999 are reserved for system-defined privileges. You can use codes of 1000 and above for user-defined privileges.

The system-defined privileges are classified into two categories: system administration privileges, and data access privileges. You can use the system administration privileges to model user data and administer and maintain the Content Manager system. You need system administration privileges to complete tasks such as configuring the system, managing the library server configuration, and managing item types. You can use the data access privileges to access and change the system data, like items and item types.

A group of privileges assigned to a user is a *privilege set*. For example, one privilege set can contain the privileges create, update, and delete. Privilege sets allow for easier system administration. You must group privileges into a set before you can use them. There is no limitation on the number of privileges a set can contain.

The Content Manager pre-defined privilege sets include: System Admin privilege

AllPrivSet; PrivSetCode: 1

A user with this privilege set can perform all functions on all Content Manager entities. The privileges contained in this privilege set include: All system-defined and user-defined privileges.

NoPrivSet; PrivSetCode: 2

Users with this privilege set cannot perform any functions on any Content Manager entities. The privileges contained in this privilege set include: None.

SystemAdminPrivSet; PrivSetCode: 3

Users with this privilege set can perform all Content Manager system administration and data modeling functions. The privileges contained in this privilege set include:

ItemAdminPrivSet; PrivSetCode: 4

Users with this privilege set can perform all Content Manager data modeling and item access functions. The privileges contained in this privilege set include:

- System define item type privilege
- Item SQL select privilege
- Item type query privilege
- Item query privilege
- Item add privilege
- Item set user defined attr privilege
- Item set system defined attr privilege
- Item delete privilege
- Item move privilege
- Item link to privilege
- Item linked privilege

- Item own privilege
- Item owned privilege
- Item add link privilege
- Item change link privilege
- Item remove link privilege
- Item check out privilege

Table 8. Privilege Codes

Privilege name	Code
ICMLogon	1
SystemAdmin	40
SystemDefineItemType	45
ItemSQLSelect	121
ItemTypeQuery	122
ItemQuery	123
ItemAdd	124
ItemSetUserAttr	125
ItemSetSysAttr	126

Users and user groups

Most likely, you have a group of users that require the same type of access to the system. For example, all of the underwriters in an insurance company require search, retrieve, and update privileges to the claims item type. You can group the underwriters and any other users with common access needs into a user group. You cannot, however, put one user group into another user group.

A user group is solely a convenience grouping of individual users who perform similar tasks. A user group consists of zero or more users. You do not assign a user group a privilege set. Each user in a user group has a privilege set. A user group makes it easier to create access control lists for objects in your system. A user group cannot belong to other groups.

Access control lists (ACLs)

When a user creates an item in the Content Manager system, that user must define the access that other users will have to that item, and what operations they can perform on that item. The list of users that have access to the item and the operations that they can perform on the item is called an *access control list* (ACL). An ACL can contain one or more individual user IDs or user groups and their associated privileges. You can associate items, item types, and worklists with an ACL. Privilege sets define an individual's maximum ability to use the system, an ACL restricts that individual's access to an item. For example, if its ACL allows the photograph item to be deleted but John doesn't have the delete privilege in his privilege set, then John cannot delete the photograph.

A controlled entity is bound to a specific ACL through the ACL code. When associated with controlled entities, ACLs define the authorization of the bound entities and do not circumvent the user privileges. An ACL is enforced, and user privileges are checked.

The users specified in access control rules can be individual users, user groups, or public. The interpretation is determined by the UserKind field of a rule. The types of rules, for illustration purposes, can be given the names ACL Rule for User, ACL Rule for Group, and ACL Rule for Public respectively. By specifying public, the ACL Rule for Public authorizes all the users to perform operations specified in the ACL Privileges on the bound entity, provided the users pass their User Privileges check. The ACL privileges on the bound entity to Public can be configured in the System level. The capability of opening a bound entity to Public can be configured system-wide. The configuration parameter is named `PubAccessEnabled` (defined in table `ICMSTSysControl`). When disabled, all the ACL Rules for Public are ignored during the access control process.

Within the same ACL, a user can be specified in more than one type of rule. The precedence of the three types, from highest to lowest, is ACL Rule for Public, ACL Rule for User, and ACL Rule for Group. When applying ACL checks, if any higher-precedence rule type passes, the authorization is resolved and the process stops. If the check for ACL Rule for Public failed, the checking process will continue on the lower-precedence rule types.

If the check for ACL Rule for the User failed, however, the checking stops. The ACL Rule for Group is not checked. There is no need to continue the check on the Group type because if a user does an individual user check, the user will be excluded from the group type access based on the access control algorithm. The access control check for individual User type and Group type is not a sequential process. It is an either-or situation, even though there is no harm in doing a sequential check.

If the user has failed to pass an individual user type check (or the user does not have a rule in the Access List table), the checking process will continue to the group type. If the user belongs to one of the groups and the check of the privilege passes, the authorization is resolved and the process stops. Otherwise, access is denied and the process also stops. When a user is specified in more than one ACL Rule for a Group, the user is authorized by the union of all those rules' ACL Privileges. A user is never specified in more than one ACL Rule for User.

The CM system provides the following pre-configured ACLs: `SuperUserACL`, `NoAccessACL` and `PublicReadACL`.

SuperUserACL
This ACL consists of a single rule that authorizes the CM pre-configured user ICMADMIN to perform all CM functions (AllPrivSet) on the bound entities.

NoAccessACL
This ACL consists of a single rule that specifies, for all CM users (public), no actions (NoPrivSet) is allowed.

PublicReadACL
This ACL consists of a single rule that specifies, for all CM users (ICMPUBLC), the read capability (ItemReadPrivSet) is allowed. This is the default value assigned to a user's DfltACLCode.

Planning a Content Manager application

This section helps you identify requirements for creating a Content Manager application and provides information about how Content Manager operates. A key part of planning your application is creating a data model that meets the needs of your business. For more information about the Content Manager data model, see

Planning and Installing Your Content Management System and the `tSItemTypeCreationICM` sample in the samples directory.

The following topics are covered in this section:

- Determining the features of your application.
- Handling errors.

Determining the features of your application

The approach you take to develop your application varies based on the needs of your organization. To produce an effective application, all interested parties in your organization should contribute to the planning and design of the application. For additional help with planning, see *Planning and Installing Your Content Management System*.

Before you can create your application, you should be able to answer all or most of the following questions:

- What types of documents does your organization use?
- What type of content is in your existing documents?
- How do you process documents?
- Can you automate your document process?
- How do you receive, display, store, and distribute documents?
- How often do you retrieve documents after they are stored?
- What is the volume of documents that your organization manages?
- What types of storage media do you want to use to store your large objects?
- Are there other applications your organization uses?
- How many users and what type of access control do you plan to have?

Use the answers to the questions above to help you determine which features to include in your application.

Handling errors

When handling errors, the most important exception to catch is the DKException class. Do not use exceptions for program logic, and do not rely on catching exceptions to detect if something exists in the content server or for any reason other than for truly exceptional cases. Using exceptions in program logic decreases performance and can render tracing and log information useless for debugging and support.

Carefully review all of the exception information. There are numerous sub-classes of DKException and depending on the program, it might be best to handle each exception individually. Table 9 contains DKException information.

Table 9. DKException information

DKException	Description
Name	Exception Class Name. Contains sub-class name.
Message	A specific message explains the error. The message can contain a lot of information, sometimes encapsulating important variable states at the time the error was detected.
Message ID	A unique Message ID identifies this error type and matches it to a core message used above.

Table 9. DKException information (continued)

DKException	Description
Error State	Might contain additional error information about the state of the OO API or library server error. If the library server detects an error, the following four pieces of information are packaged here: Return code Reason code Ext / SQL return code Ext / SQL reason code
Error Code	Might contain the library server return code.
Stack Trace	Important information indicating the failure point in the user program and exactly where the error was last detected or handled by the OOAPI.

When working in Java, you must also handle the java.lang.Exception. The `SConnectDisconnectICM` sample in the `samples` directory demonstrates how to catch and print errors. For information about logging and tracing, see *Messages and Codes*.

Working with the Content Manager samples

Content Manager provides a comprehensive set of code samples to help you complete key Content Manager tasks. The samples are a great source of API education because they provide reference information, programming guidance, API usage examples, and tools.

You can view the samples in the *Online application Programming Reference*, in the product Information Center. Additionally, the samples are located in the `CMBROOT\Samples\java\icm` and `CMBROOT\Samples\cpp\icm` directories. Note, however, that you must have selected the Samples and Tools component during EIP installation in order to have the samples in the directory.

To get the most out of the samples, be sure to read the Samples Readme. It contains a complete reference index to help you quickly find the sample that contains the concept, or topic, that you are looking for. Every sample is thoroughly documented and provides in-depth conceptual information and an explanation of each task step. Additional information contained in each sample includes:

- Detailed header information explaining the concepts shown in the sample.
- A description of the sample file including prerequisite information and command line usage.
- Fully commented code that you can easily cut, customize, and use in your applications.
- Utility function that you can use when developing your applications.

The Getting Started section in the Samples Readme helps you to quickly learn how to complete the following general tasks:

- Data modeling.
- Connecting to a server and handling errors.
- Defining attributes and attribute groups.
- Working with reference attributes.
- Defining your data model.
- Working with items.

- Working with resource items.
- Working with folders.
- Working with links.
- Defining the resource manager.
- Defining an SMS collection.
- Searching for items.

The insurance scenario sample

Content Manager provides code samples for one possible ″real world″ implementation using an insurance company. The information used to create the insurance company sample is fabricated and created only to help explain key Content Manager features. For a complete list of the samples that make up the insurance scenario, see the Samples Readme.

Creating a Content Manager application

The APIs that implement Content Manager Version 8 Release 2 functionality are grouped into what is called the ICM connector. The ICM connector APIs have an ICM suffix, as in the example `DKDatastoreICM`.

This information in this section includes:

- Understanding the software components.
- Representing items using DDOs.
- Connecting to the Content Manager system.
- Working with items.

Understanding the software components

For conceptual purposes, you can categorize the OO APIs into the following groups of services:

- Data and document modeling.
- Search and retrieve.
- Data import and delivery.
- System management.
- Document routing.

The data and document modeling module contains the APIs that enable you to map your business data model to the underlying Content Manager hierarchical data model. For example, an insurance company’s data model includes *policies*, which in the Content Manager data model are essentially items. The data and document modeling module APIs provide interfaces to define items that represent *policies*.

The search and retrieve module processes requests about managed items like documents and folders. The search module APIs enable you to perform combined text and parametric searches for items contained in the Content Manager system. The search results are returned to the application in the form of search result sets.

The data import and delivery module provides the APIs that enable you to import data into your system and deliver that data through various media, like a network or the Web.

The system management module provides you with the interfaces to configure and maintain an efficient, secure Content Manager system. For example, you can incorporate the system management APIs into your application to allow you to adjust the system control settings, manage users, assign users privileges, allow access to the system, and so forth.

The document routing module APIs help you to route business objects, like documents, through a process, as defined by the needs of your business.

Representing items using DDOs

Before you can create an application, you must understand the DDO/XDO protocol concepts explained in the "Understanding dynamic data object concepts" on page 12. The information in this section is specific to Content Manager Version 8 Release 2.

A DDO is essentially a container of attributes. An attribute has a name, value, and several properties. One of the most important properties of attributes is the attribute type. A DDO has a persistent identifier (PID) to indicate the location where the object resides in persistent storage. A DDO has some methods to populate itself, and corresponding methods to retrieve an item's information. The DDO methods include add, retrieve, update, and delete. You use these methods to move an item's data in and out of persistent storage, like Content Manager.

In memory, Content Manager items are represented as DDOs. Item attributes are represented as DDO attributes with a name, type, and a value. Links and references are represented as special types of attributes. The difference between a link attribute and a reference attribute, however, is that a reference attribute refers to another (single) DDO or XDO, and a link attribute refers to a collection (multiple) of DDOs or XDOs. XDOs are used to represent large objects (LOBs).

A reference to an item, either to an XDO or another DDO, has a name with the type property set to object reference, and value set to refer to the instance of the referenced object. Child components and links are also represented as DDO attributes with the type property set to a collection of data objects, and value set to a collection of DDOs. In the case of a child component, the attribute name is the name of the child component. The value is the collection of child components belonging to the root component. If the root item is deleted, all of the child components of the root item are also deleted.

Connecting to the Content Manager system

One of the first things that you need to do when you build a Content Manager application is connect to the server. This section helps you with the various tasks involved in connecting to, and disconnecting from, a Content Manager server.

To access the Content Manager server, your application needs to create a content server, which acts as a common server. To create and connect to a content server:

1. Create a content server object.

Java

```
DKDatastoreICM dsICM =new DKDatastoreICM();
```

C++

```
DKDatastoreICM *dsICM =new DKDatastoreICM();
```

2. Set up the connection parameters.

Java

```
String database = "icmnlsdb";
String userName = "icmadmin";
 String password = "password";
```

C++

```
char * database = "icmnlsdb";
char * userName = "icmadmin";
char * password = "password";
```

3. Call the connect operation on the content server. `databaseNameStr` is the name of the database you want to connect to.

Java

```
dsICM.connect(databaseNameStr,usridStr,pwStr,"");
```

C++

```
dsICM->connect(database, userName, password, "");
```

Depending on your system configuration, you might have several library servers and resource managers that you can connect to. To see a list of the names of the library servers that you can connect to, use `DKDatastoreICM` and call the `listDataSourceNames()` method, and then the `listDataSources()` method. The `listDataSources()` method lists the library servers that are currently available to connect to.

After you connect to a library server, use the `DKRMConfiguration` and call the `listResourceMgrs()`method, to get the list of resource managers associated with that library server.

To disconnect from the system, call the `disconnect` operation in the content server.

For more information, see `SConnectDisconnectICM`, the complete sample from which the above code snippets were extract.

Changing a password

You can allow users to change their password each time they begin a new library server session. To implement the change password option, use `DKDatastoreICM` and call the `changePassword()` method.

Java

```
changePassword(String userID, String oldPwd, String newPwd)
```

C++

```
changePassword(const char* userID, const char* oldPwd, const char* newPwd)
```

Working with items

This section describes the processes involved in creating, updating, and deleting items.

For additional information about working with items see the `SItemCreationICM` sample.

Creating an item type

Before you can create any items, you must create item types. You must define an item type for every item you create. For example, a `claim` item is the child component of the item type `policy.`

When you create an item type, you can define a classification for the item type. An *item type classification* is a categorization within an item type that further identifies the items of that item type. All items of the same item type have the same item type classification. Content Manager supplies the following item type classifications: item, resourceitem, docmodel, and docpart. If you do not specify an item type classification when you create an item type, the item type classification defaults to item (DDO). The pre-defined item type classifications and the corresponding ID constant are listed in Table 10.

Table 10. Item type classifications

Classification	ID Constant	Number	Description
Item	DK_ICM_ITEMTYPE_CLASS_ITEM	0	A standard item (DDO).
Resource	DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM	1	A resource item that describes and contains data that is stored in the resource manager. Video, images, documents, and other data archived in the resource manager are examples of resource items.
Document model	DK_ICM_ITEMTYPE_CLASS_DOC_MODEL	2	An item that models documents using parts. A document is composed of any number of parts, which are contained in the attribute `DKConstant.DK_CM_DKPARTS`.
Part	DK_ICM_ITEMTYPE_CLASS_DOC_PART	3	Items (parts) in the document model classification.

Note: Constants are located in `com\ibm\mm\sdk\common\DKConstantICM.java` for Java and `dk\icm\DKConstantICM.h` for C++.

To create an item type (see code example below):

1. Create an item type and pass it a reference to the content server.
2. Give the item type a name.
3. Add attributes to the item type, and set any desired qualifiers, like `nullable`, `textsearchable`, and `unique`.
4. Add the item type to the persistent content server.

Java

```
//This example creates an item type definition and names it.
//The item type name must be less than 15 characters in length.
DKItemTypeDefICM bookItemType = new DKItemTypeDefICM(dsICM);
bookItemType.setName("book");
bookItemType.setDescription("This is an example item type name.");
//Below, a text-searchable attribute called book title is added. The
//attribute is defined to require a unique name and also a value. The value
//does not have to be unique.
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_title");
attr.setTextSearchable(true);
attr.setUnique(true);
attr.setNullable(false);
bookItemType.add(attr);
//Here, a book_num_pages attribute is added to the book item
//type with the following characteristics: text searchable, unique,
//and a value is not required.
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();
//create new ItemType
DKItemTypeDefICM * bookItemType = new DKItemTypeDefICM(pDs);
bookItemType->setName("book");
bookItemType->setDescription("This is an example item type name.");
//Create new Attribute; add it to datastore and to the ItemType
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);
//Persist the attribute to the datastore
attr->add();
//Add the newly created attribute to the item type.
bookItemType->addAttr(attr);
//Create new Attribute; add it to datastore and to ItemType
  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);
//Add the entity, bookItemType, to the datastore.
bookItemType->add();
```

See the `SitemTypeCreationICM` sample for more information.

Listing item types

To get a list of available, defined item types:

1. Connect to a `DKDatastoreICM` content server.
2. Get a reference to the content server definition.
3. Call the `listEntityNames` method on the content server definition object to get a string array of the names of the item types.
4. Use a loop to list all of the names.

Java

```
String itemTypeName[] = 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++ Example 1

```
long larraySize = 0;
    DKString * itemTypeName = dsICM->listEntityNames(larraySize);
    for (int i = 0; i < larraySize; i++) {
           cout <<(char*)itemTypeName[i] <<endl;
    }
    delete [] itemTypeName;
```

C++ Example 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);
```

For more information, see the `SItemTypeRetrievalICM` sample.

Creating attributes

To create attributes:

1. Create an attribute definition object.
2. Describe the object that you create by setting its name, description, type, size, and so forth. Examples of types of attributes that you can create include:
   - 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. Add the new definition to the persistent content server.

Java

```
//This example defines an attribute for the title of a book.
attr = new DKAttrDefICM(dsICM);
attr.setName("book_title");
attr.setDescription("The title of the book.");
attr.setType(DKConstant.DK_CM_VARCHAR);
attr.setSize(100);
attr.add();
//This example defines an attribute for the number of pages in a book.
attr = new DKAttrDefICM(dsICM);
attr.setName("book_num_pages");
attr.setDescription("The number of pages in the book.");
attr.setType(DKConstant.DK_CM_INTEGER;
attr.setMin((short) 0);
attr.setMax((short) 100000);
attr.add();
```

C++

```
//This example defines an attribute for the title of a book.
DKDatastoreICM * dsICM; ..........
  DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
       attr->setName("book_title");
       attr->setDescription("The title of the book.");
       attr->setType(DK_CM_VARCHAR);
       attr->setSize((long) 100);
       attr->add();
//This example defines an attribute for the number of pages in a book.
DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
       attr->setName("book_num_pages");
       attr->setDescription("The number of pages in the book.");
       attr->setType(DK_CM_INTEGER);
       attr->setMin((long) 0);
       attr->setMax((long) 100000);
       attr->add();
```

For more information about creating attributes, see the `SAttributeDefinitionCreationICM` sample.

Creating, updating, and deleting attribute groups

Attribute groups make it easier for you to add entire groups of attributes to items and sub-components. An attribute can belong to any number of attribute groups, from zero to any number. An attribute can be added to multiple attribute groups. A data item (DDO) can contain multiple attribute groups. The same attribute name can appear in the DDO, but each attribute is completely separate, based on the namespace. When an attribute is added to an attribute group, it impacts only the component types that are created after the addition. Pre-existing component types remain unchanged. The following example demonstrates how to create an attribute group:

Java

```
// Create a datastore definition object given the connected datastore
DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM) dsICM.datastoreDef();
//Creating a new attribute group
DKAttrGroupDefICM attrGroup = new DKAttrGroupDefICM(dsICM);
// Set a name, maximum 15 characters
attrGroup.setName("Book_Details");
//Set a description for the new attribute group
attrGroup.setDescription("Detailed book information");
// Retrieve the definition of an attribute that will be
//added to this attribute group.
DKAttrDefICM title = (DKAttrDefICM) dsDefICM.retrieveAttr("book_title");
// Retrieve the definition of another attribute that will be added
//to this attribute group.
DKAttrDefICM publisher = (DKAttrDefICM) dsDefICM.retrieveAttr("publisher");
// Add the Attribute Definitions to the Attribute Group
attrGroup.addAttr(title);
attrGroup.addAttr(publisher);
// add it to the persistent datastore
attrGroup.add();
```

C++

```
// Create a datastore definition object given the connected datastore
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
//Creating a new attribute group
DKAttrGroupDefICM* attrGroup = new DKAttrGroupDefICM(dsICM);
// Set a name, maximum 15 characters
 attrGroup->setName("Book_Details");
//Set a description for the new attribute group
attrGroup->setDescription("Detailed book information");
// Retrieve the definition of an attribute that will be
//added to this attribute group.
DKAttrDefICM* title =
  (DKAttrDefICM *) dsDefICM->retrieveAttr("book_title");
// Retrieve the definition of another attribute that will be added
//to this attribute group.
DKAttrDefICM* publisher =
  (DKAttrDefICM *) dsDefICM->retrieveAttr("publisher");
// Add the Attribute Definitions to the Attribute Group
attrGroup->addAttr(title);
attrGroup->addAttr(publisher);
// add it to the persistent datastore
attrGroup->add();

delete(attrGroup);
```

To update the name and description of an attribute group, call the update() method in DKAttrGroupDefICM and provide an array of new attribute IDs. If this attribute group has already been associated with a component type, then you can't update this attribute group.

To delete an attribute group you also work with the DKAttrGroupDefICM class. When you delete the attribute group, the primary attributes that used to make up the attribute group remain in library server. The following exceptions apply when you delete an attribute group:

- An attribute group cannot be deleted if it is associated with a component type and is persistent.

- An attribute can not be removed from an attribute group if the attribute group is associated with a component type and is persistent.
- You cannot add an attribute to an attribute group if the attribute group is associated with a component type and is persistent.

For more information, see the SAttributeGroupDefCreationICM sample.

Listing the attributes in a content server

The following example demonstrates how to get a list of attributes in a content server.

Java

```
DKDatastoreICM dsICM;
DKDatastoreDefICM dsDefICM=DKDatastoreDefICM)dsICM.datastoreDef();
//Get a collection containingall Attribute Definitions.
DKSequentialCollection attrDefColl =
  (DKSequentialCollection)dsDefICM.listAttrs();
if ((attrDefColl!=null) &&(attrDefColl.cardinality()>0)){
  //Create an iterator to iterate through the collection
  dkIterator iter = attrDefColl.createIterator();
  while(iter.more()){
    //while there are still items in the list,continue
    dkAttrDef attrDef = (dkAttrDef) iter.next();
    System.out.println("-"+attr.getName()+":"+attr.getDescription());
  }
}
```

C++

```
DKDatastoreICM * dsICM; .........
  DKDatastoreDefICM*dsDefICM =(DKDatastoreDefICM *)dsICM->datastoreDef();
//Get a collection containingall Attribute Definitions.
DKSequentialCollection*attrDefColl =
        (DKSequentialCollection *)dsDefICM->listAttrs();
if (attrDefColl &&(attrDefColl->cardinality()>0)){
   //Create an iterator to iterate through the collection
   dkIterator*iter =attrDefColl->createIterator();
   while(iter->more()){
     //while there are still items in the list,continue
    dkAttrDef* attrDef =(dkAttrDef *)iter->next()->value();
    cout <<"-"<<attrDef->getName()<<":"<<attrDef->getDescription()<<endl;
    delete(attrDef);
  }
  delete(iter);
  delete(attrDefColl);
}
delete(dsDefICM);
```

Listing attribute names for an item type

The following example demonstrates how to get a list of attribute names for an item type. For more information, refer to the SItemTypeRetrievalICM API Education Sample.

Java

```
//Get a collection containing all attr. defs for the Item Type.
DKSequentialCollection attrColl = (DKSequentialCollection)
    dsICM.listEntityAttrs(itemTypeName);
//Accessing attribute each and printing the name and description.
System.out.println("\nAttributes of Item Type '"+itemTypeName+"':
    ("+attrColl.cardinality()+')');
//Create an iterator to iterate through the collection
dkIterator iter = attrColl.createIterator();
while(iter.more()) {
    //while there are still items in the list, continue
    DKAttrDefICM attr = (DKAttrDefICM)iter.next();
    System.out.println("-"+attr.getName()+":"+attr.getDescription());
}
```

C++

```
//Get a collection containingall Attribute Definitions for the Item Type.
DKSequentialCollection*attrColl =(DKSequentialCollection*)
   dsICM->listEntityAttrs(itemTypeName);
//Accessing attribute each and printing the name and description.
cout <<"\nAttributes of Item Type '"<<itemTypeName <<"':
   ("<<attrColl->cardinality()<<')'<<
//Create an iterator to iterate through the collection
dkIterator* iter =attrColl->createIterator();
while(iter->more()) {
   //while there are still items in the list, continue
DKAttrDefICM* attr =(DKAttrDefICM*)iter->next()->value();
cout <<"-"<<attr->getName()<<":"<<attr->getDescription()<<endl;
delete(attr);
}
delete(iter);
delete(attrColl);
```

Creating an item

An item is the entire tree of component DDOs, with at least one root component. That root component DDO must be assigned an Item Property Type or Semantic Type. When you create an item, you must assign it an item type property. The valid itemtype properties are document, folder, or item. You must specify the item type property as the second parameter of the content server's `createDDO` function. The value for the property type is stored in the DDO's property named, DK_CM_PROPERTY_ITEM_TYPE. Do not confuse this item type property with the overall item type definition that describes the structure of the item. Table 11 shows a list of available item property types.

Table 11. Item type property definitions

Property type	Constant	Definition
Document	DK_CM_DOCUMENT	Item represents a document or stored data. The information contained in this item might form a document. This item can be considered a common document since it does not rigidly mean an implementation of a specific document model.

Table 11. Item type property definitions (continued)

Property type	Constant	Definition
Folder	DK_CM_FOLDER	Item represents an object containing or referencing contents or objects. This item can be considered a common folder since it does not rigidly mean an implementation of a specific document model.
Item (default)	DK_CM_ITEM	Generic item. This item does not fit system defined or user defined semantic types.

Items are created as DKDDOs. Always use the DKDatastoreICM's createDDO() methods to create DKDDOs because the system uses the DKDatastoreICM's createDDO methods to automatically setup important information in the DKDDO structure.

When you create an item, you define the components that make up the item. For example, if you choose to create a hierarchical item, you must create child components.

1. Using the content server's `createDDO` and `createChildDDO` methods, create an item DDO and set all of its attributes and other required information. This example uses the logged on, connected `DKDatastoreICM` object named `dsICM`.
2. Create a root component.

Java Example 1

```
DKDDO myDocumentDDO = dsICM.createDDO("EmployeeDoc",
   DKConstantICM.DK_CM_DOCUMENT);
```

Java Example 2

```
DKDDO myFolderDDO   = dsICM.createDDO("DeptFolder",
   DKConstantICM.DK_CM_FOLDER);
```

C++ Example 1

```
DKDDO* myDocumentDDO = dsICM->createDDO("EmployeeDoc",DK_CM_DOCUMENT);
```

C++ Example 2

```
DKDDO* myFolderDDO = dsICM->createDDO("DeptFolder",DK_CM_FOLDER);
```

3. Create a child component under the root component ″EmployeeDoc″, for example, ″Dependent″.

Java

```
DKDDO myDDO = dsICM.createChildDDO("EmployeeDoc","Dependent");
```

C++

```
DKDDO* myDDO = dsICM.createChildDDO("EmployeeDoc","Dependent");
```

4. Add the child component to the parent.

Java

```
short dataid = myDocumentDDO.dataId(DK_CM_NAMESPACE_CHILD,"Dependent");
DKChildCollection children =
  (DKChildCollection) myDocumentDDO.getData(dataid);
children.addElement(myDDO);
```

C++

```
short dataid = myDocumentDDO->dataId(DK_CM_NAMESPACE_CHILD,"Dependent");
DKChildCollection* children =
  (DKChildCollection*)(dkCollection*)
myDocumentDDO->getData(dataid);
children->addElement(myDDO);
```

5. Use the `setData` method to populate the DDO with the appropriate values.

Java

```
myDDO.setData(.....);
```

C++

```
myDDO->setData(.....);
```

6. Save the item into the persistent store.

Java

```
myDocumentDDO.add();
```

C++

```
myDocumentDDO->add();
```

In the preceding example, the last step created a document in the content server. When a document DDO is added to a content server, all of its attributes are added, including all of the parts inside the DKParts collection. When a document DDO is added to a content server, all of its attributes are added, including all children, and all parts inside the DKParts collection.

Semantic type defines the usage or rules for an item. Content Manager comes with some pre-defined semantic types, but you can also define your own semantic types.

You can specify the semantic type as the second parameter of the content server's createDDO function. The semantic type value is stored in the DDO's property DKConstantICM.DK_ICM_PROPERTY_SEMANTIC_TYPE, which can contain the same value as the item property type. The ICM connector supports folder and document semantic types. You can also create your own semantic types. Table 12 lists the available semantic types.

Table 12. Pre-defined semantic types

Semantic type	Constant	Definition
Document	DK_ICM_SEMANTIC_TYPE _DOCUMENT	Indicates that this item is a document
Folder	DK_ICM_SEMANTIC_TYPE _FOLDER	Indicates that this item is a folder
Annotation	DK_ICM_SEMANTIC_TYPE _ANNOTATION	Indicates that this item or part is an annotation to the base part
Container	DK_ICM_SEMANTIC_TYPE _CONTAINER	Indicates that this item is a container, which can contains other items. The container-containee relationship is represented using links.
History	DK_ICM_SEMANTIC_TYPE _HISTORY	Indicates that this item or part is a history of the base part.
Note	DK_ICM_SEMANTIC_TYPE _NOTE	Indicates that this item or part is a note to the base part.
Base	DK_ICM_SEMANTIC_TYPE _BASE	Indicates that this item or part is the base part that may have an annotation, note, or history associated with it.
User defined	User defined	A user defined semantic type interpreted by the application.

Notes:

- In Java, the constants are defined in DKConstantICM.java.
- In C++, the constants are defined in DKConstantICM.h.
- In Content Manager Version 7, semantic type is called affiliated type.

If you create an item in an item type that has been defined as a resource item type, the correct XDO is returned. For more information about resources, see the `SResourceItemCreationICM` sample. You might also find it useful to review the information about item creation in the `SItemCreationICM` sample.

Setting and retrieving item attribute values

Java

Attribute values are stored and retrieved as java.lang.Objects. To set or retrieve attribute values, use the DDO's setData and the getData() methods respectively. Set the value using an object with the type corresponds to the attribute type. **Example:**

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
     nameOfAttrStr),valueObj);
Object obj = ddo.getData(ddo.dataId(DK_CM_NAMESPACE_ATTR,nameOfAttrStr));
```

The above statement sets the attribute value to the value passed in as a java.lang.Object valueObj. The nameOfAttrStr is the string name of the attribute.This attribute was defined and specified in the item type when the item type of this DDO was defined. `valueObj` is the value you must set and it must be of the right type for this attribute.

C++

When setting values for individual attributes, use the individual attribute definition name. In order to access attributes that belong to an attribute group, use this format: `<Attribute Group Name>.<Attribute Name>`. **Example:**

```
//The code snippet below shows the value of the character attribute
//"book_title" for the item represented by the DDO ddoDocument is set to
//the value "Effective C++"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
                    DKString("book_title")),DKString("Effective C++"));
//The code snippet below shows the value of the integer attribute
//"book_num_pages" for the item represented
//by the DDO ddoDocument is set to the value "250"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
                      DKString("book_num_pages")),(long)250);
//This code snippet shows how the value of the "book_title" attribute of the
//item represented by the DDO ddoDocument is retrieved into the "title"
//string variable.
DKString title =(DKString) ddoDocument->getData(ddoDocument->
   dataId(DK_CM_NAMESPACE_ATTR,DKString("book_title")));
```

Modifying an item's attributes

To modify an item's attributes, use the DDO's `setData` method and set it to the required value by specifying that value using the java.lang.Object, as shown in the example below. In the example below, `nameOfAttrStr` is the string name of the attribute and `valueObj` is the value.

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++"));
```

Updating an item

To update an item:

1. Retrieve an item or create an item and add it to the content server.
2. Modify the item, its attributes, link collections, and so forth.
3. Call the DDO's `update` operation.

Java

```
ddo.update();
```

C++

```
ddo->update();
```

If an item is enabled for versioning, you can create a new version of the item instead of updating the current item, as shown in the following example:

Java

```
ddo.update(DKConstant.DK_CM_VERSION_NEW);
```

C++

```
ddo->update(DK_CM_VERSION_NEW);
```

To update the latest version of an item, use the format shown in the following example:

Java

```
ddo.update(DKConstant.DK_CM_VERSION_LATEST);
```

C++

```
ddo->update(DK_CM_VERSION_LATEST);
```

You can also update a DDO by calling the `updateObject` method on DKDatastoreICM with the appropriate options, as shown in the example below:

Java

```
// Connected DKDatastoreICM object named ds;
// DKDDO object named 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);
```

Important: You must check out the item before calling the update method and check the item back in when your are done updating it. See "Checking in and checking out items" on page 157. For more information on updating items, refer to the `SItemUpdateICM` API Education Sample.

Defining a resource item type

A resource item is an item with additional system defined attributes that define the location, type, size, and so forth, of the object that the item represents. The object is sometimes called a ″resource″ and can be a video file, an image, a word processor document, and so forth. For additional information, see the `SItemTypeCreationICM` sample (in the samples directory).

The steps below take you through the process of defining a resource item type.

1. Get the content server definition object from the connected content server.

   Java

   ```
   DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM)dsICM.datastoreDef();
   ```

   C++

   ```
   DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
   ```

2. Create a new item type definition.

   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("Simple Resource Lob Item Type");
   ```

3. Add an attribute.

   Java

   ```
   DKAttrDefICM attr =(DKAttrDefICM)dsDefICM.retrieveAttr("S_varchar");
   itemType.addAttr(attr);
   //Resource classification indicates that this class will
   //contain data file
   itemType.setClassification
     (DKConstantICM.DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
   ```

C++

```
DKAttrDefICM * attr = (DKAttrDefICM*) dsDefICM->retrieveAttr("S_varchar");
itemType->addAttr(attr);
// Resource classification indicates that this class will contain
//data file
itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
```

4. Specify the XDO class and type of resource for this item type.

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();
```

Creating a resource item

Creating resource items is very similar to creating regular items. XDOs extend DDOs, and depending on the type of resource item, the XDO can be extended further. Table 13 contains the resource item types and the class hierarchy used to create them. For more information, see the SResourceItemCreationICM sample (in the samples directory).

Table 13. Resource item type class hierarchy

Type	DDO	XDO	Extension
LOB	DKDDO ->	DKLobICM	
Text	DKDDO ->	DKLobICM ->	DKTextICM
Image	DKDDO ->	DKLobICM ->	DKImageICM
Stream	DKDDO ->	DKLobICM ->	DKStreamICM

The steps below take you through the process of creating resource item. There are multiple ways to set and store resource content. The following process is an example for storing directly from file at the time the item is added to the content server.

1. Create the resource item. Note that it can be any semantic type and that the DKDatastoreICM::createDDO call is also used to create a resource item in the same way that it is used to create a regular item. The of type resource returned from DKDatastoreICM.createDDO is cast to the correct sub-class (in this case DKLobICM) based on the XDO classification defined during the creation of the item type on which this resource item is based.

Java

```
DKLobICM    lob = (DKLobICM)dsICM.createDDO
           ("SampleResource",DKConstant.DK_CM_DOCUMENT);
```

C++

```
DKLobICM*     lob = (DKLobICM*)
   dsICM->createDDO("SampleResource",DK_CM_DOCUMENT);
```

2. Set the content or data into the object. Once the lob is created, you can set the MIME type for the resource. In this case, the resource is a MS Word document. The MIME type describes the type of content that is being stored.

   Java

   ```
   lob.setContentFromClientFile(fileName);
   lob.setMimeType("application/msword");
   ```

   C++

   ```
   lob->setContentFromClientFile(fileName);
   lob->setMimeType("application/msword");
   ```

   For additional information about MIME types, see the `SResourceItemMimeTypesICM` sample.
3. Add the data to the content server. In this case, a sample Word document. Note that the resource item content is stored in resource manager.

   Java

   ```
   lob.add("SResourceItemICM_Document1.doc");
   ```

   C++

   ```
   lob->add("SResourceItemICM_Document1.doc");
   ```

   Important: In C++, you must clean up the memory.

   ```
   delete(lob);
   ```

   For more information, refer to the `SItemUpdateICM` sample.

Searching for items

To find items that match a set of criterion, complete the following steps:

1. Connect to a DKDatastoreICM object.
2. Process the correct query. Make sure that your query conforms to the query language. For more information see "Understanding the query language" on page 177.
3. Save the query results in a DKResult object.

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];
// only one result will be returned
options[0].set(DK_CM_PARM_MAX_RESULTS, "1");// Retrieve only one item
options[1].set(DK_CM_PARM_RETRIEVE , (long)DK_CM_CONTENT_ATTRONLY);
// Last option has to be this value
options[2].set(DK_CM_PARM_END , NULL);
//Note if a query is expected to return more than one result,
//the DKDatastoreICM::execute method
//should be used. The execute method returns a dkResultSetCursor.
DKResults* results = (DKResults*)(dkCollection*)dsICM->evaluate
  (query, DK_CM_XQPE_QL_TYPE, options);
```

For additional information, see the `SSearchICM` sample.

Retrieving an item

To explicitly retrieve an item or refresh an item retrieved through a query, you must have access to the PID object or PID string of the item to be retrieved or refreshed. If only know the ItemID, you can perform a query to retrieve the complete PID information. Given the item type name and item ID, the following scenario shows how to retrieve the DDO if you want to retrieve explicitly.

Java

1. After connecting to a datastoreICM object, search for the item using the appropriate item ID. For information about writing queries, see “Querying the Content Manager server” on page 178.
2. Save the query results in a DKResults object. In the code example below, `queryStr` is the search string in the correct query language format.`DKResults results = (DKResults)dsICM.evaluate(queryStr, DKConstantICM.DK_CM_XQPE_QL_TYPE, null);`
3. Create an iterator on DKResults and use it to get DDOs, which you can now retrieve one by one.
   ```
   ddo.retrieve();
   ```
4. Suppose you have a PID string that you obtained from a DDO as shown in this example:
   ```
   DKPidICM pid = ddo.getPidObject();
   String  pidStr = pid.pidString();
   ```

Using the PID string, you can perform a retrieve by completing the following steps.

1. Create a DDO using the DKDatastoreICM’s createDDO method. Do not use the DKDDO constructor. In the example below, the object `dsICM` is already connected to a Content Manager datastore. Also, the PID is a string named `pidStr`.
   ```
   DKDDO ddo = dsICM.createDDO(pidStr);
   ```
2. Call the DDO’s retrieve operation.
   ```
   ddo.retrieve();
   ```

C++

1. After connecting to a datastoreICM object, search for the item using the appropriate item ID. For information about writing queries, see Querying the Content Manager server.
2. Save the query results in a DKResult object. In the code example below, queryStr is the search string in the correct query language format.
   ```
   DKResults* results = (DKResults*)(dkCollection*)
      dsICM->evaluate(queryStr, DK_CM_XQPE_QL_TYPE,NULL);
   ```
3. Create a DDO using the DKDatastoreICM’s createDDO method. Do not use the DKDDO constructor. In the example below, the object dsICM is already connected to a Content Manager datastore. Also, the PID is a string named pidStr.
   ```
   DKDDO* ddo = dsICM->createDDO(pidStr);
   ```
4. Call the DDO’s retrieve operation.
   ```
   ddo->retrieve();
   ```

If an item is enabled for versioning, you can retrieve a specific version of the item by using the retrieve method of the DKDDO class with the appropriate options. To retrieve the latest version of an item, use the constant DK_CM_VERSION_LATEST as the retrieve option. By default (if no options are specified), ddo.retrieve() retrieves the latest version of an item.

Example:

```
DKDDO ddo = ds.createDDO(...);
   .... ddo.retrieve(DK_CM_VERSION_LATEST);
```

You can also retrieve a DDO by calling the `retrieveObject` method on the `DKDatastoreICM` object.

Example:

```
DKDatastoreICM ds =new DKDatastoreICM();
//connect,etc ...
DKDDO ddo =ds.createDDO(itemType,option);
//sets the PID as shown above...
ds.retrieveObject(ddo);
```

For more information on retrieving items, refer to the `SItemRetrievalICM` and `SResourceItemRetrievalICM` API Education Samples.

If an item is enabled for versioning and you want to retrieve the latest version, including its attributes and children, use the following format:

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);
```

For more information about retrieve, such as retrieval options, refer to the `SItemRetrievalICM` sample.

Deleting items

Use the delete method in the DDO to delete an item from the content server.

Java

```
ddoDocument.del();
```

C++

```
ddoDocument->del();
```

The DDO must have its item ID and object type set, and have a valid connection to a content server.

For more information about deleting items, see the `SItemDeletionICM` API Education sample.

Checking in and checking out items

To prevent two users from making changes to the same item at the same time, you must check out an item before updating it. Checking out an item grants exclusive update rights to the user who has the item checked out. When you check out an item you hold a persistent write lock on that item. The entire item is locked and unlocked as a whole, including all versions and child components. Linked and referenced items, however, are not locked with the item. The persistent lock on the item is released when you check in the item.

To check items in and out, use the DKDatastoreICM's `checkIn` and `checkOut` operations, as shown in example below.

1. Having connected to a `DKDatastoreICM` object named `dsICM`, and using a DDO called `myDataObject`, check out the item.

   Java

   ```
   dsICM.checkOut(myDataObject);
   ```

   C++

   ```
   dsICM->checkOut(myDataObject);
   ```

2. Check in the item.

   Java

   ```
   dsICM.checkIn(myDataObject);
   ```

   C++

   ```
   dsICM->checkIn(myDataObject);
   ```

For more information on checking items in and out, refer to the `SItemUpdateICM` API Education Sample.

Setting and getting an item's versioning properties

The example below, demonstrates how to set an item's versioning properties and how to retrieve and item's versioning properties.

Java

```
DKPidICM pid = (DKPidICM)ddo.getPidObject();
// Accessing version information
String version = pid.getVersionNumber();
...
// Setting the version number in the DDO
pid.setVersionNumber(version);
```

C++

```
DKPidICM * pid = (DKPidICM *)ddo->getPidObject();
// Accessing version information
DKString version = pid->getVersionNumber();
....
//Setting the version number for the PID associated with an item (DDO).
pid->setVersionNumber((char *)version);
```

Working with versioning properties

This section provides examples to help you work with versioning properties.

Retrieving the version control policy of an item type

An item has a version control policy, which contains the versioning property for the item. Following is the list of the three versioning properties available and the value used to represent each property in the version control policy.

DK_ICM_VERSION_CONTROL_ALWAYS
Versioned-always.

DK_ICM_VERSION_CONTROL_NEVER
Versioning is not supported for this item type (default).

DK_ICM_VERSION_CONTROL_BY_APPLICATION
The application determines the versioning scheme.

Java

```
short  versionControlPolicy;
DKItemTypeDefICM item = newDKItemTypeDefICM();
....
versionControlPolicy = item.getVersionControl();
```

C++

```
short versionControlPolicy = 0;
DKItemTypeDefICM * item    = NULL;
...
versionControlPolicy = item->getVersionControl();
```

Setting version control for an item type

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);
```

For a complete sample that accesses version control information of item type definitions in the system, refer to the `SItemTypeRetrievalICM` API Education Sample.

Getting the maximum number of versions allowed for an item type

Java

```
short  versionMax;
DKItemTypeDefICM item = new DKItemTypeDefICM();
....
versionMax = item.getVersionMax();
```

C++

```
 short versionMax     = 0;
DKItemTypeDefICM * item = NULL:
....
versionMax = item->getVersionMax();
```

Setting the maximum number of versions allowed for an item type

In this example, only ten versions of an item that is based on this item type are maintained by the system.

Java

```
short  versionMax=10;
DKItemTypeDefICM item = new DKItemTypeDefICM();
...
item.setVersionMax(versionMax);
```

C++

```
short versionMax    = 10;
DKItemTypeDefICM * item = NULL;
 ....
item->setVersionMax(versionMax);
```

Setting the value of the versioning type for an item type

C++

```
DKItemTypeDefICM * item = NULL;
short versionType = DK_ICM_ITEM_VERSIONING_OPTIMIZED;
...
item->setVersioningType(versioningType);
```

Working with folders

A folder is a fully supported DDO that represents an item. A folder has the full hierarchical data structure of the item type that it is created in. A folder is a DDO of semantic type folder and has an attribute, DKConstant.DK_CM_DKFOLDER, that contains a DKFolder, regardless of the item type's classification. The DKFolder object, a DKSequentialCollection, can contain any number of root component DDOs representing other items.

The procedures below include code fragments for informational purposes only. Do not copy these code fragments and paste them directly into your application program, because they will not work in their current form. For a complete folders sample that you can run, see the `SFolderICM` sample in the `samples` directory.

Creating a folder

The DKDatastoreICM.createDDO() method is used to create DDOs at runtime. As shown in the SItemCreationICM sample (in the samples directory), when creating folder items, the item property type or semantic type needs to be specified as DKConstant.DK_CM_FOLDER.

A folder item is created using the DKDatastoreICM.createDDO method with the DK_CM_FOLDER semantic type. You can create a folder by specifying the DKConstant.DK_CM_FOLDER as the second parameter to the createDDO method.

Java

```
DKDDO ddoFolder  = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
```

C++

```
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);
```

Use `setData` method to populate the `ddoFolder`. Once the folder DDO is created, it is saved to the persistent content server.

Java

```
ddoFolder.add();
```

C++

```
ddoFolder->add();
```

Adding contents to a folder

All contents that are to be placed in the DKFolder must be persistent in the content server before you call the add() or update() method on the folder DDO. To make the contents persistent, call their DKDDO.add() methods.

To add items to a folder, use the DKFolder's `addElement()` function. When you have added enough members into the folder, you can call the add or update method to save the folder-content relationships into the persistent store. This groups any number of folder modifications and into a single call to the library server. When adding items to a folder, do not add multiple copies of the same item to DKFolder. Since all folder modifications are tracked, do not cause conflicting or duplicate modifications. For example, do not add multiple copies of the same item to the folder.

Follow the steps below to add contents to a folder. Note that a folder can contain another folder (sub-folder) as one of its content items.

Java

1. Create three new items. These items will be added to a folder as its contents. Folder contents can be of any semantic type.

   ```
   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. Use the setData method to populate the DDOs. Items that are to be added as folder contents need to be persisted into the datastore.

   ```
   ddoDocument.add();
   ddoItem.add();
   ddoFolder2.add();
   ```
3. Retrieve the dkFolder attribute of the previously created and persisted folder DDO.

   ```
   short dataid = ddoFolder.dataId
      (DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKFOLDER);
   dkFolder = (DKFolder) ddoFolder.getData(dataid);
   ```
4. The folder must be checked out of the datastore before it is updated (by adding contents).

   ```
   dsICM.checkOut(ddoFolder);
   ```
5. Add the previously created items to the folder.

   ```
   dkFolder.addElement(ddoDocument);
   dkFolder.addElement(ddoItem);
   dkFolder.addElement(ddoFolder2);
   ```
6. Commit the changes to the folder and explicitly check in the changes.

   ```
   ddoFolder.update();
   dsICM.checkIn(ddoFolder);
   ```

C++

1. Create the items that will be added to the folder.

   ```
   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. Persist the created items to the datastore.

   ```
   ddoDocument->add();
   ddoItem->add();
   ddoFolder2->add();
   ```

3. Retrieve the DKFolder attribute for the folder.

   ```
   DKFolder * dkFolder;
   short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKFOLDER);
   if (dataid!=0) dkFolder =
     (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
   ```

4. Check out the folder (A folder must be checked out before it is update)

   ```
   dsICM->checkOut(ddoFolder);
   ```

5. Add the created items to the folder.

   ```
   dkFolder->addElement(ddoDocument);
   dkFolder->addElement(ddoItem);
   dkFolder->addElement(ddoFolder2); // Folders can contain sub-folders.
   ```

6. Update the folder. This implictly also checks in the folder (unlocks it).

   ```
   ddoFolder->update();
   ```

7. Explicitly check in the folder.

   ```
   dsICM->checkIn(ddoFolder);
   ```

Removing contents from a folder

Items can be removed from a folder in one of two ways. A deferred removal (the actual removal is done when the folder DDO is updated and multiple content server calls are combined into one) is done using the `removeElementXX` method(s). An immediate (and expensive since multiple calls are made to the content server) removal can be done using the dkFolder.removeMember method. See the `SFolderICM` sample (in the samples directory) for further details on working with folders.

Complete the following steps to remove contents from a folder:

Java

1. Check out the folder DDO from which we want to remove an item.

   ```
   dsICM.checkOut(ddoFolder);
   ```

2. Look for the item to remove. In this case, look for the docItem DDO created earlier. Create an iterator to iterate through the folder's content.

   ```
   dkIterator iter = dkFolder.createIterator();
   String itemPidString = ddoItem.getPidObject().pidString();
   while(iter.more()) {
      // Move the pointer to next element and return that object.
      // Compare the PID trings of the DDO returned from the iterator
      // with the DDO that is to be removed.
      DKDDO ddo =(DKDDO)iter.next();
      if(ddo.getPidObject().pidString().equals(itemPidString)) {
         // The item to be removed is found.
         // Remove it (current element in the iterator)
         dkFolder.removeElementAt(iter);
      }
   }
   ```

3. Persist the change and check in the folder DDO.

   ```
   ddoFolder.update();
   dsICM.checkIn(ddoFolder);
   ```

C++

1. Create an item and add it to a folder. Note that the folder was created earlier.

   ```
   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. Explicitly check in the folder.

   ```
   dsICM->checkIn(ddoFolder);
   ```

3. Create an iterator for the folder.

   ```
   dkIterator* iter = dkFolder->createIterator();
   ```

4. Iterate through the contents of the folder until the item that is to be removed is found. Remove the item.

   ```
   while(iter->more())
   {
           DKDDO* ddo = (DKDDO*) iter->next()->value();
             if ( ((DKPidICM*) ddo->getPidObject())->pidString() ==
             ((DKPidICM*) ddoItem->getPidObject())->pidString() )
             {
             //Found the element to  be removed. Remove it
             dkFolder->removeElementAt(*iter);
             // Now that we found the ddoItem, remove it.
             }
     }

    delete(iter);
   ```

Retrieving a folder’s contents

To retrieve folder contents, you must set the retrieval option. By default, the retrieval option is not set. If you do not set the retrieval option, the object will not contain a DKFolder or a DK_CM_DKFOLDER attribute until retrieve is called and the correct options are set.The retrieval options include the following:

- `DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND`
- `DKConstant.DK_CM_CONTENT_ITEMTREE`

To retrieve a folder item with the DKFolder attribute collection with Outbound Links specified, use the following format:

Java

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);
```

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_LINKS_OUTBOUND);
```

To retrieve a folder item where the item tree includes links and folder contents, use the following format:

Java

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_ITEMTREE);
```

C++

```
ddoFolder->retrieve(DK_CM_CONTENT_ITEMTREE);
```

Obtaining all folders containing a specific DDO

Multiple folders can contain the same item (DDO). This allows you to associate the item with different applications or users. To determine which folders currently contain a specific DDO, see the example below.

The following steps demonstrate how to find folders that contain a `ddoDocument` object that was previously created. See the `SFolderICM` sample (in the samples directory) for additional details about working with folders.

Java

1. Retrieve the datastore extension object.

   ```
   DKDatastoreExtICM dsExtICM =(DKDatastoreExtICM)
                   dsICM.getExtension(DKConstant.DK_CM_DATASTORE_EXT);
   ```

2. Call the getFoldersContainingDDO method on the extension object to find the folders containing the ddoDocument object. This returns a collection of DDOs where each returned DDO is a folder containing the ddoDocument object.

   ```
   DKSequentialCollection list =
     dsExtICM.getFoldersContainingDDO(ddoDocument);
   ```

3. Create an iterator to cycle through the returned collection of folders.

   ```
   iter =list.createIterator();
   while(iter.more()) {
         // Move iterator to next element and return that object.
         DKDDO ddo =(DKDDO) iter.next();
         // Print out the item Id of the folder DDO in order to identify
         // the returned folder object.
         System.out.println("-Item ID: " +
            ((DKPidICM)ddo.getPidObject()).getItemId());
      }
   ```

C++

```
DKDDO* ddoDocument = ....;
//Create datastore object,connect to datastore,create DDO etc.
....
//Create a new datastore extension object from the connected object
DKDatastoreExtICM* dsExtICM = (DKDatastoreExtICM*)
                              dsICM->getExtension(DK_CM_DATASTORE_EXT);
//Retrieve the PID for the created DDO
DKPidICM* pid = (DKPidICM*) ddoDocument->getPidObject();
//Retrieve a list of folders containing the DDO created earlier.
DKSequentialCollection *list =
  dsExtICM->getFoldersContainingDDO(ddoDocument);
//Create a iterator for the returned DDO collection
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;
```

Defining links between items

You can use links to associate a source item to a target item with an optional description item. You define links in DKLink objects, where you can specify the following types of link elements:

Table 14. Link data structure

DKLink	Definition
LinkTypeName	The type of link.
Source	The source item of a link.
Target	The target item of a link.

Table 14. Link data structure (continued)

DKLink	Definition
LinkItem	The description item. Optional.

Since a link represents a one to many relationship, it is represented in a DDO by a data-item under LINK namespace, of value DKLinkCollection. For more information about working with links, refer to the `SLinksICM` sample in the `samples` directory.

A link itself does not belong to either the source or the target. A link just connects a source and target. For example, if source A is linked to target B, A is always the source and B is always the target, regardless of which DDO, A or B, is being referenced.

In memory, both the source and the target DDOs contain copies of the same DKLink object. Since the DKLink object contains a reference to both the source and the target, a DDO containing a link also contains a link that refers to itself as well as to another DDO. For example, if Source A is linked to Target B, both A and B contain the same link, as shown in the example in Table 15.

Table 15. DKLink definition example

DKLink Defined for A to B	DDO A	DDO B
Source is **A**	Source is **A**	Source is **A**
Target is **B**	Target is **B**	Target is **B**

When you add or remove links in the persistent store, you only have to perform the operation on one of the two items, source or target. The link is automatically updated for the other item.

For more information about links and a complete links sample that you can run, see the `SLinksICM` sample in the `samples` directory.

Inbound and outbound links

When using links to reference a particular DDO, either the source or the target, you can consider the links to be *inbound* or *outbound*. When considering a link from the perspective from a particular DDO, if that DDO is the target, then that DDO considers it an inbound link. Meanwhile, the DDO at the other end of that link considers the same link to be outbound from its perspective.

In the example in Table 15, the link from DDO A to DDO B is outbound link, while the same link to DDO B is the inbound link.

Link type names

Link relationships have names. Within a DDO, links are grouped into link collections. There are system defined link type names and you can also define your own. You can use any number of user-defined link type names or system-defined link type names. The following system defined link type names include:

Link type name contains

Java Constant: DKConstant.DK_ICM_LINKTYPENAME_CONTAINS

C++ Constant: DK_ICM_LINKTYPENAME_CONTAINS

Link type name: DKFolder

Java Constant : DKConstant.DK_ICM_LINKTYPENAME_DKFOLDER

C++ Constant : DK_ICM_LINKTYPENAME_DKFOLDER

The DKFolder link type name is provided and used by the system to manage folders. The DKFolder object is a simplified interface to an outbound link collection to make folders easy to work with. Therefore, you should not use the DKFolder link type name to define or delete links. You should also not use the DKFolder link type name in any way with a DKLinkCollection. However, you must specify the DKFolder link type name in order to search folders using links.

See the `SLinksICM` sample, in the `cmbroot\samples` directory, for additional information about links. For information on creating user-defined link types, refer to the `SLinkTypeDefinitionCreationICM` sample.

Retrieving linked items

When you retrieve links, you have the option of retrieving only inbound, only outbound, or both types. In order to retrieve inbound and outbound, you must set a retrieve option. You can set the following options to retrieve links:

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

Remember that before you make a call to the DDO `add()` or `update()` methods, all the items that are associated with links must already be persistent.

Working with access control

If you have access to either the Content Manager system administration client or to the Content Manager APIs for writing your own administration program, you can use the access control functions to control access to the information in your Content Manager system. The various access control APIs allow you to control access to the entire system, to a closely related set of operations on the system, or to an individual item.

Some of the tasks that you can complete using the access control APIs include:

- Creating privileges.
- Associate a list of actions with information in the system.
- Give permission to users to perform actions on the information in the library server.

Creating a privilege

A privilege is represented by the class DKPrivilegeICM. The following steps and code examples show you how to create a new privilege object, make it persistent, and retrieve a privilege. To create a privilege, follow the steps below:

Java

1. Connect to a datastore

   ```
   DKDatastoreICM ds = new DKDatastoreICM();
   ds.connect("ICMNLSDB","icmadmin",password,"");
   dkDatastoreDef dsDef =(dkDatastoreDef)ds.datastoreDef();
   DKDatastoreAdminICM dsAdmin =(DKDatastoreAdminICM)dsDef.datastoreAdmin();
   ```

2. Retrieve a DKAuthorizationMgmtICM object. This class handles authorization management tasks.

   ```
   DKAuthorizationMgmtICM aclMgmt = (DKAuthorizationMgmtICM)
          dsAdmin.authorizationMgmt();
   ```

3. Create a new privilege object.

   ```
   DKPrivilegeICM priv = new DKPrivilegeICM(ds);
   ```

4. Assign a name to the privilege object.

   ```
   priv.setName("UserPriv");
   ```

5. Assign a description to the privilege object.

   ```
   priv.setDescription("This is user-defined privilege");
   ```

6. Add the new privilege to the authorization manager object.

   ```
   aclMgmt.add(priv);
   ```

7. Retrieve the newly created privilege from the authorization manager object.

   ```
   dkPrivilege aPriv = aclMgmt.retrievePrivilege("UserPriv");
   ```

8. Display information about the privilege object.

   ```
   System.out.println("privilege name = " + aPriv.getName());
   System.out.println("privilege description = " + aPriv.getDescription());
   ```

C++

1. Create the datastore object as well as all the related administration management objects.

```
//Create the datastore object
DKDatastoreICM * ds = new DKDatastoreICM();
//Connect to the underlying datastore
ds->connect("ICMNLSDB", "icmdmin", "password", "");
//Retrieve the datastore definition object (used to access
//and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process datastore
//administration functions.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
    dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
                  dsAdmin->authorizationMgmt();
```

2. Create a new privilege object and set its properties

```
DKPrivilegeICM * priv = new DKPrivilegeICM(ds);
//Set the name of the privilege object
priv->setName("UserPriv");
//Set the privilege description
priv->setDescription("This is user-defined privilege");
```

3. Add the privilege to the datastore via the authorization management object.

```
aclMgmt->add(priv);
```

Creating a privilege set

A privilege set is represented by the class DKPrivilegeSetICM. Before you can begin creating privilege sets, you must be connected to a content server. To create a privilege set, complete the following steps:

Java

1. Create new privileges (or retrieve privileges) to add to the new privilege set.

   ```
   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. Create a new privilege set.

   ```
   DKPrivilegeSetICM privSet1 = new DKPrivilegeSetICM(ds);
   ```

3. Assign a name to the privilege set.

   ```
   privSet1.setName("UserPrivSet");
   ```

4. Assign a description to the privilege set.

   ```
   privSet1.setDescription("This is a user-defined priv set");
   ```

5. Add the privileges to the privilege set.

   ```
   privSet1.addPrivilege(priv_1);
   privSet1.addPrivilege(priv_2);
   privSet1.addPrivilege(priv_3);
   ```

6. Add the newly created privilege set to the authorization manager.

   ```
   AclMgmt.add(privSet1);
   ```

7. Display information about the newly created privilege set.

   ```
   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("  privilege name = " + _priv.getName());
   }
   ```

C++

1. 1. Create the datastore object as well as all the related administration management objects.

   ```
   //Create the datastore object
   DKDatastoreICM * ds = new DKDatastoreICM();
   //Connect to the underlying datastore
   ds->connect("ICMNLSDB", "icmdmin", "password", "");
   //Retrieve the datastore definition object
   //(used to access and manipulate CM meta-data)
   dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
   //Create the class used to represent and process datastore administration
   // functions.
   DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
        dsDef->datastoreAdmin();
   //Retrieve the class used to represent and manage the authorization
   //related functionality of the ICM datastore
   DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
                dsAdmin->authorizationMgmt();
   ```
2. Create three privileges and set their properties.

   ```
   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. Create a new privilege set and set its properties.

   ```
   DKPrivilegeSetICM * privSet1 = new DKPrivilegeSetICM(ds);
   privSet1->setName("UserPrivSet");
   privSet1->setDescription("This is a user-defined priv set");
   ```
4. Add the created privileges to the new privilege set.

   ```
   privSet1->addPrivilege(priv_1);
   privSet1->addPrivilege(priv_2);
   privSet1->addPrivilege(priv_3);
   ```
5. Add the privilege set to the datastore using the authorization management object.

   ```
   aclMgmt->add(privSet1);
   ```

Displaying privilege set properties

The example below demonstrates how to display a privilege set’s properties.

C++

```
//Retrieve a privilege set using its name
DKPrivilegeSetICM * sPrivSet = (DKPrivilegeSetICM *)
   aclMgmt->retrievePrivilegeSet("UserPrivSet");
//Display privilege set properties
cout<<"privilege set name = "<< (char *)sPrivSet->getName() << endl;
cout<<"priv set descrip="<< (char *)sPrivSet->getDescription() << endl;
//Retrieve the list of privileges that are a part of this privilege set
dkCollection * coll = sPrivSet->listPrivileges();
dkIterator * iter = coll->createIterator();
while (iter->more())
{
    DKPrivilegeICM* _priv = (DKPrivilegeICM *)(void *)(*iter->next());
    cout<<"  privilege name = "<< (char *)_priv->getName() << endl;
}
delete(iter);
```

Defining an access control list (ACL)

The Content Manager access control model is applied at the level of the controlled entity. A controlled entity is a unit of protected user data. A controlled entity can be an individual item, item-type, or the entire library. Operations on controlled entities are regulated by one or more control rules. The ACL is the container for these control rules. The DKAccessControlListICM class represents an ACL. Every controlled entity in a Content Manager system must be bound to an ACL.

The examples below demonstrate how to define an ACL.

Java

```
//Define a new DKACLData object.
//A DKACLData class is used to hold ACL related data.
DKACLData aclData1 = new DKACLData();
//set the user group name for the ACL
aclData1.setUserGroupName("ICMADMIN");
//set ACL patron type.
aclData1.setPatronType(DK_CM_USER_KIND_USER);
//Set the privilege set associated with this ACL
aclData1.setPrivilegeSet(privSet_1);
//Create another DKACLData object.
DKACLData aclData2 = new DKACLData();
aclData2.setUserGroupName("ICMPUBLC");
aclData2.setPatronType(DK_CM_USER_KIND_GROUP);
aclData2.setPrivilegeSet(privSet_2);
//Create a new ACL. A DKAccessControlListICM represents a CM v8.2. ACL
DKAccessControlListICM acl1 = new DKAccessControlListICM(ds);
//Assign a new to the newly-created ACL
acl1.setName("UserACL");
//Assign a description to the newly-created ACL
acl1.setDescription("This is a user-defined ACL");
//Add the previously created ACL data objects to the ACL
acl1.addACLData(aclData1);
acl1.addACLData(aclData2);
//Add the newly-created ACL to the authorization manager
aclMgmt.add(acl1);
//Retrieve and display information about the newly created ACL
DKAccessControlListICM acl_1 = (DKAccessControlListICM)
aclMgmt.retrieveAccessControlList("UserACL");
System.out.println("ACL name  = " + acl_1.getName());
System.out.println("    desc  = " + 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("  PrivSet name  = " + _privSet.getName());
   System.out.println("  PrivSet desc  = " + _privSet.getDescription());
   String usrGrpName = aclData.getUserGroupName();
   System.out.println("    UserGroupName = " + usrGrpName);
   short patronType = aclData.getPatronType();
   System.out.println("    Patron type   = " + patronType);
}
```

C++

1. Define new DKACLData objects. Each DKACLData object is used to hold ACL related data.

   ```
   DKACLData * aclData1 = new DKACLData();
   // set the name of the user group to be associated with this ACL
   aclData1->setUserGroupName("ICMADMIN");
   // Set  the ACL patron type. Can be one of 3 possible
   //values:DK_CM_USER_KIND_USER or DK_CM_USER_KIND_GROUP or
   //DK_CM_USER_KIND_PUBLIC.
   aclData1->setPatronType(DK_CM_USER_KIND_USER);
   // Set the privilege set associated with the ACL.
   DKPrivilegeSetICM * privSet_1 = (DKPrivilegeSetICM *)
      aclMgmt->retrievePrivilegeSet("UserPrivSet");
   aclData1->setPrivilegeSet(privSet_1);
   ```

2. Create another DKACLdata object.

   ```
   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. Create a new ACL. Set property values for this new ACL.

   ```
   DKAccessControlListICM * acl1 = new DKAccessControlListICM(ds);
   //Assign a new name to the newly-created ACL.
   acl1->setName("UserACL");
   // Assign a description to the newly-created ACL.
   acl1->setDescription("This is a user-defined ACL");
   ```

4. Add the ACL, created in step three, data objects to the ACL.

   ```
   acl1->addACLData(aclData1);
   acl1->addACLData(aclData2);
   ```

5. Add the ACL, created in step three, to the authorization manager.

   ```
   aclMgmt->add(acl1);
   ```

The complete sample program is in the samples directory.

Retrieving and displaying ACL information

To retrieve and display ACL information, complete the following steps.

1. Retrieve the ACL from the authorization management object using the ACL's name.

C++

```
DKAccessControlListICM * acl_1 = (DKAccessControlListICM *) aclMgmt->
                        retrieveAccessControlList("UserACL");
cout<<"ACL name = "<< (char *)acl_1->getName() << endl;
cout<<"    desc = "<< (char *)acl_1->getDescription() << endl;
```

2. Retrieve the ACL data associated with the 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<<"privSet name = "<< (char *)_privSet->getName() << endl;
       cout<<"privSet desc="<< (char *)_privSet->getDescription() << endl;
       DKString usrGrpName = aclData->getUserGroupName();
       cout<<"    UserGroupName = "<< (char *)usrGrpName << endl;
       short patronType = aclData->getPatronType();
       sprintf(szpatronType, "%d", patronType);
       cout<<"    Patron type   = "<< szpatronType << endl;
}
delete(iter);
```

Assigning an ACL to an item type

Once an ACL is created, you can associate it to a specific item type. Following are the steps you follow to assign an ACL to an item type:

1. Set up anew item type.

Java

```
DKItemTypeDefICM  it = new DKItemTypeDefICM(dsICM);
//Assign a name to this item type
it.setName("TextResource1");
//Assign a description to this item type
it.setDescription("CMv8.2 Text Resource Item Type.");
//Assign an ACL code to this item type
it.setItemTypeACLCode((int)DK_ICM_ITEMACL_BIND_AT_ITEM);
....
it.add();  // make the item type persistent
...
```

C++

```
DKItemTypeDefICM * itemType = new DKItemTypeDefICM(dsICM);
// Assign a name to this item type.
itemType->setName("TextResource1");
// Assign a description to this item type.
itemType->setDescription("CMv8.2 Text Resource Item Type.");
```

2. Retrieve the ACL data associated with the ACL.

Java

```
int itemTypeACLCode = it.getItemTypeACLCode();
// By setting the item type's ACL flag to TRUE(1) we confirm that ACL
// binding is at the item type level. By setting the item type's ACL
// flag was set to FALSE(0), we would be saying that the ACL binding
// is not at the item type level
it.setItemLevelACLFlag(1); // - true
//Determine whether the ACL binding is at the item type level or not
int itemTypeACLFlag = it.getItemLevelACLFlag();
```

C++

```
itemType->setItemTypeACLCode((long) 1);
// By setting the item type's ACL flag to TRUE (1),
//we confirm that ACL binding is at the item type level.
//By setting the item type's ACL flag to FALSE(0),
// we would be saying that the ACL binding is not at the item type level.
itemType->setItemLevelACLFlag((short)1);
itemType->add();
// make the item type persistent.
```

The complete sample program is available in the `samples` directory.

Assigning an ACL to an item

To enable item level access control, you must bind an ACL to the item. To do this, you must create the item using the add() method on the DDO, as shown in the code fragment below. In the code fragment below, it is assumed that you already have a connection to the content server and have created the ACL. The complete program is in the `samples` directory.

Java

```
//Create a new DDO
DKDDO ddoItem =dsICM.createDDO("myItemType",DKConstantICM.DK_CM_ITEM);
//create a new ACL (access contrl list)
DKAccessControlListICM acl1 =new DKAccessControlListICM(ds);
//Assign a name to the ACL
acl1.setName("MyACL");
//Add a new property to the DDO.This property will be
//called DK_ICM_PROPERTY_ACL
int propId =ddoItem.addProperty(DK_ICM_PROPERTY_ACL);
//Set the previously created (or retrieved)ACL as the value of this property
ddoItem.setProperty(propId,"MyACL");
//Persist the DDO into the data store
ddoItem.add();
```

C++

1. Create a new item (DDO) based on an existing item type.

   ```
   DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);
   ```

2. Create a new ACL (Access Control List) and set the ACL's properties.

   ```
   DKAccessControlListICM * acl1 = new DKAccessControlListICM(dsICM);
   ```

3. Assign a name to the ACL.

   ```
   acl1->setName("MyACL");
   ```

4. Add a new property to the DDO. This property will be called DK_ICM_PROPERTY_ACL.

   ```
   ushort propId = ddoItem->addProperty(DK_ICM_PROPERTY_ACL);
   ```

5. Set the ACL, created above, as the value for this property. Note that a user can choose to retrieve an existing ACL and use it as the ACL this item.

   ```
   ddoItem->setProperty(propId, "MyACL");
   ```

6. Persist the DDO into the data store.

   ```
   ddoItem->add();
   ```

Understanding the query language

A powerful XML-based query language is available in Content Manager 8.2. When an application searches for items stored in the Content Manager system, the underlying engine performs the processing of the search based on a query. To efficiently traverse Content Manager's hierarchical data model, you use the Content Manager query language. The query language provides the following benefits:

- Supports the full data model.
- Support for versions, such as searching for a specific version and for the latest version.
- Enables searches within component type view hierarchies, across linked items, and references.
- Combines parametric and text search.
- Provides SORTBY capabilities.
- Enforces Content Manager access control.
- Conforms to XQuery Path Expressions (XQPE), a subset of W3C XML Query working draft.
- Executes high performance searches.

The Content Manager query language is an XML-based query language that, unlike proprietary languages, conforms to XQuery Path Expressions (XQPE), a subset of W3C XML Query working draft. The query language searches hierarchical item types and locates items quickly and easily. Before you begin to write queries, you must understand the query language concepts, syntax, and grammar.

All queries are done on component type views. Therefore, the names that you use for root components or child components in your query strings can be the names of either base component type views that are created for you by the system (for example, Journal, Journal_Article) or the names of user-defined component type views (for example, My_Journal, My_Book_Section). In sys admin, they call component type views as component type subsets.

When you submit a query for a document, folder, object, and so forth, your request is directed to the Content Manager query engine, which processes the query and translates it to the appropriate SQL. The library then adds in the security checks (ACLs) and runs the query.

Querying the Content Manager server

To query the Content Manager library server:

1. Query its content server by creating a query string to represent your search conditions.
2. Call the evaluate, execute, or executeWithCallback method with your query string and query options.
3. Receive the results through a DKResults object, a dkResultSetCursor, or a dkCallback object.

The query APIs perform the query processing tasks, like preparing and executing a query, monitoring the status of a query execution, and retrieving the results.

Logically, a query string can contain one of three types of queries: parametric, text, and combined. A parametric query uses conditions like equality and comparison. A text query uses text search functions to make the search more powerful. A combined query is composed of both text and parametric conditions.

To run a query, you can use one of the following methods: evaluate, execute, or executeWithCallback. The evaluate method returns all results as a collection of DDO's and is useful for small result sets. The execute method returns a dkResultSetCursor object, which has the following characteristics:

- The dkResultSetCursor works like a content server cursor.
- You can use it for large result sets because the DDOs it returns are retrieved in blocks as the user requests them.
- You can use dkResultSetCursor to rerun a query, by calling the close and open methods.
- You can use the dkResultSetCursor to delete and update the current position of the result set cursor.

The executeWithCallback method executes a query in asynchronous manner and sends the results to the specified callback object in blocks. As such, the executeWithCallback method frees up the main thread of execution from the task of retrieving query results. For more information on query methods, refer to the `SSearchICM` sample. Refer to the Application Programming Reference for methods evaluate, execute, and executeWithCallback in DKDatastoreICM class.

Applying the query language to the Content Manager data model

To help you understand the query language, you can conceptually view the library server as an XML document. The XML document analogy is used only for the purpose of explaining the query language. Therefore, it is very important to remember that the XML representation of the library server is only a virtual representation and items in a library server are not XML elements, nor do you get XML elements when you perform a query. In the XML representation of the library server, Content Manager data model elements are represented as follows:

Items In general, each CM item is represented by nested XML elements, where the top level XML element represents the root component and the nested XML elements represent the descendent components. The nesting of XML elements thus represents the component hierarchy.

Root components

A root component is represented by the first level of an XML element. A root component has the following XML attributes: ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID, INTEGER SEMANTICTYPE, and any other user-defined attributes of the component. In the library server, the ITEMID is unique.

Child components

A child component is represented by a nested XML element and has the following attributes: STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID, and any other user-defined attributes of the component.

Note that COMPONENTID alone is only unique within a Content Manager component.The ITEMID and the VERSIONID are exactly the same as the child's root component ITEMID and VERSIONID.

User-defined attributes

Each user-defined attribute is represented by a nested XML attribute within the XML element representing the containing component.

Links Although the inbound and outbound links are not a part of an item itself in the CM data model (they are stored separately in the links table), for the purpose of querying it is very convenient to conceptually think of them as being a part of the XML element representing the item. This relieves applications from writing joins explicitly in the queries. Links represent a one-to-many relationship between items. Note that this relationship is only between root components (a link cannot originate or end in a child component).

The links originating at an item are represented by <OUTBOUNDLINK> XML elements with the following attributes: IDREF LINKITEMREF, IDREF TARGETITEMREF and STRING LINKTYPE. The LINKITEMREF is a reference to an item that contains meta-data for the link. The TARGETITEMREF is a reference to the item pointed to by the link. The LINKTYPE is the type of the link Similarly, links pointing to an item are represented by <INBOUNDLINK> XML elements with the following attributes: IDREF LINKITEMREF, IDREF SOURCEITEMREF and STRING LINKTYPE. The SOURCEITEMREF is a reference to the item where the link originates. For more information on link semantics, refer to the SLinksICM and SSearchICM samples.

References (reference attributes)

Reference attributes are represented by XML attributes of type IDREF. A reference represents a one-to-one relationship between an item or a component and another item. Therefore, the target of a reference can only be a root component, not a child. A reference attribute, however, can originate in either root or child components.

Reference attributes can be either system-defined (SYSREFERENCEATTRS) or user-defined (PublicationRef in sample queries below). References can be traversed in both directions.

Reverse traversal of references is performed in a way similar to reverse traversal of links, as described above. You can conceptually think of the item that is being referenced as having an XML element called REFERENCEDBY that contains an XML attribute called REFERENCER (of type IDREF), which points to the component that references this item. This is similar to the INBOUNDLINK element with the SOURCEITEMREF attribute for reverse traversal of links.

References also support delete semantics (restrict, cascade, set null, or no action). For more information on reference attributes, refer to the `SReferenceAttrDefCreationICM` and `SSearchICM` samples.

Documents

Folder functionality provided in Content Manager through the DKFolder is stored as a set links of the ″DKFolder″ (DK_ICM_LINKTYPENAME_DKFOLDER) link type. Each folder-content relationship is modeled as an outbound link from the folder and an inbound link to the contents making the folder the source and the contents the targets of each link. Follow the semantics of searching and traversing links in order to search and traverse folders. For more information, refer to the `SFolderICM` and `SSearchICM` samples.

Understanding parametric search

Items are often retrieved by initiating a search on selected attributes. A single query can examine both system-defined and user-defined attributes of the items in the content server. Simple search conditions consist of an attribute name, an operator, and a value that are combined into a clause. Content Manager provides you with many comparison operators to complete parametric searches. The operators include:

″=″

″< ″

″<=″

″>″

″>=″

″!=″

″LIKE″

″NOT LIKE″

″BETWEEN″

″NOT BETWEEN″

″IS NULL″

″IS NOT NULL″

You can specify complex search conditions by combining simple search conditions into a clause using the Boolean operators AND, OR, and NOT. Refer to the query examples for more details.

Understanding text search

Using the DB2 Universal Database Net Search Extender (NSE), formerly known as Text Information Extender (TIE) in CM 8.1, Content Manager provides two types of text search: text search of attributes that contain text in components and text search of objects. The main difference between the two types of text search is how the content is stored. When you define an attribute to be text searchable, you are indicating that one can search text contained in the column of that attribute. To make an attribute (column) text searchable, NSE creates a text index. The text index holds information about the text to be searched. This information is used to perform text search efficiently. For example, Fred, a system administrator, creates an item type called `Journal` that has a child component type view `Journal_Article`, which he wants to enable for text search. One of the attributes for `Journal_Article` is `Title`, which Fred enables for text search. When Lily, an underwriter, searches for `Title` that contains the word ″Java″, the system searches the `Title` text index for any hits on ″Java″.

For information about TIE used in CM 8.1, see the DB2 Universal Database Text Information Extender (TIE) documentation. For information about NSE used in CM 8.2, see the DB2 Universal Database Net Search Extender (NSE) documentation. In the discussion of text search in ICM query language, NSE and TIE can be used interchangeably. From the perspective of ICM query language, there are no differences in text search syntax or functionality between NSE and TIE.″

Searching for object contents

Searching for contents of objects works a little differently. Instead of indexing a column directly, the system uses a reference to the object's location on a resource manager. NSE uses the reference to fetch the content when it creates a text index. An end user performing a search does not notice any difference when searching for objects stored in a resource manager. A system administrator, however, has to set up a text resource item type view in order for the search mechanism to locate the content in the resource manager. The text search is performed on the resource item type's attribute ″TIEREF″, which refers to the contents stored on the resource manager for text search purposes.

Searching for documents

You can perform text search on the contents of document parts. A virtual component type view ″ICMPARTS″ is supported in query as a child of every document in the system. The ″TIEREF″ attribute under the ″ICMPARTS″ component type view refers to the contents of all the text-searchable parts of that document for text search purposes. Refer to query examples for specific usage of this functionality.

Making user-defined attributes text searchable

You can make your user-defined attributes text searchable by using the DKAttrDefICM and DKItemTypeDefICM APIs. Default properties of the created text index can be modified by using the DKTextIndexDefICM class. For more information on the APIs, see the online API reference or the `SItemTypeCreationICM` sample.

Understanding text search syntax

You can perform text search queries by using either basic or advanced text search syntax.

Basic text search

Since the majority of text searches are done by simply listing a few words one after the other, basic (simplified) text search syntax was designed specifically to make this most common case easy for users. The syntax also allows for use of ″+″ and ″-″, as well as for use of quoted phrases. Simplified text search is done by using ″contains-text-basic″ and ″score-basic″ functions. The ″contains-text-basic″ function is used to search within attributes or within content of resources or documents. The ″score-basic″ function uses the same syntax as the ″contains-text-basic″ function and is used for sorting results based on the rank of the text search results. Don't forget to equate the ″contains-text-basic″ function to 1 to check if it is true, and to equate it to 0 to check if it is false. Refer to query examples for how these functions are used.

Additional information about basic text search syntax includes the following:

- Performs case-insensitive text search (just like advanced syntax by default). See the NSE documentation for case-sensitive search options.
- Terms within quotes are assumed to be a phrase.
- Use of + (plus) - (minus)
  - + (plus) = document must include this word.
  - - (minus) = document must not include this word.
  - When a + or - is not specified, the query engine uses an algorithm to match the words to the text.
- Boolean operators (AND, OR, NOT) are not valid and are ignored.
- Parentheses in the basic syntax are not supported.
- Valid wildcards
  - ? (question mark) = represents a single character
  - * (asterisk) = represents any number of arbitrary characters

For more information on basic text search, refer to the `SSearchICM` sample.

Advanced text search

Advanced text search syntax is used to allow the user to specify more complex conditions for text search. The text search uses the NSE text search syntax, and allows such powerful features as proximity search and fuzzy search. Advanced text search syntax uses ″contains-text″ and ″score″ functions similar to the way the ″contains-text-basic″ and ″score-basic″ functions are used for basic text search. The strings that are supplied to the advanced functions should be in NSE syntax, except as follows: change double quotes to single quotes, and vice-versa. For example, CONTAINS (description,′ ″IBM″ ′)=1 condition in NSE would become contains-text(@description, ″ ′IBM′ ″)=1 in CM query language. This needs to be done to support simplicity of writing queries with minimal use of escape characters. Don′t forget to equate the ″contains-text″ function to 1 to check if it is true, and to equate it to 0 to check if it is false. Refer to query examples for more details on advanced text search.

For more information on advanced text search, refer to the `SSearchICM` sample.

Creating combined parametric and text search

Searches can be performed based on virtually any piece of an item or component, text within an item or component, or text within resource content. Search can be one of the following types:

Parametric search
Searching is based on item and component properties, attributes, references, links, folder contents, and so forth.

Text Search
Searching within text.

Combined Search
Searching using both parametric and text search.

Follow the steps below to create a combined parametric and text search.

Java

1. Create an ICM datastore and connect to it.
2. Generate the combined query string.

   ```
   String queryString =
          "//Journal_Article [Journal_Author/@LastName = \"Richardt\"" +
          " AND contains-text (@Text, \" 'Java' & 'XML' \")=1]";
   ```

3. If you want to use retrieve options that are different from the defaults, use the DKNVPair array to package the options.

   ```
   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. Execute the combined query in one of three different ways: evaluate, execute, and executeWithCallback.

   ```
   DKResults resultsCollection =
   (DKResults)dsICM.evaluate(queryString,
   DKConstant.DK_CM_XQPE_QL_TYPE, parms);
   ```

5. Process the results. The procedure for handling the results depends on which execution method you used.

For a complete sample and additional documentation, refer to the `SSearchICM` API Education Sample in `CMBROOT\Samples\java\icm`.

C++

1. Specify the search options. Note that the options array always has to have an end element. For example, if you want to specify two options, the options array must have three elements.

   ```
   DKNVPair * parms = new DKNVPair[3];
   DKNVPair * pparm = NULL;
   DKString strMax  = "5";
   DKAny * anyNull  = new DKAny();
   //Allow a maximum of 5 items to be returned from the search
   pparm = new DKNVPair(DK_CM_PARM_MAX_RESULTS, strMax);
   parms[0] = *pparm;
   delete pparm;
   //Specify what content is to be retrieved
   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. Execute the search. There are three ways to execute a search:

   evaluate
   Returns all the results as a collection; good for small sets.

   execute
   Returns a result set cursor, which the caller uses to iterate over the results.

   executeWithCallback
   Creates a thread that iterates over the result set and calls the callback object for each block of results. Caller uses the callback object to get the results

   In the example below, only five results are desired, so the DKDatastoreICM.evaluate method is used.

   ```
   DKResults * resultsCollection = (DKResults *)(dkCollection *)
           dsICM->evaluate(queryString,DK_CM_XQPE_QL_TYPE, parms);
   ```

3. Display the results of the search.

   ```
   // Create an iterator to go through Results collection.
   dkIterator* iter = resultsCollection->createIterator();

   cout << "Results:" << endl;
   cout << "     - Total:  " << results->cardinality() << endl;

   while(iter->more())
   {
   //Each element in the returned array is an item (DDO)
   DKDDO* ddo = (DKDDO*) iter->next()->value();
     cout << "     - Item ID:  " << ((DKPidICM*)ddo->getPidObject())
     ->getItemId() << "  (" << ((DKPidICM*)ddo->getPidObject())
     ->getObjectType() << ")" << endl;
   }
   ```

4. Clean up.

   ```
   delete(iter);
   delete[] parms;
   ....
   ```

For a complete sample and additional documentation, refer to the SSearchICM API Education Sample in CMBROOT\Samples\cpp\icm.

Query examples

To help you better understand the query language and to get you started with writing queries, this section provides you with the following information:

- A sample data model.
- An XML document representation of the data model.
- Sample queries.
- Query language grammar.

The sample queries in the following sections are based on the query examples data model outlined in Figure 13 on page 186. Refer to the data model illustration when you review the sample queries.

For additional query syntax and examples based on an alternate data model, refer to the SSearchICM sample

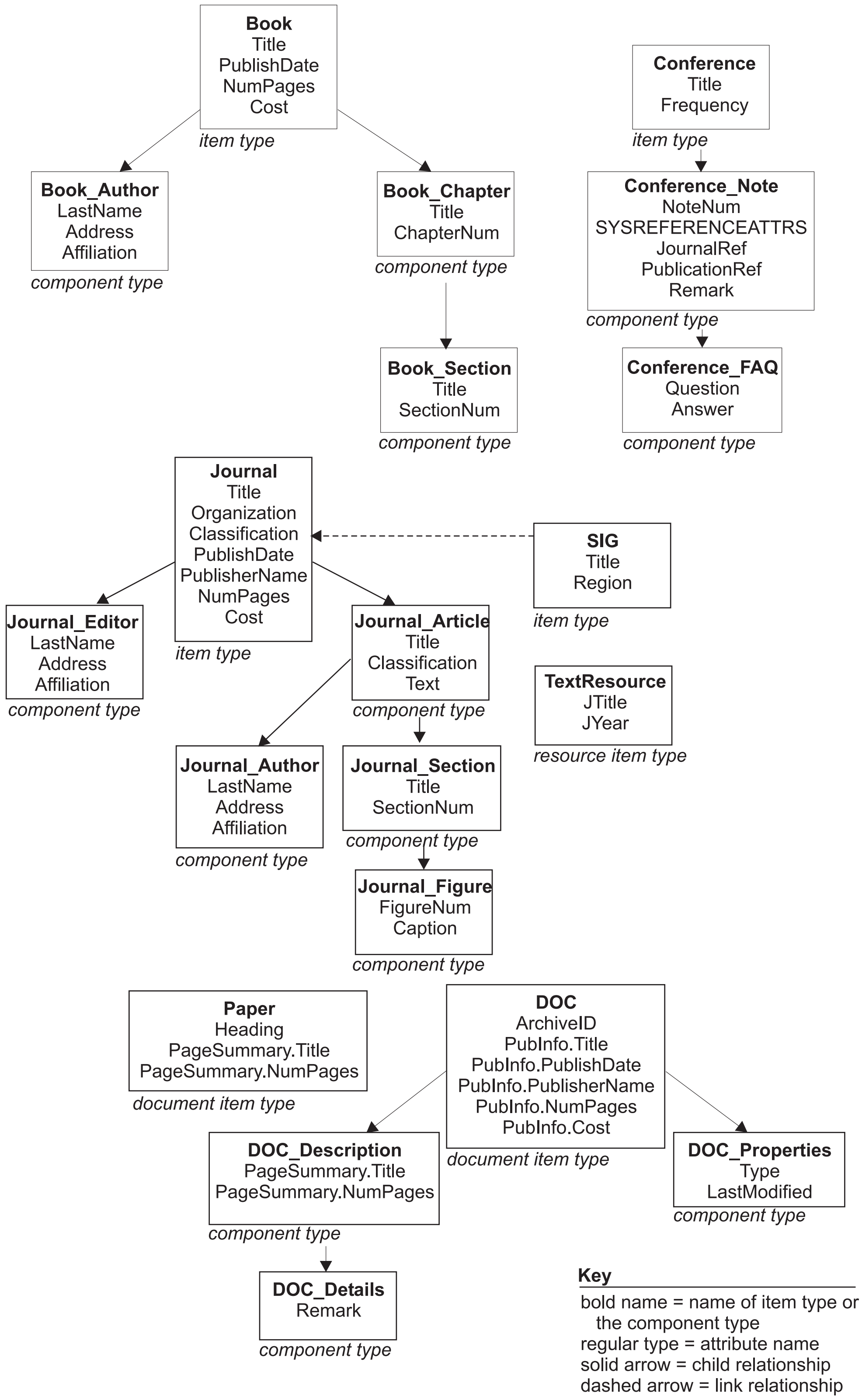


Figure 13. Query examples data model

The XML document below is a representation of the data model in figure Figure 13.

XML representation of the query examples data model:

```
<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>)
```

Query examples

The sample queries provided in this section are based on the sample data model, Figure 13 on page 186, and the sample XML document on page 187. Here are some hints to help you understand the query examples:

- Follow the query string as you would follow a directory structure
- ″/″ single slash indicates a direct child relationship
- ″//″ double slash indicates either a child relationship or a descendant relationship
- ″.″ (DOT) represents the current component in the hierarchy
- ″..″ (DOT-DOT) represents the parent of the current component
- ″@″ (AT sign) denotes an attribute
- ″[ ]″ (square brackets) denote a conditional statement or a list
- ″=>″ (DEREFERENCE operator) represents linking or referencing action
- The result of the query must be a component (for example, an attribute cannot be the last thing in the path)

Additional examples and documentation are provided in the SSearchICM sample.

Example 1: access to components

This query finds all journals.

```
/Journal
```

Explanation: The "/" starts at the implicit root of the XML document, which in this case is the entire library server. Each item type is an element under this root. If LS.xml is the XML document that contains the entire model as described above, then the explicit document root is document (LS.xml).

Example 2: access to attributes

This query finds all journal articles with a total of 50 pages in them.

```
/Journal[@NumPages=50]
```

Explanation: The predicate @NumPages = 50 evaluates to true for all journals that have the Content Manager attribute "NumPages"set to 50.

Example 3: multiple item types

This query finds all books or journals that have"Williams" as one of the authors and have a section title beginning with "XML".

```
(/Book | /Journal)
[(.//Journal_Author/@LastName = "Williams"
OR .//Book_Author/@LastName = "Williams")
AND (.//Book_Section/@Title LIKE "XML%"
OR .//Journal_Section/@Title LIKE "XML%")]
```

OR

```
(/Book[.//Book_Author/@LastName = "Williams"
AND .//Book_Section/@Title LIKE "XML%"])
| (/Journal[.//Journal_Author/@LastName = "Williams"
 AND .//Journal_Section/@Title LIKE "XML%"])
```

Explanation: The above two queries produce the same result. ".//Journal_Author" means that a component Journal_Author should be found either directly under the current component in the path (which in the first case is either a Book or a Journal) or somewhere deeper in the hierarchy. Note that the LIKE operator is used in conjunction with a wildcard character, in this case" %".

Example 4: arithmetic operations in conditions

This query finds all journals with the number of pages between 45 and 200.

```
/Journal[@NumPages BETWEEN 49-4 AND 2*100]
```

Explanation: Note that you can perform arithmetic operations to calculate the resulting values to be used with the BETWEEN operator.

Example 5: traversal of links in the forward direction

This query finds all articles in journals edited by "Williams" that are contained in SIGs with title "SIGMOD".

```
/SIG[@Title = "SIGMOD"]/OUTBOUNDLINK
 [@LINKTYPE = "contains"]/@TARGETITEMREF =>
 Journal[Journal_Editor/@LastName = "Williams"]
 /Journal_Article
```

Explanation: This is an example of following links in the forward direction. The virtual XML component OUTBOUNDLINK and its attribute

TARGETITEMREF are used to traverse to all Journals and then finally the underlying Journal_Articles. The last component in the path is what is returned as the result of the query. The result can be constrained by traversing only specific link types (“contains” in this example) to a specific type of items (Journal in this example). Since the conceptual XML representation of the library server looks at inbound and outbound links as being parts of items, the dereferencing operator can be used to relieve applications from writing explicit joins.

Example 6: traversal of links in the backward direction

This query finds all items of any type that have journals which cost less than five dollars with articles by author“ Nelson”.

```
/Journal[@Cost < 5
AND .//Journal_Author/@LastName = "Nelson"]
/INBOUNDLINK[@LINKTYPE = "contains"]
/@SOURCEITEMREF => *
```

Explanation: This is an example of following links in the backward direction. The wildcard“*”, following the dereference operator “=>” ensures that items of ANY type are returned as the result.

Example 7: text search (contains-text and score functions)

This query finds journal articles with author“ Richardt” that contain the text “Java” and the text “XML”. The results are ordered by the text search score.

```
//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text(@Text, " 'Java' & 'XML' ")=1]
SORTBY(score(@Text, " 'Java' & 'XML' "))
```

Explanation: This is an example of performing text search with the contains-text function. For the syntax supported by this function, see the documentation for DB2 Net Search Engine (NSE). Note that the contains-text function should be equated with 1 to be true and 0 to be false. The score function uses the ranking information returned by NSE, which is used in this case to sort the resulting journal articles through SORTBY.

Example 8: text search (contains-text function and attribute sorting)

This query finds all journals that have either the word “Design” or the word“ Index” in their title and sorts the results in descending order by their title.

```
/Journal
[Journal_Article[contains-text(@Title, " 'Design' | 'Index'  ")=1]]
SORTBY (@Title DESCENDING)
```

Explanation: This is another example of performing text search using the contains-text function. The sorting in this case uses the DESCENDING operator on the “Title ”attribute. The default for the SORTBY is ASCENDING.

Example 9: text search (contains-tex-basic and score-basic functions)

This query finds all journal articles that contain the text “Java” and the text “JDK 1.3” but not the text “XML” using the simplified (basic) text search syntax and sort the results by the text search score.

```
//Journal_Article
[contains-text-basic(@Title, " +Java -XML +'JDK 1.3'")=1]
SORTBY (score-basic(@Title, " +Java -XML +'JDK 1.3' "))
```

Explanation: This is an example of performing text search using the simplified text search syntax. Use a“ +” to indicate the words or phrases that should be present in the attribute“ Title”, and, similarly, use a “-” to exclude other words or phrases. The score-basic function works similarly to the score function in the previous example, but uses a simplified syntax.

Example 10: text search on resource items

This query finds text resources in a text resource item type “TextResource” that contain the text “Java” and the text “XML”.

```
/TextResource[contains-text(@TIEREF, " 'Java' & 'XML'
")=1]
```

Explanation: This is an example of performing text search inside of the resources in the resource manager. Note that the “TIEREF” attribute is used as a representation of the resource that is represented by the item of type “TextResource”. TIE syntax is used as usual in this case inside the contains-text function. For the syntax supported by this function, see the documentation for DB2 Net Search Engine (NSE).

Example 11: traversal of references in the forward direction

This query finds all the frequently asked questions for conferences, for which the conference notes refer to books with titles mentioning EIP.

```
 /Conference/Conference_Note [@PublicationRef =>
Book[@Title LIKE "%EIP%"]]
/Conference_FAQ
```

Example 12: traversal of references in the forward direction

This query finds all chapters of books referenced in the notes of conferences related to Internet.

```
/Conference[@Title LIKE "%Internet%"]
/Conference_Note/@PublicationRef =>
*/Book_Chapter
```

Example 13: traversal of references in the reverse direction

This query finds all the components that have references pointing to any books.

```
/Book/REFERENCEDBY/@REFERENCER => *
```

Example 14: traversal of references in the reverse direction

This query finds all the frequently asked questions under conference notes that refer to books about XML.

```
/Book[@Title LIKE "XML"]/REFERENCEDBY/@REFERENCER =>
Conference_Note/Conference_FAQ
```

Explanation: Note that since the reference attributes originate inside of the Conference_Note component, this is the component that must appear as the first component after the dereference operator. This query produces an empty result set if, for example, Conference follows the“ =>” operator.

Example 15: traversal of references in the reverse direction

This query finds all the components that contain XML in their remarks and that have references pointing to books.

```
/Book/REFERENCEDBY/@REFERENCER =>
*[@Remark LIKE "%XML%"]
```

Example 16: latest version function

This query finds all the journals of the latest version. By default, all versions of the indicated component type view that match the query are returned. VERSIONID is a system-defined attribute that is contained in every component type.

```
/Journal[@VERSIONID = latest-version(.)]
```

Example 17: latest version function on the target of the dereference

This query finds all the books of the latest version that are referenced in the notes of any conferences.

```
/Conference/Conference_Note/@SYSREFERENCEATTRS =>
Book[@VERSIONID = latest-version(.)]
```

Example 18: latest version function on wildcard components

This query finds all the components of the latest version that have references pointing to any books.

```
/Book/REFERENCEDBY/@REFERENCER => *
[@VERSIONID = latest-version(.)]
```

Example 19: system-defined attributes

This query finds all the root components with a specific item ID.

```
/*[@ITEMID =
"A1001001A01J09B00241C95000"]
```

Example 20: text search on document model

This query finds all documents that contain the word “XML” in any one of its parts.

```
/Doc[contains-text(.//ICMPARTS/@TIEREF, " 'XML' ")=1]
```

Explanation: The query language offers a virtual component “ICMPARTS” that allows access to all the ICM Parts item types contained under a specific item type of Document classification.

Example 21: document model (access to ICM Parts)

This query finds all the parts of the document with the storage ID of 555.

```
/Doc[@ArchiveID = 555]/ICMPARTS/
@SYSREFERENCEATTRS => *
```

Example 22: document model (access to ICM Parts)

This query finds all the parts in all of the documents in the system.

```
 //ICMPARTS/@SYSREFERENCEATTRS => *
```

Explanation: Because both the Doc and Paper item types have been defined as being Documents in the system, the ICM Parts from both of them are returned in the result.

Example 23: existence of attributes

This query finds all root components that have a title.

```
  /*[@Title]
```

Explanation: To eliminate the restriction that only root components should be returned, the query can be rewritten to start with a double-slash

```
//*[@Title]
```

Example 24: list of both literals and expressions

This query finds all journals that have a title that is equal to either its article’s title, its section’s title, or ″IBM Systems Journal″.

```
/Journal[@Title = [Journal_Article/@Title,
.//Journal_Section/@Title,"IBM Systems Journal"]]
```

Example 25: list of numeric literals

This query finds all books that cost either $10, or $20, or $30.

```
 /Book[@Cost = [10, 20, 30]]
```

Example 26: list of a result of query

This query finds all journals or all books with the title "Star Wars".

```
 [/Journal, /Book[@Title = "Star Wars"]]
```

Example 27: attribute groups

This query finds all details on documents in which the description is at least 20 pages long.

```
/Doc[Doc_Description/@PageSummary.NumPages >=
20]//Doc_Details
```

Explanation: Note that if an attribute (for example, "NumPages") is contained in an attribute group (for example, "PageSummary"), then you must refer to that attribute as GroupName.AttrName (for example, PageSummary.NumPages). The attribute "@NumPages" would not be found under Doc_Description.

Intermediate results obtained by INTERSECT/EXCEPT cannot be combined with arithmetic(unary/binary) or comparison operators. They can be combined by set operators (UNION/INTERSECT/EXCEPT) or appear by themselves.

Examples of valid usage of UNION/INTERSECT/EXCEPT:

1.
   ```
    (/Journal/Journal_Article[@Title = "Content Management"]
   EXCEPT
   //Journal_Article[@Classification =
   "Security"])/Journal_Section
   ```

   This query is valid because the result of the EXCEPT is the result of the entire query - it is not combined using any operators.

2.
   ```
   /Journal[(Journal_Editor/@LastName
   UNION .//Journal_Author/@LastName) = "Davis"]
   ```

   This query is valid because there is no restriction on UNION operator.

3.
   ```
   /Journal[Journal_Article[Journal_Section/@Title INTERSECT
   .//Journal_Figure/@Caption]/@Title = "Content Management"]
   ```

   This query is valid because the result of INTERSECT is not combined using any operator.

4.
   ```
   /Journal[@Title = "VLDB"]
   UNION /Journal[@Cost = 20]
   INTERSECT /Journal[@Organization = "ACM"]
   ```

   This query is valid because the result of INTERSECT operator is combined using a set operator (UNION).

Examples of invalid usage of INTERSECT/EXCEPT:

1.
   ```
   /Journal[(Journal_Editor/@LastName
   INTERSECT .//Journal_Author/@LastName) = "Davis"]
   ```

   This query is invalid because the result of INTERSECT operator is combined using a comparison operator (=).

2.
   ```
   /Journal[(.//Journal_Section/@SectionNum
   EXCEPT .//Journal_Figure/@FigureNum) + 5 = 10]
   ```

   This query is invalid because the result of EXCEPT is combined using an arithmetic operator (+).

Understanding the query language

To support advanced features of the query language (like the wildcards ″%″ or ″_″ inside of text strings), escape sequences are used to differentiate between the cases when wildcards are treated as regular characters versus when they are given the special meaning of wildcard characters. For the user, it is important to know which characters are used as wildcards because wildcard characters, when intended to be treated as regular characters, must be preceded by an escape character. Escape sequences are also used to handle single and double quotes.

You need to add escape sequences when the strings used in queries contain either special characters (double-quote, apostrophe) or wildcard characters (percent sign, underscore, star, question mark) or a default escape character (a backlash). This handling is the simplest for strings used in comparison conditions and becomes a bit more involved for the LIKE operator and text search functions. Proper handling of special characters will ensure successful execution of queries and correctness of query results.

Important: Use wildcard characters sparingly as using them in your queries can increase the size of your result list significantly, which can decrease performance and return unexpected search results.

Using escape sequences with comparison operators (″=″, ″!=″, ″>″, ″<″, ″BETWEEN″ and others)

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[@Title = "Analysis of ""The Time Machine"" by H.
G. Wells himself"]
```

Since the article’s title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark (apostrophe) ′

You do not need to escape in this case.

Example:

/Book[@Title != ″Uncle Tom’s Cabin″]

Using escape sequences with the LIKE operator

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[@Title LIKE "Analysis of ""The Time Machine"" %"]
```

Since the article’s title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark (apostrophe) ′

You do not need to escape in this case.

Example:

```
 /Book[@Title LIKE "Uncle Tom's Cabin"]
```

Wildcards (″%″, ″_″)

The percent sign ″%″ is a wildcard character used to represent any number of arbitrary characters in a string used with the LIKE operator. The underscore ″_″ is a wildcard character used to represent a single arbitrary character. If you want these wildcard characters to be treated as regular characters, you need to do the following:

1. Precede the wildcard character with an escape character
2. Add an ESCAPE clause with the escape character after the LIKE phrase.

Example A:

```
/Book[@Title LIKE "Plato%s%S_mposium"]
```

This example shows how wildcards ″%″ and ″_″ are used to find a book whose title's spelling is uncertain.

Example B:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ in query"
ESCAPE "!"]
```

Since the search string in this example contains the underscore ″_″ as a regular character (not a wildcard), you can escape the underscore with an exclamation point character ″!″. Any single character can be used as an escape character.

Example C:

```
//Journal_Article[@Title LIKE "_sage of underscore \_ in%" ESCAPE
"\"]
```

In this query, wildcard characters are used as both regular characters (″_″ escaped by ″\″) and as wildcards (″_″) to catch both uppercase and lowercase versions of the word ″Usage″, as well as ″%″ to catch multiple endings of the string.

Example D:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ on Yahoo!!"
ESCAPE "!"]
```

You can also use an escape character as a regular character. To do so, precede the escape character with itself, as in the example to search for ″Yahoo!″ below.

Using escape sequences with advanced text search (″contains″ and ″score″ functions)

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[contains-text (@Title, " 'Analysis of ""The Time
Machine"" %' ")=1]
```

Since the article’s title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark(apostrophe) ’

Precede the apostrophe with another apostrophe. A single apostrophe is not allowed in advanced text search because a set of apostrophes is used to enclose a term or a phrase. If an apostrophe appears inside a term, then the apostrophe needs to be escaped to differentiate it from the apostrophe that ends the term or the phrase.

Example A:

```
/Book[contains-text (@Title, " 'Uncle Tom''s Cabin' ")=1] SORTBY
(score (@Title, " 'Uncle Tom''s Cabin' "))
```

Note that Tom’’s has two apostrophes.

Example B:

```
/Book[contains-text (@Title, " ('Greek' & 'Plato''s Symposium') &
NOT ' Socrates' ")=1] SORTBY (score (@Title, " ('Greek' & 'Plato''s
Symposium') & NOT ' Socrates' "))
```

Note that Plato’’s has two apostrophes.

Wildcards (″%″, ″_″)

Just as the LIKE operator, advanced syntax uses ″%″ and ″_″ as wildcards. The percent sign ″%″ is a wildcard character used to represent any number of arbitrary characters. The underscore ″_″ is a wildcard character used to represent a single arbitrary character. If you want a wildcard character to be treated as a regular character, you need to do the following:

1. Precede the wildcard character with an escape character
2. Add an ESCAPE clause after EACH term where you use the escape character

Example A:

```
/Book[contains-text (@Title, " 'Usage of underscore !_ in query'
ESCAPE '!' ")=1] SORTBY (score (@Title, " 'Usage of underscore !_ in
query' ESCAPE '!' "))
```

In this example, an exclamation mark ″!″ is used as an escape character before the underscore.

Example B:

```
/Book[contains-text (@Title, " 'Usage of underscore !_ in query'
ESCAPE '!' | 'Yahoo! For Dummies' | 'Usage of underscore !_ on
Yahoo!!' ESCAPE '!' | 'War and Peace' ")=1]
```

Note that an ESCAPE clause must be added after every term in your text search string where you escape wildcards, even if the escape character is the same in all the terms.

Using escape sequences with basic text search (″contains-text-basic″ and ″score-basic″ functions)

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[contains-text-basic (@Title, "Analysis of '""The
Time Machine""' ")=1]
```

Since the article's title contains the name of the book in double quotes, "The Time Machine", these internal double quotes need to be escaped. The book title is inclosed in apostrophes to keep it as a phrase.

Single quotation mark (apostrophe) '

Precede the apostrophe with another apostrophe. Basic text search syntax allows terms enclosed within single quotes, so that a term can contain a space. The doubling of the apostrophe is therefore necessary to differentiate the case of an apostrophe occurring within a term from the case of an apostrophe starting a new term.

Example A:

```
/Book[contains-text-basic (@Title, "Uncle Tom''s Cabin")=1]SORTBY
(score-basic (@Title, "Uncle Tom''s Cabin"))
```

Note that Tom''s has two apostrophes.

Example B:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato''s Symposium'
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek +'Plato''s
Symposium' -Socrates "))
```

Note that Plato''s has two apostrophes and 'Plato's Symposium' is enclosed in single quotes since it is a phrase.

Wildcards ("*", "?" and "\")

Precede "*", "?", and "\" characters with a backslash "\" if these characters are not to be treated as wildcards. Star "*" is a wildcard character used to represent any number of arbitrary characters in basic text search for the functions contains-text-basic and score-basic. The question mark "?" is a wildcard character used to represent a single arbitrary character. For basic text search, the query language provides an escape character backslash "\" to be used when the term to be searched contains the wildcard character in it and you want to treat that wildcard character as a regular character.

Example A:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato*s*S?mposium'
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek
+'Plato*s*S?mposium' -Socrates "))
```

This example shows how to use basic text search when the spelling of a term is not certain. The "*" and "?" characters are meant to be wildcards in this case, so they are not escaped.

Example B:

```
/Book[contains-text-basic (@Title, "Why forgive\?")=1] SORTBY
(score-basic (@Title, "Why forgive\?"))
```

In this example, the title contains the question mark "?" as a normal character, so this character can be escaped with a backslash.

Example C:

```
//Journal_Section[contains-text-basic (@Title,
"C:\\OurWork\\IsNeverDone")=1] SORTBY (score-basic (@Title,
"C:\\OurWork\\IsNeverDone"))
```

Each backslash that naturally occurs in the search term"`C:\OurWork\IsNeverDone`" must be escaped with another backslash.

Using escape sequences in Java and C++

Precede special characters (for example, double quotes and backslash) with a backslash.

Example:

Query:

/Book[contains-text-basic (@Title, ″Why forgive\?″)=1]

Java

```
String query = "/Book[contains-text-basic (@Title, \"Why forgive\\?\")=1]";
```

C++

```
DKString query ("/Book[contains-text-basic (@Title, \"Why forgive\\?\")=1]");
```

Note how the internal double quotes and the backslash before the question mark are preceded by a backslash. This handling is inherent to Java and C++ programming languages. For more information, refer to the specifications for these languages.

The query language grammar

The query language formal grammar is as follows:

- (* keywords *)
- 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″);
- KEYWORD = (AND | ASCENDING | BETWEEN | DESCENDING | DIV | EXCEPT | INTERSECT | LIKE | MOD | NOT | OR | SORTBY | UNION | IS | NULL) ;
- (* literals *)
- 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 is the set of all unicode characters and escape sequences. It’s definition is not included in this document *) (* String literals are delimited by double quotes and can contain any character except double quote. To include a double quote as the part of the string literal, specify two consecutive double quotes i.e. a double quote is escaped by another double quote. These will be treated as one double quote character *)
- STRING_LITERAL = ′″′, { (UNICODE_CHARACTER - ′″′) | (′″′, ′″′) } , ′″′ ;
- (* Escape sequence is a single character delimited by double quotes. To specify a double quote itself as the escape character, specify two consecutive double quotes, as in a double quote is escaped by another double quote. These will be treated as one double quote character. For the complete explanation of the legal values for ESCAPE_CHARACTER, please see the DB2 SQL Reference section on the LIKE Predicate. *)
- 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″ | ″_″ | ″$″) ;
- (* An IDENTIFIER begins with a letter (a-z, A-Z) or an underscore or a dollar character, followed by zero or more letters, underscores, dollar characters or digits (0-9). A keyword can be an IDENTIFIER only if it is enclosed within single quotes *)
- 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;

Working with the resource manager

A Content Manager resource manager controls a collection of managed resources (objects). It also manages the necessary storage and Hierarchical Storage Management (HSM) infrastructure, but you must first configure the resource manager to support HSM. Resource managers have facilities to support type-specific services for more than one type of object, such as streaming, zipping, unzipping, encrypting, encoding, transcoding, searching, or text mining.

A single resource manager is used exclusively by one library server. Each resource manager delivered by the Content Manager system provides a common subset of native data access APIs through which it is accessible by the controlling library server, by other Content Manager components, and by applications, either locally (on the same network node) or remotely.

Other data access APIs allow remote access to a resource manager using the resource manager's own client support or a standard network access protocol such as CIFS, NFS, or FTP. For remote access, use a client-server connection. Clients communicate with Content Manager resource managers using HTTP through the use of a standard Web server. Data delivery is based on HTTP, FTP, and FS data transfer protocols. Using HTTP, any application or Content Manager component that needs to access Content Manager-managed content can dynamically form a triangle with a library server and resource manager. This triangle forms a direct data access path between the application and each resource manager, and a control path between the library server and the resource manager. You can map this conceptual triangle to any network configuration, ranging from a single-node configuration to a geographically distributed one.

The new architecture also accommodates resource managers that an application is not able to access directly, such as a host-based subsystem, a single-user system that does not handle access control, or a system containing highly sensitive information where direct access by an application is not allowed by business policy. In this case, access to such resource managers is indirect. Both the pull and the push paradigms of data transfer are accommodated by the Content Manager system as well as synchronous and asynchronous calls.

For information about how to configure a resource manager see *Planning and Installing Your Content Management System* and the `SResourceMgrDefCreationICM` sample in the `samples` directory, `cmbroot\samples\java\icm`.

Working with resource manager objects

Within Content Manager, every managed entity is called an item. Items come in two types, the type that represent pure logical entities, such as documents or folders, or entities that represent physical data objects, such as the text data of a word processing document, the scanned image of a claim or the video clip of an automobile accident. Objects have a special state and behavior needed to handle the physical data associated to a logical document.

Resource objects also represent things like files in a file system, video clips in a video server, and BLOBs. At run time, resource objects are used to access the physical data they point to. For that reason, resource objects in Content Manager have a type. That is, they have a specific state and behavior. The library server and the resource manager share a schema to store the state of an object. The base object types provided by Content Manager are: generic BLOBs or CLOBs, Text, Image, and Video content objects. You can also create sub-classes of the pre-defined types. A resource object can also have user-defined attributes, which are used for search and retrieval.

From the Content Manager system perspective, each object is represented by a unique logical identifier, its Uniform Resource Identifier (URI). The library server manages the URI name space. On request, the library server maps URIs onto Uniform Resource Locators (URL). URLs are used to gain access to the physical data. URLs do not point directly to a storage area managed by the resource manager. Instead, the resource manager uses a local name space to convert logical object names to physical file names. Object URIs are created by the specific resource manager. The library server or the end-user can suggest an object URI (its name), but the decision is made by the resource manager.

You can access an object using the Content Manager resource manager APIs (store, retrieve, update, delete, and so forth). In some cases, you can use APIs that are native to the object (stream, multicast, and stage) or file system.

For information about how to work with resource manager objects, see the SResourceItemCreationICM sample in the samples directory, CMBROOT\Samples\java\icm or CMBROOT\Samples\cpp\icm.

Managing documents in Content Manager

Content Manager implements a flexible document management data model that you can use for managing business objects. The basic elements of the data model includes folders, documents, and objects.

As mentioned earlier, documents, folders, and other objects are all represented by items in the Content Manager system. From the API level, the only difference between a document and a folder is the semantic type and the respective DKFolder functionality. A document is comprised of attributes, or metadata, that describe the document, including single valued attributes (document name, date, subject), multi-valued attributes (keywords), and collections of multi-valued attributes (address, consisting of street, city, state, and zip).

The document management data model uses document parts to associate objects (resource items) with the document. This model supports more than one part to construct a document. For example, each page could be a separate part. In order for an application to determine the order of the parts that make up a document, a part number is stored in the document parts. The document parts contains a pointer to the object (a reference attribute) which contains other information about the part such as MIME type, size, the resource manager ID which contains the part, the collection name on that resource manager and so forth. Every object can have different attributes. For example, an annotation might have X and Y coordinates, while a note log might have the CCSID of the text of the note.

To better understand the document management data model, consider the following scenario of a user who imports a document using a client application:

- A window is displayed to the user.
- The user enters (or selects) the name of the file to import into the system. For example, a police report.
- The user selects the type of document (memo, claim, design).
- A new window opens, where the user can enter attributes that describe the document. The date, a claim number, and insurance policy number, for example.
- The user defines a document parts and enters some values for the attributes. The police report is a document parts of a claim, for example.
- The user finishes entering the document descriptions and completes the task. The police report is created.

Using either the APIs or the JavaBeans, the client application then connects to the library server and the resource manager. The system creates two items (a non-resource item and a resource item) to store the document. Two items are created because the police report contains a photograph, which is stored in the resource manager. The document is created with a single API call. The object is then stored in the resource manager and the resource manager returns the timestamp, and other metadata for the object. The resource manager creates a reference attribute for the object, and inserts the reference attribute into the child component of the document. A final call to the library server is made to store the child component and to update the attributes. The entire process is bound by a transaction so that any API failure does not result in a partially created document.

After you create a document, you can update it. You can perform two types of updates: change the metadata or change the content. The library server automatically creates a new item record with the next version number (if the item is version-enabled) and copies all of the child components associated with the item.

Creating the document management data model

This section helps you complete the main tasks associated with the document management data model:

- Creating a document item type.
- Create a document.
- Updating a document.
- Retrieving and deleting a document.
- Versioning of parts in the document management data model.

Creating a document item type

Java:

1. Create a document ItemType with ItemType classification = DK_ICM_ITEMTYPE_CLASS_DOC_MODEL, set the following.

   ```
   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. Create ItemType relation to add Parts to document. Retrieve EntityDef for each Part.

   ```
   Parttype = (DKItemTypeDefICM) dsDef.retrieveEntity(PartName);
   ```

3. For each part, create a ItemType Relation and set the values.

   ```
   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. Add the ItemType relation to the Document (Source).

   ```
   docItemTypeDef.addItemTypeRelation(itRel);
   ```

5. Add the document ItemType to persistent store.

   ```
   docItemTypeDef.add();
   ```

C++:

1. Retrieve the content server definition object.

   ```
   DKDatastoreDefICM * dsDefICM =  (DKDatastoreDefICM *)dsICM->datastoreDef();
   ```

2. Retrieve the content server administration object.

   ```
   DKDatastoreAdminICM * pdsAdmin =
     (DKDatastoreAdminICM *)dsDefICM->datastoreAdmin();
   DKItemTypeDefICM *itemType = NULL;
   DKItemTypeRelationDefICM *itemTypeRel = NULL;
   DKAttrDefICM* attr = NULL;
   ```

3. Retrieve an attribute for the document item type. If the attribute does not exist, create it.

```
dkAttrDef *pAttr = dsDefICM->retrieveAttr("docTitle1");
if(pAttr == NULL)
{
     attr = new DKAttrDefICM(dsICM);
     attr->setName("docTitle1"); //attribute name column name
     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("This is a test Item Type");
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. Create a relation between the document that you just created and part1. To do that, first retrieve EntityDef for each Part.

   ```
   DKItemTypeDefICM *itemTypePart1 = (DKItemTypeDefICM *)
      dsDefICM->retrieveEntity("ICMBASE");
   //int part1ITypeid = itemTypePart1->getItemTypeId();
   int part1ITypeid = itemTypePart1->getIntId();
   ```

5. For each part, create an item type relation and set the values.

   ```
   DKItemTypeRelationDefICM *itemTypeRelPart1=
     new DKItemTypeRelationDefICM(dsICM);
   itemTypeRelPart1->setTargetItemTypeID(part1ITypeid);
   //sets the default resource manager
   itemTypeRelPart1->setDefaultRMCode((short)1);
   //sets the default ACL code
   itemTypeRelPart1->setDefaultACLCode(1);
   ```

6. Set the default collection where item resources pertaining to this item type are to be stored.

   ```
   itemTypeRelPart1->setDefaultCollCode((short)1);
   ```

7. Set the default prefetch collection where item resources pertaining to this item type are to be stored.

   ```
   itemTypeRelPart1->setDefaultPrefetchCollCode((short)1);
   itemTypeRelPart1->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);
   itemTypeRelPart1->setSourceItemTypeID(itemType->getIntId());
   ```

8. add the item type relation to the document (source).

   ```
   itemType->addItemTypeRelation(itemTypeRelPart1);
   ```

9. Create a relation between the document that you just created and part2. To do that, first retrieve EntityDef for each part.

   ```
   DKItemTypeDefICM *itemTypePart2 = (DKItemTypeDefICM *)
      dsDefICM->retrieveEntity("ICMANNOTATION");
   int part2ITypeid = itemTypePart2->getIntId();
   ```

10. For each part, create an item type relation and set values.

    ```
    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. Add the item type relation to the document (source).

    ```
    itemType->addItemTypeRelation(itemTypeRelPart2);
    ```

12. Update the definition of the item type in the library server.

    ```
    itemType->add();
    ```

For additional information, see the SItemTypeCreationICM sample.

Creating a document

An item with the ″Document″ semantic type can contain attributes (like items of other semantic types) and multiple ″parts″ (unlike items of other semantic types) inside it. The steps below take you through the process of creating an item (based on a pre-defined document item type) that contains one attribute and one ″part″. Note that in the steps below, it is assumed that an item type called ″s_simple,″ with one attribute, called ″S_varchar, ″ and one part (″ICMBASE″) has already been defined.

Java

1. Create the document DDO.

```
DKDDO ddoDocument = dsICM.createDDO("S_simple",
       DKConstant.DK_CM_DOCUMENT);
short    dataId   = 0;
String attrValue  = "Test";
```

2. Set the document's attribute. In this case, we assume that the item type has only one attribute.

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
DKParts  parts    = null;
// Document's parts
```

3. Access the document's parts collection.

```
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. Create a part of pre-defined type "ICMBASE". This part will be added to the created document. It is assumed that the document created below is based on an item type with only one part.

```
DKLobICM pLobPart = (DKLobICM) dsICM.createDDO("ICMBASE",
   DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
pLobPart.setPartNumber(1);
// Set the mime type for added part
pLobPart.setMimeType("text/plain");
String partValue = "This is a base part";
pLobPart.setContent(partValue.getBytes());
```

5. Add the created part to the "parts" collection. Note that this is a deferred save (the change is not committed to the datastore until the document DDO is persisted).

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. Persist document to the datastore.

```
ddoDocument.add();
```

C++

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Create a new DDO of type DocModelTest and semantic type DK_CM_DOCUMENT
DKDDO* ddoDocument = dsICM->createDDO("DocModelTest",DK_CM_DOCUMENT);
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
      DKString("docTitle1")),DKString("this is a string value"));
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
      pdsDef->retrieveEntity("DocModelTest");
// Retrieve the collection of DKItemTypeRelationDefICM object for given source
// item type from the persistent store
dkCollection* pRelationColl = itemType->retrieveItemTypeRelations();
dkIterator* pIter = pRelationColl->createIterator();
int noOfPartsToCreate = pRelationColl->cardinality();
DKParts* pPartColl= NULL;
// Create the parts collection for the object if it does not exist
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;
// CV v8 BLOB
DKLobICM* pPart =NULL;
DKString str = "This is to test the document model with two parts";
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);
}
//Add the DDO to the datastore
ddoDocument->add();
```

For more information, see the `SDocModelItemICM` sample.

Updating a document

The steps below take you through the process of updating an item of semantic type ″Document.″ In the steps below, a new part is added and the attribute value is updated.

Java

1. Update the attribute value for the document item.

   ```
   String attrValue = "New Value";
   short dataId=ddoDocument.dataId
     (DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
   ddoDocument.setData(dataId,attrValue);
   ```

2. Access the document's parts collection.

   ```
   DKParts  parts      = null;
   // Document's parts
   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. Create data for the new part.

   ```
   String partValue = "This is an annotation";
   ```

4. Create a part of pre-defined type ″ICMANNOTATION″. This part will be added to the created document. Here, it is assumed that the document being created is based on an item type with only one part. Once the new part is added, the document will have two parts.

   ```
   DKLobICM pLobPart = (DKLobICM)dsICM.createDDO("ICMANNOTATION",
      DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
   pLobPart.setContent(partValue.getBytes());
   pLobPart.setPartNumber(2);
   ```

5. Add the created part to the ″parts″ collection. Note that this is a deferred save (the change is not committed to the datastore till the document DDO is persisted).

   ```
   parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
   ```

6. Persist the changed document to the datastore.

   ```
   ddoDocument.update();
   ```

C++

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Create a Document DDO from a PID string
DKString pidString = ....;
//...
DKDDO* ddoDocument = dsICM->createDDO(pidString);
DKPidICM* pPID = (DKPidICM*) ddoDocument->getPidObject();
DKString* pstrItemType = &pPID->getObjectType();
// Retrieves the definition for the item type "DocModelTest" from the
// persistent datastore. The definition is returned as a the *dkEntityDef
// object that is in turn typecast to a DKItemTypeDefICM object
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
          pdsDef->retrieveEntity(*pstrItemType);
ddoDocument->setData(ddoDocument->dataId(
                     DK_CM_NAMESPACE_ATTR,DKString("docTitle1")),
                     DKString("this is a new string value"));

DKString updateString = "This is updated part";
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; // quit
pPart = (DKLobICM *) pSeqIter->next();
// Update the existing part with the new content
pPart->setContent(updateString);
// Add a new part to the Document
DKLobICM* pPart1 =
  (DKLobICM*) dsICM->createDDO ("ICMNOTELOG", DK_CM_RESOURCE);
pPart1->setPartNumber(3);
DKString pTempData = "This is to test the document model with two parts";
pPart1->setContent(pTempData);
DKAny any = (dkDataObjectBase*)(DKDDO*) pPart1;
pParts->addElement(any);
//Update the DDO information in the datastore.
ddoDocument->update();
delete pSeqIter;
```

For additional information, see the SDocModelItemICM sample.

Retrieving and deleting a document

To retrieve a document, call ddo.retrieve(option). If you set option to DK_ICM_CONTENT_YES, the parts TOC list, as well as the parts, is retrieved. Otherwise, only the TOC list of the parts is retrieved.

To delete a document, call ddo.del(). The document and its attached parts is deleted. For more information about retrieving and deleting documents with parts, refer to the SDocModelItemICM API sample.

Versioning of parts in the document management data model

Versioning properties of document parts are determined by the versioning properties of the item type relation to each part type selected in the document's item type.Versioning characteristics of document parts include the following:

- Like regular documents, parts can have one of three versioning models: versioned-always, versioned-never (default) and application-controlled versioning.
- If an item type has a version policy of versioned-never, its parts also have a versioning policy of versioned-never.
- If an item type T has a version-policy of versioned-always and an item I of item type T and you modify (by adding/deleting or updating a part) either its attributes or its parts collection, a new version of item I is created.
- Parts of documents, unlike documents themselves, do not have a maximum number of versions.
- You can obtain part-level versioning rules from the item type relation object for the part of interest (base, note, annotation etc.).

The examples below show how to get the versioning rules for an item type’s base part.

Java

```
String itemTypeName="book";  //example item type
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM item =null;
DKDatastoreICM ds = new DKDatastoreICM();
...
item = (DKItemTypeDefICM)ds.datastoreDef.retrieveEntity(itemTypeName;
DKItemTypeRelationDefICM itemRelation =
(DKItemTypeRelationDefICM)item.retrieveItemTypeRelation(partId);
versionControlPolicy = itemTypeRelation.getVersionControl();
```

C++

```
DKString itemTypeName="book";  //example item type
//Specify the part id for which we need versioning information
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM  * itemType;
//Retrieve the entity corresponding to the "book" item type
itemType =
 (DKItemTypeDefICM * )dsICM->datastoreDef()->retrieveEntity(itemTypeName);
//Retrieve the relation object for the specified part id
DKItemTypeRelationDefICM * itemTypeRelation =
  (DKItemTypeRelationDefICM*)itemType->retrieveItemTypeRelation(partId);
//Retrieve the version control policy for the specified part
int versionControlPolicy = itemTypeRelation->getVersionControl();
```

Working with transactions

Transactions allow Content Manager to maintain consistency between the library server and any adjoining resource manager. A transaction is a user-determined, recoverable, unit of work, that consists of one or more consecutive API calls made through a single connection to the library server. The sequence of consecutive DKDatastoreICM method calls are made either directly or indirectly, through the DDOs and XDOs.

The scope of a transaction and the amount of work within that transaction is by default the work performed by a single API method (implicit transaction). This type of transaction is recommended and is the best performing scope of a transaction. You can, however, change the scope of a unit of work, making it larger to include multiple method calls (explicit transaction), but using this type of transaction can introduce some performance overhead.

When a transaction ends, the entire transaction is either committed or rolled back. If it is committed, all of the Content Manager server changes made by API calls within the transaction are permanent. If a transaction is rolled back or fails, all the changes made within the transaction are reversed during rollback processing.

The commit and rollback of a transaction are done automatically in the case of implicit transactions. In the case where explicit transactions are in use, the transaction commit is controlled by the application, whereas a transaction rollback can be initiated by an application or automatically by the Content Manager system. The Content Manager system initiates a rollback when a severe error occurs or when it is necessary to resolve a deadlock between the library server and the database.

Within a transaction, uncommitted resource manager changes are not visible to the application that made the changes until the transaction is committed. For example, you make changes to a resource manager item and you store it. If you retrieve that item before the transaction is committed, the item will not reflect the changes that you just made. You will not see the updated item until the transaction is committed.

Concurrent or overlapping transactions through a single library server connection are not supported. To maintain concurrent transactions, you must make multiple connections between the library server and the database, or initiate multiple client processes or threads if you are working with a client application. Applications like IBM WebSphere ® Application Server handle processes, connections, and sessions.

The execute() and executeWithCallback() methods in DKDatastoreICM automatically create an additional connection to the database when invoked. The new database connection is then used to execute the query. Since queries use a separate database connection, they also have a separate transaction scope from other content server operations. The connection to the database is closed (or returned to the pool, if pooling is enabled) when the DKResultSetCursor is closed.

Things to consider when designing transactions in your application

If a client node or library server fails before the transaction is committed, the database recovery function rolls back the transaction on the library server immediately. The resource manager changes made during the failure are undone immediately if the client node and resource manager are both active. If the client node itself failed, you should put the resource manager through a cycle of the Asynchronous Recovery Utility in order to restore consistency between the resource manager and the library server. Before the utility runs, the servers still have data integrity. What is affected are operations on the in-progress items that had the failure, which will be rejected until the RM is recovered. Failure during in-progress update of an object prevents another update of that same object, until the first failure is reconciled.

If the resource manager fails, you should run the asynchronous recovery utility to remove inconsistencies. On OS/390, the resource manager has native transaction capabilities, such as Object Access Method (OAM), which are used to recover more expediently.

Caution when using explicit transactions

For explicit transactions, where you control the transaction scope using DKDatastoreICM.startTransaction() and DKDatastoreICM.commit(), use caution when developing an application where you work with Content Manager Documents with parts and when performing DKLobICM create, retrieve, update, and delete (CRUD) operations. When performing these operations, you should perform CRUD operations as close as possible to the end of the transaction. You should also keep a transaction as short as possible, since a long transaction increases the potential for database locking problems.

Locking problems are most apparent when updating an item, and the application chooses to not commit the transaction immediately. As long as the transaction is not committed, the item that is being updated, is still visible to other applications. When another user attempts to access or view the item, that user is locked out until the update transaction is committed. The same problem (database locking) occurs when creating new items in a folder. If the folder is visible to another user, and that user attempts to retrieve the new item, the user is locked out until the transaction is committed. The amount if time prior to the transaction commit is the amount of time the user is locked out.

The best approach to avoiding database locking is to commit transactions often and to avoid long running transactions. If you must perform CRUD operations within a transaction, it is recommended that you perform these operations when it is understood that no one else will access the items being updated.

Using check-in and check-out in transactions

Content Manager supports check-out and check-in operations on items. The check-out operation is called to acquire a persistent write lock for items. When an item is checked out by a user, other users can not update it although they can still retrieve and view it. You need to call the check-out operation prior to updating or re-indexing an item, regardless of the transaction mode (implicit or explicit) that you use. When you are done with the item, call the check-in operation to release the persistent lock and make the item available for other users to update. After you create an item, you have the option to keep it in checked out state to prevent other users from changing it until you are completely done with the work. If you check-out (or check-in) an item using an explicit transaction, the checkout is undone if the transaction is rolled back. If you check-out an item using an implicit transaction, the checkout is committed. It is the application's responsibility to check the item back in, using `checkin` options or methods.

Processing transactions

The transaction scope can be controlled by a client API call, but it must be designed carefully. To group a set of API calls into a transaction, you must build it explicitly by completing the following steps:

1. Call the `startTransaction()` method of the `DKDatastoreICM` class. You work with the `DKDatastoreICM` methods to complete all the transaction steps.
2. Call all of the APIs that you want to include in the transaction in the order that you want them called.
3. Call the `commit` or `rollback` methods to end the transaction.

All of the API calls made between the startTransaction() and either commit() or rollback(), are treated as one transaction.

All APIs can be included in transactions, unless specifically noted. See the Online API Reference for details. Some administrative APIs cannot be included in explicit transactions. For example, the method to define or update item types.

Below is the list of class methods involved in Content Manager transactions in relation to item creation and update.:

DKDatastoreICM.startTransaction()
Starts an explicit transaction.

DKDatastoreICM.commit()
Commits transaction changes to the database.

DKDatastoreICM.rollback()
Rolls back or removes transaction changes from the database.

DKDatastoreICM.checkOut()
Acquires a persistent write lock on an item.

DKDatastoreICM.checkIn()
Releases a previously acquired persistent write lock.

DKDatastoreICM.add()
Creates a new item in the database.

DKDatastoreICM.updateObject()
Updates an item. The item must be checked out prior to calling this method.

DKDatastoreICM.retrieveObject()
Retrieves an item from the database.

DKDatastoreICM.deleteObject()
Deletes an item from the database.

DKDatastoreICM.moveObject()
Re-index an item. Moves an item from one item type to another item type. The item must be checked out prior to calling this method.

Another great source for information about transactions is the `SItemUpdateICM` sample.

Routing a document through a process

Content Manager provides an integrated document routing service to help you route documents through a business process. The document routing APIs enable you to build new applications using document routing, or add document routing functionality into your existing applications. Document routing provides you with the following features:

- Synchronization of all items in a document routing process because document routing functions are included in Content Manager transactions.
- Presentation of only the work that the user can access.
- Single audit trail that includes records for document creation, modification, and routing.

For basic document routing concepts and terminology, see the *System Administration Guide*. Also see the samples, because they are a great source of additional document routing information.

Understanding the document routing process

Document routing consists of processes, work nodes, work lists, and work packages. The system administrator creates the work nodes, processes, and work lists through the system administration client. A process consists of work nodes. Each work node in the process is a separate step in the process. You can create a process that branches out in several directions. The user determines which branch the work node goes to next. The user can choose from a list of possible selections that the system administrator defines. You can define a server exit when you define a work node. You can define server exits for entering a work node, leaving a work node, and to notify the user when the overload limit is reached. When a process is started, a work package is created. The work package is the routing element and contains the attributes of the work. The attributes of the work package consist of the item PID, priority, owner, and so forth.

Collection points are work nodes with additional function. A work package at a collection point node continues to the next work node in the process once the specified number of items of a specified item type exist in the specified folder. Work lists define the work packages assigned to a user. You can have one or more work lists. Each work list can include one or more work nodes. You can specify the order of the work packages in the work list by priority, or date. You can also define the order of work nodes in the work list.

When you retrieve work lists, you can filter the results to include or exclude suspended work. Work packages can also be in notify state. *Notify* state is when work packages have been at the node for longer than the time specified by the administrator. Remember that a work node can be in more than one worklist. The number of packages returned in a work list is defined by the system administrator.

The basic operations you can perform using document routing include:

- Start a process
- End a process
- Continue a process
- Suspend a process
- Resume a process
- Get work from a work list
- Get the next item from a work list
- Define, update, and delete a process
- Define, update, and delete work node
- Define, update, and delete a work list

Setting up a document routing process

There are nine APIs that you can use to implement document routing functionality into your application. You can find the details about these APIs and methods in the *online API reference*. The nine document routing APIs include:

DKDocRoutingServiceICM

This class provides the methods for managing a process: start, terminate, continue, suspend, and resume.

DKDocRoutingServiceMgmtICM

This class provides the methods to manage the helper classes:

DKProcessICM, DKWorkNodeICM, and DKWorkListICM. You can access the DKDocRoutingServiceMgmtICM object from the DKDocRoutingServiceICM object.

DKProcessICM
This class represents a process in the library server.

DKWorkNodeICM
This class represents a work node in the library server.

DKWorkListICM
This class represents a worklist in the library server.

DKRouteListEntryICM
This class defines the route that a process can take (from, to). A process object (that is, DKProcessICM) contains a collection of route entry objects (that is, DKRouteListEntryICM).

DKCollectionResumeListEntryICM
This class represents an entry in the resume list for the work nodes. A work node can contain a collection of DKCollectionResumeListEntryICM objects.

DKWorkPackageICM
This class represents a work package in the library server. When a process is started, a work package is created.

DKResumeListEntryICM
This class represents a resume list. A work package can contain a collection of resume lists.

Creating document routing service objects

The examples below demonstrate how to create a document routing object.

Java

```
//The DKDocRoutingServiceMgmtICM object is a helper class that provides
// methods to manage DKProcessICM, DKWorkNodeICM, and DKWorkListICM
// and the meta-data required to define these objects
DKDocRoutingServiceMgmtICM routingMgmt = new
        DKDocRoutingServiceMgmtICM(dsICM);

//The DKDocRoutingServiceICM class provides the core routing services
//like starting, terminating, continuing, suspending, and resuming a process.

DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);
```

C++

```
//provides methods to manage DKProcessICM, DKWorkNodeICM,
//and DKWorkListICM and the meta-data required to define these objects
DKDocRoutingServiceMgmtICM* routingMgmt = new
   DKDocRoutingServiceMgmtICM(dsICM);

//The DKDocRoutingServiceICM class provides the core routing services
// such as starting, terminating, continuing, suspending, and resuming
//a process.
DKDocRoutingServiceICM* routingService = new
   DKDocRoutingServiceICM(dsICM);
```

For a complete sample, refer to the SDocRoutingDefinitionCreationICM sample.

Defining a new regular work node

A work node is a step in a document routing process definition. It may be exited by the user application at any time. The user application is responsible for validating its own exit criteria.

Java

```
// Create new Work Node Object.
 DKWorkNodeICM workNode1 = new DKWorkNodeICM();
//Choose a Name that is 15 characters or less length
workNode1.setName("S_fillClaim");
//Choose a Description for more information than the name.
workNode1.setDescription("Claimant Fills Out Claim");
// Sets the value of the maximum time that an item can spend at this
// work node (in minutes).
workNode1.setTimeLimit(100);
// Specify max number of work packages that can be at this node
//at any given time
workNode1.setOverloadLimit(200);

// Set the type to be a regular work node
workNode1.setType(0);

//Add the new work node definition to the document routing
//managment object
routingMgmt.add(workNode1);
```

C++

```
// Create new Work Node Object.
 DKWorkNodeICM* workNode1 = new DKWorkNodeICM();
// Choose a Name that is 15 characters or less length
 workNode1->setName("ValidateCreditCard");
// Choose a Description for more information than the name.
workNode1->setDescription("Buyer's credit card is validated with
                 the credit card agency");

// Sets the value of the maximum time that an item can spend at
//this work node (in minutes).
workNode1->setTimeLimit(100);
// Specify max number of work packages that can be at this node
// at any given time
workNode1->setOverloadLimit(200);

// Set the type to be a regular work node
workNode1->setType(0);

//Add the new work node definition to the document routing
//management object
routingMgmt->add(workNode1);

// Free memory. This object will no longer be needed.
delete(workNode1);
```

For a complete example creating work notes, refer to the SDocRoutingDefinitionCreationICM sample.

Listing work nodes

The `listWorkNodeNames` method lists all work node names in the library server, and the `listWorkNodes` method returns a collection of DKWorkNodeICM objects representing a work node in the library server.

Java

```java
// Obtain the document routing  management object.
 // Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt = new
   DKDocRoutingServiceMgmtICM(dsICM);

// Obtain all Work Nodes in the System.
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++

```cpp
// Obtain the document routing  management object.
DKDocRoutingServiceMgmtICM* routingMgmt =
  new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the collection containing all the  work nodes in the system.
dkCollection* workNodes = routingMgmt->listWorkNodes();
if (workNodes &&  ( workNodes->cardinality()>0) )
{
    cout << "Work Nodes in System: (" << workNodes->cardinality() << ")"
      << endl;
    dkIterator* iter = workNodes->createIterator();
    while(iter->more())
    {
        DKWorkNodeICM* workNode = (DKWorkNodeICM*) iter->next()->value();
        if(workNode->getType()==0)
        {
              cout << " Normal Node - " << workNode->getName() << ": " <<
              workNode->getDescription() << endl;
        }
        else
        {
             cout << " Collection Pt - " << workNode->getName() << ": " <<
             workNode->getDescription() << endl;
        }
        delete(workNode);
  }
  delete(iter);
  delete(workNodes);
}
delete(routingMgmt);
```

For more information on listing work nodes, refer to the `SDocRoutingListingICM` sample.

Defining a new collection point

A collection point is a work node with system-defined exit criteria, specifically applicable to routing folders. A set of requirements can be specified that must be met before the process can resume or advance past this point.

Java

```
// Create a new Work Node Object. This will be
//the collection point
DKWorkNodeICM collectionPoint = new DKWorkNodeICM();

// Choose a Name with 15 characters or less.
collectionPoint.setName("S_gatherAll");

// Choose a Description for more information than the name.
collectionPoint.setDescription("Gather Claim,Police Report,Policy,& Photos");

// Sets the value of the maximum time that an item can spend at this
// work node (in minutes).
 collectionPoint.setTimeLimit(100);

// Specify max number of work packages that can be at this node.
collectionPoint.setOverloadLimit(200);
// Set the type of node to be a collection point.
collectionPoint.setType(1);

// Create the "Resume" List, which is the list of document types
//that the process must  wait for before moving on to the next node.
//A list will be created to  hold "resume entries" which are descriptions
//of requirements that must be met before the process can move on.
// Create a List/Collection to hold all Resume Entries.
dkCollection resumeList = new DKSequentialCollection();
// Create as many requirements, or "Resume List Entries", which
//specify what Item Types it must wait for.  The process cannot
//pass this collection point unless the specified number of Item
//(DDO) of the specified Item Type reaches this collection point.
DKCollectionResumeListEntryICM resumeRequirement = new
     DKCollectionResumeListEntryICM();
// Set the Item Type Name Folder Item that is being routed.
resumeRequirement.setFolderItemTypeName("S_simple");
// Make the collection wait for an Item of the specified Item Type
//to be added to the folder before proceeding.
resumeRequirement.setRequiredItemTypeName("S_autoClaim");
// Specify the number of Items of the specified Item Type that it
//must wait for.
resumeRequirement.setQuantityNeeded(1);
// Add the requirement (Entry) to the List of Requirements (Resume List).
resumeList.addElement(resumeRequirement);
resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement.setFolderItemTypeName("S_simple");
resumeRequirement.setRequiredItemTypeName("S_policeReport");
resumeRequirement.setQuantityNeeded(1);
// When all requirements (resume list entries) have been added to the
//list of requirements (resume list), set the resume list in the
//collection point.
collectionPoint.setCollectionResumeList(resumeList);
// Add the new collection point definition to the document routing
// managment object
routingMgmt.add(collectionPoint);
```

C++

```
// Create a new Work Node Object.This will be the collection point
DKWorkNodeICM* collectionPoint = new DKWorkNodeICM();
// Choose a Name with 15 characters or less.
collectionPoint->setName("GatherOrderDetails");
// Choose a Description for more information than the name.
collectionPoint->setDescription("Gather all the information related to the
  order, shipping mechanism and shipping address");
// Sets the value of the maximum time that an item can spend at this
//work node (in minutes).
 collectionPoint->setTimeLimit(100);
// Specify max number of work packages that can be at this node.
collectionPoint->setOverloadLimit(200);

// Set the type of node to be a collection point.
collectionPoint->setType(1);
// Create the "Resume" List, which is the list of document types
//that the process must  wait for before moving on to the next node.
//A list will be created to  hold "resume entries" which are descriptions
//of requirements that must be met before the process may move on.

// Create a List / Collection to hold all Resume Entries.
dkCollection* resumeList = new DKSequentialCollection();

// Create as many requirements, or "Resume List Entries", which specify
//what Item Types it must wait for. The process cannot pass this
//collectionpoint unless the specified number of Item (DDO) of the
//specified Item Type reaches this collection point.
 DKCollectionResumeListEntryICM* resumeRequirement = new
    DKCollectionResumeListEntryICM();
// Set the Item Type Name Folder Item that is being routed.
resumeRequirement->setFolderItemTypeName("book");

// Make the collection wait for an Item of the specified Item Type to
//be added to the folder before proceeding.
resumeRequirement->setRequiredItemTypeName("AnItemType");

//Specify the number of Items of the specified Item Type that
//it must wait for.
resumeRequirement->setQuantityNeeded(1);

// Add the requirement (Entry) to the List of Requirements (Resume List).
resumeList->addElement(resumeRequirement);
resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement->setFolderItemTypeName("book");
resumeRequirement->setRequiredItemTypeName("AnotherItemType");
resumeRequirement->setQuantityNeeded(1);
resumeList->addElement(resumeRequirement);

// When all requirements (resume list entries) have been added to
//the list of requirements (resume list), set the resume list in the
//collection point.
collectionPoint->setCollectionResumeList(resumeList);
// Add the new collection point definition to the document routing
// managment object
routingMgmt->add(collectionPoint);

//Free the memory associated with this collection point
delete(collectionPoint);
```

For more information on defining collection points, refer to the `SDocRoutingDefinitionCreationICM` sample.

Defining a work list

A worklist consists of one or more work nodes from which a user obtains a list of work packages or the ″next″ work package. A work node can be in more than one worklist. Using a worklist allows an administrator or a user application to dynamically change work assignments without contacting end users.

Java

```
// Create a new work list.
 DKWorkListICM workList = new DKWorkListICM();
// Choose a name of 15 characters or less.
workList.setName("S_fillClaimWL");
workList.setDescription("Work List Covering Fill/Submit Claim Work Node.");
//Specify that work packages returned by the work list will be sorted by time
workList.setSelectionOrder(DKConstantICM.DK_ICM_DR_SELECTION_ORDER_TIME);
//Specify that the work packages returned will be the one that are not
// in the suspend state
workList.setSelectionFilterOnSuspend
    (DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that work packages returned are not the ones in the notify state
workList.setSelectionFilterOnNotify
     (DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

// Specify that at most 100 work packages should be listed in
// this work list
workList.setMaxResult(100);
String[] wnNames = {"S_fillClaim"};
workList.setWorkNodeNames(wnNames);

//Add the new work list definition to the document routing
//management object
routingMgmt.add(workList);
```

C++

```
// Create a new worklist.
 DKWorkListICM* workList = new DKWorkListICM();
//Choose a name of 15 characters or less.
workList->setName("ValidateWorkList");
workList->setDescription("worklist Covering the credit card
   validation work node.");

//Specify that work packages returned by the worklist will be
//sorted by time
workList->setSelectionOrder(DK_ICM_DR_SELECTION_ORDER_TIME);

//Specify that the work packages returned will be the one that are not
//in the suspend state
workList->setSelectionFilterOnSuspend(DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that work packages returned are not the ones in the notify state
workList->setSelectionFilterOnNotify(DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that at most 100 work packages should be listed in this worklist
workList->setMaxResult(100);

DKString* wnNames = new DKString[1];
wnNames[0] = "ValidateCreditCard";
//Add the work node to the worklist
workList->setWorkNodeNames(wnNames,1);

//Add the new worklist definition to the document routing management
//object
routingMgmt->add(workList);

//Free the memory associated with this worklist
delete(workList);
```

For more information on defining work lists, refer to the `SDocRoutingDefinitionCreationICM` sample.

Listing worklists

The `listWorkListNames` method lists all worklist names in the library server, and the `listWorkLists` method returns a collection of DKWorkListICM objects representing a worklist in the library server.

Java

```
// Obtain the document routing management object.
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt =
  new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain all Work Lists in the System.
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(); // Obtain all
// Work Lists in the System.

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); // Free Memory
    }
    delete(iter); // Free Memory
    delete(workLists);
}
```

For the complete example, see the `SDocRoutingListingICM` sample.

Defining a new process and associated route

A document routing process is the defined routes that a work package follows. Multiple routing processes can reuse the same nodes and you can also use multiple routes between nodes.

Java

```
// Create a new Process Definition
DKProcessICM process = new DKProcessICM();
process.setName("S_claimProcess");
process.setDescription("Process for an Insurance Claim");

// Define all possible Routes.

// Create a list of all possible routes between nodes.
dkCollection routes = new DKSequentialCollection();

// Connect the Work Nodes using Route List Entries.  A simple route
//between two work nodes is specified by associating a 'From' work
//node and a 'To' work node.
// A Route List Entry simply connects two nodes with an implied direction.
// Multiple routes may exist between nodes.  A specific route may be selected
// by a user-defined "selection" keyword.  Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", etc.
// Create a new connection between two nodes.
DKRouteListEntryICM nodeRoute = new DKRouteListEntryICM();

// Every process must start with the start node.
nodeRoute.setFrom(DKConstantICM.DK_ICM_DR_START_NODE);
nodeRoute.setTo("S_fillClaim");
// Choose any user-defined name for an action that will make the
//transition from the 1st node to the second
nodeRoute.setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_fillClaim");
nodeRoute.setTo("S_gatherAll");
// Choose any user-defined name for an action that will make the
// transition take place.
nodeRoute.setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_gatherAll");
nodeRoute.setTo(DKConstantICM.DK_ICM_DR_END_NODE);

// Choose any user-defined name for an action that will make
//the transition take place.
nodeRoute.setSelection("Complete");
// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);
// Set the route in the process.
process.setRoute(routes);
// Add the process to the routing Management.
routingMgmt.add(process);
```

C++

```
//A document routing process is the defined routes that a work
//package being routed will follow. Multiple routing processes may
//re-use the same nodes and multiple routes between nodes  may be used.

// Create a new Process Definition
DKProcessICM* process = new DKProcessICM();
process->setName("Buy_Book");
process->setDescription("Purchase a book online");
// Define all possible Routes.

// Create a list of all possible routes between nodes.
dkCollection* routes = new DKSequentialCollection();

// Connect the Work Nodes using Route List Entries.  A simple route
// between two work nodes is specified by associating a 'From' work node
// and a 'To' work node.
// A Route List Entry simply connects two nodes with an implied direction.
// Multiple routes may exist between nodes.  A specific route may be selected
// by a user-defined "selection" keyword.  Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", etc.

// Create a new connection between two nodes.
DKRouteListEntryICM* nodeRoute = new DKRouteListEntryICM();

// Every process must start with the start node.
nodeRoute->setFrom(DK_ICM_DR_START_NODE);
nodeRoute->setTo("ValidateCreditCard");
// Choose any user-defined name for an action that will make the transition
// from the 1st node to the 2nd
nodeRoute->setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("ValidateCreditCard");
nodeRoute->setTo("GatherShippingDetails");

// Choose any user-defined name for an action that will make the transition
// take place.
nodeRoute->setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("GatherOrderDetails");
nodeRoute->setTo(DK_ICM_DR_END_NODE);

// Choose any user-defined name for an action that will make the transition
// take place.
nodeRoute->setSelection("Complete");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

// Set the route in the process.
process->setRoute(routes);

// Add the process to the routing Management.
routingMgmt->add(process);

delete(process);
```

For the complete example, see the `SDocRoutingDefinitionCreationICM` sample.

Starting a document routing process

The example below shows you how to start a document routing process.

Java

```
//First create a document or folder that will be routed.
//An item type of name "s_simple" must be pre-defined before a
// DDO of that name can be created.

DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);

// Save the created folder to the persistent datastore.
ddoFolder.add();


//Create the core document routing service object.
DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);

//Start a process with the name "S_claimProcess" (which must be pre-defined.

//The PID string of the work pkg that will be routed is returned by
//this call.
String workPackagePidStr = routingService.startProcess
    ("S_claimProcess", ddoFolder.getPidObject().pidString(),1,"icmadmin");
```

C++

```
//First create a document or folder that will be routed.
//An item type of name "book"must be pre-defined before a DDO of
// that name can be created.
DKDDO* ddoFolder = dsICM->createDDO("book", DK_CM_FOLDER);

// Save the created folder to the persistent datastore.
ddoFolder->add();

//Set the priority for this document routing process
int Priority = 1;

//Create the core document routing service object.
DKDocRoutingServiceICM* routingService = new DKDocRoutingServiceICM(dsICM);

//Start a process with the name "Buy_Book" (which must be pre-defined.
//The PID string of the work pkg that will be routed is returned by
//this call.
DKString workPackagePidStr=routingService->startProcess("Buy_Book",
  ((DKPidICM*)
   ddoFolder->getPidObject())->pidString(), Priority, "icmadmin");
```

For the complete example, see the `SDocRoutingProcessingICM` sample.

Ending a process

A process can be explicitly terminated before it reaches the end node. When you terminate a process, the work package being routed is removed from the system. To terminate a process requires, you must know the PID string of the work package being routed by the process instance

Java

```
routingService.terminateProcess(workPackagePidStr);
```

C++

```
routingService->terminateProcess(workPackagePidStr);
```

For more information on terminating a process, refer to the `SDocRoutingProcessingICM` sample.

Continuing a process

The continueProcess() method routes the item referenced by the item PID in the specified work package from the current work node to the next work node that is determined by the selection. The specified work package is removed from the library server, and a new work package is created for the specified owner. The item referenced by the item PID is checked in, if it has been checked out. The PID of the new work package is returned. A null is returned if the process has ended.

In the code snippet below, the name of the selection that will cause the transition from the current work node to the next work node is ″Continue″. Note that the work package PID string of the current work package is specified in the method call. The method call returns the PID string of the new work package

Java

```
workPackagePidStr = routingService.continueProcess
     (workPackagePidStr, "Continue", "icmadmin");
```

C++

```
char * userName = "icmadmin";

workPackagePidStr = routingService->continueProcess(workPackagePidStr,
   "Continue", userName);
```

For the complete example, see the `SDocRoutingProcessingICM` sample.

Suspending a process

An instance of a document routing process can be suspended for a period of time (in minutes) or pending a set of requirements that must be met before it is resumed. This does not relate to processes and threads in a programming environment. The thread or process in the C++ runtime environment will not be stopped.

Java

```
dkCollection  requirements = new DKSequentialCollection();
//Process will be suspended for 2 minutes.
routingService.suspendProcess(workPackagePidStr, 2, requirements);
```

C++

```
dkCollection * requirements = new DKSequentialCollection();
//If no requirements are to be provided and the process is only to be
//suspended for a fixed period of time, the user can also pass in a
//NULL collection to this method.
//dkCollection * requirements = NULL;

//Process will be suspended for 2 minutes.
routingService->suspendProcess(workPackagePidStr, 2, requirements);
delete(requirements);
```

For the complete example, see the `SDocRoutingProcessingICM` sample.

Resuming a process

A suspended process (process in the suspended state) can be resumed explicitly, or implictly after the specified time has expired, or after the defined requirements have been met. This takes the process out of the suspended state, returning it to normal operation.The `resumeProcess` method resets the suspend flag of the specified work package to false before the suspension reaches the specified duration time and the resume list is satisfied. No routing or checkout of the associated work item will be performed.

Java

```
routingService.resumeProcess(workPackagePidStr);
```

C++

```
routingService->resumeProcess(workPackagePidStr);
```

For more information on resuming a process, refer to the `SDocRoutingProcessingICM` sample.

Listing work package persistent identifier strings in a worklist

The following code sample shows how to list the PID strings for all the work packages in a specified worklist.

Java

```
String[] workPackagePIDs =
  routingService.listWorkPackagePidStrings(workListName,processOwner);

// Print Work Package PIDs
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; // Size to be set by the API.
DKString* workPackagePIDs = routingService->
  listWorkPackagePidStrings("workListName",processOwner,arraySize);

// Print Work Package PIDs
cout << "Work Packages in Work List: (" << arraySize << ")" << endl;
for(int i=0; i< arraySize; i++)
    cout << " - PID: " << workPackagePIDs[i] << endl;

delete[] workPackagePIDs; // Free Memory
```

For the complete example, see the SDocRoutingListingICM sample.

Retrieving work package information

When an instance of a document routing process is in progress, a work package is the vehicle through which an item (instance of an item type) moves along through the routing process. A work package contains all the necessary information about the process and about the item that it is transporting. The work package is the object that an application uses and manipulates as required.

The retrieveWorkPackage method returns the DKWorkPackageICM object referenced by the specified work package PID (wpPidStringStr).

Java

```
//Use an established document routing service
//Specifying false in this method call makes sure that the work package
// is not checked out
DKWorkPackageICM workPackage =
        routingService.retrieveWorkPackage(workPackagePidStr,false);
System.out.println("----------------------------------------");
System.out.println("                Work Package");
System.out.println("----------------------------------------");
System.out.println(" Process Name:  " + workPackage.getProcessName());
System.out.println("  work Node Name:  " + workPackage.getWorkNodeName());
System.out.println("           Owner:  " + workPackage.getOwner());
System.out.println("        Priority:  " + workPackage.getPriority());
System.out.println(" User Last Moved:  " + workPackage.getUserLastMoved());
System.out.println(" Time Last Moved:  " + workPackage.getTimeLastMoved());
System.out.println("   Suspend State:  " + workPackage.getSuspendState());
System.out.println("    Notify State:  " + workPackage.getNotifyState());
System.out.println("     Notify Time:  " + workPackage.getNotifyTime());
System.out.println("     Resume Time:  " + workPackage.getResumeTime());
System.out.println("Work Package Pid:  " + workPackage.getPidString());
System.out.println("     Item Pid:  " + workPackage.getItemPidString());
```

C++

```
cout << "---------------------------------------------" << endl;
cout << " Work Package" << endl;
cout << "---------------------------------------------" << endl;
cout << " Process Name: " << workPackage->getProcessName() << endl;
cout << " work Node Name: " << workPackage->getWorkNodeName() << endl;
cout << " Owner: " << workPackage->getOwner() << endl;
cout << " Priority: " << workPackage->getPriority() << endl;
cout << " User Last Moved: " << workPackage->getUserLastMoved() << endl;
cout << " Time Last Moved: " << workPackage->getTimeLastMoved() << endl;
cout << " Suspend State: " << workPackage->getSuspendState() << endl;
cout << " Notify State: " << workPackage->getNotifyState() << endl;
cout << " Notify Time: " << workPackage->getNotifyTime() << endl;
cout << " Resume Time: " << workPackage->getResumeTime() << endl;
cout << "Work Package Pid: " << workPackage->getPidString() << endl;
cout << " Item Pid: " << workPackage->getItemPidString() << endl;
```

For the complete example, see the `SDocRoutingProcessingICM` sample.

Listing document routing processes

The following example shows you how to list document routing processes.

Java

```
//The listProcessNames method lists all process names in the
//Library Server, and the listProcesses method returns a collection
//of DKProcessICM objects representing a process in the Library Server.
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM routingMgmt =
  new DKDocRoutingServiceMgmtICM(dsICM);
 // Obtain the list of all running document routing processes
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt =
  new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain list of all routing processes running.
dkCollection processes = routingMgmt.listProcesses();
System.out.println("Running Processes:  ("+processes.cardinality()+")");

dkIterator iter = processes.createIterator();
while(iter.more())
{
// Move pointer to next element and obtain that next element.
DKProcessICM proc = (DKProcessICM) iter.next();
     System.out.println(" - "+proc.getName()+": "+proc.getDescription());
}
```

C++

```
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM* routingMgmt =
  new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the list of all running document routing processes
dkCollection* processes = routingMgmt->listProcesses();
if (processes && (processes->cardinality()>0))
{
   cout << "Running Processes: (" << 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);
```

A print function is provided in the `SDocRoutingListingICM` sample.

Ad hoc routing

Below is an ad hoc routing example procedure. In the example, the system administration client is used to set up the work nodes, processes, and worklists.

1. Create two work nodes, N1 and N2 for example.
2. Create two one-node processes, P1 and P2 such that P1 has one work node, N1 and P2 has one work node, N2.

   P1 looks like this:

From:	Action:	To:
START	Continue	N1
N1	Continue	END

   P2 looks like this:

From:	Action:	To:
START	Continue	N2
N2	Continue	END

3. Create two worklists, WL1 and WL2 such that WL1 has one work node, N1 and WL2 has one work node, N2.

At run time, complete the following steps to implement ad-hoc routing:

1. Start process P1 with a document PID (Example, ABC). A work package, WP1, is created. The worklist WL1 displays the work package WP1 at work node N1.
2. To move the document ABC from process P1 to process P2, terminate work package WP1 and start process P2 with the same document (ABC). Work package WP2 is created.

The worklist WL2 shows the work package WP2 at work node N2.

To see additional examples, see the `SDocRoutingDefinitionCreationICM` sample.

Document routing example queries

This sections contains example queries. For more information about writing queries, see "Understanding the query language" on page 177.

Example 1

Finds car documents, and returns only the associated work packages that are active.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@SUSPENDFLAG =
0]
```

Example 2

Finds car documents and returns the associated work packages that are in the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]
```

Example 3

Finds car documents where the name is ″Honda″, and returns the associated work packages that are in the ″AccidentInvesigation″ process.

```
/Car[@Name = "Honda" AND @VERSIONID = latest-version(.) AND
@SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER =>
WORKPACKAGE[@PROCESSITEMID = /ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]
```

Example 4

Finds car documents and returns the associated work packages that are in the ″UnderReview″ step of the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]/@ITEMID
AND ../@WORKNODENAME =  "UnderReview"]
```

Example 5

Finds car documents and returns only the suspended work packages that are in the ″UnderReview″ step of the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]
/@ITEMID AND ../@WORKNODENAME = "UnderReview" AND
@SUSPENDFLAG = 1]
```

Granting privileges for document routing

In order for a user to perform document routing operations, the user must have the appropriate privileges. The privileges associated with document routing are listed in the following table. The general privileges for items are applicable to processes, work nodes, and worklists.

Table 16. Document routing privileges

Privilege	Description	Related API
ICM_PRIV_ITEM _UPDATE_WORK	Used to see if the user is authorized to do the following for a work package: set the priority set the owner set the resume list set the duration for suspension	suspendProcess resumeProcess setWorkPackagePriority setWorkPackageOwner
ICM_PRIV_ITEM _ROUTE_START	Used to see if the user is authorized to start a process.	startProcess
ICM_PRIV_ITEM _ROUTE_END	Used to see if the user is authorized to terminate a process.	terminateProcess
ICM_PRIV_ITEM _GET_WORKLIST	Used to see if the user is authorized to get the count or work packages from a worklist.	getCount listWorkPackagePidStrings
ICM_PRIV_ITEM _GET_WORK	Used to see if the user is authorized to get a work package.	getNextWorkPackagePidString getNextWorkPackage checkOutItemInWorkPackage retrieveWorkPackage
ICM_PRIV_ITEM _GET_ASGN_WORK	Used to see if the user is authorized to get a work package that is owned by a different a different user.	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings
ICM_PRIV_ITEM _ROUTE	Used to see if the user is authorized to route a work package.	continueProcess

Working with access control lists for document routing

When an ACL is defined for a document routing entity such as a process, work node, and worklist, the operations allowed on the entity are impacted. The effect of ACLs on Content Manager document routing entity and their associated privileges are listed in the following table.

Table 17. Access control lists and document routing

Privilege	Description	Related API
Process	startProcess	ICM_PRIV_ITEM_ROUTE_START
Work node	continueProcess suspendProcess resumeProcessterminateProcess setWorkPackagePriority setWorkPackageOwner	ICM_PRIV_ITEM_ROUTE ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_ROUTE_END ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_UPDATE_WORK

Table 17. Access control lists and document routing (continued)

Privilege	Description	Related API
Worklist	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings checkOutItemInWorkPackage	ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK 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

Document routing constants

You define document routing constants in DKConstantICM. Following is the list of document routing constants:

- public final static int DK_ICM_DR_SELECTION_FILTER_NO = 0;
- public final static in 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″;

Working with other content servers

You use the dkDatastore classes to define an appropriate content server for the content servers in your application. The content server is the primary interface to the Enterprise Information Portal. Each content server has a separate content server class.

To create a content server, use the DKDatastore*xx* classes, where *xx* identifies the specific content server. Table 7 on page 34 shows these classes.

Table 18. Server type and class name terminology

Content server	Class name
Content Manager Version 8.2	DKDatastoreICM
Earlier 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
Relational databases	DKDatastoreDB2, DKDatastoreJDBC (for Java), DKDatastoreODBC

When creating a content server for a content server, implement each of the following classes and interfaces:

dkDatastore
Represents the content server and manages the connection, communications, and execution of content server commands. dkDatastore is an abstract version of the query manager class. It supports the evaluate method.

dkDatastoreDef
Uses the methods to access items stored in the content server. Also creates, lists, and deletes its entities. It maintains a collection of dkEntityDefs. Examples of concrete classes for this interface are:

- DKDatastoreDefDL
- DKDatastoreDefOD

dkEntityDef
Uses the methods to access entity information. Also creates and deletes entities and attributes. The methods of this class support accessing multiple-level entities. If a content server does not support subentities, they generate DKUsageError objects. If a content server supports multiple-level entities, you must implement methods to overwrite the exceptions for subclasses for these content servers. Examples of concrete classes for the dkEntityDef interface are:

- DKIndexClassDefDL
- DKAppGrpDefOD

The class hierarchy for an entity definition is illustrated in Figure 14:

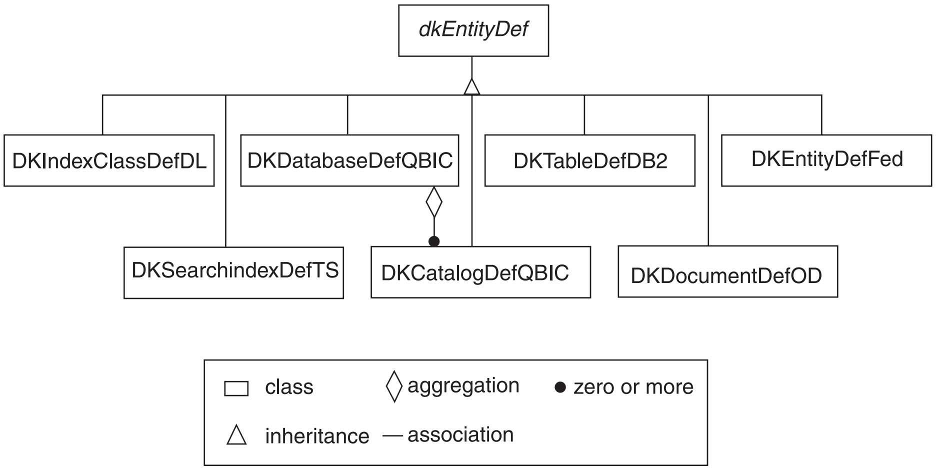


Figure 14. Class hierarchy

dkAttrDef
Defines methods to access attribute information and to create and delete attributes. Examples of concrete classes for dkAttrDef are:

- DKAttributeDefDL
- DKFieldDefOD

dkServerDef
Defines methods to access server information. Examples of concrete classes for dkServerDef are:

- DKServerDefDL
- DKServerDefOD

dkResultSetCursor
Creates a content server cursor that manages a collection of DDO objects. To use the addObject, deleteObject, and updateObject methods, set the content server option `DK_CM_OPT_ACCESS_MODE` to `DK_CM_READWRITE`.

dkBlob
Declares a common public interface for binary large object (BLOB) data types in each content server. The concrete classes derived from dkBlob share this common interface, allowing processing of BLOBs from heterogeneous content servers. Examples of concrete classes for dkBlob are:

- DKBlobDL
- DKBlobOD

The data definition classes and their class hierarchy are represented in Figure 15 on page 237:

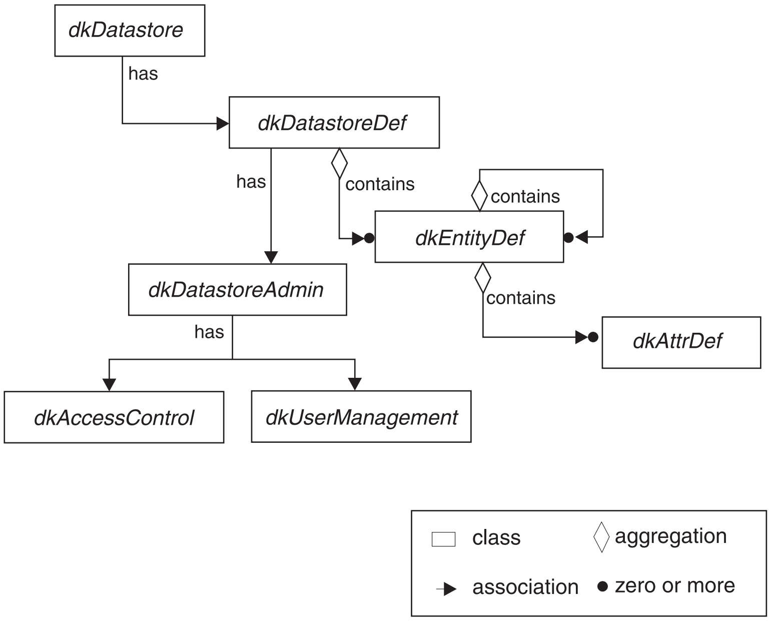


Figure 15. Data definition class hierarchy

For more information on dkDatastore and other common classes, see "Developing custom content server connectors" on page 364.

Working with earlier Content Manager

This section describes how to access data in Content Manager servers, and how to perform the following tasks:

- Handle large objects
- Use DDOs.
- Use XDOs in a search engine
- Use combined query.
- Use Text Search Engine.
- Use image search (QBIC).
- Use workflows and workbaskets.

Handling large objects

| In the earlier Content Manager connector, you can retrieve large objects piece by
| piece using asynchronous retrieval. For the sample application, refer to
| `TxdoAsyncRetDL` in the `CMBROOT\dk\samples` directory.

Setting Java heap size (Java only)

Java has a limitation for the default initial and maximum heap size. The default initial heap size is 1 048 576 and the default maximum heap size is 16 777 216 bytes. If your Java application program tries to use objects larger than the heap size, your program will fail during execution. To increase maximum heap size for your application, use the `-mx` option when you execute your Java application program. For example:

Java

```
java -mx40000000 yourApplication
```

Using DDOs to represent earlier Content Manager content

A DDO associated with DKDatastoreDL has some specific information to represent the Enterprise Information Portal document model: document, folder, parts, item, item ID, rank, and so forth. The following sections describe how you access this information.

DDO properties

The type of an item, whether it is a document or folder, is a property under the name DK_CM_PROPERTY_ITEM_TYPE. To get the item type of the DDO, you call:

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;
}    ...            // do something
```

After the property is called, the item_type is equal to DK_CM_DOCUMENT for a document, or DK_CM_FOLDER for a folder. The if statement ensures that the property exists. See “Adding properties to a DDO” on page 41 and “Getting the DKDDO and attribute properties” on page 44 for more information.

Persistent identifier (PID)

The PID contains important pieces of information specific to Enterprise Information Portal: the object type indicates the index class the DDO belongs to; the PID contains the item ID of the associated item from the content server. See “Creating a persistent identifier (PID)” on page 41.

Representing documents

A DDO representing a document has the property DK_CM_PROPERTY_ITEM_TYPE set to DK_CM_DOCUMENT. Its PID contains the index class name as the object type. The PID ID the same as the item ID.

The parts inside a document are represented as DKPartsDL objects, which are collections of binary large objects (BLOBs), each of which is represented as a DKBlobDL object.

A document DDO has a specific attribute named DKPARTS, whose value is a DKParts object.

To get to each part in a document, retrieve the DKParts first, then create an iterator to iterate over the parts. If the document does not have any parts, DKParts is null or the cardinality of DKParts is zero.

Documents associated with a combined query (a combination of a parametric and text query) can have a transient attribute named DKRANK, whose value is an object containing an integer rank computed by the Text Search Engine.

For more information on creating and processing a DKParts object, see "Creating, updating, and deleting documents or folders", "Retrieving a document or folder" on page 247, and "Creating documents and using the DKPARTS attribute" on page 83.

Representing folders

A DDO representing a folder has a property DK_CM_PROPERTY_ITEM_TYPE equal to DK_CM_FOLDER. Similar to a document DDO, its PID contains the index class name as the object type, and item ID in the PID's ID.

A DKFolder object represents the table of contents inside a folder. A DKFolder object is a collection of DDOs. Each DDO represents an item in the folder, either a document or another folder. A folder DDO has an attribute named DKFOLDER, whose value is a DKFolder object.

To get to each DDO member of the folder, retrieve the DKFolder object first; then create an iterator to access each item member. If the folder does not have a member, DKFolder is null, but the DKFOLDER attribute is always present in a folder DDO created by the content server.

For more information on creating and processing a DKFolder object, see "Creating, updating, and deleting documents or folders", "Retrieving a document or folder" on page 247, and "Creating folders and using the DKFOLDER attribute" on page 86.

Creating, updating, and deleting documents or folders

This section describes the processes involved in creating, updating, and deleting documents and folders.

Creating a document

To create a document and save its persistent data in a content server, you must create a DDO and set all of its attributes (and other information) except for the item ID. The item ID is assigned and returned by the content server. Some of the previous examples are combined in the following example:

Java

```
// ----- Step 1: create a datastore and connect to it
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// ----- Step 2: create a document (or folder) DDO
//         and set all its attributes and other required information
DKPid pid = new DKPid();
pid.setObjectType("GRANDPA");    // Set the index-class name it belongs to
DKDDO ddo = new DKDDO(dsDL,pid); // Create a DDO with PID and
...                              //   associate it to dsDL

// ----- Step 2.a: add attributes according to index class GRANDPA
Object obj, vstr;
Boolean yes = new Boolean(true);
Boolean  no = new Boolean(false);

short data_id = cddo.addData("Title");  // add new attribute "Title"
vstr = new Short(DK_CM_DATAITEM_TYPE_STRING);
// ----- Add type properties VSTRING and nullable
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstr);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, no);

data_id = cddo.addData("Subject");  // add new attribute "Subject"
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE,vstr);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE,yes);

// ----- Add some more attributes as necessary
  ....

// ----- Step 2.b: add DKPARTS attribute
DKParts parts = new DKParts();  // create a new DKParts, collection of parts
DKBlobDL blob = new DKBlobDL(dsDL);   // create a new XDO blob
DKPidXDODL pidXDO = new DKPidXDODL(); // create PID for this XDO object

pidXDO.setPartId(5);                          // set part number to 5
blob.setPidObject(pidXDO);                    // set the PID for the XDO blob
blob.setContentClass(DK_DL_CC_GIF);           // set content class type GIF
blob.setRepType(DK_REP_NULL);                 // set rep type for the part
blob.setContentFromClientFile("choice.gif");  // set the blob's content
blob.setInstanceOpenHandler("xv");            // the viewer program on AIX

parts.addElement(blob);               // add the blob to the parts collection

....                                  // create and add some more blobs to
....                                  // to the collection as necessary

// ----- Create DKPARTS attribute and set it to refer to the DKParts object
short data_id = ddo.addData(DKPARTS);        // add attribute "DKParts"
obj = new Short(DK_CM_COLLECTION_XDO);       // add type property
ddo.addDataProperty(data_id,DK_CM_PROPERTY_TYPE,obj);
ddo.addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE,yes); / add nullable prop
ddo.setData(data_id, parts);                 // sets the attribute value

// ----- Step 2.c: sets the item type : document
obj = new Short(DK_CM_DOCUMENT);
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, obj);

// ----- Step 3: make item persistent; add item to the datastore
ddo.add();                                // document created in datastore
```

C++

```
// step 1: create a datastore and connect to it
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

// step 2: create a document (or folder) DDO
//         and set all its attributes and other required information
//         See the section on "Using DDO"
DKPid pid;
// set the index-class name it belongs to
pid.setObjectType("GRANDPA");
// create a DDO with PID and associated with dsDL
DKDDO* ddo = new DKDDO(&dsDL,pid);

// step 2.a: add attributes according to index-class GRANDPA
DKAny any;
DKBoolean yes = TRUE;
DKBoolean no = FALSE;
// add a new attribute named "Title"
unsigned short data_id = cddo->addData("Title");
// add type property VSTRING
any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
// add nullable property: non-nullable
any = no;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// add a new attribute named "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);

// add some more attributes as necessary
//  ...

// step 2.b: add DKPARTS attribute
// create a new XDO blob
DKParts* parts = new DKParts;
DKBlobDL* blob = new DKBlobDL(&dsDL);

DKPidXDODL pidXDO;                     // create PID for this XDO object

pidXDO.setPartId(5);                   // set part number to 5
blob->setPid(&pidXDO);                 // set the PID for the XDO blob
blob->setContentClass(DK_CC_GIF);      // set content class type GIF
blob->setRepType(DK_REP_NULL);         // set rep type for the part
blob->setContentFromClientFile("choice.gif"); // set the blob's content

DKAny any = (dkDataObjectBase*) blob;
parts->addElement(any);               // add the blob to the parts collection

...                                     // create and add some more blobs
...                                     // to the collection as necessary
// continued...
```

C++ (continued)

```
// create DKPARTS attribute and sets it to refer to the DKParts object
// add attribute "DKParts"
unsigned short data_id = ddo->addData(DKPARTS);
any = DK_COLLECTION_XDO;
// add type property
ddo->addDataProperty(data_id,DK_CM_PROPERTY_TYPE,any);
any = (DKBoolean) TRUE;
// add nullable property
ddo->addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE ,any);
any = (dkCollection*) parts;
// sets the attribute value
ddo->setData(data_id, any);

// step 2.c: sets the item type : document
any = DK_CM_DOCUMENT;
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);

// step 3: make it persistent
// a document is created in the datastore
ddo->add();
```

The last step of the preceding example created a document in the content server (with the information). Whenever a document DDO is added to a content server, all of its attributes are added, including all of the parts inside the DKParts collection.

You use the same process for adding a folder DDO. The DKFOLDER collection members are added to the content server as a component of the folder. The folder contains a table of contents of its members, which are existing documents and folders. Therefore, create all folder members in the content server before adding a folder DDO.

Java

You can add the same document to a different content server of the same type. To add this document to the Content Manager server LIBSRVRN, which has an index class LIBSV2 with the same structure as LIBSV, use the following example:

```
// ----- Create datastore and connect to LIBSRVRN
DKDatastoreDL dsN = new DKDatastoreDL();
dsN.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");

// ----- Update the PID
pid = ddo.getPidObject();
pid.setObjectType("LIBSV2");        // set the new index class
pid.setPrimaryId("");               // make the item ID blank
pid.setDatastoreName("LIBSRVRN");   // set the new datastore name
ddo.setPidObject(pid);              // update the PID
ddo.setDatastore(dsN);              // re-associate the DDO with dsN
ddo.add();                          // add the DDO
```

C++

You can add the same document to a different content server of the same type. For example, to add the document to the server LIBSRVRN, which has an index class GRANDPA2 with the same structure as GRANDPA:

```
// create datastore and connect to LIBSRVRN
DKDatastoreDL dsN;
dsN.connect("LIBSRVRN","FRNADMIN","PASSWORD");
// update the Pid
pid = ddo->getPid();
pid.setObjectType("GRANDPA2");             // set the new index-class
pid.setId("");                             // blank the item-id
pid.setDatastoreName("LIBSRVRN");          // set the new datastore name
ddo->setPid(pid);                          // update the PID
ddo->setDatastore(&dsN);                   // re-associate it with dsN
ddo->add();                                // add it
```

Updating a document or a folder

To update a document or folder:

1. Set the item ID and the object type.
2. Update the appropriate attributes, or add to the DKParts collection.
3. Call the update method to store the change.

Java

```
// ----- Update the value of attribute Title
String newTitle = "Accident Report";
short data_id = ddo.getDataByName("Title");
ddo.setData(data_id, newTitle);
ddo.update();
```

C++

```
// update the value of attribute Title
DKAny any = DKString("Guess who is behind all this");
unsigned short data_id = ddo->getDataByName("Title");
ddo->setData(data_id, any);
ddo->update();
```

After the call to the update method, the value of the attribute Title in the content server is updated. The parts in this document are not updated unless their content has changed. The connection to the server must be valid when you call the update method.

Update a folder DDO using similar steps: update the attribute values, or add or remove elements from DKFolder; then call the update method.

Updating parts

Represent the collection of parts in a document using a DKParts object.

DKParts is a subclass of DKSequentialCollection. In addition to inheriting the sequential collection functions, DKParts has two additional methods for adding a part to, and removing a part from, the collection. These methods also immediately save the changes to the content server.

The document must already exist in the content server.

Adding and removing a member: The following examples add a part to a document.

Java

```
DKDDO addo = new DKDDO();    // create a document DDO
DKBlobDL newPart = new DKBlobDL();   // create the new part to be added
....                                 // initialized the DDO and new part
DKParts parts = (DKParts) addo.getDataByName(DKPARTS);  // get DKParts
parts.addMember(ddo, newPart);       // assume none of these values are NULL
```

C++

```
// a document DDO
DKDDO* ddo;
// a new part to be added
DKBlobDL* newPart;
// ddo and newPart are
// initialized somewhere along the line
...
...
// get DKParts
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
// assume none of these values are NULL
parts->addMember(ddo, newPart);
```

To remove `newPart` from the collection and the content server, you would use:

Java

```
parts.removeMember(addo, newPart);
```

C++

```
parts->removeMember(ddo, newPart);
```

The removeMember method in DKParts actually deletes the persistent copy of the part from the content server.

Differences between update, add, and remove on a document DDO: The addMember and removeMember methods of DKParts provide conveniences for adding and removing a part in the collection and the content server. They are faster than the update method in a document DDO. The update method on a DDO updates all of the attributes in the DDO, including DKParts and all of its members that changed. The steps are:

Java

```
....
// ----- Get DKParts, assume it exists and not null
DKParts parts = (DKParts) addo.getDataByName(DKPARTS);
parts.addElement(newPart);         // add a new part to parts
addo.update();                     // updates the whole ddo
....
```

C++

```
...
DKAny any = ddo->getDataByName(DKPARTS);
// get DKParts, assume it exists
DKParts* parts = (DKParts*) any.value();
// assume it is not NULL
any = (dkDataObjectBase*) newpart;
parts->addElement(any);
// updates the whole ddo
ddo->update();
...
```

Updating folders

You represent the collection of documents and folders within a folder using a DKFolder object. In the content server, a folder holds a table of contents referring to its objects instead of keeping all actual objects.

DKFolder is a subclass of DKSequentialCollection. In addition to inheriting the sequential collection methods, it has two additional members for adding a member (a document or a folder) to, or removing a member from, the collection and immediately stores those changes.

The document or folder to be added or removed must already exist in the content server.

Adding and removing a member: The following example illustrates adding another document or folder DDO to a folder DDO:

Java

```
DKDDO folderDDO = new DKDDO();  // Created the folder DDO
DKDDO newMember = new DKDDO();  // Create the new DDO to be added
....                            // The folder DDO and newMember are
....                            //    initialized
// ----- Get the DKFolder, assuming it exists, and the value not null
DKFolder folder = (DKFolder) folderDDO.getDataByName(DKFOLDER);
folder.addMember(folderDDO, newMember);
```

C++

```
// a folder DDO
DKDDO* folderDDO;
// a new DDO to be added as a member of this folder
DKDDO* newMember;
...      // folderDDO and newMember are
...      // initialized somewhere along the line
DKAny any = folderDDO->getDataByName(DKFOLDER);
// get DKFolder, assume it exists
DKFolder* folder = (DKFolder*) any.value();
// assume non NULL
folder->addMember(folderDDO, newMember);
```

Both `newMember` and `folderDDO` must exist in the content server for another document or folder to be added to it.

Similarly, to remove `newMember` from the collection and the content server use the following example:

Java

```
folder.removeMember(folderDDO, newMember);
```

C++

```
folder->removeMember(folderDDO, newMember);
```

Important: Removing a member from a folder only removes that member from the folder table of contents. If you use the `removeElementAt`, then function it does not delete the member from memory or from the content server.

Differences between update, add, and remove on a folder DDO: The addMember and removeMember methods of DKFolder provide conveniences for adding and removing a document or folder in the collection and in the content server. They are faster than the update method in a folder DDO

The update method on a DDO updates all of the attributes in the DDO, including DKFolder and all of its members, whereas the addMember and removeMember methods only add or remove a member in the folder table of contents.

Deleting a document or a folder

Use the del method in the DDO to delete a document or folder from the content server.

Java

```
ddo.del();
```

C++

```
ddo->del();
```

The DDO must have its item ID and object type set, and have a valid connection to the content server.

Use the statement above to delete a folder as well. Only persistent data is deleted, the in-memory copy of the DDO does not change. Therefore, you can add this DDO back to the same or different content server later, in the application. See “Creating a document” on page 239 for more information.

Retrieving a document or folder

To retrieve a document from a DKDatastoreDL (representing an earlier Content Manager content server), you must know the document’s index class name and item ID. You also must associate the DDO with a content server and establish a connection.

Java

```
DKDDO ddo = new DKDDO(dsDL,pid);
// ----- Create the datastore and establish a connection
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

DKPid pid = new DKPid();
pid.setObjectType("Claim"); // set the index-class name it belongs to
pid.setPrimaryId("LN#U5K6ARLGM3DB4");  // set the item-id
// ----- create a DDO with the PID and associated with the datastore

ddo.retrieve(); // retrieve the document
```

C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
DKPid pid;
// set the index-class name it belongs to
pid.setObjectType("GRANDPA");
// set the item-id
pid.setId("LN#U5K6ARLGM3DB4");
// create a DDO with pid and associated to dsDL
DKDDO* ddo = new DKDDO(&dsDL, pid);
// retrieve it
ddo->retrieve();
```

After a call to retrieve, all of the DDO’s attribute values are set to the value of the persistent data stored in the content server. If the document has parts, the DKPARTS attribute is set to a DKParts object. However, the content of each part in this collection is not retrieved. Because a part might be large, you should not retrieve all of them into memory at once. It is better to explicitly retrieve the part you want.

If the DDO is a parametric query result that ran with the query option CONTENT=NO, the DDO is empty (does not have the attribute values). However, all information required to retrieve it is already set.

Retrieving parts

After you retrieve a DDO, you can retrieve its parts that are stored in the DKPARTS attribute, as follows:

Java

```
DKParts parts = (DKParts) ddo.getDataByName(DKPARTS);
```

C++

```
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
```

This example assumes that the DKPARTS attribute exists. If it does not exist, an exception is generated. See “Retrieving a folder” on page 249 for an example of extracting an attribute value by getting the data ID first and testing it for zero.

To retrieve each part, you must create an iterator to step through the collection and retrieve each part. See “Creating documents and using the DKPARTS attribute” on page 83.

Java

```
// ----- Create an iterator and process the part collection members
if (parts != null) {
    DKBlobDL blob;
    dkIterator iter = parts.createIterator();
    while (iter.more()) {
           blob = (DKBlobDL) iter.next();
           if (blob != null) {
               blob.retrieve();  // retrieve the blob's content
               blob.open();
            ....                 // other processing, as needed
           }
    }
}
```

C++

```
// create iterator and process the part collection member one by one
if (parts != NULL) {
    DKAny* element;
    DKBlobDL* blob;
    dkIterator* iter = parts->createIterator();
    while (iter->more()) {
           element = iter->next();
           blob = (DKBlobDL*) element->value();
           if (blob != NULL) {
               // retrieve the blob's content
               blob->retrieve();
               // other processing, as needed
               blob->open();
            }
    }
    delete iter;
}
```

Similar to the DDO results of a parametric query, each part XDO inside the DKParts collection is empty (does not have its content set). However, it has all the information needed for retrieval. To bring its content and related information into memory, call the retrieve method:

Java

```
blob.retrieve();
```

C++

```
blob->retrieve();
```

Retrieving a folder

Retrieve a folder DDO in the same way that you retrieve a document DDO. After being retrieved, the folder DDO has all of its attributes set, including the attribute, DKFOLDER. This attribute value is set to a DKFolder object, a collection of the DDO members in the folder. Like the parts in a DKParts object, these member DDOs contain only enough information to retrieve them. You can retrieve a folder DDO as follows:

Java

```
data_id = ddo.dataId(DKFOLDER);      // get DKFOLDER data-id
if (data_id == 0)                    // folder not found
    throw new DKException(" folder data-item not found");

DKFolder fCol = (DKFolder) ddo.getData(data_id); // get the folder collection

// ----- Create iterator and process the DDO collection members one by one
if (fCol != null) {
   DKDDO item;
   dkIterator iter = fCol.createIterator();
   while (iter.more()) {
      item = (DKDDO)  iter.next();
      if (item != null) {
        item.retrieve();     // retrieve the member DDO
         ....                // other processing
       }
   }
 }
```

C++

```
// get DKFOLDER data-id
data_id = ddo->dataId(DKFOLDER);
// folder not found
if (data_id == 0) {
    DKException exc(" folder data-item not found");
    DKTHROW exc;
}
// get the folder collection
any = ddo->getData(data_id);
DKFolder* fCol = (DKFolder*) any.value();
// create iterator and process the DDO collection member one by one
if (fCol != NULL) {
    DKAny* element;
    DKDDO* item;
    dkIterator* iter = fCol->createIterator();
    while (iter->more()) {
           element = iter->next();
           item = (DKDDO*) element->value();
           if (item != NULL) {
               // retrieve the member DDO
               item->retrieve();
               // other processing
               ...
           }
    }
    delete iter;
}
```

See also “Creating folders and using the DKFOLDER attribute” on page 86.

Understanding text searching (Text Search Engine)

The Text Search Engine product supports various query types:

- “Boolean query” on page 251
- “Free text query” on page 251
- “Hybrid query” on page 251
- “Proximity query” on page 252
- “Global text retrieval (GTR) query” on page 252

You can use the text search item ID, part number, and ranking information (returned by the query) to create an XDO that retrieves the document from an earlier Content Manager server.

Use a DKDatastoreTS object to represent the Text Search Engine. Text Search Engine does not actually store the data, it merely indexes the data stored in earlier Content Manager to support a text search on them. The result of a text search is an item identifier describing the location of the document in Content Manager. Use these identifiers to retrieve the document.

The DKDatastoreTS object does not support `add`, `update`, `retrieve`, and `delete` functions. You can query this content server. Refer to “Loading data to be indexed by Text Search Engine” on page 260 for information on adding data to Content Manager that is indexed by Text Search Engine.

Boolean query

A boolean query is made up of words and phrases, separated by boolean operators. Enclose a phrase in single quotes (′). Phrases are treated as a literal strings.

The following example creates a query string to search for all text documents with the word www or the phrase web site in the TMINDEX text search index:

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)";
```

Free text query

A free text query is made up of words, phrases, or sentences enclosed in braces ({ }). The words are not required to be adjacent to each other. The following example creates a query string to search for all text documents with the free text web site in the TMINDEX text search index:

Java

```
String cmd = "SEARCH=(COND=({web site}));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd  = "SEARCH=(COND=({Web site}));";
         cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Hybrid query

A hybrid query is made up of a boolean query followed by a free text query. The following example creates a query string to search for all text documents with the words IBM and UNIX, as well as the free text web site in the TMINDEX text search index:

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)";
```

Proximity query

A proximity query looks for a sequence of search arguments found in the same document, paragraph, or sentence. The following example creates a query string to search for all text documents with the phrase rational numbers and the word math in the same paragraph using the TMINDEX text search index:

Java

```
String cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd  = "SEARCH=(COND=($PARA$ {'rational numbers' math}));";
         cmd += "OPTION=(SEARCH_INDEX=TMINDEX)"
```

This type of query requires at least two search arguments.

Global text retrieval (GTR) query

A GTR query is optimized for double-byte character set (DBCS) languages like Japanese or Chinese. GTR also supports single-byte character set (SBCS) languages. Enclose all double-byte characters in single quotes (′). Make sure that the phrase to be searched for is in the specified character code set and language.

The following example shows a GTR search for all text documents that contain the phrase IBM marketing. The MATCH keyword is set to indicate the degree of similarity for the phrase.

Java

```
String cmd = "SEARCH=(COND=($CCSID=850,LANG=6011,MATCH=1$ " +
             "'IBM marketing'));" +
             "OPTION=(SEARCH_INDEX=TMINDEX)";
```

C++

```
DKString cmd = "SEARCH=(COND=($CCSID=850, LANG=6011,MATCH=1$ ";
         cmd += "'IBM marketing'));";
         cmd += "OPTION=(SEARCH_INDEX=TMGTRX)";
```

Make sure that the text search content server options DK_OPT_TS_CCSID (coded character set identifiers) and DK_OPT_TS_LANG (language identifiers) are set properly. The default for DK_OPT_TS_CCSID is DK_CCSID_00850. The default for DK_OPT_TS_LANG is DK_LANG_ENU. These values are used as the global defaults for the text query. For more information, see the online API reference.

You can also enter specific CCSID and LANG information as shown in the following example. You must specify both CCSID and LANG; one value cannot be specified without the other.

Representing Text Search Engine information using DDOs

You use a DDO (associated with a DKDatastoreTS object) to represent the results from text searches.

DKDastastoreTS does not have a property item type as a DKDatastoreDL object does. The format of its ID is also different. A DDO resulting from a text query corresponds to a text part inside an item. It contains the following standard attributes:

DKDLITEMID
The item ID that this text is part of. Use this item ID to retrieve the whole item from the content server.

DKPARTNO
An integer part number for this text part. Use the part number with the item ID to retrieve the text part from the content server.

DKREPTYPE
The RepType of this text part. Use this attribute with the item ID and part number to retrieve the text part from the content server.

DKRANK
An integer rank signifying the relevance of this part to the results of a text query. A higher rank means a better match. See the online API reference for further information.

DKDSIZE
An integer number representing word occurrences (in the results of boolean queries). See the online API reference for further information.

DKRCNT
An integer number representing boolean search matches. See the online API reference for further information.

The PID for a text search DDO has the following information:

content server type
TS.

content server name
The name used to connect to the content server.

object type
Text search index.

ID Text Search Engine document ID.

Establishing a connection

The DKDatastoreTS object provides two functions for connecting and a function for disconnecting. Normally, you create a DKDatastoreTS object, connect to it, run a query, and then disconnect when done. The following example shows the first connection function using the text search server `TM`.

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect("TM", "", "", "");
....                    // run a query
dsTS.disconnect();
```

The complete sample application from which this example was taken (`TConnectTS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

C++

```
DKDatastoreTS dsTS;
dsTS.connect("TM","","","");
...                  // do some work
dsTS.disconnect();
```

The complete sample application from which this example was taken (TConnectTS.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

The following example shows the second connection function using the text search server with the hostname apollo, port number 7502, and TCP/IP communication type DK_CTYP_TCPIP:

```
dsTS.connect("apollo", "7502", DK_CTYP_TCPIP);
```

The following example shows the first connection function using the text search server hostname apollo, port number 7502, communication type T (TCP/IP):

```
dsTS.connect("apollo", "", "", "PORT=7502; COMMTYPE=T");
```

The following example shows the first connect method using the text search server name TM and using library server LIBSRVR2, user ID FRNADMIN and password PASSWORD:

The following example shows the first connection function using the text search server name TM, library server LIBSRVRN, user ID FRNADMIN, and password PASSWORD:

```
dsTS.connect("TM", "", "", "LIBACCESS=(LIBSRVRN, FRNADMIN, PASSWORD)");
```

You can use the last parameter LIBACCESS, also called the connect string, to pass a sequence of parameters.

Tip: To prevent the Text Search Engine connection from timing out, connect to Text Search Engine, run your queries, and immediately disconnect. Do not leave the connection open.

Getting and setting text search options

Text search provides some options that you can set or get using its functions. See the online API reference for the list of options and their descriptions. The following example shows how to set and get the option for a text search character code set.

Java

```
DKDatastoreTS dsTS = new DKDatastoreTS();
Integer input_option  = new Integer(DK_TS_CCSID_00850);
Integer output_option = null;

dsTS.setOption(DK_TS_OPT_CCSID, input_option);
output_option = (Integer) dsTS.getOption(DK_OPT_TS_CCSID);
```

C++

```
DKAny input_option = DK_CCSID_00850;
DKAny output_option;
dsTS.setOption(DK_OPT_TS_CCSID,input_option);
dsTS.getOption(DK_OPT_TS_CCSID,output_option);
```

The `ouput_option` is an object, but is usually cast to an `Integer`.

Tips: The search options `CCSID` and `LANG` go together. If one is specified, the other must also be specified. The default `CCSID` and `LANG` are specified by the DKDatastoreTS options, `DK_OPT_TS_CCSID` and `DK_OPT_TS_LANG`. Refer to the online API reference for the list of the content server options and their descriptions.

You can specify more than one search option for a query term. The search options are separated by commas. An example of multiple search terms is given in “Global text retrieval (GTR) query” on page 252.

If both the `SC` (single required character) and the `MC` (sequence of optional characters) search options, you must specify the `SC` search option first. For example, `$SC=?,MC=*$ U?I*`.

Listing servers

The DKDatastoreTS object provides a function to list the text search servers that it can connect to. The following example shows how to retrieve the list of servers.

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);
}
```

The complete sample application from which this example was taken (`TListCatalogTS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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;
```

The complete sample application from which this example was taken (`TListCatalogTS.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

The list of servers is returned in a DKSequentialCollection of DKServerInfoTS objects. After you get a DKServerInfoTS object, you can retrieve the server name and location. You can then use the server name to establish a connection to it.

Listing schema

A DKDatastoreTS object provides functions for listing the schema. For text search, these are text search indexes. The following example shows how to retrieve the index list.

The list of indexes is returned in a DKSequentialCollection object of DKIndexTS objects. After you get a DKIndexTS object, you can retrieve information about the index, such as its name and library ID, which you can use to form a query.

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();
   ...           \\  Process the list as appropriate
}
```

The complete sample application from which this example was taken (`TListCatalogTS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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();
```

The complete sample application from which this example was taken (`TListCatalogTS.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory. Also, refer to “Managing memory in collections (C++ only)” on page 95 for information about deleting the collection.

Indexing XDOs by search engine

Before searching data items with the Text Search Engine and image search, you must first index the data. Indexes require three values: SearchEngine, SearchIndex and SearchInfo.

The value of the SearchIndex property is a combination of two names: the search service name and search index name. For example, if you defined a text search server named TM in the system administration client and a search index named TMINDEX associated with it, the value for the SearchIndex is TM-TMINDEX.

For an object that is to be indexed by Text Search Engine, the value of SearchEngine must be SM, for a data item to be indexed by query by image search, the value of SearchEngine must be QBIC (for more on image search, see “Understanding image search terms and concepts” on page 272).

The SearchIndex for QBIC is a combination of three values: QBIC database name, QBIC catalog name, and QBIC server name. For example, if the QBIC database name is SAMPLEDB, the QBIC catalog name is SAMPLECAT, and the QBIC server name is QBICSRV, then the correct value for the SearchIndex would be SAMPLEDB-SAMPLECAT-QBICSRV.

Refer to LoadSampleTSQBICDL and LoadFolderTSQBICDL in the CMBROOT\Samples directory for examples of how to load data, or create a folder and load data.

Important: When adding a part object to be indexed by a search engine, don’t set the RepType. Currently, the Text Search Engine works only with the default RepType FRN$NULL.

Adding an XDO to be indexed by Text Search Engine: The following example shows how to add an XDO that is to be indexed:

Java

```
// ----- Declare variables for part ID, item ID, and file
  int    partId = 20;
  String itemId = "CPPIORH4JBIXWIY0";
  String fileName = "g:\\test\\testsrch.txt";
  try {
    DKDatastoreDL dsDL = new DKDatastoreDL(); // create datastore
     ...                                      // connect to datastore
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
    DKBlobDL axdo = new DKBlobDL(dsDL);      // create XDO
    DKPidXDODL  apid = new DKPidXDODL();     // create PID
    apid.setPartId(partId);                  // set partId
    apid.setPrimaryId(itemId);               // set itemId
    axdo.setPidObject(apid);                 // setPid to XDO
    axdo.setContentClass(DK_DL_CC_ASCII);    // set ContentClass to text

    // --- set the searchEngine
    DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
    aSrchEx.setSearchEngine("SM");
    aSrchEx.setSearchIndex("TM-TMINDEX");
    aSrchEx.setSearchInfo("ENU");
    axdo->setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
    ...
    // ----- Set file content to buffer area
    axdo.setContentFromClientFile(fileName);
    axdo.add();                              //add from buffer
    ...
    // ----- Display the partId after add
    System.out.println("after add partId = " + ((DKPidXDODL)
             (axdo.getPidObject())).getPartId());

    dsDL.disconnect();                      //disconnect from datastore
    dsDL.destroy();
  }
  // ----- Catch any exception
  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
 {
    //replace following with your library server, user ID, password
    dsDL.connect("LIBSRVN","FRNADMIN","PASSWORD");

    cout << "datastore connected" << endl;

    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= "<<axdo->getPartId()<<endl;
    cout<<"repType= "<<axdo->getRepType()<<endl;

// continued...
```

C++ (continued)

```
    //--- set 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;
}
```

Important: When adding a part object to be indexed by a search engine, don't set the RepType (representation type). The Text Search Engine works only with the default RepType FRN$NULL.

Loading data to be indexed by Text Search Engine: To load data into Content Manager to be indexed by Text Search Engine, you must create both an index and a text search index.

Before you can create a text search index, the text search server must be running. Make sure that your environment is properly set up. To do so, you can customize and run the samples: TListCatalogDL and TListCatalogTS in the CMBROOT\Samples directory.

To create parts in Content Manager that are indexed by the Text Search Engine, refer to "Working with extended data objects (XDOs)" on page 48.

After the data is loaded into Content Manager, use the wakeUpService function in the DKDatastoreDL class to place the documents on the document queue. This function takes a search engine name as a parameter.

Then from the Content Manager text search Administration window, complete the following steps:

1. Double-click the text search server.
2. Double-click the text search index.
3. Click **INDEX**.

This indexes the documents on the document query. After the indexing is complete, you can perform text search queries.

For more information on text search administration, refer to the *System Administration Guide*.

Using text structured document support

Text structured documents are composed of tagged text (for example, an HTML file). A document model defines the structure of the document, and the Text Search Engine can search on words or phrases between the tags.

You can perform text queries on structured documents as follows:

1. Create a document model. The document model contains sections, and each section includes the section name and document tag used. For example:

   ```
   <HTML>
   <HEAD>
   <TITLE>Acme Corp<br></TITLE>
   </HEAD>
   <BODY>
   <H1>Acme Corp<BR></H1>
   <P><B>Acme Corp<BR></B><BR>
   <P>John Smith <BR>
   <P><ADDRESS>Acme Corporation<BR></ADDRESS>
   <HR>
   <H2>Acme Corp Business Objectives</H2>
   <HR>
   <P>
   <H2><A NAME="Header_Test" HREF="#ToC_Test">Marketing</A></H2>
   <P>We need to increase our time to market by 25%.
   <P>We need to meet our customers needs.
   </BODY>
   </HTML>
   ```

2. Create a text search index that uses the Content Manager document model.
3. Set the indexing rules for the text search index and specify the default document format (for example, `DK_TS_DOCFMT_HTML` for HTML files)
4. Add parts objects to the Content Manager server.
5. Start the indexing process for the text search index.

This example shows how to list the document models defined in your system.

Java

```
// ----- Initialize the variables
DKSequentialCollection pCol = null;
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;
dkIterator pIter = null;
DKDocModelTS pDocModel = null;
int ccsid = 0;
String strDocModelName = null;
int i = 0;

// ----- Create the datastore and connect
DKDatastoreTS dsTS = new DKDatastoreTS();
dsTS.connect(srchSrv,"",' ');

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Get list of document models
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();
```

The complete sample application from which this example was taken (`TListDocModelsTS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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();

// list document models
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();
```

The complete sample application from which this example was taken (`TListDocModelsTS.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

The following example shows how to create a document model:

Java

```
// ----- Create datastore and connect
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;

// ----- Create an instance of a document model object
DKDocModelTS docModel = new DKDocModelTS();

// ----- Create 2 instances of a document section objects for the model
DKDocSectionTS docSection = new DKDocSectionTS();
DKDocSectionTS docSection2 = new DKDocSectionTS();

// ----- Describe the document model for text document structure
//       for files like tstruct.htm above
docModel.setCCSID(DK_TS_CCSID_00850);
docModel.setName(docModelName);
docSection.setName("SAMPTITLE");
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();

// ----- Create the document model
dsAdmin.createDocModel("",docModel);

dsTS.disconnect();
dsTS.destroy();
```

Refer to `TCreateDocModelTS.java` and `TCreateStructDocIndexTS.java` in the `CMBROOT\Samples\java\dl` directory for more examples.

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;

// create an instance of a document model object
DKDocModelTS* docModel = new DKDocModelTS();

// create 2 instances of a document section objects for the model
DKDocSectionTS* docSection = new DKDocSectionTS();
DKDocSectionTS* docSection2 = new DKDocSectionTS();

// Describe the document model for text document structure
// for files like tstruct.htm above

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();

// create doc model
dsAdmin->createDocModel("",docModel);

// delete document model & sections
delete docModel;

dsTS.disconnect();
```

The complete sample application from which this example was taken (TCreateDocModelTS.cpp) and (TCreateStructDocIndexTS.cpp) are available in the Cmbroot/Samples/cpp/dl directory.

The following example shows how to create and set the indexing rules for a text search index that uses a document model:

Java

```
// ----- Create the datastore and index rules object
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;
DKIndexingRulesTS indexRules = new DKIndexingRulesTS();

// ----- Create an instance of a document model object
DKDocModelTS docModel = new DKDocModelTS();

// ----- Create 2 instances of a document section objects for the model
DKDocSectionTS docSection = new DKDocSectionTS();
DKDocSectionTS docSection2 = new DKDocSectionTS();

// ----- Create the document model instance for indexing rules
DKDocModelTS docModel2 = new DKDocModelTS();
docModel2.setCCSID(DK_TS_CCSID_00850);
docModel2.setName("SAMPCORPMOD");

// ----- Describe the document model for text document structure
//       for files like tstruct.htm above
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);

// ----- Connect to the datastore
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";
// ----- For AIX us the following form for the file
//    String strIndexFileDir = "/home/cltadmin/tsindex/tstruct";
pEnt.setIndexDataArea(strIndexFileDir);
String strWorkFileDir =  "e:\\tsindex\\work\\tstruct";
// ----- For AIX us the following form for the file
//     String strWorkFileDir = "/home/cltadmin/work/tstruct";
pEnt.setIndexWorkArea(strWorkFileDir);

// ----- Associate document model with index
pEnt.addDocModel(docModel);

// ----- Create text search index that supports sections
dsDef.add((dkEntityDef)pEnt);

// continued...
```

Java (continued)

```
indexRules.setIndexName("TSTRUCT");
indexRules.setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules.setDefaultDocModel(docModel2);

dsAdmin.setIndexingRules(indexRules);

dsTS.disconnect();
dsTS.destroy();
```

The complete sample application from which this example was taken (`TCreateStructDocIndexTS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKIndexingRulesTS* indexRules = new DKIndexingRulesTS();

// create an instance of a document model object
DKDocModelTS* docModel = new DKDocModelTS();

// create 2 instances of a document section objects for the model
DKDocSectionTS* docSection = new DKDocSectionTS();
DKDocSectionTS* docSection2 = new DKDocSectionTS();

// doc model instance for indexing rules
DKDocModelTS* docModel2 = new DKDocModelTS();
docModel2->setCCSID(DK_TS_CCSID_00850);
docModel2->setName("SAMPCORPMOD");

// Describe the document model for text document structure
// for files like tstruct.htm above

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);

// continued...
```

C++ (continued)

```
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);

// This index is text structure document section enabled
pEnt->setIndexProperty(DK_TS_PROPERTY_SECTIONS_ENABLED);

pEnt->setLibraryId("LIBSUM");
pEnt->setLibraryClientServices("IMLLSCDL");
pEnt->setLibraryServerServices("IMLLSSDL");
DKString strIndexFileDir = "e:\\tsindex\\index\\tstruct";
//**** for AIX ***************************************
//DKString strIndexFileDir = "/home/cltadmin/tsindex/index/tstruct";
pEnt->setIndexDataArea(strIndexFileDir);
DKString strWorkFileDir = "e:\\tsindex\\work\\tstruct";
//**** for AIX ***************************************
//DKString strWorkFileDir = "/home/cltadmin/tsindex/work/tstruct";
pEnt->setIndexWorkArea(strWorkFileDir);

// Associate document model with index
pEnt->addDocModel(docModel);

// Create text search index that supports sections
dsDef->add(pEnt);

delete pEnt;

indexRules->setIndexName("TSTRUCT");
indexRules->setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules->setDefaultDocModel(docModel2);
dsAdmin->setIndexingRules(indexRules);

delete indexRules;

dsTS.disconnect();
```

The complete sample application from which this example was taken (`TCreateStructDocIndexTS.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

The following example shows how to start the indexing process, which is asynchronous. Using the system administration program, you can start the indexing process and check its status.

Java

```
// ----- Declare datastore and administration
DKIndexFuncStatusTS pIndexFuncStatus = null;
DKDatastoreTS dsTS = new DKDatastoreTS();
DKDatastoreDefTS dsDef = null;
DKDatastoreAdminTS dsAdmin = null;

dsTS.connect("TMMUF","","","");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Start the indexing process
dsAdmin.startUpdateIndex(indexName);

// ----- Get indexing status
pIndexFuncStatus = dsAdmin.getIndexFunctionStatus(indexName,
                     DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS);
....  //  Process the status as appropriate

// ----- Show the scheduled document queue
System.out.println("*getScheduledDocs "
    + pIndexFuncStatus.getScheduledDocs());

// ----- Show the primary document queue
System.out.println("*getDocsInPrimaryIndex "
    + pIndexFuncStatus.getDocsInPrimaryIndex());

// ----- Shows the secondary document queue
System.out.println("*getDocsInSecondaryIndex " +
   pIndexFuncStatus.getDocsInSecondaryIndex());
System.out.println("*getDocMessages "
    + pIndexFuncStatus.getDocMessages());
if (pIndexFuncStatus.isCompleted() == true)
{
   // ---- Processing if indexing is completed
}

if (pIndexFuncStatus.getReasonCode() != 0)
{
   dsAdmin.setIndexFunctionStatus(indexName,
      DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS, DK_TS_INDEX_ACTID_RESET);
}

dsTS.disconnect();
dsTS.destroy();
```

The complete sample application from which this example was taken (TIndexingTS.java) is available in the CMBROOT\Samples\java\dl directory.

C++

```
DKDatastoreTS dsTS;
DKDatastoreDefTS* dsDef = 0;
DKDatastoreAdminTS* dsAdmin = 0;
DKIndexFuncStatusTS* pIndexFuncStatus = 0;

dsTS.connect(srchSrv,"","");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// starts the indexing process
dsAdmin->startUpdateIndex(srchIndex);

// Get indexing status
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)
      {
      // indexing completed
      }
      if (pIndexFuncStatus->getReasonCode() != 0)
      {
       dsAdmin->setIndexFunctionStatus(srchIndex,
     DK_TS_INDEX_FUNCID_INDEX_DOCUMENTS,DK_TS_INDEX_ACTID_RESET);
      }
delete pIndexFuncStatus;
dsTS.disconnect();
```

The complete sample application from which this example was taken (TIndexingTS.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Refer to TCheckStatusTS in the CMBROOT\Samples directory for an example of checking status. The example checks whether a queued request has been moved from the scheduled document queue to the primary or secondary queues. If an indexing error occurs, then you can check the imldiag.log file in the text search instance directory.

The following example shows how to execute a structure document text query based on the document model and the text search index defined above.

Java

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect
dsTS.connect("TMMUF","","","");
// ----- Generate the query string
String cmd = "SEARCH=(COND=($CCSID=850," +
             "DOCMOD=(DOCMODNAME=SAMPCORPMOD," +
             "SECLIST=(SAMPCORPTITLE,SAMPCORPBODY))$ Corp));" +
             "OPTION=(SEARCH_INDEX=TMSTRUCT;MAX_RESULTS=5)";
// ----- Execute the query
pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);

// ----- Process the results
.....
dsTS.disconnect();
dsTS.destroy();
```

The complete sample application from which this example was taken (`TExecuteStructDocTS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

C++

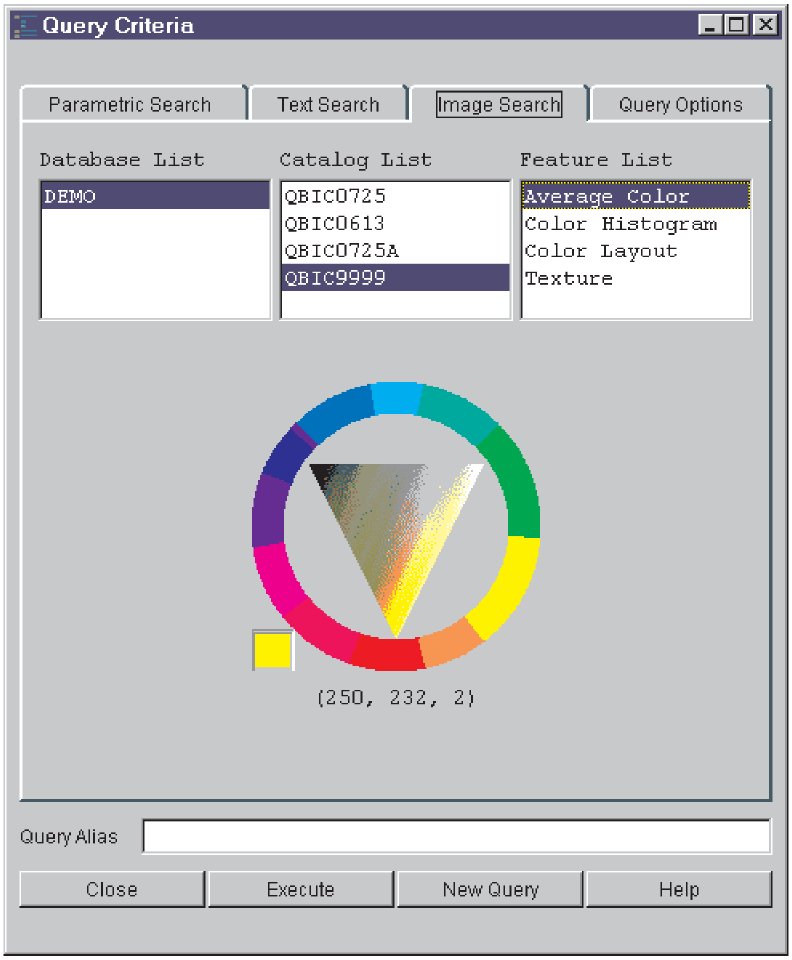
```
DKDatastoreTS dsTS;
dkResultSetCursor* pCur = 0;

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);

// process the results

dsTS.disconnect();
```

The complete sample application from which this example was taken (`TExecuteStructDocTS.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

Searching images by content

The Image Search server can search for stored images when you specify the image type, or provide an example of the image.

Figure 16 on page 272 shows a sample application that connects to the image search server. The image search server uses Query by Image Content (QBIC) technology to search for similar colors, layouts, and patterns.



Database List
DEMO

Catalog List
QBIC0725
QBIC0613
QBIC0725A
QBIC9999

Feature List
Average Color
Color Histogram
Color Layout
Texture

(250, 232, 2)

Query Alias

Figure 16. Image search sample client

Understanding image search terms and concepts

This section describes the image search components: the server, databases, catalogs, and the relationship of the image search server to earlier Content Manager. It also describes *features* that are the searchable visual characteristics of images.

Understanding image search servers, databases, and catalogs: Earlier Content Manager uses an image search server to search for images. Content Manager applications store image objects in the object server. The image search server analyzes and indexes the image information. The image search server does not store images themselves.

A content server defined by a DKDatastoreQBIC object represents the image search server. The results of an image search include identifiers (item IDs) that describe the location of the image in the Content Manager server. You can use these identifiers with other results, such as the part number and RepType, to retrieve the image.

You can perform queries on the content server. However, the content server for image search does not support add, update, retrieve, and delete operations. Figure 17 on page 273 shows an example of an image search server.

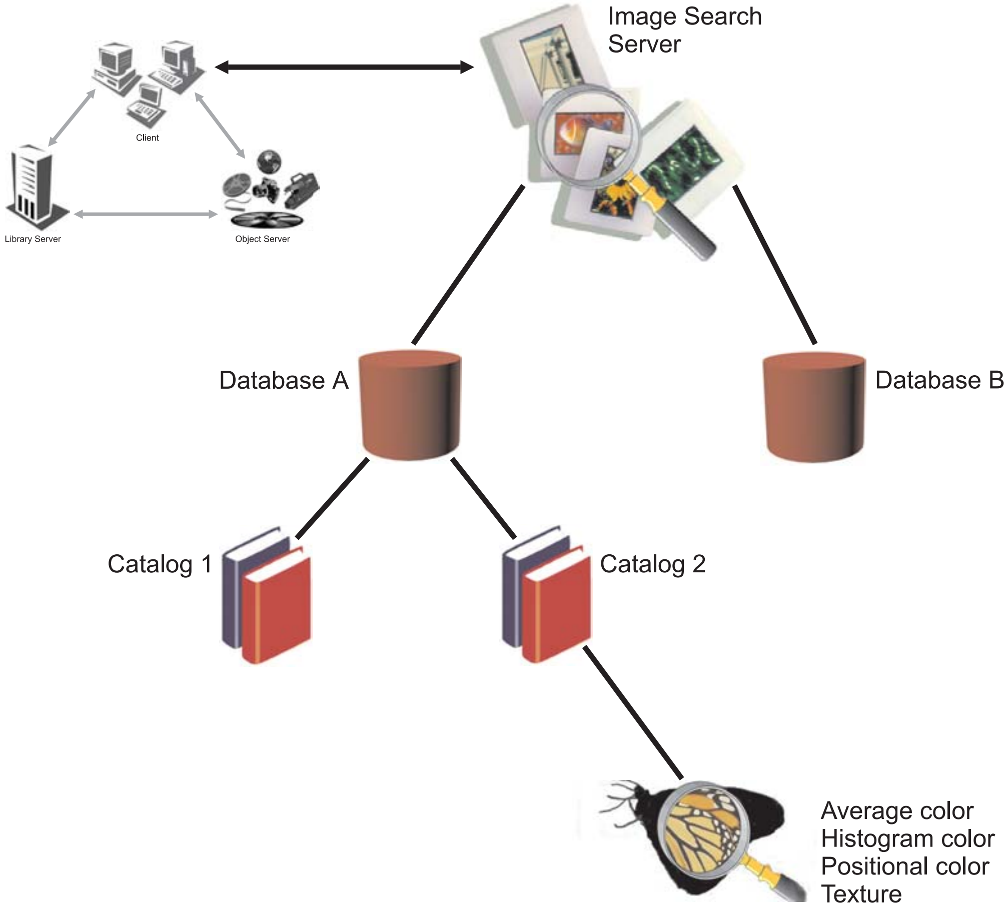


Figure 17. An image search server in an earlier Content Manager system

The image search server can contain one or more databases. Each database can contain one or more catalogs, which hold information about the visual features of images. These four image search features are:

- Average color.
- Histogram color.
- Positional color (color layout).
- Texture.

Understanding image search features: The four image search features and their purposes are defined in this section:

Average color Use average color to search for images that have a predominant color. Images with similar predominant colors have similar average colors. For example, images that contain equal portions of red and yellow have an average color of orange.

QbColorFeatureClass if the feature name for average color.

Histogram color

Measures the percentages of color distribution of an image. Histogram analysis separately measures the different colors in an image. For example, an image of the countryside has a histogram color that shows a high frequency of blue, green, and gray.

Use histogram color to search for images that contain a similar variety of colors. QbColorHistogramFeatureClass is the feature name for histogram color.

Positional color (color layout)
Positional colors measure the average color value for the pixels in a specified area of an image. For example, images with bright red objects in the middle have a positional color of bright red.

QbDrawFeatureClass is the feature name for positional color.

Texture
Use texture to search for images that have a particular pattern. Texture measures the coarseness, contrast, and directionality of an image. Coarseness indicates the size of repeating items in an image. Contrast identifies the brightness variations in an image. Directionality indicates whether a direction predominates in an image. For example, an image of a wood grain has a similar texture to other images that contain a wood grain.

QbTextureFeatureClass is the feature name for texture.

Using image search applications

Image search client applications create image queries, run them, and then evaluate the information returned by the image search server. Before an application can search images by content, the images must be indexed, and the content information must be stored in an image search database.

Restriction: You cannot index existing images in your object server. You can index only the images you create in your object server after you install the image search server and client. Figure 18 on page 275 shows an example of the client and retrieve images.

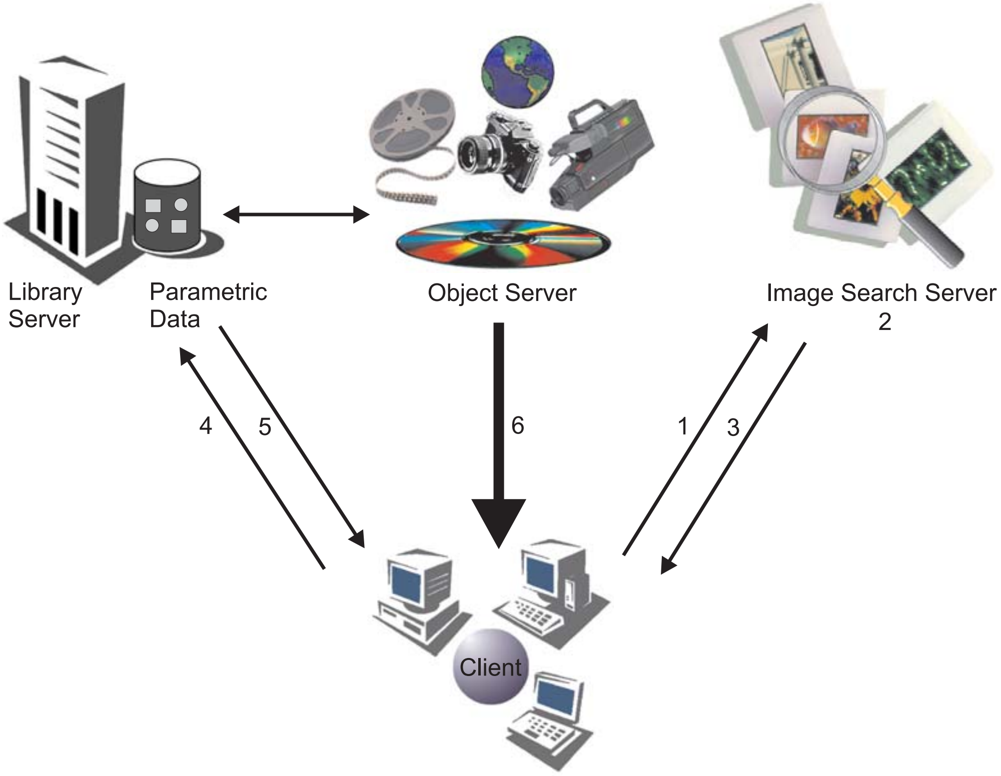


Figure 18. How image search clients search and retrieve images

To perform an image search:

1. A client builds a QBIC query string and sends it to an image search server.
2. Image search server receives the query string and searches the cataloged images for matches.
3. Client receives the matches as a list of identifiers. The identifier for each matching image consists of the:
   - Item ID.
   - Part number.
   - RepType.
   - Rank.
4. Client requests the image part and index information from a library server.
5. Library server returns index information, such as a text description, to the client. The library server also requests that an object server send specified image parts to the client.
6. Object server sends image parts and the client acknowledges receiving them.

Creating queries

When you create queries, you construct a query string that the application passes to the image search server. An image query is a character string that specifies the search criteria. The search criteria consists of:

An image query is a character string that specifies the search criteria. The search criteria consists of:

Feature name The features used in the search.

Feature value The values of those features. Table 19 shows the image search feature names and the values that can be passed in a query string.

Feature weight
The relative weight or emphasis placed on each feature. The weight of a feature indicates the emphasis that the image search server places on the feature when calculating similarity scores and returning results for a query. The higher the specified weight, the greater the emphasis.

Maximum results
In addition to defining the type of images a query will look for, you can specify the maximum number of matches that the query will return.

A query string has the form: `feature_name value`, where `feature_name` is an image search feature name, and `value` is a value associated with the feature. If you use more than one feature in a query, then you must specify a feature name-value pair for each feature. The string ″and″ separates each pair.

Image search queries have the following syntax:

```
feature_name value
feature_name value weight
```

You cannot repeat a feature within a single query. You can specify multiple features in a query. When you specify multiple features in a query, you can assign a weight to one or more of the features. Queries that include the emphasis for each feature have the form: `feature_name value weight`, where `feature_name` is an image search feature name, `value` is a value associated with the feature, and `weight` is the weight assigned to the feature. `weight` is the combination of the keyword `weight`, an equal sign (=), and a real number greater than 0.0.

You can also specify the maximum number of matches that a query returns. To specify the maximum results, append `and max_results` to your query. `max_results` consists of the keyword `max`, an equal sign (=), and an integer greater than 0. Table 19 describes feature names and values.

Table 19. Image search query: valid feature values

Feature name	Values

QbColorFeatureClass or QbColor	**color = < rgbValue , rgbValue , rgbValue >**
	where rgbValue is an integer from 0 to 255.
	file = < fileLocation , ″ fileName ″ >
	where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides. For example, you can search using an average color and a texture value. The texture value is provided by an image in a client file. The weight of the texture is twice that of the average color:
	`QbColorFeatureClass color=`
	`<50, 50, 50> and QbTextureFeatureClass`
	` file=<client, "\patterns\pattern1.gif">`
	` weight=2.0`

Table 19. Image search query: valid feature values (continued)

Feature name	Values
QbColorHistogramFeatureClass or QbHistogram	**histogram = < histList >** where histList consists of one or more `histClause` separated by a comma (,). A `histClause` is specified as (`histValue, rgbValue , rgbValue , rgbValue`), where `histValue` is an integer from 1 to 100 (a percentage value), and `rgbValue` is an integer from 0 to 255. **file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

QbDrawFeatureClass or QbDraw	**description = < ″ descString ″ >**
	where descString is a special encoded string describing a picker file. Format of the description string:
	1. Dw,h specifies the outer dimensions of the image itself with width w and height h.
	2. Rx,y,w,h,r,g,b specifies that a rectangle of width w and height h is to be positioned with its upper left corner at the coordinates (x,y)—with respect to an origin in the upper left corner of the image rectangle—and this rectangle should have color values r (red), g (green), and b (blue).
	3. Use the colon character (:) is used as a separator.
	For example, you can search for color layout (QbDrawFeatureClass) described by the description string:
	`QbDrawFeatureClass description=`
	`<"D100,50:R0,0,50,50,255,0,0"`
	file = < fileLocation , ″ fileName ″ >
	where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

QbTextureFeatureClass or QbTexture	**file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

Query examples:

1. Search for average color red:

   ```
   QbColorFeatureClass color=<255,0,0>
   ```

2. Search using a histogram of three colors, 10% red, 50% blue, and 40% green:

   ```
   QbColorHistogramFeatureClass histogram=
   <(10, 255, 0, 0) (50, 0, 255, 0), (40, 0, 0, 255)>
   ```

3. Search using an average color and a texture value. The texture value is provided by an image in a file on the client. The weight of the texture is twice that of the average color:

   ```
   QbColorFeatureClass color=
   <50, 50, 50> and QbTextureFeatureClass file=<client, "\patterns\pattern1.gif">
    weight=2.0
   ```
4. Search for the described color layout:

   ```
   QbDrawFeatureClass description=<"D100,50:R0,0,50,50,255,0,0">
   ```
5. Search for average color red and limiting the returned matches to five:

   ```
   QbColorFeatureClass color=<255,0,0> and max=5
   ```

Running queries and evaluating search results

Applications use the image search API to issue queries and evaluate search results. If information in the image search database matches the image search criteria, then an identifier of the matching image or images is returned. This identifier is a dynamic data object (DDO) that corresponds to an image part inside a Content Manager object.

Establishing a connection in QBIC

Image search provides functions for connecting and disconnecting to the content server. The following example shows how to connect to an image search server named QBICSRV using the user ID QBICUSER and the password PASSWORD.

Java

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
  ...           //  Process as appropriate
dsQBIC.disconnect();
```

The complete sample application from which this example was taken (TConnectQBIC.java) is available in the CMBROOT\Samples\java\dl directory.

C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
...                        // do some work
dsQBIC.disconnect();
```

The complete sample application from which this example was taken (TConnectQBIC.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

The image search connection allows an application to connect to an image search server.

After connecting, your program can use functions that access the image search server, except for the functions that are not related to image search catalogs, such as listDatabases. An openCatalog function is required to open a catalog for processing. A closeCatalog function is called after processing is done. The following example shows how to connect, open a catalog, close the catalog, and disconnect.

Java

```
// ----- Create a QBIC datastore and connect
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
// ----- open the catalog
dsQBIC.openCatalog("DEMO", "QBIC0725");
  ...         //   Do some processing
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

C++

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "QBIC0725");
...                       // do some work
dsQBIC.closeCatalog();
dsQBIC.disconnect();
```

Listing image search servers

The image search server provides a function for listing the image search servers that it can connect to. The following example shows how to retrieve (in a DKSequentialCollection object) the list of servers that contain DKServerInfoQBIC objects. After you get a DKServerInfoQBIC object, you can retrieve the server name, the host name, and the port number.

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();
     .....      // Process each server as appropriate
}
```

The complete sample application from which this example was taken (`TListCatalogQBIC.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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;
```

The complete sample application from which this example was taken (`TListCatalogQBIC.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

Listing image search databases, catalogs, and features

DKDatastoreQBIC provides a function for listing all of the image search databases, catalogs, and features on an image search server. The list is returned in a DKSequentialCollection object that contains DKIndexQBIC objects. After you get a DKIndexQBIC object, you can retrieve the database, catalog, and feature name. The following example shows how to retrieve the list of databases, catalogs, and features.

Java

```
 // ----- Create the datastore and connect
 DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
 dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");

 // ---- Get the list of servers
 col = (DKSequentialCollection)dsQBIC.listDataSources();
 iter = col.createIterator();
 while (iter.more()) {
    srvDef = (DKServerDefQBIC)iter.next();
    .....      // Process each server as appropriate
 }

// ----- Get the list of QBIC Databases
col = (DKSequentialCollection)dsQBIC.listEntities();
iter = col.createIterator();
while (iter.more()){
   dbDef = (DKDatabaseDefQBIC)iter.next();
   // ----- Get the list of catalogs for the database
   col2 = (DKSequentialCollection)dbDef.listSubEntities();
   iter2 = col2.createIterator();
   while (iter2.more()){
      catDef = (DKCatalogDefQBIC)iter2.next();
      // ----- Get the list of features for the catalog
      col3 = (DKSequentialCollection)catDef.listAttrs();
      iter3 = col3.createIterator();
      while (iter3.more()){
        featDef = (DKFeatureDefQBIC)iter3.next();
        .... // Process the features as appropriate
      }
   }
 }
dsQBIC.disconnect();
dsQBIC.destroy();
....
```

The complete sample application from which this example was taken (`TListCatalogQBIC.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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();
         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;
    }
// continued...
```

C++ (continued)

```
     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();
```

The complete sample application from which this example was taken (`TListCatalogQBIC.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

Representing image search information with a DDO

A DDO associated with DKDatastoreQBIC contains specific information for representing image search results. A DDO resulting from an image query corresponds to an image part inside an item; it has the following set of standard attributes:

DKDLITEMID
The item ID for the item to which this image part belongs. Use the item ID to retrieve the whole item from the content server.

DKPARTNO
An integer part number of this image part. Use this with the item ID to retrieve this part from the content server.

DKREPTYPE
A string for representation type (RepType). The default value is `FRN$NULL`. This attribute is reserved.

DKRANK
An integer rank indicating the relevance of this part to the results set of the image query. The rank is within the range 0 to 100. A higher rank means a better match.

The PID for an image search DDO has the following information:

content server type
QBIC.

content server name
The server name used to connect to the content server.

ID The zero-based sequence number of the DDO in the results set.

As a convention, the attribute value is always an object.

Working with image queries

This section describes how to run and evaluate image queries.

Running an image query

Using an instance of dkQuery from DKDatastoreQBIC, you can create a query object to run the query and obtain the results. The following example shows how to create an image query object and run it. After you run a query, the results are returned in a DKResults collection.

Java

```
// ----- Generate a query string; then create the datastore and connect
String cmd = "QbColor color=<255, 0, 0>";
DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
 // ----- Open the catalog
dsQBIC.openCatalog("DEMO", "qbic0725");

  ...   // Process as appropriate

// ----- Create the query and run it
dkQuery pQry = dsQBIC.createQuery(cmd, DK_IMAGE_QL_TYPE, parms);
pQry.execute(parms);
// ----- Get the results and process
DKResults pResults = (DKResults)pQry.result();
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
   item = (DKDDO)pIter.next();
     // Process the DKDDO
 }
 dsQBIC.closeCatalog();
 dsQBIC.disconnect();
 dsQBIC.destroy();
 ...
```

The complete sample application from which this example was taken (`SampleIQryQBIC.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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();
  // Process the DKDDO
  ...
  }
delete pIter;
delete pResults;
delete pQry;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

The complete sample application from which this example was taken (TSampleIQryQBIC.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Running an image query from the content server

As an alternative, you can use the execute function of DKDatastoreQBIC to run a query. The results are returned in a dkResultSetCursor object. The following example shows how to run an image query on the content server. Results are returned in a dkResultSetCursor object.

Java

```
// ----- Generate a query string; then create the datastore and connect
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");
 // ----- Execute the query from the datastore
 dkResultSetCursor pCur = dsQBIC.execute(cmd, DK_IMAGE_QL_TYPE, parms);
 while (pCur.isValid())
 {
    item = pCur.fetchNext();
    ....     // Process the DKDDO
  }
  pCur.destroy();
  dsQBIC.closeCatalog();
  dsQBIC.disconnect();
  dsQBIC.destroy();
```

The complete sample application from which this example was taken (TExecuteQBIC.java) is available in the CMBROOT\Samples\java\dl directory.

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)
   {
     // Process the DKDDO
     ...
     delete item;
   }
 }
delete pCur;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

The complete sample application from which this example was taken (TExecuteQBIC.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Evaluating an image query from the content server

DKDatastoreQBIC also provides a function to evaluate a query. The following example shows how to evaluate an image query from the content server. Results are returned in a DKResults collection.

Java

```
// ----- Generate a query string; then create the datastore and connect
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");

// ----- Use evaluate to run the query
DKResults pResults=(DKResults) dsQBIC.evaluate(cmd,DK_IMAGE_QL_TYPE,parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
    item = (DKDDO)pIter.next();
     ...   // Process the 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();
  // Process the DKDDO
  ...
  }
delete pIter;
delete pResults;
dsQBIC->closeCatalog();
dsQBIC->disconnect();
```

Using the image search engine

You can use the image search server to specify a query based on one of the following features: average color, color percentages, color layout, and textures. You can also specify multiple features in a query. The query results contain the item ID, part number, representation type, and ranking information. You can use this information to create an XDO for retrieving the image contents.

Loading data to be indexed for image search

To load data into a Content Manager server to be indexed by the image search server, you must create a Content Manager index class, an image search database, and an image search catalog. The database is a collection of image search catalogs. A catalog holds data about the visual features of images.

The image search features need to be added to the catalog for indexing. You should add all supported features to the catalog.

The image search server must be running when you create an image search database and catalog. Make sure your environment is set up properly.

After the data is loaded into Content Manager, you can place the image in the image queue. In the system administration program, select **Process Image Queue**. After the indexing is complete, you can run image searches.

Indexing an existing XDO using search engines

You can index an existing XDO using a specified search engine. The following example calls the setToBeIndexed function of the DKBlobDL class.

Java

```
try
{
    // ----- Create the datastore and connect
    DKDatastoreDL dsDL = new DKDatastoreDL();
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");

    // ----- Create the XDO and PID and set attributes
    DKBlobDL axdo = new DKBlobDL(dsDL);
    DKPidXDODL  apid = new DKPidXDODL();
    apid.setPartId(partId);
    apid.setPrimaryId(itemId);
    axdo.setPidObject(apid);

    // ----- Set search engine information
    DKSearchEngineInfoDL aSrchEx = new DKSearchEngineInfoDL();
    aSrchEx.setSearchEngine("SM");
    aSrchEx.setSearchIndex("TM-TMINDEX");
    aSrchEx.setSearchInfo("ENU");
    axdo.setExtension("DKSearchEngineInfoDL", (dkExtension)aSrchEx);
    // ----- Call setToBeIndexed on the XDO
    axdo.setToBeIndexed();

    dsDL.disconnect();
    dsDL.destroy();
}
catch (DKException exc)
{
    ... // Handle the DKException
}
catch (Exception exc)
{
    ... // Handle the Exception
 }
```

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
 {
    //replace following with your library server, userid, password
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
    cout << "datastore connected" << endl;

    DKBlobDL* axdo = new DKBlobDL(&dsDL);
    DKPidXDODL*  apid = new DKPidXDODL;
    apid ->setPartId(partId);
    apid ->setId(itemId);
    axdo ->setPid(apid);
    axdo ->setRepType(repType);
    cout<<"itemId= "<<(axdo->getPid())->getId()<<endl;
    cout<<"partId= "<<((DKPidXDODL*)(axdo->getPid()))->getPartId()<<endl;
    cout<<"repType= "<<axdo->getRepType()<<endl;

// continued...
```

C++ (continued)

```
    //--- set 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;
}
```

Using combined query

Use a *combined query* to execute a combination of parametric and text queries, with or without a scope. A *scope* is a DKResults object formed from a previous parametric or text query. The result is an intersection between the scopes and the results of each query. Therefore, if you are not careful when formulating the query and including scopes, individual query results might not intersect and the result of the combined query is empty.

If there is at least one parametric and one text query, the resulting DDO has the attribute DKRANK, which signifies the highest rank of the matching part belonging to the document.

Restriction: For each query in a combined query, you must use a different connection to the search engine; you cannot route multiple queries through the same connection.

Combined parametric and text queries

To run a combined query made up of one parametric and one text query, without a scope, you must create a combined query object and pass the two queries as input parameters to be run by the combined query. For example:

Java

```
// ----- Create a pre-Version 8.1 Content Manager datastore and connect
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ------ Create a text search datastore and connect
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');  // TM is a local alias for
...                         //     the Text Search Engine server

// ----- Generate the parametric query string and create the query
String pquery = "SEARCH=(INDEX_CLASS=GRANDPA, COND=(DLSEARCH_Date > 1994));";
DKParametricQuery pq =
 (DKParametricQuery) dsDL.createQuery(pquery, DK_CM_PARAMETRIC_QL_TYPE, null);

// ----- Generate the text query string and create the query
String tquery = "SEARCH=(COND=(Tivoli)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
    (DKTextQuery) dsTS.createQuery(tquery, DK_CM_TEXT_QL_TYPE, null);

// ----- Create the combined query
DKCombinedQuery cq = new DKCombinedQuery();

// ----- Package the queries in DKNVPair as input parameters
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);    // to signal the end of parameter list

// ----- Execute the combined query
cq.execute(par);

// ----- Get the results
DKResults res = (DKResults) cq.result();
if (res != null) {
    ...  // process the results
}
```

C++

```
DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

DKDatastoreTS dsTS;
// TM is a local alias for the Text Search Engine server
dsTS.connect("TM","",' ');
// create a parametric query
DKString pquery="SEARCH=(INDEX_CLASS=GRANDPA,COND=(DLSEARCH_Date > 1994));";
DKParametricQuery* pq =
  (DKParametricQuery*) dsDL.createQuery(pquery,DK_PARAMETRIC_QL_TYPE, NULL);

// create a text query
DKString tquery = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery* tq =
    (DKTextQuery*) dsTS.createQuery(tquery,DK_TEXT_QL_TYPE, NULL);

// create a combined query
DKCombinedQuery* cq = new DKCombinedQuery();

// package the queries in DKNVPair as input parameters
DKNVPair par[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
// to signal the end of parameter list
par[2].setName(DK_PARM_END);

// execute the combined query
cq->execute(par);

// get the results
DKAny any = cq->result();
DKResults* res = (DKResults*) any.value();
if (res != NULL) {
    // process the results
    ...
}
```

The last `if` statement is necessary to ensure that the DKResults object is not null.

Using a scope

If you have a DKResults object that you want to use as the scope, pass it as an additional query parameter. The following example illustrates using a DKResults object to function as a scope for a combined query:

Java

```
// ----- This scope is the result of a parametric query
DKResults scope;
// ------ This scope is the result of a previous text query
DKResults tscope;

// ----- Package the query in array if DKNVPairs as input parameters
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);

// ----- Execute the combined query
cq.execute(par);
....
```

C++

```
DKResults* scope;    // assume that this is the scope
                     // initialized somewhere as a result of
                     // some parametric query
DKResults* tscope    // assume that this is the scope
                     // initialized somewhere as a result of
                     // some text query


...
// package the query in DKNVPair as input parameters
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);
// execute the combined query
cq->execute(par);
...
```

The results of one combined query can also be used as a scope for another combined query, and sometimes you can query the results.

Ranking

If the combined query contains at least one text query, then the resulting DDO has the attribute DKRANK. This attribute is not stored, but is computed each time by the Text Search Engine. The value of the rank corresponds to the highest rank of the part in the document that satisfies the text query conditions.

Tips

If you have several parametric queries and scopes, it is more efficient to run one complete query. This is also true for text queries.

The query option "MAX_RESULTS=nn" limits the number of returned results. Usually, this option is more applicable to text queries, because the result is sorted in descending order by rank. If this option is set to 10, for example, it means that the caller only wants the 10 highest matching results.

The meaning of the query option "MAX_RESULTS=nn" is different for parametric queries. Because there is no notion of rank, the caller gets the first 10 results. The results are intersected with the result from the text query. Therefore, when combining parametric and text queries, it is advisable not to specify the query option "MAX_RESULTS=nn" for the parametric query.

Understanding the earlier Content Manager workflow and workbasket functions

This section describes the earlier Content Manager workflow and workbasket functions.

Understanding the earlier Content Manager workflow service

A *workbasket* is a container that holds documents and folders that you want to process. A *workflow* is an ordered set of workbaskets that represent a specific business process. Folders and documents move between workbaskets within a workflow, allowing your applications to create simple business models and route work through the process until completion.

The workflow model in Content Manager follows these rules:

- A workbasket does not need to be located in a workflow.
- A workbasket can be located in one or more workflows.
- A workbasket can be in the same workflow more than once.
- A document or folder can only be stored in one workflow at a time.
- A document or folder can be stored in a workbasket that is not located in a workflow.

The Enterprise Information Portal APIs provide classes to work with Content Manager workflow.

The DKWorkFlowServiceDL class represents the workflow service of Content Manager. This class provides the capability to start, change, remove, route, and complete a document or folder in a workflow. Additionally, you can use the DKWorkFlowServiceDL class to retrieve information about workbaskets and workflows.

The DKWorkFlowDL and DKWorkBasketDL classes are the object-oriented representations of a workflow item and a workbasket item, respectively.

Establishing a connection

You must establish a connection to a Content Manager server before you can use the workflow service. The content server provides connection and disconnection functions.

The following example shows how to connect to a Content Manager server named LIBSRVRN, using the user ID FRNADMIN and the password PASSWORD.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
...                  //  Process as appropriate
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (`TListWorkFlowWFS.java`) is available in the `CMBROOT\Samples\java\dl` directory.

C++

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
...                                  // do some work
dsDL.disconnect();
```

The complete sample application from which this example was taken (`TListWorkFlowWFS.cpp`) is available in the `Cmbroot/Samples/cpp/dl` directory.

Creating a workflow

Use DKWorkflowServiceDL to create a workflow. To do this, you typically complete the following six steps:

1. Create an instance of DKWorkFlowDL.
2. Set the workflow name (″GOLF″).
3. Set the workbasket sequence (″NULL″) to indicate that this workflow contains no workbaskets.
4. Set the privilege (″All Privileges″).
5. Set the disposition (DK_WF_SAVE_HISTORY).
6. Call the add function add ().

The example follows the six steps to create a workflow.

Java

```
// ----- Create the datastore and the CM workflow services
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);

// ----- Set the access option and connect
Object input_option = new Integer(DK_SS_CONFIG);
dsDL.setOption(DK_OPT_DL_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");

// ------ Create the CM workflow
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();
...     //  Processing as appropriate
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TCreateDelWorkFlow.java) is available in the CMBROOT\Samples\java\dl directory.

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();
...                                 // do some work
dsDL.disconnect();
```

The complete sample application from which this example was taken (TCreateDelWorkFlowWFS.cpp) is available in the Cmbroot/Samples/cpp/dl directory

Important: If you connect to the content server as a normal user (DK_SS_NORMAL), you do not get the workflow defined after you connect. Therefore, this sample uses DK_SS_CONFIG.

Listing workflows

DKWorkflowServiceDL provides a function for listing the workflows in the system as shown in the following example. The list is returned in a sequential collection of DKWorkFlowDL objects.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Get a list of the CM workflows
DKSequentialCollection wfList=(DKSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
{
   dkIterator pIter = wfList.createIterator();
   DKWorkFlowDL pwf1;
   while (pIter.more())
   {
      pwf1 = (DKWorkFlowDL)pIter.next();
      pwf1->retrieve();
      ...                       // Process as appropriate
   }
}
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TListWorkFlowWFS.java) is available in the CMBROOT\Samples\java\dl directory.

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 (pIter->more())
      {
         pwf1 = (DKWorkFlowDL *)((void*)(*pIter1->next()));
         pwf1->retrieve();
         ...                       // do some work
         delete pwf1;
      }
   }
dsDL.disconnect();
```

The complete sample application from which this example was taken (TListWorkFlowWFS.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Creating a Content Manager workbasket

Use DKWorkflowServiceDL to create a workbasket. To do this, you typically complete the following steps:

1. Create an instance of DKWorkBasketDL.
2. Set the workbasket name (Hot Items).
3. Set the privilege (All Privileges).
4. Call the add function.

The following example follows these steps to create a workbasket. If you connect to the content server as a normal user (DK_SS_NORMAL), you do not get the workbasket defined after you connect. Therefore, this sample uses 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");
// ----- Create the CM workbasket and set properties
DKWorkBasketDL newwb = new DKWorkBasketDL(wfDL);
newwb.setName("Hot Items");
newwb.setAccessList("All Privileges");
newwb.add();
...     // Process as appropriate
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TCreateDelWorkBasket.java) is available in the CMBROOT\Samples\java\dl directory.

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();
...                               // do some work
dsDL.disconnect();
```

The complete sample application from which this example was taken (TCreateDelWorkBasket.cpp) is available in the CMBROOT\Samples\cpp\dl directory.

Listing workbaskets

DKWorkflowServiceDL provides a function for listing the workbaskets in the system as shown in the following example. The list is returned in a sequential collection of DKWorkBasketDL objects.

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 pwb1;
      while (pIter1.more())
      {
         pwb1 = (DKWorkFlowDL)pIter1.next();
         pwb1->retrieve();
         ...                       // do some work
      }
   }
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TListWorkBasketWFS.java) is available in the CMBROOT\Samples\java\dl directory.

C++

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1 =
  (DKSequentialCollection *)wfDL.listWorkBaskets();
if (wbList1 != NULL)
   {
      dkIterator * pIter1 = wbList1->createIterator();
      DKWorkBasketDL * pwb1;
      while (pIter->more())
      {
         pwb1 = (DKWorkBasketDL *)((void*)(*pIter1->next()));
         pwb1->retrieve();
         ...                       // do some work
         delete pwb1;
      }
   }
dsDL.disconnect();
```

The complete sample application from which this example was taken (TListWorkBasketWFS.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Listing items in an earlier Content Manager workflow

DKWorkflowServiceDL provides a function for listing the item IDs of the items in a workflow as shown in the following example. The list is returned in a sequential collection of DKString objects.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ----- Get the list of CM workflows
KSequentialCollection wfList = (DKSequentialCollection)wfDL.listWorkFlows();
if (wfList != null)
{
  dkIterator pIter = wfList.createIterator();
  while (pIter.more())
  {
    DKWorkFlowDL pwf1 = (DKWorkFlowDL)pIter.next();
    // ----- Get the list of items in the CM workflow
DKSequentialCollection itemList=(DKSequentialCollection)pwf1.listItemIDs();
    if (itemList != null)
    {
       dkIterator iter1 = itemList.createIterator();
       String itemid;
       while (iter1.more())
       {
          itemid = (String)iter1.next();
          // ----- Process the items using the item ID
       }
    }
  }
}
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TListItemsWFS.java) is available in the CMBROOT\Samples\java\dl directory.

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());
         ...                       // do some work
      }
   }
dsDL.disconnect();
```

The complete sample application from which this example was taken (TListItemsWFS.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Executing an earlier Content Manager workflow

DKWorkflowServiceDL provides functions for executing a workflow. The following example demonstrates how to start an item in a workflow, how to route an item to a workbasket, and how to complete an item in a workflow. To use this sample you must modify it to:

- Use a valid item ID instead of EP8L8OR9MHH##QES.
- Use a valid workflow ID instead of HI7MOPALUPFQ1U47.
- Use a valid workbasket ID instead of E3PP1UZOZUFQ1U3M.

Java

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
DKString itemID   = new String("EP8L8OR9MHH##QES");
DKString itemIDWF = new String("HI7MOPALUPFQ1U47");
DKString itemIDWB = new String("E3PP1UZOZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkFlowItem(itemID,                  // itemID
                       itemIDWF,                // itemIDWB
                       NULL,                    // default (1st workbasket)
                       TRUE,                    // overload
                       DK_WIP_DEFAULT_PRIORITY  // initial_priority
                       );
...                                         // do some work
wfDL.routeWipItem(itemID,                   // itemID
                  itemIDWF,                 // itemIDWB
                  TRUE,                     // overload
                  DK_NO_PRIORITY_CHANGE     // initial_priority
                  );
...                                         // do some work
wfDL.completeWorkFlowItem(itemID);
dsDL.disconnect();
dsDL.destroy();
```

The complete sample application from which this example was taken (TProcessWFS.java) is available in the CMBROOT\Samples\java\dl directory.

C++

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
DKString itemID   = DKString("EP8L8OR9MHH##QES");
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
DKString itemIDWB = DKString("E3PP1UZOZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkFlowItem(itemID,                   // itemID
                       itemIDWF,                 // itemIDWB
                       NULL,                     // default(1st workbasket)
                       TRUE,                     // overload
                       DK_WIP_DEFAULT_PRIORITY   // initial_priority
                      );
...                                         // do some work
wfDL.routeWipItem(itemID,                   // itemID
                  itemIDWF,                 // itemIDWB
                  TRUE,                     // overload
                  DK_NO_PRIORITY_CHANGE     // initial_priority
                      );
...                                         // do some work
wfDL.completeWorkFlowItem(itemID);
dsDL.disconnect();
```

The complete sample application from which this example was taken (TProcessWFS.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Working with OnDemand

Enterprise Information Portal provides a connector and related classes for accessing content on Content Manager OnDemand servers. The OnDemand classes and APIs allow you to:

- Connect to and disconnect from OnDemand servers.
- List application groups and application group fields.
- List OnDemand folders.
- Query an application group.
- Search and retrieve documents using the folder or the application group interface.
- Treat OnDemand folders as native entities in federated searches.
- Search synchronously and asynchronously in either the application group or folder mode.
- Retrieve entire OnDemand documents or document segments.
- Retrieve the logical view data of a given OnDemand document.
- Retrieve the resource group for a given OnDemand document.
- Retrieve annotation data for a given OnDemand document.
- Create and modify annotations.

Restriction: OnDemand does not support Text Search Engine and QBIC search or Combined query.

Representing OnDemand servers and documents

You represent a Content Manager OnDemand content server using a DKDatastoreOD in an Enterprise Information Portal application, and you represent an OnDemand document as a DDO using a DKDDO. OnDemand DDOs contain the following information:

- Document attribute names and their values
- Document data and annotations (represented as DKParts)
- Collection of logical views for a document
- Resource group data

An OnDemand document's attributes are stored in a DKDDO as properties. An OnDemand document's segments and notes are stored as DKParts.

All other document data (resource group and views, both fixed and logical), are stored as special properties in an OnDemand DDO with the following property names are reserved for the OnDemand:

DKViews
A collection of logical views

DKFixedView
Contains fixed view information

DKResource
Contains resource group data

Notes:

1. An Enterprise Information Portal administrator must properly define the OnDemand connector by specifying the string in the **Additional Parameters** field on the **Initialization Parameters** page. The string should look like this: `ENTITY_TYPE=TEMPLATES;;` Be sure to include the two semicolons.
2. In Enterprise Information PortalVersion 8.2, `DKViews`, `DKFixedView`, and `DKResource` replace `DKViewDataOD`, `DKFixedViewDataOD`, and `DKResouceGrpOD` repsectively.

Connecting to and disconnecting from the OnDemand server

To log into an OnDemand content server, pass in the server name (for example, `ODServer.mycompany.com`), user ID, and password through the `connect` method.

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
System.out.println("connecting to datastore ...");
dsOD.connect(ODServer, UserID, Password, "");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD();
cout << "connecting to datastore ..." << endl;
dsOD->connect(ODServer, UserID, Password, "");
```

Use the `disconnect` method to log out of the OnDemand server.

Java

```
System.out.println("disconnecting from the datastore ...");
dsOD.disconnect();
dsOD.destroy(); // Finished with the datstore
```

C++

```
cout << "disconnecting from the datastore ..." << endl;
dsOD->disconnect();
delete dsOD;
```

Listing information on OnDemand

You can list application groups and folders for OnDemand servers.

Listing application groups

You can list application groups in OnDemand using the listEntities() method of DKDatastoreOD. The following example illustrates how to use this method.

Java

```
...
pCol = (DKSequentialCollection) dsOD.listEntities(); //get application groups
pIter = pCol.createIterator();
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++

```
  // ----- Show the application groups
  // ----- First get the groups
  pCol = (DKSequentialCollection*)dsOD.listEntities();
  pIter = pCol->createIterator();
  int i = 0;
  // ---- Process the list
  while (pIter->more() == TRUE)
  {
    i++;
    agDef = (DKAppGrpDefOD*)((void*)(*pIter->next()));
    strAppGrp = agDef->getName();
    cout << "  app group name[" << i << "]: ==>" << strAppGrp << endl;
    . . .
```

The following example illustrates getting the attribute information for each application group:

Java

```
...
pCol2 = (DKSequentialCollection) dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2.createIterator();
j = 0;

while (pIter2.more() == true)
 {
   j++;
   attrDef = (DKFieldDefOD)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());
   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("  " + j + " attribute(s) listed for " +
                    strAppGrp + " app grp\n");
...
```

C++

```
// ----- Get the attributes for each of the entities(application groups)
pCol2 = (DKSequentialCollection*)dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2->createIterator();
int j = 0;
// ----- List the attributes
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;
   // ----- Clean up the attribute
   delete attrDef;
}
cout << "  " << j << " attribute(s) listed for the " << strAppGrp
     << " app group\n" << endl;
// ----- Clean up the iterators and collections
if ( pIter2 )
   delete pIter2;
if ( pCol2 )
   delete pCol2;
 . . .
```

Listing OnDemand folders

To get a list of folders in an OnDemand content server, you use the listSearchTemplates() function.

Java

```
...
dsDef = (DKDatastoreDefOD)dsOD.datastoreDef();
pCol = (DKSequentialCollection) dsDef.listSearchTemplates();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
   i++;
   folderName = (String)pIter.next();
   ....   // Process the folder as appropriate
}
dsOD.disconnect();
dsOD.destroy();
```

C++

```
  . . .
 // ----- List the folders
 dsDef = (DKDatastoreDefOD*)dsOD.datastoreDef();
 pCol = (DKSequentialCollection*)dsDef->listSearchTemplates();
 pIter = pCol->createIterator();
 i = 0;
 // ----- Process the list of folders
 while (pIter->more() == TRUE)
 {
    i++;
    folderName = (DKString)(*pIter->next());
    cout << "folder name [" << i << "] - " << folderName << endl;
 }
 // ----- Disconnect
 dsOD.disconnect();
 . . .
```

Retrieving an OnDemand document

In an OnDemand server, you can retrieve documents. You can also display documents with their parts and attributes.

Searching for a particular document

The following example searches against an application group (OnDemand Publications) in the OnDemand content server.

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;
```

The following example searches against an application group (`OnDemand Publications`), and retrieves the documents returned.

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("create a new DDO with a cloned pid to retrieve!");
      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(); // Finished with the cursor
```

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;
}
```

Displaying documents and their parts and attributes

The following example displays the documents found by the query with their parts and attributes:

Java

```
//-----For each data item, get the attributes
//-----numDataItems is the number of data items
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));
}
//-----Check for the type of the attribute
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)
{
  System.out.println("      Attribute Value is collection");
  pCol = (dkCollection)a;
  pIter = pCol.createIterator();
  i = 0;
  while (pIter.more() == true)
  {
    i++;
    a = pIter.next();
    pDO = (dkDataObjectBase)a;

// continued...
```

Java (continued)

```
if (pDO.protocol() == DK_XDO)
{
  System.out.println("     dkXDO object " + i + " in collection");
  pXDO = (dkXDO)pDO;
  DKPidXDO pid2 = pXDO.getPidObject();
  System.out.println("           XDO pid string: " +
                                   pid2.pidString());
          //----- Retrieve and open instance handler for an XDO
          pXDO.retrieve();
          // pXDO.open();
       }
     }
   }
   else if (a != null)
   {
     System.out.println("       Attribute Value: " + a.toString());
     if (strDataName.equals("DKResource")  ||
         strDataName.equals("DKFixedView") ||
         strDataName.equals("DKLargeObject"))
     {
       pDO = (dkDataObjectBase)a;

       if (pDO.protocol() == DK_XDO)
       {
         System.out.println("         dkXDO object ");
         pXDO = (dkXDO)pDO;
         DKPidXDO pid2 = pXDO.getPidObject();
         System.out.println("           XDO pid string: " +
                                         pid2.pidString());
         // Retrieve and open instance handler for an XDO
            pXDO.retrieve();
         // pXDO.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 . . .
   {
      // ----- Handle each of the other types
   }
   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 = (DKPidXDOOD*)pXDO->getPid();
         cout << "    XDO PID string: " << pidXDO->pidString() << endl;
         // ----- Retrieve and open instance handler for an XDO
         pXDO->retrieve();
       }
    }
  }
  else cout << " Attribute Value is NULL" << endl;
```

For the complete application, refer to TRetrieveOD.cpp in the CMBROOT\samples\cpp\od directory.

Enabling the OnDemand folder mode

To enable the OnDemand folder mode, the string

```
ENTITY_TYPE=TEMPLATES
```

must be passed to the OnDemand connector as part of the connection string or the configuration string. An sample configuration string would look like this:

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

A sample connect string would look like this:

Java

```
DKDatastoreOD dsOD = new DKDatastoreOD();
dsOD.connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

C++

```
DKDatastoreOD* dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
dsOD->connect(hostname, userid, password, "ENTITY_TYPE=TEMPLATES");
```

Asynchronous search

The OnDemand connector supports both federated and direct asynchronous searches. An asynchronous search does not tie up the main thread and allows the search to be cancelled at any time; press the **Stop Search** button on the Search Template Viewer to terminate the search. The max hits property on the Search Template Viewer limits the maximum number of results returned.

The OnDemand connector also supports synchronous and asynchronous searches in the application group mode from an AIX client.

If you use a search criterion such as `WHERE userid LIKE '%'`, the resulting number of documents returned to the client can consume all available memory on the client's machine. By issuing an asynchronous search using the executeWithCallback() method, you can set the value for the maximum number of documents returned and cancel the search at any time.

You may also have to increase the default Java Virtual Machine (JVM) stack size if your result set is too large. The default stack size for each Java thread is 400k, which allows 3920 return items before the stack overflows. Increasing the JVM stack size to 800k doubles the capacity to 7840 items. If necessary, you can further increase the JVM stack size.

To raise the JVM stack size, use the Java command line option `-oss` followed by `nnnK` or `nnM`, where `K` stands for Kilobytes and `M` for Megabytes.

For examples of using asynchronous search, see the TRetrieveWithCallbackOD, TRetrieveFolderWithCallbackOD and TCallbackOD sample programs.

OnDemand folders as search templates

Three of the EIP visual JavaBeans, CMBSearchTemplateList, CMBSearchTemplateViewer, and CMBSearchResultsView, use EIP search templates. You can use these beans for federated searches by setting the CMBConnection dsType to Fed.

You can use these beans for direct searches on OnDemand servers as well. Set the following properties before login:

```
connection.setDsType("OD");
connection.setServerName(<odserver>);
connection.setConnectString("ENTITY_TYPE=TEMPLATES");
```

OnDemand folders as native entities

The OnDemand connector can also map OnDemand folders as native entities by specifying the connect string "ENTITY_TYPE=TEMPLATES". Using OnDemand folders as entities can be useful in federated searches, where folder definitions may be easier to work with than OnDemand application groups, the default native entity for OnDemand.

For federated searches, specify the server definition in Enterprise Information Portal administration. You can then define and map federated entities to the folders on the OnDemand server.

Create and modify annotations

When using the OnDemand viewer, which is launched by the CMBDocumentViewer bean, you can create, modify, and delete annotations for OnDemand documents by using the CMBDataManagement bean and the associated CMBAnnotation class.

Tracing

You can trace events with the OnDemand connector. To enable the connector java API trace, place the trace INI file (cmbodtrace.ini for Java or cmbodCtrace.inifor C++), in the root of the C drive (C:\) or in the directory specified in the CMBROOT variable. For AIX, place this file /usr/lpp/cmb/cmgmt. For Solaris, place this file in /opt/IBMcmb/cmgmt.

The default output directory for trace files is C:\Ctrace. To write the trace information elsewhere, edit the trace INI file. For AIX, note that the file names must be all lower case.

Make sure that the path name specified in the trace file is valid and that the line containing CMBODTRACEDIR is not preceded with a # sign. Here are the sample trace INI files:

Java

```
#===============================================================
# This is a java property file - not a real INI file!
# ************************************************************************#
# For windows systems, make sure to use TWO BACK SLASH CHARACTERS (\\)
# to separate the directory names!!!    =========================
# ************************************************************************#
#
# ********** On windows systems, this file must be located in c:\ *********
#
# OD Trace File Directory Name property - CMBODTRACEDIR
#
# The CMBODTRACEDIR property defines the directory where the trace files
# will be written to.  If the directory name does not exist, it will be
# created.
#
# Please make sure that the directory names are separated by two back slash
# characters to avoid undesirable results.
#
# Please make sure the path name does not point to an existing file name.
# Otherwise, no trace files will be created.
#
# The trace output directory name can be changed to point to a drive
# where more space is available.  But it is recommended not to change the
# trace output directory name in the middle of an active trace session.
#
# CMBODTRACESCOPE controls how much trace information to generate.
#
# CMBODTRACESCOPE=ENTRY_EXIT_JNI_ONLY
# Trace the entry & exit points in JNI only. Produce the least amount
# of trace.
#
# CMBODTRACESCOPE=ENTRY_EXIT_ONLY
# Trace the entry and exit points in Java methods and JNI functions.
#
# CMBODTRACESCOPE=JNI_ONLY
# Full trace for the JNI functions only.
#
# If CMBODTRACESCOPE is missing, or set to anything else,
# a full trace will be taken.
#
# To disable the trace, add a leading # character in column 1.
#
# AIX: change the following line to CMBODTRACEDIR=/usr/lpp/cmb/cmgmt/trace
# Sun: change the following line to CMBODTRACEDIR=/opt/IBMcmb/cmgmt/trace
CMBODTRACEDIR=c:\\trace
```

C++

```
#===============================================================
# OnDemand Trace INI file
#
# OnDemand Trace File Directory Name key - CMBODTRACEDIR
#
# The CMBODTRACEDIR key defines the directory where the trace files will
# be written to.  If the directory name does not exist, it will be created.
#
# Please make sure the path name does not point to an existing file name.
# Otherwise, no trace files will be created.
#
# The trace output directory name can be changed to point to a drive
# where more space is available.  But it is recommended not to change
# the trace output directory name in the middle of an active trace
# session.
#
# CMBODTRACESCOPE controls how much trace information to generate.
#
# CMBODTRACESCOPE=ENTRY_EXIT_ONLY
# Trace only the entry and exit of all C++ methods and functions.
#
# If CMBODTRACESCOPE is missing, or set to anything else, a full trace
# is taken.
#
# To disable the trace, add a leading # character in column 1 on
# the CMBODTRACEDIR line.
#
[ODCTRACE]
# For AIX: change next line to CMBODTRACEDIR=/usr/lpp/cmb/cmgmt/ctrace
CMBODTRACEDIR=D:\Ctrace
#CMBODTRACESCOPE=ENTRY_EXIT_ONLY
```

Working with Content Manager ImagePlus for OS/390

Enterprise Information Portal APIs support the following features when working with Content Manager ImagePlus for OS/390 content servers:

- Connecting and disconnecting from one or more ImagePlus servers.
- Retrieving categories.
- Retrieving attribute fields.
- Retrieving folders.
- Retrieving documents.

You represent an ImagePlus for OS/390 content server using DKDatastoreIP in your application.

Restriction: ImagePlus for OS/390 does not support:

- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

Listing entities and attributes

After creating a content server for an ImagePlus for OS/390 content server as a DKDatastoreIP object and connecting, you can check the entities and attributes for the content server. The following example lists all of the entities for an ImagePlus for OS/390 content server:

Java

```
// ----- After creating a datastore and connecting
//          dsIP is a DKDatastoreIP object
DKEntityDefIP entDef = null;
DKAttrDefIP attrDef = null;

DKSequentialCollection pCol = (DKSequentialCollection)dsIP.listEntities();
dkIterator pIter = null;

if ( pCol == null )
{
   // ----- Handle if the collection of entities is null
}
else
{
... // ----- Process as appropriate
```

The complete sample application from which this example was taken (TListCatalogIP.java) is available in the CMBROOT\Samples\java\dl directory.

C++

```
// List entities...
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
{
...
```

The complete sample application from which this example was taken (TListCatalogIP.cpp) is available in the Cmbroot/Samples/cpp/ip directory.

The following example lists all of the attributes associated with each entity using the getAttr and listAttrNames functions of DKEntityDefIP.

Java

```
// ----- List attributes using listAttrNames and getAttr methods

pIter = pCol.createIterator();
while (pIter.more())
{
   // ----- Iterate over the each entity
  entDef = (DKEntityDefIP)pIter.next();
  System.out.println(" Entity type     : " + entDef.getType() );
  System.out.println(" Entity type name: " + entDef.getName() );

  // ----- Get a list of attributes for the entity
  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++

```
// Method 1:
cout << "List attributes using listAttrNames and getAttr functions" << 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;

// Upon return, tmpCount contains the number of elements in the list.
  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;
```

The following example shows an alternative way to list the attributes associated with each entity by using the `listEntityAttrs` method of DKDatastoreIP.

Java

```
//  --- List attributes using listEntityAttrs method
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 )
  {
     // ----- Handle if the collection of attributes is null
  }
  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++

```
//  Method 2:
cout << "---List attributes using listEntityAttrs function---" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
  docDef=(DKEntityDefIP*)(pIter->next()->value()); //iterator returns DKAny*
  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++;
      // ----- The iterator returns a pointer to 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;
}
delete pCol;
```

ImagePlus for OS/390 query syntax

The following example shows the query syntax for ImagePlus for OS/390:

Java

```
SEARCH = (COND=(search_expression), ENTITY={entity_name | mapped_entity_name}
        [, MAX_RESULTS = maximum_results]);
        [OPTION=([CONTENT={YES | ATTRONLY | NO};][PENDING={YES | NO};])]
```

C++

```
SEARCH=(COND=(search_expression),ENTITY={entity_name | mapped_entity_name}
      [,MAX_RESULTS=maximum_results]);
      [OPTION=([CONTENT={YES | ATTRONLY | NO};][PENDING={YES | NO};])]
```

The query uses the following parameters:

search_expression

Each search expression consists of one or more search criteria. You can only use the boolean operator AND between search criteria.

The search criteria has the form:

```
{attr_name | mapped_attr_name} operator literal
```

where:

attr_name

Name of the entity attribute on which to base the search.

mapped_attr_name

Attribute name mapped with the attribute on which to base the search.

operator

All attributes support equality (==). For attributes of type DATE, you can use the following additional operators:

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

literal

A literal. For numeric attributes, do not use quotation marks (″), for example:

```
FolderType == 9
```

For date, time, and timestamp attributes, quotation marks or apostrophes (′) are not necessary, but are tolerated, for example:

```
ReceiveDate == 1999-03-08
ReceiveDate == '1999-03-08'
```

For string attributes, quotation marks or apostrophes (′) are not necessary, but are tolerated. If the string contains an apostrophe (′), the string must be specified using two apostrophes, for example for a value of Folder'1:

```
FolderId == 'Folder''1'
```

entity_name

Name of the entity to be searched.

mapped_entity_name
Entity name mapped to the entity to be searched.

maximum_results
Maximum number of results to return.

The option keywords are:

CONTENT Controls the amount of information returned in the results

YES (default)
Sets the PIDs, attributes and their values for a document or folder. If there are parts in a document, the XDO PIDs are set. If there are documents in a folder, the document PIDs are set.

NO Only sets the document or folder PIDs.

ATTRONLY
Sets only the PIDs, attributes and their values for a document or folder.

PENDING Controls whether to include pending documents that do not have any parts. This option only applies when `ENTITY` is set to `DOCUMENT` or to an entity mapped to `DOCUMENT`.

YES Includes pending documents in the results.

NO (default)
Does not include pending documents in the results.

Working with Content Manager for AS/400

The API classes provided for Content Manager for AS/400 (VisualInfo for AS/400) are similar to those provided for Content Manager.

Restriction: Content Manager for AS/400 does not support:

- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

Listing entities (index classes) and attributes

You represent a Content Manager for AS/400 content server as a DKDatastoreV4. After creating the content server and connecting to it, you can list the entities (index classes) and attributes for the Content Manager for AS/400 server (refer to the example).

Java

```
// ----- After creating a datastore (dsV4) and connecting, get index classes
pCol = (DKSequentialCollection) dsV4.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
   i++;
   icDef = (DKIndexClassDefV4)pIter.next();
   strIndexClass = icDef.getName();
   ...  // ---- Process the index classes as appropriate
  // ----- Get the attributes
  pCol2 = (DKSequentialCollection) dsV4.listEntityAttrs(strIndexClass);
  pIter2 = pCol2.createIterator();
  j = 0;

   while (pIter2.more() == true)
   {
    j++;
    attrDef = (DKAttrDefV4)pIter2.next();
     ...  // ----- Process the attributes
    }
  }

dsV4.disconnect();
dsV4.destroy();
```

The complete sample application from which this example was taken (TListCatalogV4.java) is available in the CMBROOT\Samples\java\dl directory.

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;
```

The complete sample application from which this application was taken (`TListCatalogV4.cpp`) is available in the `Cmbroot/Samples/cpp/v4` directory.

Running a query

The following example runs a query in Content Manager for AS/400, and processes the results.

Java

```
// ----- After creating a datastore (dsV4) and connecting, build the
//       query and parameters and execute it
pCur = dsV4.execute(cmd,DK_CM_PARAMETRIC_QL_TYPE,parms);
...

if (pCur == null)
{
   // ---- Handle if the cursor is null
}

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 :
         {
            ... // Handle if the item is a document ");
            break;
         }
         case DK_CM_FOLDER :
         {
             ...  // Handle if the item is a folder
             break;
         }
     }
 }

 for (j = 1; j <= numDataItems; j++)
 {
    a = p.getData(j);
    strDataName = p.getDataName(j);
    ...  // Process the attributes as appropriate
    if (strDataName.equals(DKPARTS) == false
        && strDataName.equals(DKFOLDER) == false)
    {
    System.out.println("       attribute id: "
               + p.getDataPropertyByName(j,DK_CM_PROPERTY_ATTRIBUTE_ID));
   }
// continued...
```

Java (continued)

```
  if (a instanceof String)
  {
      System.out.println("      Attribute Value: " + a);
  }
  else if (a instanceof Integer)
       ...  // ---- Handle each type for attribute {
  else if (a instanceof dkCollection)
  {
     // ---- Handle if attribute value is a collection
     pCol = (dkCollection)a;
     pIter = pCol.createIterator();
     i = 0;
     while (pIter.more() == true)
     {
        i++;
        a = pIter.next();
        pDO = (dkDataObjectBase)a;

       if (pDO.protocol() == DK_CM_PDDO)
       {
          //  Process a DDO
          pDDO = (DKDDO)pDO;
          ...
       }
       else if (pDO.protocol() == DK_CM_XDO)
       {
          // Process an XDO
          pXDO = (dkXDO)pDO;
          DKPidXDO pid2 = pXDO.getPid();
          ...
       }
     }
   }
   else if (a != null)
   {
      //  Process the attribute
   }
   else ... // Handle if the attribute is null
   }
  }
}
pCur.destroy(); // Delete the cursor when you're done
```

The complete sample application from which this example was taken (TExecuteV4.java) is available in the CMBROOT\Samples\java\dl directory.

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);

       switch (a.typeCode())
       {
         case DKAny::tc_string :
         {
           strData = a;
           cout << "  attribute name: " << strDataName
                << ", value: " << strData << endl;
           break;
          }
// continued...
```

C++ (continued)

```
         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)
             {
               pDDO = (DKDDO*)pDOBase;
               cout << "  DKDDO object " << b << " in " << strDataName
                    << " collection " << endl;
               k = pDDO->propertyId(DK_CM_PROPERTY_ITEM_TYPE);

               if (k > 0)
               {
                 a = pDDO->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;
                }
              }
// continued...
```

C++ (continued)

```
             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;

    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;
```

The complete sample application from which this application was taken (`TExecuteV4.cpp`) is available in the `Cmbroot/Samples/cpp/v4` directory.

Running a parametric query

The following example runs a parametric query.

Java

```
// ----- Create the query string and the query object
String cmd = "SEARCH=(INDEX_CLASS=V4DEMO)";
pQry = dsV4.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Run the query
pQry.execute(parms);

System.out.println("number of query results = " + pQry.numberOfResults());

// ----- Processing the query results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
...
```

The complete sample application from which this example was taken (`TSamplePQryV4.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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;
```

The complete sample application from which this application was taken (`TSamplePQryV4.cpp`) is available in the `Cmbroot/Samples/cpp/v4` directory.

Working with Domino.Doc

Domino.Doc is the Lotus Domino solution for organizing, managing, and storing business documents, and making them accessible within and outside of a business.

Domino.Doc supports the open document management API (ODMA), so that you can create, save, and retrieve documents using ODMA-enabled applications. ODMA connects to a Domino.Doc server using an HTTP or Lotus Notes™ protocol.

Domino.Doc includes the following features:
- Connecting and disconnecting from one or more Domino.Doc servers.
- Ability to search binders.
- Ability to retrieve documents.
- ODMA compliance so that users can work in familiar applications.

Restriction: Domino.Doc does not support:

- Add, update, and delete document methods.
- Text Search Engine and QBIC search.
- Combined query.
- Workbasket and workflow.

When using the API to work with a Domino.Doc object, you must build an expression to return the objects. This section describes the design of the Domino.Doc API, how objects fit into the hierarchy, and how to build the expression. Figure 19 shows the relationship between the Domino.Doc object model and its components.

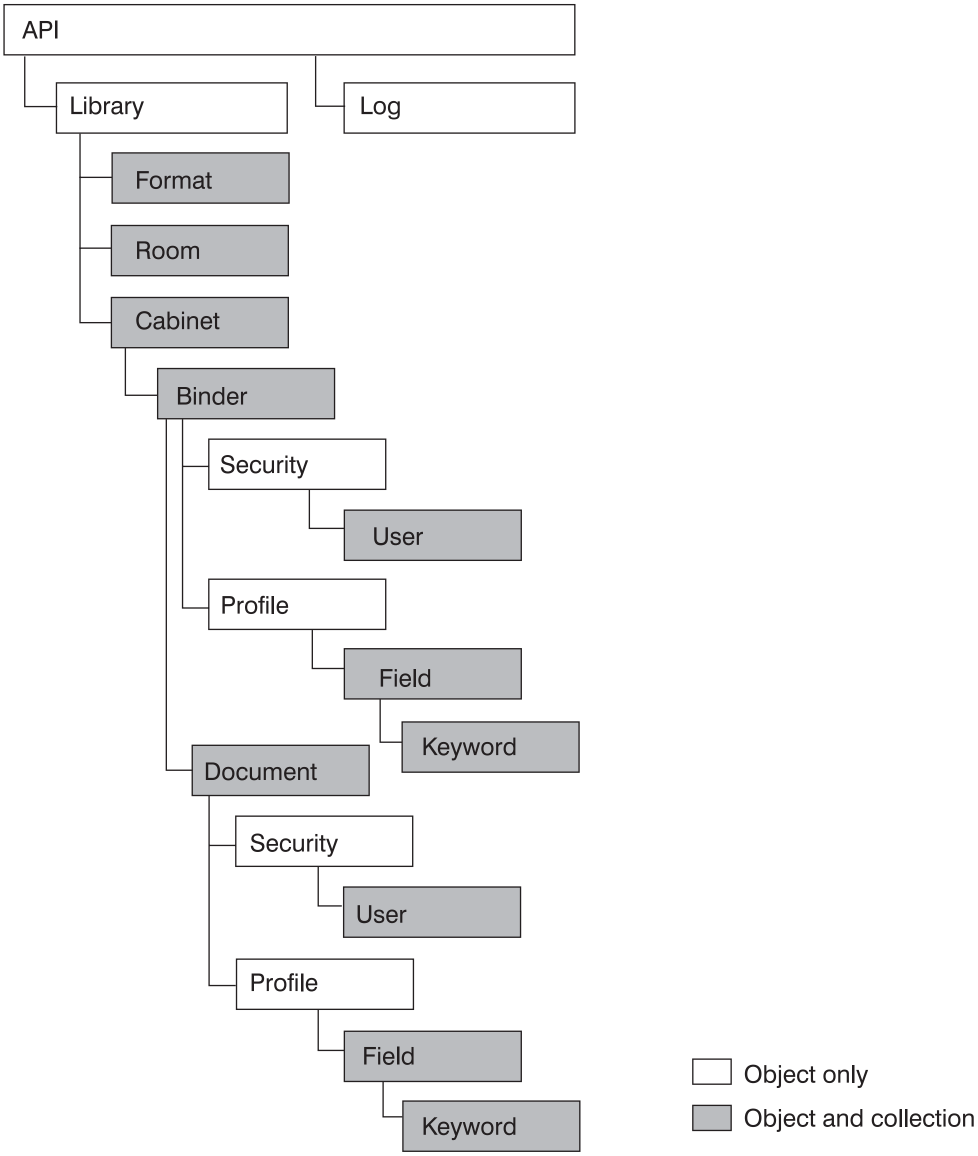


Figure 19. Domino.Doc object model

The elements contained in Domino.Doc (and the APIs to represent them) are arranged such that:

- The library contains rooms (DKRoomDefDD objects) and cabinets (DKCabinetDefDD objects).
- Each cabinet contains binders (DKBinderDefDD object).

- Each binder contains a profile (DKAttrProfileDefDD object) and security.
- Each binder contains documents (DKDocumentDefDD objects).
- Each document contains a profile (DKAttrProfileDefDD object) and security.
- Each profile contains fields (DKAttrFieldDefDD objects).
- Each field can contain keywords (DKAttrKeywordDefDD objects).

Listing entities and subentities

The following example lists the entities (in this example, rooms) in Domino.Doc and their subentities (in the example, cabinets, binders, and documents).

Java

```
...
// ----- Get a list of rooms
dkCollection rooms = dsDD.listEntities();
// ----- Iterate thru the rooms and their subEntities
dkIterator itRooms = rooms.createIterator();
itRooms.reset();
while( itRooms.more() ) {
   ...  // Process the rooms and entities
```

The complete sample application from which this example was taken (`TListSubEntitiesDD.java`) is available in the `CMBROOT\Samples\java\dd` directory.

C++

```
dkCollection* pColl = domDoc.listEntities();

long nbrEnts = pColl->cardinality();

dkIterator* itEnts = pColl-> createIterator();
while( itEnts->more() )
{ // For each returned dkEntityDef...
  DKRoomDefDD* pEnt = (DKRoomDefDD*)itEnts->next()->value();
  cout << "Room title: " << pEnt->getName() << endl;
  cout << "  Has SubEntities: " << pEnt->hasSubEntities() << endl;

  // print subEntities (Cabinets->Binders->Documents)
  printSubEnts(pEnt, domDoc, 1);

  delete pEnt;
}
delete itEnts;
delete pColl;
```

The complete application from which this example was taken (`TListEntitiesDD.cpp`) is available in the `Cmbroot/Samples/cpp/dd` directory.

The following example lists document attributes and keywords:

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();
      // ---- get the keywords
      dkCollection keywords = ((DKAttrFieldDefDD)aField).getKeywords();
      if( keywords != null )
      {
         if( keywords.cardinality() > 0 )
         {
           dkIterator itKeywords = keywords.createIterator();
           while( itKeywords.more() )
           {
               DKAttrDefDD aKeyword = (DKAttrDefDD)itKeywords.next();
               // ----- Process the keyword
...
```

The following example lists the subentities (cabinets, binders, and documents) associated with an entity (in this case a room).

C++

```
void printSubEnts( DKEntityDefDD* pEnt, DKDatastoreDD& domDoc, int indents )
{
  // indents: 1=Cabinets; 2=Binders; 3=Documents
  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;
}
```

Listing cabinet attributes

Cabinets are the only items that contain any useful attributes. If you try to list entity attributes for rooms, nothing will appear in the collection. Therefore, when DKDatastoreDD lists searchable entities, it only lists cabinets.

The following example lists cabinets and their attributes.

Java

```
...
dkCollection cabinets = dsDD.listSearchableEntities();
dkIterator itCabinets = cabinets.createIterator();
while( itCabinets.more() )
{
   // ----- For each cabinet, list it's attributes.
   dkEntityDef aCabinet = (dkEntityDef)itCabinets.next();
   cabinetName = aCabinet.getName();
   // ----- List Document Profiles without sub-attributes
   System.out.println("\n" + Me + ": calling listAttrs for" + cabinetName );
   DKSequentialCollection coll=(DKSequentialCollection) aCabinet.listAttrs();
...
```

The complete sample application from which this example was taken (`TListAttributes.java`) is available in the `CMBROOT\Samples\java\dd` directory.

Building queries in Domino.Doc

`ENTITY=` must be the first word in the query string if you want to limit the query to one cabinet. If the `ENTITY` parameter and its value are missing, then the entire library is searched. Also, the value must be enclosed in quotation marks ("). For example, `"Diane Cabinet"`.

`QUERY=` is a required parameter.

In Domino.Doc a query string looks like this:

```
"ENTITY=<"cabinetTitle"> QUERY=<"lotusQueryString">"
```

Use the `FTSearch` function to query the Domino.Doc content server. The Domino.Doc content server must be fully text indexed for this function to work efficiently. To test for an index, use the `IsFTIndexed` property. To create an index, use the `UpdateFTIndex` function.

The `FTSearch` function searches all of the documents in a content server—to search documents within a particular view, use the `FTSearch` function in NotesView. To search documents within a particular document collection, use the `FTSearch` function in NotesDocumentCollection.

If you do not specify a sort option, documents are sorted by relevance. If you want to sort by date, you do not get relevance scores with the sorted results. If you pass the resulting DocumentCollection to a NotesNewsletter instance, results are sorted by either the document creation date or the relevance score, depending on which sort options you use.

Using query syntax

The syntax rules for a query are in the following list. Use parentheses to override precedence and to group operations.

Plain text

Use plain text to search for a word or phrase as-is. Enclose search keywords and symbols in apostrophes ('). Remember to use quotation marks (") whenever you are inside a LotusScript literal.

Wildcards

Use the question mark (?) to match any single character in any position within a word. Use the asterisk (*) to match zero to *n* (where *n* is any number) characters in any position in a word.

Logical operators

Use logical operators to expand or restrict your search. The operators and their precedents are:

1. ! (not)
2. & (and)
3. , (accrue)
4. | (or)

You can use either the keyword or symbol.

Proximity operators

Use proximity operators to search for words that are close to each other. These operators require word, sentence, and paragraph breaks in a full-text index. The operators are:

- near
- sentence
- paragraph

Field operator

Use the field operator to restrict your search to a specified field. The syntax is `FIELD` ***`field-name operator`***, where *operator* is CONTAINS for text and rich text fields, and is one of the following symbols for number and date fields: =, >, >=, <, <=

Exactcase operator

Use the exactcase operator to restrict a search for the next expression to the specified case.

Termweight operator

Use the termweight *n* operator to adjust the relevance ranking of the expression that follows, where *n* is 0-100.

Working with Extended Search (ES)

Enterprise Information Portal supports IBM Extended Search 7.1, while deprecating support for ES 3. Extended Search allows you to query and retrieve documents from:

- Lotus Notes databases.
- NotesPump databases.
- File systems.
- Web search engines.

Enterprise Information Portal classes and APIs support the following Extended Search features:

- Connecting and disconnecting from one or more ES servers.
- Listing ES servers.

- Listing databases and fields.
- Using Generalized Query Language (GQL) to perform searches.
- Retrieving documents.
- Using the ES C++ classes and APIs on AIX.
- Performing parametric text searches from the federated layer and through a direct ES connection.
- Using CONTAINS_TEXT and CONTAINS_TEXT_IN_CONTENT text search operators from the federated layer.
- Using Enterprise Information Portal JavaBeans.

Restriction: ES does not support:

- Adding, updating, and deleting documents.
- Text Search Engine and QBIC search.
- Combined query.
- Workbasket and workflow.

All ES features are accessed and controlled by the ES configuration database. Use the configuration database to assign database definitions for data sources to be searched, network addresses, access control information, and other related information.

Listing Extended Search servers

In order to provide access to multiple ES servers, you can create a file named `cmbdes.ini` that contains the server information. Store this file in *C*:\CMBROOT (where *C* is the drive letter). The `cmbdes.ini` file must contain one line for each server, in the following format:

```
DATASOURCE=TCP/IP address;PORT=port number
```

where *TCP/IP* is the TCP/IP address of the ES server and the *port number* is the port number defined in order to access the server (for example: `PORT=80`).

Listing entities (databases) and attributes (fields)

When you build a query to search a ES server, you must know the available database and field names. The DKDatastoreDES object provides the `listEntities` function to list the databases and the `listEntityAttrs` function to list the fields for each database. The following example shows how to retrieve a list of databases and their fields.

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());
     }
     System.out.println("  " + j + " attributes listed for
                        " + strDatabase + " database");
    }
    System.out.println(i + " databases listed");
    dsDES.disconnect();
    System.out.println("datastore disconnected");
    }
// continued...
```

Java (continued)

```
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();
}
```

The complete sample application from which this example was taken (`TListCatalogDES.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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 << "NumValus:  " << 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 << "Atrr 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;
    delete pIter2;
    delete pCol2;
    delete pEnt;
 }
delete pIter;
delete pCol;
cout << i << " entities listed\n" << endl;
...
```

The complete sample application from which this example was taken (TListCatalogDES.cpp) is available in the Cmbroot/Samples/cpp/ES directory.

Using Generalized Query Language (GQL)

Extended Search uses the Generalized Query Language (GQL) to perform searches. Table 20 contains examples of valid GQL expressions.

Table 20. GQL expressions

GQL Expression	Description
"software"	Search for documents that contain the word software.
(TOKEN:WILD "exec*")	Search for documents that contain any word beginning with exec.
(AND "software" "IBM")	Search for documents that contain both words software and IBM.
(START "View" "How")	Search for documents in which the View field begins with the word How.
(EQ "View" "How Do I?")	Search for documents in which the View field contains the exact string How Do I?.
(GT "BIRTHDATE" "19330804")	Search for documents in which the BIRTHDATE field is greater than August 4, 1933.

ES uses the query type DK_DES_GQL_QL_TYPE. This query type has the following syntax:

```
SEARCH=(DATABASE=(db_name | db_name_list | ALL);
                   COND=(GQL expression));
[OPTION=([SEARCHABLE_FIELD=(fd_name, ...);]
                    [RETRIEVABLE_FIELD=(fd_name, ...);]
                    [MAX_RESULTS=maximum_results;]
                    [TIME_LIMIT=time)]
```

where db_name_list is a list of database names (db_name) separated by commas and ALL means search all of the available databases. The default time limit for a search is 30 seconds.

This example uses the query string to search for documents in the Notes Help database, where the View field is How Do I? and the maximum expected results are five.

```
String cmd = "SEARCH=(DATABASE=(Notes Help);" +
             "COND=(EQ \"View\" \"How Do I?\"));" +
             "OPTION=(MAX_RESULTS=5)"
```

This example runs a GQL query against ES. After the query is executed, the results are returned in a dkResultSetCursor object.

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\" \"How Do I?\"));"
             "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(); // Finished with the cursor
System.out.println("query results processed");
dsDES.disconnect();
System.out.println("datastore disconnected");
```

The complete sample application from which this example was taken (`TExecuteDES.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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;
...
```

The complete sample application from which this example was taken (`TExecuteDES.cpp`) is available in the `Cmbroot/Samples/cpp/ES` directory.

Identifying the DDO item type in Extended Search

A DDO in ES always has the type `DK_CM_DOCUMENT`. To get the item type of the DDO, you call:

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
 }
```

Creating PIDs in Extended Search

The persistent identifier (PID) contains specific information about a document. The object type identifies the database where the document was found. A PID is created by using the database name, followed by the | character and the document ID, for example:

```
database name|documentId ()
```

For more information on PIDs see “Understanding persistent identifiers (PID)” on page 14 and “Creating a persistent identifier (PID)” on page 41.

Processing the contents of an Extended Search document

Each item in the DDO represents either a field, a collection, or a DKParts object.

Field
The field name for a single field is inside the item name. The field value is also inside the item value. The field property can be:

- DK_CM_VSTRING
- DK_CM_FLOAT
- DK_CM_XDOOBJECT
- DK_CM_DATE
- DK_CM_SHORT

Collection
When a field has multiple values, the field name is in the item name. The item value is a DKSequentialCollection object. The property can be a DK_CM_COLLECTION, or a DK_CM_COLLECTION_XDO if the field is a BLOB.

DKParts
A document DDO has a specific attribute with a reserved name `DKPARTS`. Its value is a DKParts object. The DKPARTS object can store the Web address (URL) information about a document. DKPARTS can also contain an XDO with its contents as a string representing the URL of a document.

This example processes the contents of a 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 );
  // determine if data item has a single value or multiple values
  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();
      System.out.println("   Value" + i + " " + blob.getContent());
     }
   }

  else
  System.out.println("   Value " + anObject);
  }
}
```

The complete sample application from which this example was taken (TExecuteDES.java) is available in the CMBROOT\Samples\java\dl directory.

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;
DKPidXDODES *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;
  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);
// continued...
```

C++ (continued)

```
      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 = (DKPidXDODES*)pXDO->getPid();
cout << "XDO pid database name " << pidXDO->getDatabaseName() << endl;
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;
// continued...
```

C++ (continued)

```
          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 = (DKPidXDODES*)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;
```

The complete sample application from which this example was taken (TExecuteDES.cpp) is available in the Cmbroot/Samples/cpp/ES directory.

Retrieving a document

To retrieve a document from a DKDatastoreDES object, you must know the name of the database that contains the document and the document ID. You must also associate the DDO to a content server and establish a connection. This example retrieves a document:

Java

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dsDES.connect(libSrv, userid, pw, connect_string);
DKPid pid = new DKPid();
// ----- Primary ID is 'database name' followed by the
//      character '|' followed by the document ID
pid.setPrimaryId("Notes Help" + DK_DES_ITEMID_SEPARATOR + "215e");
pid.setObjectType("Notes Help");
DKDDO ddo = new DKDDO(dsDES, pid);
ddo.retrieve();
```

The complete sample application from which this example was taken (`TRetrieveDDODES.java`) is available in the `CMBROOT\Samples\java\dl` directory.

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();
...
```

The complete sample application from which this example was taken (`TRetrieveDDODES.cpp`) is available in the `Cmbroot/Samples/cpp/ES` directory.

Retrieving a binary large object (BLOB)

To retrieve a BLOB from a DKDatastoreDES object, you must know the name of the database where the document resides, the document ID that contains the BLOB, and field name that contains the BLOB. You must also associate the DDO to a content server and establish a connection.

In the following example, the file system database named `ES files` contains an HTML file named `D:\desdoc\README.html`. The field that contains the HTML file is named `Doc$Content`. The following example retrieves the HTML file and saves it as `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");
```

The complete sample application from which this example was taken (TRetrieveXDODES.java) is available in the CMBROOT\Samples\java\dl directory.

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;
...
```

The complete sample application from which this example was taken (TRetrieveXDODES.cpp) is available in the Cmbroot/Samples/cpp/ES directory.

Associating MIME types with documents

ES does not directly support the identification of Multipurpose Internet Mail Extension (MIME) types. However, you must know the MIME type of an XDO that you want to display within a Web browser.

The CMBCC2MIME.INI file is used to determine the MIME type of a document. When a ES query from NotesPump or FileSystem databases returns a BLOB, the CMBCC2MIME.INI file is searched to determine if a MIME type can be assigned to the BLOB. The default MIME type is text/html. A sample file named cmbcc2mime.ini.samp is available in the samples directory.

Using federated searching in Extended Search

When you create federated queries, the syntax used in ES is similar to SQL syntax. The federated query expressions are converted to GQL syntax before they are submitted to ES. Because SQL and GQL grammar have differences however, only a subset of the SQL grammar is supported by Enterprise Information Portal.

Table 21 summarizes the SQL to GQL conversion of the supported comparison and logical operators.

Table 21. SQL and GQL operators

SQL operator	GQL operator
AND	AND
OR	OR
NOT	not supported
IN	not supported
BETWEEN	BETWEEN
EQ	EQ
NEQ	not supported
GT	GT
LT	LT
LIKE	not supported
GEQ	GTE
LEQ	LTE
NOTLIKE	not supported
NOTIN	not supported
NOTBETWEEN	not supported

Working with Panagon Image Services (Java only)

You can use The Enterprise Information Portal API classes provided for Panagon Image Services (FileNET) to access content on Panagon Image Services servers. Support for Panagon Image Services is available only as a special feature of EIP.

Restriction: Panagon Image Services does not support Text Search Engine and QBIC search or combined queries.

Data modeling

Enterprise Information Portal has some data modeling concepts that need to be mapped to the corresponding objects in the content server that a connector supports. The table below shows the EIP data models in the FileNET environment.

EIP Concept	FileNET
content server	Panagon Image Services Server
server	server
entity	document class
attribute	index
DDO	document (folder) Folders in FileNET are used to organize documents temporarily; they are not currently modeled in the connector.

EIP Concept	FileNET
XDO	page content, annotation
PID for a DDO	document_id, as the primary ID
PID for page content XDO	System_serial_num + document_id + page#
PID for annotation XDO	System_serial_num + document_id + page# + annotation_id

FileNET connector only returns user-defined indexes or attributes in FileNET, which can be used in federated entity and attribute mappings and search result attribute display lists. The following table shows the mapping of FileNET supported index or attribute data types to the corresponding EIP attribute data types in the FileNET Connector:

FileNET index data type	EIP attribute data type
numeric	double
date	varchar with max_length set to the maximum length defined for the FileNET index
menu	varchar with max_length = 14 (maximum for menu names)

Every XDO in EIP has a MIME type. EIP clients use the MIME type to determine how to display the document. FileNET Connector assumes that all XDOs of the same document are of the same MIME type. The MIME type of the XDO is set according to the F_DOCFORMAT FileNET system attribute value of the belonging document. If the value is null, the MIME type is then set to `image\tiff` if the F_DOCTYPE FileNET system attribute indicates the image type; otherwise, it is set to the default MIME type of `application/octet-stream`.

Documents and pages in Panagon Image Services

To use a DDO to represent a document in Panagon Image Services, four fields in the DDO's PID need to be set properly:

1. DatastoreType – this is the constant DK_FN_DSTYPE defined in DKConstantFN. You can get this value from the datastoreType() method of the DKDatastoreFN.
2. DatastoreName – this is the FileNET domain-organization name. You can get this value from the datastoreName() method of the DKDatastoreFN.
3. ObjectType – this is the FileNET document class (native entity name in EIP) name to which the document belongs.
4. PrimaryID – this is the FileNET document number, assigned by FileNET.

Once the PID is set, application programs can get attributes or contents of the DDO by calling the retrieve() method either in DKDatastoreFN or DKDDO class.

A document DDO in EIP has a special attribute with a reserved name DKPARTS, whose value is a DKParts object. The DKParts object is a collection of binary large objects (BLOBs). The FileNET parts (pages/annotations) within a document are represented as DKBlobFN objects (a subclass of DKXDO) which are elements in the DKParts collection.

A FileNET page is not a conventional page; a single page (or single part) document in FileNET may contain multiple physical pages but they are stored in a single page within a FileNET document. A multiple-page document in FileNet refers to a document that contains either multiple parts or multiple files that are created using the Panagon Capture Software or the Batch Entry System.

Listing entities and attributes

You represent a Panagon Image Services server as a DKDatastoreFN. After creating the content server and connecting to it, you can list the entities (document x classes) and attributes for the Panagon Image Services server. The following example illustrates doing this:

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();
// ----- Connect to the server using the server name, user ID , and password
      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());
         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);
     // ----- Disconnect and clean up
      dsFN.disconnect();
      dsFN.destroy();
    }
    catch(DKException exc)
```

Refer to `CMBROOT\Samples\java\fn` directory for sample applications.

Querying

EIP supports three basic forms of query for Panagon Image Services:

- query object
- command string
- DKCQExpr

Both query object and command string forms of query are typically used in applications that are accessing the Panagon Image Services server directly. The EIP federated content server and query uses only the DKCQExpr form of the query to interface with individual content servers.

Sample applications for querying Panagon Image Services servers are available in the `CMBROOT\Samples\java\fn` directory.

Query String Syntax and Parameters

FileNET Connector supports the use of DKParametricQuery to submit and execute a query. The DKParametricQuery can be constructed with a command string. The command string represents the query specification for the FileNET filter condition. Other query options (ENTITY_NAME, MAX_RESULTS, CONTENT) and FileNET key condition (KEY) will be accepted as parameters to the evaluate() or execute() methods in (name, value) pairs (e.g., DKNVPair).

Note that the values are always of String type and the parameter names are defined in DKConstant and DKConstantFN. The return results are a collection of document DDOs for evaluate() and the result set cursor pointing to the results for execute().

The conditional expression syntax for the command and the KEY parameter follow the FileNET query syntax supported by FileNET WAL PRS_Parse() function. It accepts the following operators:

Operator	Use
`AND`	logical and
`OR`	logical or
`NOT`	logical not
`<`	less than
`<=`	less than or equal to
`=`	equal to
`>`	greater than
`>=`	greater than or equal to
`!=`	not equal to
`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`IN RANGE`	a range is specified with the syntax`IN RANGE` *`op constant  op constant`*, where *op* is `<`, `<=`, `>`, or `>=`, and *constant* is a numeric or string constant

Operator	Use
`LIKE`	Similar to. The pattern on the right hand side of the `LIKE` operator must be a single quoted character string that can contain ? as a wild character (to indicate match any character), and * as a wild card.
`DEFINED(`*x*`)`	The defined function returns non-zero if the given column name *x* is non-null, and zero if the value is null.

The operands are numeric constants, string constants and attribute names. Since command string is used for searching FileNET content server directly, the attribute names are all local FileNET attribute/index names. No federated attribute names and no schema mapping would be supported nor required

Numbers are in the format *+ddd.dddE+eee* where *ddd* stands for 0 or more numeric digits, + stands for + or - (+ is optional), *eee* stands for 0 to three digits of the exponent, and . is a decimal point (optional), and E is the beginning of the exponent (optional).

String constants are surrounded by single quotes, and an embedded single quote within a string is represented by two consecutive single quotes.

The conditional expression for the command string may contain any number of operands and operators. Parenthesis may be also used in the filter to override precedence. Examples of filter conditions are as follows:

```
PROD_PRICE > 100 AND (PROD_DATE > '10/15/92' OR PROD_NAME LIKE 'comp*')
PROD_PRICE >= 100 AND (NOT (PROD_ID < 30))
```

The conditional expression for the KEY parameter must contain just one attribute name, which is defined as a retrieval key in FileNET, and one condition. Note that key names defined in FileNET must be used, and a non-inverted column name may not be used. Examples of key strings are:

```
PROD_PRICE = 100000
PROD_PRICE IN RANGE >= 100000 <= 200000
```

Additional search options

The other supported optional parameters for evaluate() and execute() are:

(ENTITY_NAME, entity_Name):

Specify the entity name (FileNET document class name) as the scope of query. Sample:

```
parms[1] = new DKNVPair (DK_FN_PARM_ENTITY_NAME, entity_Name);
```

The actual query string submitted to the Panagon Image Services server is appended with AND (F_DOCCLASSNUMBER = doc_cls#), where = doc_cls# is the matching document class number of the specified entity name. If not specified, documents in all FileNET document classes are searched.

(MAX_RESULTS, Max_results)

Only the specified maximum number of results are returned in the result collection. If the value is zero or not specified, the maximum number of results allowed by FileNET is returned. Sample:

```
parms[0] = new DKNVPair(DK_CM_PARM_MAX_RESULTS, Max_results);
```

(CONTENT, YES / NO / ATTRONLY)

This parameter will overwrite the DK_CM_OPT_CONTENT option set at the content server level.

- `YES` - the resulting documents will have their contents retrieved (default)
- `NO` - only the document IDs are set in the result DDOs
- `ATTRONLY` - the document IDs, the data attributes and their values are set in the result DDOs

Sample:

```
dsFN.setOption(DK_CM_OPT_CONTENT, DK_CM_CONTENT_YES);
```

For example, content value `YES` would cause the resulting document DDOs to have their PID, object type, properties and all attributes set. This option also causes the DKPARTS attribute to be set to a collection of content parts (XDOs for parts and their annotations) with PID information set but no actual content. If the CONTENT value is `ATTRONLY`, the resulting document DDOs PID, object type, properties, and all data attributes are set. If the CONTENT value is `NO`, the resulting document DDOs PID, object type, and properties are set. The document part or XDO content can be retrieved explicitly when needed using the retrieve() method of the DKBlobFN class.

Exception handling and messages

The Panagon Image Services connector employs the same set of Java exception classes defined in EIP. The text message included in an exception also uses one of the commonly defined messages in DKMessageID whenever is possible. For example, the DKUsageError exception includes a message as (DKMessage.getMessage(DK_CM_MSG_NOTIMP) + this.getClass().getName() + *XXXX* , DK_CM_MSG_NOTIMP); where *XXX* is the method name that the program is attempting to invoke.

DKDatastoreAccessError exception will include the FileNET specific message text in the following format:

```
WAL_function_name [IMS_SESSION= sss, ERR_CAT=ccc, ERR_FUN=fff, ERR_NUM=nnn]
```

You can use the FileNET command `msg` (accessible via the `c:\fnsw\client\bin` directory) to further display the native FileNET error message text using the three numbers *ccc*, *fff* and *nnn* (e.g., `msg` *`ccc,fff,nnn`*.

The EIP message IDs used uniquely for the Panagon Image Services servers are from 3401 to 3600.

Working with relational databases

The Enterprise Information Portal API classes support IBM DB2 Universal Database, and other relational databases using Java Database Connectivity (JDBC), Open Database Connectivity (ODBC) for C++, or IBM DB2 DataJoiner.

Connecting to relational databases

To represent a relational database, create a DKDatastore*xx* object, where the *xx* is DB2 for a DB2 database, JDBC for Java Database Connectivity, or ODBC for Open Database Connectivity. The following sample connects to the DB2 sample database:

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). . .
```

Use the database name when connecting.

Connection strings

When connecting to a relational database, you can specify a connection string and pass it as a parameter. If you specify multiple connection strings, separate them with a semi-colon (;). Connection strings can take the following forms:

Connection strings for DB2, DataJoiner, and ODBC:

`NATIVECONNECTSTRING=(`*`native connect string`*`)`

Specifies a native connect string to be passed to the database when connecting. Check the information for your content server to determine the valid native connections strings.

`SCHEMA=`*`schema name`*

Specifies the database schema name to be used when running the listEntities, listEntityAttrs, listPrimaryKeyNames, listForeignKeyNames methods.

Connection strings for JDBC:

`SCHEMA=`*`schema name`*

Specifies the database schema name to be used when running the listEntities, listEntityAttrs, listPrimaryKeyNames, listForeignKeyNames methods.

Configuration strings

You can specify a configuration string and pass it as a parameter to the configuration method of. If you specify multiple configuration strings, separate them with a semi-colon (;). Configuration strings have the following forms:

Configuration strings for DB2, DataJoiner, and ODBC:

`CC2MIMEURL=(`*`URL`*`)`

Specifies the `cmbcc2mime.ini` file as a uniform resource locator address. Use this form of the configuration string or CC2MIMEFILE, depending on the location of the file.

`CC2MIMEFILE=(`*`filename`*`)`

Specifies the `cmbcc2mime.ini` file by name.

`DSNAME=(`*`content server name`*`)`

Specifies the name of the content server. For federated queries and other federated functions, Enterprise Information Portal sets this automatically.

`AUTOCOMMIT=ON | OFF`

Sets autocommit on or off. Default is off. When this content server is used for federated queries and other federated functions, autocommit is on by default.

Configurations strings for JDBC:

`CC2MIMEURL=(`*`URL`*`)`

Specifies the `cmbcc2mime.ini` file as a uniform resource locator address. Use this form of the configuration string or CC2MIMEFILE, depending on the location of the file.

Specifies the `cmbcc2mime.ini` file by name.

`JDBCSERVERSURL=(`*`URL`*`)`

Specifies the `cmbjdbcsrvs.ini` file in a uniform resource locator address. This file contains the list of JDBC servers.

`JDBCSERVERSFILE=(`*`filename`*`)`

Specifies the `cmbjdbcsrvs.ini` file that contains the list of JDBC servers as a filename.

`JDBCDRIVER=(`*`JDBC driver`*`)`

Specifies the JDBC driver that you want to use. This is automatically set when you use the system administration client client program.

`DSNAME=(`*`content server name`*`)`

Specifies the name of the content server. For federated queries and other federated functions, Enterprise Information Portal sets this automatically.

`AUTOCOMMIT=ON | OFF`

Sets autocommit on or off. Default is off. When this content server is used for federated queries and other federated functions, autocommit is on by default.

Listing entities and entity attributes

After creating the content server for the relational database and connecting to it, you can list the entity and entity attributes. The following example shows how to retrieve list and step through it:

Java

```
// ----- After creating a datastore and connecting, get index classes
pCol = (DKSequentialCollection)dsDB2.listDataSources();
pIter = pCol.createIterator();
while (pIter.more() == true)
{
   i++;
   pSV = (DKServerDefDB2)pIter.next();
   strServerName = pSV.getName();
   .... // Use the server name as appropriate
 }
 // ----- Connect to datastore
 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 + "]");
 }
 // ----- List the tables
 pCol = (DKSequentialCollection) dsDB2.listEntities();
 pIter = pCol.createIterator();
 i = 0;
 while (pIter.more() == true)
 {
    i++;
    tableDef = (DKTableDefDB2)pIter.next();
    strTable = tableDef.getName();
    // ----- List attributes (columns for the table)
    pCol2 = (DKSequentialCollection) dsDB2.listEntityAttrs(strTable);
    pIter2 = pCol2.createIterator();
    j = 0;
    while (pIter2.more() == true)
    {
       j++;
       colDef = (DKColumnDefDB2)pIter2.next();
        .... // Process the information as appropriate
     }
 }
 // ----- Commit and disconnect
 dsDB2.commit();
 dsDB2.disconnect();
 dsDB2.destroy();
```

Refer to TListCatalogDB2.java, TListCatalogJDBC.java, and TListCatalogDJ.java in the CMBROOT\Samples\java\dl directory for complete examples.

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;

   // ----- Connect to datastore and set value for schema name
      . . .
   // ----- Create a datastore definition and set the schema name
   dsDefDB2 = (DKDatastoreDefDB2 *) dsDB2.datastoreDef();
   if (schema != "")
   {
     dsDefDB2->setSchemaName(schema);
   }

   // ----- Get a list of the entities (tables)
   pCol = (DKSequentialCollection*)((dkCollection*)dsDB2.listEntities());
   pIter = pCol->createIterator();
   i = 0;
   // ----- List the attributes (columns) for each entity (table)
   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;
       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;
     }
// continued...
```

C++ (continued)

```
    delete pIter2;
    delete pCol2;
    delete pEnt;
  }
  delete pIter;
  delete pCol;
  dsDB2.disconnect();
}
catch(DKException &exc)
  . . .
```

Refer to `TListCatalogDB2.cpp`, `TListCatalogODBC.cpp`, and `TListCatalogDJ.cpp` in the `CMBROOT\Samples\cpp\db2`, `odbc`, and `dj` directories for complete examples.

Running a query

To run a query, you must first create the query string and then execute the query. The following example runs a query and processes the results.

Java

```
 // ----- After creating a datastore and connecting, build the
 //       query and parameters and execute it
 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);
 // ----- Connect to datastore
 dsDB2.connect(database,userid,pw,"");
 // --- Create the query string
 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;

// continued...
```

Java (continued)

```
// ----- Execute the  query
pCur = dsDB2.execute(cmd,DK_CM_SQL_QL_TYPE,parms);
if (pCur == null)
{
    // Handle if the cursor is null
}
while (pCur.isValid())
{
    p = pCur.fetchNext();
    if (p != null)
    {
       cnt++;
       i = pCur.getPosition();
       //  Get item information
       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++)
       {
         a = p.getData(j);
         strDataName = p.getDataName(j);
         //  Handle the attributes ;
      if (a instanceof String)
      {
      System.out.println("   Attribute Value " + a);
      }
      .......  //  Handle for various types )
      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)
     {
        // dkXDO object
           pXDO = (dkXDO)pDO;
           pidXDO = pXDO.getPidObject();
      }
      }
      .......  //  Handle for various types
      {
    }
 }
 // Delete the cursor when you're done, commit and disconnect
 pCur.destroy(); // Finished with the cursor
 dsDB2.commit();
 dsDB2.disconnect();
 dsDB2.destroy();
```

Refer to `TExecuteDB2.java`,`TExecuteJDBC.java`, and `TExecuteDJ.java` in the `CMBROOT\Samples\java\dl` directory for complete examples.

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);
  // ---- Create a datastore and connect
   . . .
  // ----  Create a query string containing the select
  DKString qstrng = "SELECT * FROM EMPLOYEE";
  // ---- Execute the query
  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);
        val = a;
                  switch (val)
        {
          case DK_CM_DOCUMENT :
          {
            cout << "Item is document " << endl;
            break;
     }
        }
                }
      cout << "Number of Data Items " << numDataItems << endl;
```

C++ (continued)

```
       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;
      }
           // ---- Handle each type in a similar fashion
           . . .
         }
    }
    // ----- Delete the cursor and disconnect
    if (pCur != 0)
           delete pCur;
    dsDB2.disconnect();
}
catch(DKException &exc)
   . . .
```

Refer to TExecuteDB2.cpp,TExecuteODBC.cpp, and TExecuteDJ.cpp in the CMBROOT\Samples\cpp directories for complete examples.

Creating custom content server connectors

You can create your own server definitions for custom content servers (that are not currently included in Enterprise Information Portal). If you integrate a custom server into EIP, then you must provide your own Java or C++ classes to support the definition.

Developing custom content server connectors

The object-oriented API framework is designed with the following objectives:

- Additional data storage systems, or content servers, can be added into the framework.
- Ability to map to any complex content server data type.
- A common object model for all content server data access.
- A flexible mechanism to use a combination of different types of search engines, such as Text Search Engine, image search (QBIC), and so forth.
- Client/server implementation for Java application users.

For information on specific object-oriented APIs see the online API reference.

If you are integrating a custom content server into Enterprise Information Portal you must:

- Import the `com.ibm.mm.sdk.common` package.
- **Java only:** Link to the `cmbcm81.jar` (Java) file in order to access the common framework.
- **C++ only:** Link to the `cmbcm817.dll`, non-debugged version, and `cmbcm8167.dll`, debugged version, files in order to access the common framework.

Enterprise Information Portal database infrastructure

The dkDatastore classes serve as the primary interface between Enterprise Information Portal and the content servers. Each content server has a separate class that implements the dkDatastore class to provide implementation information for a specific content server. Each content server type is represented by a class called DKDatastore*xx*, where *xx* identifies the name or type of the specific content server. Table 7 on page 34 lists the current content servers provided in Enterprise Information Portal.

You must specify the DKDatastore class you create for your content server in the EIP system administration client when you create your server definition.

Common classes in Enterprise Information Portal

dkDDO

The dkDDO class provides a representation and a protocol to define and access an object's data, independent of the object's type. The DDO protocol is implemented as a set of functions to define, add, and access each data item of an object. You can use this protocol to dynamically create an object and get it from the content server regardless of the content server's type.

When implementing a content server, you can utilize schema mapping information, registered in the content server class. The schema maps each individual persistent data item to its underlying representation in the content server.

A DDO has a set of attributes; each attribute has a type, value, and properties associated with it. The DDO itself can have properties that belong to the DDO as a whole. For example, you can map which class to an item in Content Manager content server, or a document in OnDemand.

Java

This diagram represents the hierarchy for the dkDDO class:

```
java.lang.Object
   |
   +----com.ibm.mm.sdk.common.dkDataObjectBase
           |
           +----com.ibm.mm.sdk.common.dkDataObject
                   |
                   +----com.ibm.mm.sdk.server.DKDDOBase
                           |
                           +----com.ibm.mm.sdk.server.DKDDO
```

dkXDO

The dkXDO class represents complex user-defined types or large objects (LOBs) which can exist stand-alone or as a part of DDO. Therefore, it has a persistent identifier (PID) and create, retrieve, update, and delete functions.

The dkXDO class extends the public interface of dkXDOBase by defining independent content server access, create, retrieve, update, and delete functions. These functions enable an application to store and retrieve the object's data to and from a content server without the existence of an associated DDO class object or stand-alone XDO.

You must set the PID for a stand-alone XDO to locate its position in the content server. If you are using the XDO with a DDO, the PID is set automatically. For example you can map which class to an item for the Content Manager content servers, and mapped to notes for the OnDemand content servers.

Java

Here is the class hierarchy for the dkXDO class:

```
java.lang.Object
   |
   +----com.ibm.mm.sdk.server.dkDataObjectBase
           |
           +----com.ibm.mm.sdk.server.dkXDOBase
                   |
                   +----com.ibm.mm.sdk.server.dkXDO
```

dkCollection

The dkCollection class is a collection of objects. dkCollection cannot evaluate a query. A collection might have a name (the default name is an empty string). For example, DKParts is a subclass of DKSequentialCollection, which is in turn a subclass of dkCollection.

DKResults

DKResults is a subclass of dkQueryableCollection, therefore it supports sorting and bi-directional iterators, and it is queryable. The element members of a DKResults class are objects, instances of the dkDDO class that represent query results. The iterator created by this class is dkSequentialIterator.

Java

Here is the class hierarchy for the DKResults class:

```
java.lang.Object
   |
   +----com.ibm.mm.sdk.server.DKSequentialCollection
           |
           +----com.ibm.mm.sdk.server.dkQueryableCollection
                   |
                   +----com.ibm.mm.sdk.server.DKResults
```

dkQuery

dkQuery is an interface for a query object associated with a specific content server. Objects that implement this interface are created by content server classes. The result of a query is usually a DKResults object. Examples of a concrete implementation of the dkQuery interface are DKParametricQuery, DKTextQuery and DKImageQuery, which are created by their associated content servers.

dkCQExpr

The dkCQExpr class represents a compound or combined query expression. It can contain a dkQExpr query expressions tree, which can contain a combination of parametric, text, and image query. If you want each content server to allow a federated search, the content server must be able to process this dkCQExpr object.

dkSchemaMapping

dkSchemaMapping is the an interface that defines an associative mapping between a federated entity and a native entity in content server. The content server must understand this mapping class to unmap and remap federated entities and attributes to native entities and attributes for a query and return results.

dkDatastore and related classes

You must implement one concrete class for each of the following classes or interfaces for your content server. For example in an OnDemand server, the concrete class that implements the dkDatastore interface is DKDatastoreOD.

dkDatastore

dkDatastore represents and manages a connection to the content server, its transactions and commands. It supports the evaluate function, so it can be considered a subclass of the query manager.

The main methods in the dkDatastore interface are:

connect()
Connects to the content server.

disconnect()
Disconnects from the content server.

evaluate(), execute(), executeWithCallback()
Queries the content server.

commit(), rollback()
Performs transactions in the content server.

Restriction: Some content servers do not support these functions.

registerServices(), unregisterServices()
Registers search engines.

changePassword(userid, oldPasswd, newPasswd)
Changes the login password for the current logon user ID from the content server.

listDataSources()
Returns a collection of content server user ID objects to use for logon. You do not need to be connected to the content server to use this function.

listDataSourceNames()
Returns an array of content server names.

getExtension(String)
Gets the dkExtension object from the content server. If the given extension does not already exist but is supported by the content server, a newly created object is returned, otherwise, a null value is returned.

addExtension(String, dkExtension)
Adds a new extension object (XDO) to this content server.

createDDO(String,int)
Creates a data object based on the given object type and flag. Create DDO returns a new DKDDO object with all the properties and attributes set. The calling program must provide the attribute values for this data object.

The data object manipulation methods in the dkDatastore interface are:

addObject(dkDataObject)
Adds a new document or folder to the content server.

retrieveObject(dkDataObject)
Retrieves a document or folder from the content server.

deleteObject(dkDataObject)
Deletes a document or folder from the content server.

updateObject(dkDataObject)
Updates a document or folder in the content server.

moveObject(dkDataObject, String)
Moves a folder or document from one entity to another.

The schema mapping related methods in the dkDatastore interface are:

registerMapping(*DKNVPair***)**
Registers the mapping information to this content server.

unRegisterMapping(*String***)**
Removes the mapping information from this content server.

listMappingNames()
Returns an array of mapping names from this content server.

getMapping(*String***)**
Returns a dkSchemaMapping object.

dkDatastoreDef

The dkDatastoreDef interface defines functions to access content server information and to create, list, and delete its entities. It maintains a collection of dkEntityDef objects.

Table 22 contains examples of concrete classes for the dkDatastoreDef interface.

Table 22. Concrete classes for dkDatastoreDef

Server type	Class name
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
Earlier Content Manager	DKDatastoreDefDL

The main methods in the dkDatastoreDef interface are:

listEntities()
Lists entities.

listEntityAttrs()
Lists entity attributes.

addEntity()
Adds an entity.

getEntity(name)
Gets an entity.

Each concrete class can also have its own content server-specific functions with names that are familiar to that content server. For example, the DKDatastoreDefDL class contains these specific functions:

- listIndexClassNames()
- listIndexClasses()

The DKDatastoreDefOD class contains these specific functions:

- listAppGrps()
- listAppGrpNames()

dkEntityDef
The dkEntityDef class defines functions to:

- Access entity information.
- Create and delete attributes.
- Create and delete the entity.

In the dkEntityDef class, all functions that are related to subentities generate a DKUsageError indicating that the default content server does not support subentities. However, if the content server does support this kind of multiple level entity, as does Domino.Doc, for example, the subclass for this content server must implement the proper functions to overwrite the exceptions.

Table 23 contains examples of concrete classes for the dkEntityDef class.

Table 23. Concrete classes for dkEntityDef

Server type	Class name
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
Earlier Content Manager	DKIndexClassDefDL

The main functions in the dkEntityDef class are:

listAttrs()
Lists the entity attributes.

getAttr(*String attrName***)**
Gets a specified entity attribute.

addAttr(*DKAttrDef***)**
Adds an attribute to an entity.

getName()
Gets the name of the entity.

setName(*String*)
Sets the name of the entity.

hasSubEntities()
Determines whether the entity contains subentities.

getSubEntity(*String*)
Gets the subentity.

addSubEntity(*dkEntityDef*)
Adds a subentity to the entity.

listSubEntities()
Lists the subentities of the entity.

removeAttr(*String*)
Removes a subentity from the entity.

add() Adds the entity to the content server.

update()
Updates the entity in the content server.

retrieve()
Retrieves the entity values from the content server.

del() Deletes the entity from the content server.

dkAttrDef
The dkAttrDef class defines functions for accessing attribute information and creating and deleting attributes. Table 24 contains examples of concrete classes for the dkAttrDef class.

Table 24. Concrete classes for dkAttrDef

Server type	Class name
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
ODBC	DKColumnDefODBC
Earlier Content Manager	DKAttrDefDL

The main methods in the dkAttrDef class are:

listAttrs()
Lists the attributes.

getAttr(String *attrName*)
Gets a specified attribute.

getName()
Gets the name of the attribute.

getDescription()
Gets the description of the attribute.

add() Adds the entity to the content server.

dkServerDef
The dkServerDef class provides the server definition information for each content server. Table 25 contains examples of concrete classes for the dkServerDef class.

Table 25. Concrete classes for dkServerDef

Server type	Class name
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
earlier Content Manager	DKServerDefDL

The main functions in the dkServerDef class are:

setDatastore(dkDatastore *ds*)
Sets the reference to the content server object.

getDatastore()
Gets the reference to the content server object.

getName()
Gets the name of the content server.

setName(String *name*)
Sets the name of the content server.

datastoreType()
Gets the content server type.

dkResultSetCursor
dkResultSetCursor is a content server cursor in the query result set that you can use to manage a virtual collection of DDO objects. The collection is a query result set. Each element of the collection is not created until the content server retrieves the element.

The main functions in the dkResultSetCursor class are:

isScrollable()
Returns `TRUE` if the cursor can be scrolled forward and backward.

isUpdatable()
Returns `TRUE` if the cursor can be updated.

isValid()
Returns `TRUE` if the cursor is valid.

isBegin()
Returns `TRUE` if the cursor is positioned at the beginning of the result set.

isEnd()
Returns TRUE if the cursor is positioned at the end of the result set.

isInBetween()
Returns TRUE if cursor is positioned between data elements in the result set.

getPosition()
Gets the current position of the cursor.

setPosition(int position, *Object value***)**
Sets the cursor to the specified position.

setToNext()
Sets the cursor to point to the next element in the result set.

fetchObject()
Retrieves the current element from the result set and returns it as a DDO.

fetchNext()
Retrieves the next element from the result set and returns it as a DDO.

findObject(int position, String predicate)
Finds the data object that satisfies the specified predicate, moved the cursor to that position, and then retrieves the object.

addObject(DKDDO ddo)
Adds a new element of the same type, represented by the specific DDO, to the content server.

deleteObject()
Deletes the current element from the content server.

updateObject(DKDDO ddo)
Updates the current element at the current position in the content server, using the specified DDO.

newObject()
Creates an element of the same type and returns it as a DDO.

open() Opens the cursor, and if necessary, runs the query to create the result set.

close() Closes the cursor and the result set.

isOpen()
Returns TRUE if the cursor is open.

destroy()
Deletes the cursor; this allows for cleanup before the cursor is collected as garbage.

datastoreName()
Gets the name of the content server name to which the cursor belongs.

datastoreType()
Gets the content server type to which the cursor belongs.

handle(int type)
Gets the result set handle that is associated with the result set cursor, by type.

Requirement: In order to use the `addObject`, `deleteObject` and `updateObject` functions, you must set the content server option DK_DL_OPT_ACCESS_MODE to DK_READWRITE.

dkBlob

dkBlob is an abstract class that declares a common public interface for basic binary large object (BLOB) data types. The concrete classes derived from the dkBlob class share this common public interface which allows polymorphic processing of collections of BLOBs originating from heterogeneous content servers. There is also a dkClob and a dkDBClob class which can have concrete classes.

Table 26 contains examples of concrete classes for the dkBlob class.

Table 26. Concrete classes for dkBlob

Server type	Class name
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
Earlier Content Manager	DKBlobDL

The main methods in the dkBlob class are:

getContent()
Returns a byte array containing the BLOB data of the object.

getContentToClientFile(String *afileName, int fileOption***)**
Copies the BLOB data from the object to the specified file.

setContent(byte[] *aByteArr***)**
Sets the LOB data for the object with the contents of the byte array.

setContentFromClientFile(String *afileName***)**
Replaces the LOB data of the object with the contents of the file *afileName*.

add(String afileName)
Adds the content of the specified file to the content server.

retrieve(String afileName)
Retrieves the content of the content server into the specified file.

update(String afileName)
Updates the object and the content server with the content of the specified file

del(boolean flush)
Deletes the object's data from the content server, if *flush* is `TRUE`; otherwise the current content is preserved.

concatReplace(dkBlob aBlob), concatReplace(byte[] *aByteArr***)**
Concatenates this object with another dkBlob object or byte array.

length()
Returns the length of the LOB content of the object.

indexOf(String aString, int startPos), indexOf(dkBlob aBlob, int startPos)
Starting the search at offset start positions, returns the byte offset of the first occurrence of the search argument within this object.

subString(int startPos, int length)
Returns a string object that contains a substring of the LOB data of this object.

remove(int startPos, int aLength)
Starting at *startPos* for *aLength* bytes, deletes a portion of the LOB data of this object.

insert(String *aString***, int startPos), insert(dkBlob aBlob, int** *startPos***)**
Inserts the argument data, following the *startPos* position in the LOB data of the object.

open(String *afileName***)**
Unloads the object contents to the file *afileName* and then runs a default file handler.

setClassOpenHandler(String aHandler, boolean newSynchronousFlag)
Identifies, by executable program name, the file handler for an entire class. This function also indicates whether to run the handler synchronously or asynchronously for the file object.

setInstanceOpenHandler(String aHandler, boolean newSynchronousFlag)
Identifies, by executable program name, the file handler and indicates whether to run it synchronously or asynchronously for this object.

getOpenHandler()
Gets the executable program name of the file handler for an entire class.

isOpenSynchronous()
Returns the current synchronization setting for the file handler.

dkClob
dkClob is an abstract class that declares a public interface for storing character large object (CLOB) data types, such as documents.

Table 27 contains examples of concrete classes for the dkClob class.

Table 27. Concrete classes for dkClob

Server type	Class name
DB2 Universal Database	DKClobDB2
ODBC	DKClobODBC

The main functions in the dkClob class are:

open() Open() is a member inherited from dkXDOBase. Open() will be implemented or overridden by concrete subclasses of dkClob.

dkXDO Members: dkXDO& add(), dkXDO retrieve(), dkXDO update(), dkXDO del()

Inherited as protected members from dkXDO. Where necessary, these protected members will be implemented or overridden by concrete subclasses of dkClob.

The following list contains members defined by dkClob:

add(String *afileName***)**

Adds the content of the specified file to the content server.

retrieve(String *afileName***)**

Retrieves the content of the content server into the specified file.

update(String *afileName***)**

Updates the object and the content server with the content of the specified file.

del(DKBoolean flush)

Deletes the object's data from the content server, if *flush* is TRUE; otherwise the current content is preserved.

getContentToClientFile(String *afileName, int fileOption***)**

Copies the CLOB data from the object to the specified file.

setContentFromClientFile(String *afileName***)**

Replaces the LOB data of the object with the contents of the file *afileName*.

indexOf(String& *aString***, long startPos=1), indexOf(dkClob&** *adkClob***, long startpos=1)**

Starting the search at offset start positions, returns the byte offset of the first occurrence of the search argument within this object.

subString(long startpos, long length)

Returns a string object that contains a substring of the LOB data of this object.

remove(long startpos, long *aLength***)**

Starting at *startPos* for *aLength* bytes, deletes a portion of the LOB data of this object.

insert(DKString *aString***, long startpos), insert(dkClob&** *adkClob***, long startpos)**

Inserts the argument data following the *startPos* position in the CLOB data of the object.

open(String *afileName***)**

Unloads the object contents to the file *afileName* and then runs a default file handler.

setInstanceOpenHandler(String *ahandler***, DKBoolean newSynchronousFlag)**

Identifies, by executable program name, the file handler and indicates whether to run it synchronously or asynchronously for this object.

setClassOpenHandler(String *ahandler***, DKBoolean newSynchronousFlag)**

Identifies, by executable program name, the file handler for an entire class. This function also indicates whether to run the handler synchronously or asynchronously for the file object.

getOpenHandler()
Gets the executable program name of the file handler for an entire class.

isOpenSynchronous()
Returns the current synchronization setting for the file handler.

dkAnnotationExt
dkAnnotationExt is the interface class for all annotation objects. If your content server supports annotation data, you must implement this interface. This annotation object is an extension of your DKBlob*xx* class, where the dkBlob object is the representation of the binary annotation data and the DKParts collection.

dkDatastoreExt
The dkDatastoreExt class defines the standard content server extension classes.

Table 28 contains examples of concrete classes for the dkDatastoreExt class.

Table 28. Concrete classes for dkDatastoreExt

Server type	Class name
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
Earlier Content Manager	DKDatastoreExtDL

The main functions in the dkDatastoreExt class are:

getDatastore()
Gets the reference to the owning content server object.

setDatastore(dkDatastore ds)
Sets the reference to the owning content server object.

isSupported(String functionName)
Determines whether the specified function name is supported by this extension.

listFunctions()
Lists all supported function names for the extension.

addToFolder(dkDataObject folder, dkDataObject member)
Adds a member to this folder and to the content server.

removeFromFolder(dkDataObject folder, dkDataObject member)
Removes a member from this folder and the content server.

checkOut(dkDataObject item)
Checks out a document or folder item from the content server. While the item is checked out, you have exclusive updating privileges to the item and other users have read access only.

checkIn(dkDataObject item)
Checks in a document or folder item previously checked out from the content server. By checking in the file, you release all write privileges with this function.

getCommonPrivilege()
Gets the common privilege of a specific content server.

isCheckedOut(dkDataObject item)
Determines whether a document or folder item was checked out from the content server.

checkedOutUserid(dkDataObject item)
Gets the user ID that checked out the item from the content server.

unlockCheckedOut(dkDataObject item)
Unlocks the item from the content server.

changePassword (String userId, String oldPwd, String newPwd)
Changes the password on the content server for the specified user ID.

moveObject (dkDataObject dataObj, String *entityName*)
Moves the *entityName* object from one entity to another.

retrieveFormOverlay(String id)
Retrieves the form overlay object.

DKPidXDO

The DKPidXDO class represents the persistent identification of the BLOB data in the content server.

Table 29 contains examples of concrete classes for the DKPidXDO class.

Table 29. Concrete classes for DKPidXDO

Server type	Class name
Earlier Content Manager	DKPidXDODL
OnDemand	DKPidXDOOD
Content Manager for AS/400	DKPidXDOV4
ImagePlus for OS/390	DKPidXDOIP
Domino.Doc	DKPidXDODD
Extended Search	DKPidXDODES
DB2 Universal Database	DKPidXDODB2
JDBC	DKPidXDOJDBC
ODBC	DKPidXDOODBC

dkUserManagement

The dkUserManagement class represents and processes all of the content server's user management functions.

Table 30 contains examples of concrete classes for the dkUserManagement class.

Table 30. Concrete classes for dkUserManagement

Server type	Class name
Content Manager	DKUserMgmtICM
Content Manager for AS/400	DKUserMgmtV4

Table 30. Concrete classes for dkUserManagement (continued)

Server type	Class name
ImagePlus for OS/390	DKUserMgmtIP
Earlier Content Manager	DKUserMgmtDL

DKConstant

All common constants are defined in the DKConstant class. Each content server has its own DKConstant*xx* class for defining constants specific to that content server.

Recommendation: All content servers use the common messages whenever possible.

DKMessageId

All common message IDs are defined in this class. Each content server has its own DKMessageId*xx* class for defining its own message IDs.

Recommendation: All content servers should use the common messages whenever possible.

These property files contain common warning and error messages:

For Java:

- `DKMessage_en.properties`
- `DKMessage_es.properties`

For C++:

- `DKMessage_en_US.properties`
- `DKMessage_es_ES.properties`

Each content server has its own DKMessage*xx_yy_zz*.properties files for its warning and error messages.

Using the FeServerDefBase class (Java only)

The FeServerDefBase class is the abstract class that you must extend in order to create a custom server definition. The Java class that extends this base class must have a constructor that accepts the following parameters and passes them to the super class:

String connectString

The connect string for the server.

String[] serverList

The list of defined servers.

String[] associatedServerList

The list of servers associated with this server (null if none).

String[] serverTypes

The list of defined server type IDs.

String[] serverTypeDescriptions

The list of descriptions for defined server types.

When you create the Java class that extends the FeServerDefBase class you must determine how to handle the data for the new server dialog. You can use the same class or a separate model class. If the custom content server requires more than fields for the connect string, you must use the Enterprise Information Portal database and Java APIs as a model in order for additional functions to perform properly.

When the content servers are selected in the Enterprise Information Portal Administration program, the `New` menu will contain the list of server types stored in the FASERVERTYPES table in the Enterprise Information Portal database. This table contains the name of the Java class to be instantiated when the menu item is selected.

If you support password verification, you must place your Java class in the same directory as the Enterprise Information Portal Administration .jar file, you can dynamically instantiate that Java class and invoke the `verify` method with the user input password as a parameter. The `verify` method will return null for a valid password or return an array of strings with the information for an invalid password.

Building EIP workflow applications

Using the EIP classes and APIs, you can create or extend your own applications to use the EIP workflow support. Typically, you perform a federated search, and start the workflow with the search result (a content item or a folder of multiple content items). You use the APIs to access a worklist and then to display the worklist contents. As each activity completes, the workflow moves to the next activity in the workflow.

Connecting to workflow services

To use Enterprise Information Portal workflow in your applications, start by creating an instance of DKWorkFlowServicesFed. Then connect to it. The following example starts workflow services:

Java

```
// ----- Create the strings for the name of the
//service, user ID and Password
String wfsrv = "icmnlsdb";
String userid = "icmadmin";
String pw = "password";
// ----- Create a federated datastore
DKDatastoreFed dsFed = new DKDatastoreFed();
dsFed.connect(wfsrv, userid, pw,"");
//----- Create the workflow service
DKWorkFlowServiceFed svWF =new DKWorkFlowServiceFed ();
// ----- Set the datastore in the workflow service
svWF.setDatastore(dsFed);
// ----- Connect to the service
svWF.connect (wfsrv, userid, pw,"");
```

C++

```
 // ----- Create the strings for the name of the service, user ID
// -----    and Password
DKString wfsrv = "icmnlsdb";
DKString userid = "icmadmin";
DKString pw = "password";
// ----- Create a federated datastore
DKDatastoreFed* dsFed = new DKDatastoreFed();
dsFed->connect(wfsrv, userid, pw,"");
//----- Create the workflow service
DKWorkFlowServiceFed* svWF =new DKWorkFlowServiceFed ();
// ----- Set the datastore in the workflow service
svWF->setDatastore(dsFed);
// ----- Connect to the service
svWF->connect (wfsrv, userid, pw,"");
```

When you are finished using the workflow service, you must disconnect by calling the disconnect() and the delete() functions.

Java

```
svWF.disconnect();
dsFed.disconnect();
svWF.destroy();
dsFed.destroy();
```

C++

```
svWF->disconnect();
dsFed->disconnect();
delete svWF;
delete dsFed;
```

Starting a workflow

After you create the workflow, you must start it. To start a workflow complete the following steps:

1. Create a DKWorkFlowFed object and set the workflow name.
2. Create a workflow instance using a valid workflow template, which is a workflow definition defined in the Enterprise Information Portal workflow builder.
3. Set the PID and priority in the container.
4. Start the workflow.

The following example uses these steps to start a workflow:

Java

```
  // ----- Create the DKWorkFlowFed object and set the name
  DKWorkFlowFed WF = new DKWorkFlowFed(svWF);
  WF.setName("wfl");
  // ----- Create an instance of a workflow with the workflow template name
  WF.add("WD1");
  // ----- Refresh the workflow object
  WF.retrieve();
  // ----- Construct the container object for the workflow
  DKWorkFlowContainerFed con = WF.inContainer();
  // ----- Retrieve the container data
  con.retrieve();
  // ----- Add a PID string referring to an Extended Search document
  con.setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
  con.setPriority(100);
  // ----- Update the container
  con.update();
  // ----- Start the workflow
  WF.start(con);
```

C++

```
// - Create the DKWorkFlowFed object and set the name
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF);
WF->setName("wfl");
//Create an instance of a workflow with the workflow template name
WF->add("WD1");
// ----- Refresh the workflow object
WF->retrieve();
// ----- Construct the container object for the workflow
DKWorkFlowContainerFed* con = WF.inContainer();
// ----- Retrieve the container data
con->retrieve();
// Add a PID string referring to a content item from Extended Search
con->setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
// ----- Assign a priority of 100
con->setPriority(100);
// ----- Update the container
con->update();
// ----- Start the workflow
WF->start(con);
. . .
// When you are done, clean up by deleting the container and workflow
delete con;
delete WF;
```

Terminating a workflow

You can terminate a workflow by calling the terminate() or del() function as shown in the following example:

Java

```
//-----Retrieve the status of the workflow named WF
WF.retrieve();
int state =WF.state();
//-----Check the status and either terminate or delete
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++

```
/---Construct a DKWorkFlowFed instance
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF, "Test");
//-----Retrieve the status of the workflow named WF
WF->retrieve();
int state = WF->state();
//---Check the status and either terminate or delete
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;
```

Listing all the workflows

You can list all the workflows in a workflow service by using the listWorkFlows() function. The following example lists the name and description of all the workflows in a workflow service referenced by the DKWorkFlowSerivceFed object `svWF`.

Java

```
// ----- Call the listWorkFlows method
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++

```
// ----- Call the listWorkFlows function
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;
```

Suspending a workflow

You can suspend a running workflow with either a specific time or indefinitely. The following example shows how to suspend a workflow until a certain time. If you provide a null DKTimestamp, then EIP suspends the workflow indefinitely.

Java

```
// ----- Construct a DKWorkFlowFed object
DKWorkFlowFed WF = new DKWorkFlowFed(svWF, "Test");
WF.retrieve();
// ----- Call the suspend method if the workflow is in the running state
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_RUNNING)
{
  // ----- Suspended until 2000-07-27-16.30.00.000000
  // -----   The timestamp uses the base year 1900; months are
  // -----   numbered 0 to 11
  DKTimestamp suspension = new DKTimestamp(100, 6, 27, 16, 30, 0, 0);
  WF.suspend(suspension);
}
```

C++

```
// ----- Construct a DKWorkFlowFed instance
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF, "Test");
WF->retrieve();
// ----- Call the suspend function if the workflow is in
the running state
if (WF->state() == DK_FED_FMC_PS_RUNNING)
{
  // ----- Suspended until 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;
```

Resuming a workflow

You can resume a suspended workflow by calling the resume() function. The following example resumes a suspended workflow.

Java

```
// ----- Construct a DKWorkFlowFed object
DKWorkFlowFed WF = new DKWorkFlowFed(svWF, "Test");
WF.retrieve();
// ---- Call resume() if the workflow is in the suspended state
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_SUSPENDED)
{
  WF.resume();
}
```

C++

```
// ----- Construct a DKWorkFlowFed instance
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF, "Test");
WF->retrieve();
// ----- Check whether the workflow is suspended and call resume
if (WF->state() == DK_FED_FMC_PS_SUSPENDED)
{
  WF->resume();
}
delete WF;
```

Listing all the worklists

You can list all the worklists in a workflow service by calling the listWorkLists() function on the workflow service. The following example lists the name and description of all the worklists in a workflow service referenced by the DKWorkFlowServiceFed instance svWF.

Java

```
// ----- Call the listWorkLists method
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++

```
// ----- Call the listWorkLists function
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 << "name = " << WL->getName() << " description = "
             << WL->getDescription() << endl;
      cout << "Threshold = " << WL->getThreshold() << endl;
      delete WL;
    }
    delete iterWL;
}
delete collWL;
```

Accessing a worklist

You can access a worklist by creating an instance of DKWorkListFed that refers to the worklist which you created using the system administration client. The following example accesses a worklist named WL0712 and displays the information contained in that worklist.

Java

```
  // ----- Create the DKWorkListFed
  DKWorkListFed WL = new DKWorkListFed(svWF, "WL0712");
  WL.retrieve();
  // ----- Display information about the worklist
  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++

```
// ----- Create the DKWorkListFed
DKWorkListFed* WL = new DKWorkListFed(svWF, "WL0712");
WL->retrieve();
// ----- Display information about the worklist
cout << "worklist name = " << WL->getName() << endl;
cout << "description = " << WL->getDescription() <<
        " owner = " << WL->getOwner() <<
        " filter = " << WL->getFilter() <<
        " threshold = " << WL->getThreshold() <<
        " sort criteria = " << WL->getSortCriteria() << endl;
// -----  Delete the worklist when you are done
delete WL;
```

Accessing work items

After you create the DKWorkListFed, you can retrieve the work items as a collection. The following example retrieves the work items.

Java

```
// ----- Create a collection and an iterator
DKSequentialCollection coll = (DKSequentialCollection)WL.listWorkItems();
dkIterator iter = (DKSequentialIterator) coll.createIterator ();
Object a;
DKWorkItemFed item;
String nodename;
String workflowname;

// ----- Step through the collections
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;
      // ----- Create a collection and an iterator
      coll = (DKSequentialCollection*)WL->listWorkItems();

      if (coll != NULL)
      {
         iter = coll->createIterator();
         cout << "listWorkItems" << endl;
         // ----- Step through the collections
         while (iter->more ())
         {
           item = (DKWorkItemFed*)((void*)(*iter->next()));

           if (item != NULL)
           {
              //item.retrieve ();
              nodename = item->name();
              workflowname = item->workFlowName();
              cout << "workitem node = " << nodename
                   << "  workflow name = " << workflowname << endl;
              delete item;
           }
         }
         delete iter;
         delete coll;
      }
```

Moving items in the workflow

As a workflow advances, you move work items from one activity to the next by using the checkOut() and checkIn() functions. The following example shows how to move the work items. Note that only the workflow user currently being assigned to perform the work item can check out and check in the work item.

Java

```
  DKWorkItemFed item =new DKWorkItemFed(svWF, "wf1", "node1", wfuser);
  item.retrieve();
  // ----- Call the checkOut method to lock the workitem
  item.checkOut();
  // ----- Call the checkIn method
  item.checkIn(null);
```

C++

```
DKWorkItemFed* item =new DKWorkItemFed(svWF, "wf1", "node1", wfuser);
item->retrieve();
// ----- Call the checkOut method to lock the workitem
item->checkOut();
// ----- Call the checkIn method
item->checkIn(NULL);
delete item;
```

Listing all the workflow templates

You can list all the workflow templates in a workflow service by calling the listWorkFlowTemplates() function. The following example lists the name and description of all the workflow templates in a workflow service referenced by the DKWorkFlowSerivceFed object svWF.

Java

```
// ----- Call the listWorkFlowTemplates method
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++

```
// ----- Call the listWorkFlowTemplates function
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 << "name = " << WT->name() << " description = "
                << WT->description() << endl;
       delete WT;
    }
    delete iterWT;
 }
 delete collWT;
```

Creating your own actions (Java only)

You can create your own actions that you can use in a workflow. You define the actions and add them to actions lists in Enterprise Information Portal Administration. You create actions using action objects (DKWorkFlowActionFed objects). An action object is a meta data container that records detailed instructions about how a particular task is intended to be executed at the client node. Action objects (meta data containers) only record instructions; they do not initiate the invocation of the tasks that are described in the action meta data.

Actions can be grouped into an action list (DKWorkFlowActionListFed). A workflow container carries the name of the action list (not the contents of the action list) in which a set of actions relating to the work item are associated. A client must retrieve the action list and then iterate through the entries (actions) in the list and react accordingly. The sample below shows you how to work with actions and action lists. The sample completes the following tasks:

1. Retrieves the work item.
2. Retrieves the container that is routed along with the work item.
3. Retrieves the action list from the container.
4. Gets the list of actions from the action list.
5. Starts the actions accordingly.

```
wit.retrieve(); // wit is a DKWorkItemFed object
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");
```

Building applications with non-visual and visual JavaBeans

This chapter describes the non-visual and visual JavaBeans provided in Enterprise Information Portal.

The EIP JavaBeans can be divided into the following categories:

Non-visual beans You can use the non-visual beans to build Java and Web client applications that require a customized user interface. The non-visual beans support the standard bean programming model by providing default constructors, properties, events and a serializable interface. You can use the non-visual beans in builder tools that support introspection.

Visual beans The visual beans are customizable, Swing-based, graphical user interface components. Use the visual beans to build Java applications for Windows. You can place them within windows and dialogs of Java-based applications. Because the visual beans are built using the non-visual beans (as a data model), you must use them in conjunction with the non-visual beans when building an application.

Java Viewer related beans The Java Viewer beans are a set of visual and non-visual components. You can use them to build document viewers and to convert documents. The Java Viewer beans are used in both the eClient viewer applet and in the middle-tier of the eClient.

Understanding basic beans concepts

JavaBeans (hereinafter referred to as Beans) are reusable software components that are written in the Java programming language and can be manipulated using builder tools that are Beans-aware. Because the Beans are reusable, you can use them to construct more complex components, build new applications, or add functionality to existing applications. You can do all of this visually, using a builder, or manually, by calling the beans methods from a program.

Beans are essentially Java classes. You can turn almost any existing programming component and Java class into a Bean. In general, any Java class that adheres to specific conventions regarding property and event interface definitions can be considered a Bean.

Beans define a design-time interface that allows application designer tools, or builder tools, to query components to determine the kinds of properties these components define and the kinds of events they generate or to which they respond. You do not have to use special introspection and construction tools when working with Beans. The pattern signatures are well defined and can be easily recognized and understood by visual inspection.

Beans have the following characteristics:

Introspection

Introspection is the process by which a builder tool determines and analyzes how a Bean works at design and run time. Because the Beans are coded with predefined patterns for their method signatures and class definitions, tools that recognize these patterns can ″look inside″ a Bean and determine its properties and behavior. Each Bean has a related Bean information class, which provides property, method, and event information about the Bean itself. Each Bean information class implements a BeanInfo interface, which explicitly lists the Bean features that will be exposed to application builder tools.

Properties
Properties control a Bean's appearance and behavior. Builder tools introspect on a Bean to discover its properties and to expose those properties for manipulation. This allows you to change a Bean's property at design time.

Customization
The exposed properties of a Bean can be customized at design time. Customization allows you to alter the appearance and/or behavior of a Bean. Beans support customization by using property editors or by using special, sophisticated Bean customizers.

Events
Beans use events to communicate with other Beans. Beans can fire events, which means that the Bean sends an event to another Bean. When a Bean fires an event it is considered a source Bean. A Bean can also receive an event, in which case it is considered a listener Bean. A listener Bean registers its interest in the event with the source Bean. Builder tools use introspection to determine those events that a Bean sends and those events that it receives.

Persistence
Beans use Java object serialization, by implementing the java.io.Serializable interface, to save and restore states that might have changed as a result of customization. For example, the state is saved when an application customizes a Bean in an application builder, so that the changed properties can be restored at a later time.

Methods
All Bean methods are identical to methods of other Java classes. Bean methods can be called by other Beans or through scripting languages. By default, all of a Beans' public methods are exported.

Using JavaBeans in builders

This section explains how to use JavaBeans in IBM Websphere Studio Application Developer, and in other builders.

To use builders other than IBM Websphere Studio Application Developer, make sure that the builder supports Java 2. Follow the builder's instructions for adding new jar files to add the jars specified in the instructions below. Then, follow the builder's instructions for adding beans from a jar to add the EIP beans, which are in cmb81.jar.

Important: To use the beans, you must have at least Java JDK 1.3.1 or above.

The `CMBROOT\Samples\java` directory contains code samples of the non-visual beans.

Using IBM Websphere Studio Application Developer

You can use the non-visual beans to build servlets and JSP pages in Webphere Studio Application Developer by completing the following steps:

1. Create a Web project for your Web application.

2. In the properties for the project, in Java Build Path | Libraries, specify the following JAR files:

   \CMBROOT\lib\cmb81.jar

   \CMBROOT\lib\cmbview81.jar

   \CMBROOT\lib\cmbsdk81.jar

   \CMBROOT\lib\esclisrv.jar

   \SQLLIB\java\db2java.zip
3. If you plan on using the EIP servlets and JSP taglib, you must also specify the following files:

   \CMBROOT\lib\cmbservlet81.jar

   \CMBROOT\lib\cmbtag81.jar

   For the tag library, you also need to import the taglib.tld file for EIP's JSP taglib into your web application:
   - Copy \CMBROOT\lib\taglib.tld to the webApplication\WEB-INF directory in your web application.
   - Configure the taglib in the webApplication\WEB-INF\web.xml file in you web application by adding the following:

     ```
     <taglib>
                 <taglib-uri>cmb</taglib-uri>
                 <taglib-location>/WEB-INF/taglib.tld</taglib-location>
     </taglib>
     ```
4. Since the JARs listed above contain J2EE classes, you need to include the J2EE JAR, usually located in: `\Program Files\IBM\Application Developer\plugins\com.ibm.etools.websphere.runtime\lib\j2ee.jar`

Invoking the EIP Java Beans

The Beans in the EIP layer can be called in one of two ways. You can call them directly by using their public interfaces (public methods). In this case, explicit Java exceptions are thrown to indicate error events.

Another method for calling the functionality on the session-wide beans is to wire any instances of this type of bean to other EIP Beans using request and reply events. When using this method, remember the following:

- The CMBConnection bean listens to connection request events and replies by firing connection reply events.
- The CMBDataManagement beans listens to data request events and fires data reply events in return.
- The CMBSchemaManagement beans listens to schema request events and fires schema reply events in return.
- The CMBQueryService bean listens to search request events and fires search reply events in return.
- The CMBWorkflowDataManagement beans listens to workflow data request events and fires workflow data reply events in return.
- The CMBWorkflowQueryService beans listens to worklist request events and fires worklist reply events in return.

Non-visual beans

EIP provides a set of non-visual JavaBeans that you can use to build Java applications. The non-visual beans are a set of Java classes that follow the Beans conventions. They are built using the EIP and CM Java connector classes. A typical use is in building Servlets or Java Server Pages (JSPs), although they can also be used in command line or in Windows applications.

The benefits of using the non-visual beans are as follows:

- Provide a federated access mechanism and common programming model for the many different connectors that ship with EIP.
- Allow you to program at a higher level of abstraction.
- Hide the complexity and details of individual connectors.
- Allow you to leverage Beans support built into most commercial development environments.

Using the beans makes building basic applications easier; however, there are some limitations that you must be aware of before you begin using them. The Beans do not:

- Provide all the features available in the Java API layer, for example, administrative or configuration functionality.
- Support batch import. The beans only support casual single item type import capabilities. Batch processes for importing and exporting large amounts of data should be written using the connector interfaces.
- Provide all functionality supported by all servers. Some functionality is server specific. For example, the functionality related to sub-items within items is tied to the functionality available in the Content Manager Version 8 connector only.

The limitations listed above can, in some cases, be bridged by using accessor methods that allow access to the underlying Java API's.

Important: The EIP beans are not Enterprise Java Beans (EJB). Therefore, they cannot be hosted directly inside the managed environment provided by containers like IBM Websphere. However, they can been used from inside EJBs as the underlying connection mechanism to unstructured data repositories.

Non-visual bean configurations

Non-visual beans have local, remote and dynamic configurations.

local Connects directly to the content server.

remote
Connects to a content server using an RMI server.

dynamic
Enables an application that dynamically switches between local and remote based on the `cmbcs.ini` file. The `cmbcs.ini` file specifies whether the content server is local or remote.

Understanding the non-visual beans features

You can use the EIP Beans in JSPs since their properties are typically simple types like strings and arrays. In essence, they act as the model component for Web applications because they are modeled using the Model View Controller (MVC) design pattern. Note that the view component is typically comprised of JSPs and the controller component of servlets (like the ones in the EJB servlet kit). The following is a list of the EIP non-visual beans features:

- Provide access to the schema definitions in the library server.
- Provide CRUD (create, retrieve, update, delete) methods for documents, simple (non-resource) items, and resource items in all of the repositories supported by EIP.
- Provide functionality to search and retrieve documents, simple items, and resource items in all of the repositories supported by EIP.
- Support conversion of data types to viewable formats.
- Act as a federating layer that enforces a consistent set of semantics across the many content management repositories supported by EIP.
- Integrate and expose the functionality provided in the EIP workflow and information mining services.
- Provide document extraction and conversion services as well as support for managing document annotations.
- Provide sorting and conversion functionality.
- Provide events that are fired for key actions occurring on the constituent beans, like connect and disconnect events, search results notification events, and content change notification events.

Non-visual beans categories

The non-visual beans can be divided into the following categories:

Datastore beans

These beans exist across a typical user session and present specialized services to the user. The session-wide beans include the following:

- **CMBConnection** This bean maintains the connection to a backend server which could be a native content server or a federated server. This bean is required in order to use any of the JavaBeans.
- **CMBSchemaManagement** Used to work with repository metadata.
- **CMBDataManagement** Used to work with repository data.
- **CMBQueryService and CMBSearchResults** Used to run queries and work with the results from the queries.
- **CMBWorkflowDataManagement and CMBWorkflowQueryService** Used to work with EIP advanced workflow processes.
- **CMBDocRoutingDataManagement and CMBDocRoutingQueryService** Used to work with CM v8 document routing processes.
- **CMBDocumentServices** Used to provide document streaming and annotation services.

Helper beans

The helper beans exist in the context of one or more of the session-wide beans and are primarily used for encapsulation of data values and for providing services to the session-wide beans. The helper beans available include the following:

- **CMBEntity** Represents data item definitions available in the content management repositories. For example, for a CM v8 repository, a CMBEntity represents both item types and child component definitions, while for a CM v7 repository, a CMBEntity represents an index class. CMBEntity is a helper class for CMBSchemaManagement.
- **CMBAttribute** Represents attribute definitions in the repositories. CMBAttribute is a helper class for CMBSchemaManagement.

- **CMBSearchTemplate** Represents a federated search template. CMBSearchTemplate is a helper class for CMBSchemaManagement.
- **CMBSTCriterion** Represents a search criterion that is a part of a federated search template. CMBSTCriterion is a helper class for CMBSchemaManagement.
- **CMBItem** Represents instances of documents, resource items and non-resource items. CMBItem is a helper class for CMBDataManagement.
- **CMBObject** Represents instances of resource items, base parts and notelog parts. CMBObject is also used to represent BLOB attributes. CMBObject is a helper class for CMBDataManagement.
- **CMBAnnotation** Represents instances of annotation parts for CM v8 repositories and notes for OnDemand repositories.
- **CMBPrivilege** Provides the functionality required to retrieve privilege related information from a CM or EIP supported repository.

Ancillary Beans

The ancillary Beans are not essential in applications, but can be useful for enhancing the functionality. Following is a list of the ancillary beans.

- **CMBConnectionPool** Used to provide pooling services to CMBConnection beans. The Java API API class DKDatastorePool is used to maintain the DKDatastore instances that are used by CMBConnection instances. By moving the pooling to the Java API level, the content servers can be better managed, since they can be pooled more intelligently based on the type of server.
- **CMBUserManagement**Used in connections to federated repositories to manage the mappings of federated users to native server users.
- **CMBExceptionSupport** Provides a framework for common exception event handling.
- **CMBTraceLog** Provides a common trace event-handling framework and provides listener capabilities for trace events fired by other beans.

Workflow beans

The workflow beans provide workflow services. The workflow services provided by the EIP Beans layer support two types of workflow systems: advanced workflow and document routing. Advanced Workflow functionality is built using MQSeries Workflow, while document routing is a workflow system integrated into the CM V8 product and API set. The beans layer provides the full set of objects required to create workflow definitions, execute workflow instances based on the created definitions, and manage instances of executing workflows. The following beans are the main components of the advanced workflow support:

- **CMBWorkflowDataManagement** This bean is used to create and work with advanced workflow instances. An instance of this bean can be retrieved from the CMBConnection object. This bean provides support for:
  - Starting, terminating, suspending, and resuming a workflow instance.
  - Transferring work-items and work notifications from one user to another.
  - Canceling work notifications.
- **CMBWorkflowQueryService** The CMBWorkflowQueryService provides an interface for querying advanced workflow related information. An

instance of this bean can be retrieved from the CMBConnection object. This bean provides support for retrieving the following information:
  - Information on workflows in the system.
  - Information on the work items moving through the system as part of active workflows.
  - Work list related information.
  - Information on all the registered work notifications.

The following beans are the main components of the document routing support.

- **CMBDocRoutingManagementICM** This bean is used to create and manage document routing processes. You can obtain an instance of this bean from an instance of the CMBConnection object. This bean provides support for the following features:
  - Starting, terminate, suspending and resuming a document routing instance.
  - Checking out a CM item contained inside a work package.
  - Setting properties for work packages being routed by a document routing instance.
- **CMBDocRoutingQueryServiceICM** This bean provides an interface for querying information related to document routing processes. You can obtain an instance of this bean from an instance of the CMBConnection object. This bean provides support for retrieving the following information:
  - Information relating to the work packages being routed through the document routing system.
  - Information relating to the processes that are currently active in the system.
  - Information on all the worklists that are present in the system.
  - All the work nodes that are part of the system.

Information Mining Beans

The EIP Information Mining beans allow an application to incorporate text analysis and mining technology. The information mining beans offer the following functionality:

- Text summarization and categorization, creation of a document summary.
- Categorization, assignment of a category to a document.
- Support for extracting relevant information from a document.
- Support for identifying the language that a document is written in.
- Support for clustering similar documents in a document collection.
- Text searching of documents across the catalog or restricted to documents of a certain category.
- Support for accessing documents that are continually retrieved from the Web using the IBM Web Crawler.

The information mining beans include the following:

- **CMBCatalogService** Provides an interface for retrieving information related to a catalog. It also provides support for importing items created from information mining operations into the content server. All information mining operations are bounded by a catalog. Each catalog is associated with a taxonomy which in turn consists of a hierarchy of categories.

- **CMBAdvancedSearchService** Provides support for completing text searches on information in the catalog. Methods on this bean allow you to control the catalog against which the search is to be run as well as the content and size of the results generated from the text search.
- **CMBCategorizationService** This bean is used to determine document categories based on a specified catalog.
- **CMBSummarizationService** Provides the functionality required to generate summaries of documents.
- **CMBClusteringService** Used to group documents into clusters based on the similarity of their content.
- **CMBWebCrawlerService** Provides an interface for managing the results of Web Crawler actions. It allows the user to initiate web crawling requests on specific webspaces and manage the results. The created results can then be categorized, summarized, and imported into a backend content server.

Document Services Beans

The document services beans provide document streaming and document annotation services. The following beans are part of the document services sub-section:

- CMBDocumentServices - The CMBDocumentServices bean provides services needed when working with documents, including the functionality needed to render, convert, and reconstitute the pages of one or more documents.
- CMBDocument - This bean represents the entity created when loading a document using CMBDocumentServices. In essence, CMBDocument is a container for the pages in the document. CMBDocument also allows you to query and set properties that control a document's characteristics.
- CMBPage - This bean provides a representation of a particular page in a document. The functionality in this class allows you to specify and control the properties of a set of renderable images that can be generated for the page.
- **CMBPageAnnotation** This bean models an annotation that can be associated with a page in a document. All supported annotations are modeled by sub-classes of this bean. Also, the CMBPageAnnotation itself contains properties like the page the annotation is on and the annotation type. The sub-classes include the following:
  - CMBArrowAnnotation
  - CMBCircleAnnotation
  - CMBHighlightAnnotation
  - CMBLineAnnotation
  - CMBNoteAnnotation
  - CMBPenAnnotation
  - CMBRectAnnotation
  - CMBStampAnnotation
  - CMBTextAnnotation

Other Beans classes

The Bean classes listed in this section provide a variety of functionality, as described below.

- **BeanInfo Classes** Allow for explicit exposure of the features of the EIP beans in a separate, associated class that implements the BeanInfo interface.
- **Exception classes** Used to encapsulate exceptions in the Beans layer. All the exception classes inherit from the base class CMBException. Each of the sub-classes of CMBException indicates a specific error condition. In each case, properties of the exception object can be used to obtain detailed error information on the error condition that led to the exception being thrown. When the beans are used in the event-driven fashion, exceptions are thrown inside events.
- **Event and listener classes** Implement the standard Beans event listener model. Classes have to explicitly request an event by implementing the listener interface associated with the event and by registering that listener interface with the object that generates an event. The EIP Beans layer provides events and listener pairs for schema access operations, data access operations, workflow operations and search operations.
- **Session listeners** These are the listener classes that exist at the session-wide level. In the EIP Beans, there are currently session listeners that track connection requests and connection reply events.

Considerations when using the non-visual beans

You can use the non-visual beans to enable general-purpose applications with the functionality required to access content management repositories supported by EIP. This section contains some tips on specific usage patterns in the Beans.

Singletons in the Beans CMBConnection has methods to obtain access to instances of the other session-wide EIP Beans. When the session-wide beans like CMBSchemaManagement and CMBDataManagement are obtained in this way, they are already wired to the CMBConnection bean (from which they are obtained) to be informed of a connection or a disconnection, and to share trace and exception event handlers. Only a single instance of each of the other session-wide beans is created. If these methods are called repeatedly, the same instance is returned (singleton design pattern). If session-wide beans are created in the application, and not by the CMBConnection bean, they must be wired to a CMBConnection bean to be used.

Threading considerations in the Beans

A single instance of the CMBConnection bean can only be used on a single thread at any point in time. This restriction extends to all other beans that are associated to a CMBConnection bean (through the connection property of the associated bean). That means that you must create separate connections for each thread. Alternatively, multiple threads can obtain and free connections using the CMBConnectionPool bean. Therefore, each thread should obtain, use, and free a connection.

All the session-wide beans have affinities to an instance of the CMBConnection from which they were retrieved or with which they have been associated after creation. This implies that an instance of the session-wide beans such as CMBSchemaManagement can only be used by a single thread at any given time. If the session-wide bean instance is used by multiple threads, you must perform explicit synchronization in your application to ensure that only a single thread is actively using the session-wide bean instance at any given time.

All session-wide beans also listen to connection reply events generated by the CMBConnection beans. This allows them to recognize that the underlying content repository with which the CMBConnection bean instance is associated has changed, so that the beans can take appropriate action.

Unlike the CMBConnection, the CMBConnectionPool bean is designed for multithreaded use. Multiple threads can simultaneously call the methods related to obtaining and freeing connection objects. Any connection obtained from the pool is an instance of CMBConnection and is restricted to single-thread access. Any connection obtained from the connection pool bean should be returned to the pool as soon as possible after its use, so that it may be made available to other threads that might be requesting connections from the pool.

Tracing and Logging in the Beans

You can enable tracing on all session-wide beans in the EIP Beans layer. Enabling tracing on an instance of the CMBConnection bean also enables tracing on any EIP Bean obtained from this connection bean, including schema management, data management, query service, and workflow beans.

When tracing is enabled, tracing events are fired. A utility bean, CMBTraceLog can listen to trace events and write trace records to a defined log, stdout, stderr, or window (when used with the visual beans).

All session-wide beans also listen to tracing events. The tracing functionality in the beans writes logging information to the same log file as the Java API if log4j is used for logging.

Understanding properties and events for non-visual beans

Each non-visual bean provides the following:

- Imported properties, vetoable or not

  The property value is determined by other beans at run time by PropertyChange or VetoableChange events. Beans that have import properties must listen to PropertyChange or VetoableChange events.
- Exported properties, vetoable or not

  A non-visual bean may have a constrained property and some other beans might have interest in its value. Whenever its value is changed, the bean is responsible for generating a PropertyChange or VetoableChange event.
- Stand-alone properties

  No other beans have interest in this property value.
- Events generated by this bean
- Events in which this bean is interested

Building an application using non-visual beans

A sample non-Graphical User Interface (GUI) application

The example in this section uses non-visual beans to create a sample non-GUI application. The sample application includes every bean except the CMBUserManagement bean. The complete sample application from which this example was taken (`DemoSimpleAppl.java`) is available in the `Cmbroot/Samples/java/beans` directory. The sample application shows how to:

1. Connect to the Enterprise Information Portal (federated) server
2. Get a list of search template names

3. Use the search template name to get a list of search criteria names
4. Select a search template and gets its search criteria
5. Complete the search values and submits a query
6. Print the result using the search results bean
7. Select a result row and displays it
8. Disconnect from the server

Working with visual beans

Visual beans allow you to integrate the functionality of Enterprise Information Portal or other content servers into Java applications based on Swing. Visual beans perform basic tasks that are common to many applications, such as logging on, searching, displaying and viewing results, updating documents, and viewing version information.

Each visual bean has a Connection property. This property must reference an instance of CMBConnection, the nonvisual bean that maintains the connection to the content servers. Any application built with the EIP visual beans must also contain an instance of the CMBConnection nonvisual bean.

CMBLogonPanel
This bean displays a panel to login to Enterprise Information Portal or to content servers such as Content ManagerVersion 8.2 (CM 8.2). It also provides the window where federated users can modify the UserIDs and passwords on the content servers.

CMBSearchResultsViewer
This bean displays search results. When the search result returns folders, use CMBSearchResultsViewer bean to ″drill-down″ into the folder to see its contents. Items in the search results or folders can be selected and opened for viewing in a Windows Explorer style window

CMBSearchTemplateList
For servers that support search templates, this bean displays a list of available search templates and allows selection of a template.

CMBSearchTemplateViewer
For servers that support search templates, this bean displays a search template and provides fields for users to enter search criteria. It performs a search based on those criteria.

CMBSearchPanel
For all servers, the search panel displays a list of available entities, and provides fields for users to enter search criteria. It performs a search based on those criteria. The CMBSearchPanel is useful for performing searches on content servers that do not support search templates.

CMBFolderViewer
Displays the contents of one or more folders in a Windows Explorer style window

CMBItemAttributesEditor
Displays a window where users can update the index class and indexing attributes for an item

CMBDocumentViewer
Displays one or more documents by launching the appropriate viewer

CMBVersionsViewer
Displays version information for a document, if versioning is enabled.

CMBLogonPanel bean

The CMBLogonPanel bean (see Figure 20) displays a window that lets users login to a content server, update user mappings, and change a password.

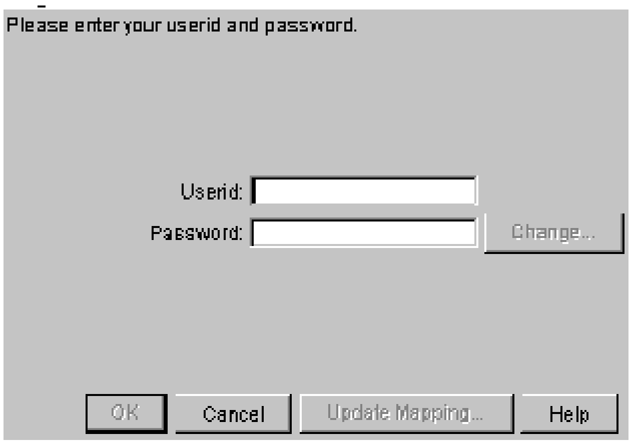


Figure 20. CMBLogonPanel bean window

In the CMBLogonPanel bean, when a user clicks **Change**, the **Change Password** window appears (see Figure 21.) The user enters the old password, and enters the new password twice.

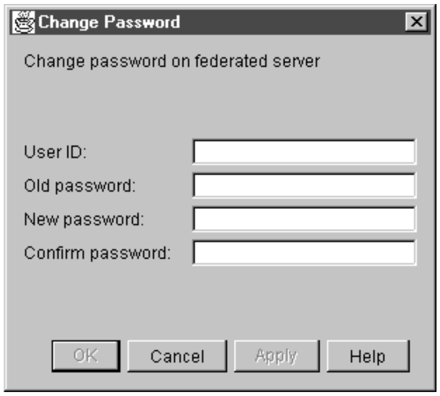
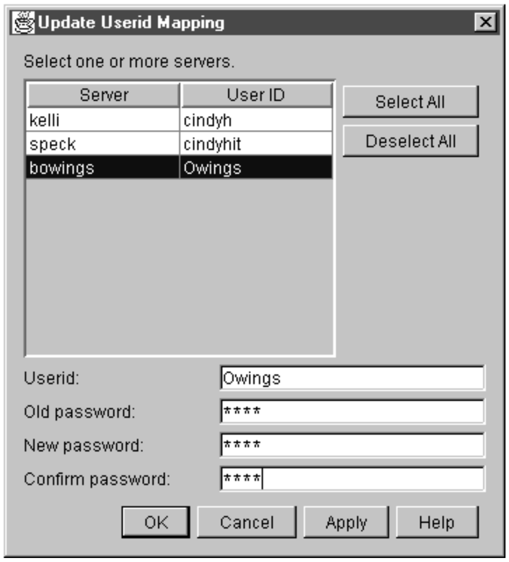


Figure 21. Change Password window

In the CMBLogonPanel bean, when a user clicks **Update Mapping** in the Logon window, the **Update Userid Mapping** window is displayed (see Figure 22 on page 405). When you Update Mapping, you update the user ID and password specified for a server. This function is available only when logging on to the Enterprise Information Portal federated database.



Select one or more servers.

Server	User ID
kelli	cindyh
speck	cindyhit
bowings	Owings

Userid: Owings

Old password: ****

New password: ****

Confirm password: ****

Figure 22. Update Userid Mapping window

At the top of the window is a list of all servers and a corresponding user ID. Users can select one or more servers from the list. Click **Select All** to select all servers. Users can specify a new user ID and (optionally) password after you select one or more servers. If you select one server, the user ID appears in the **Userid** field. If users select more than one user ID, the **Userid** field is blank.

Deselect All
Removes all server selections.

Apply Click to apply mapping and password changes without closing the window.

OK Click to accept changes and close the window.

Cancel
Click to close window without making changes.

CMBSearchTemplateList bean

The CMBSearchTemplateList bean has three styles. The image style, shown in Figure 23, uses one image for the backgrounds of the selected items and another for the unselected items. Figure 24 on page 406 shows the simple template list style. Figure 25 on page 406 shows the drop-down template list style.

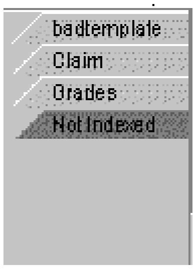


Figure 23. Image template list style

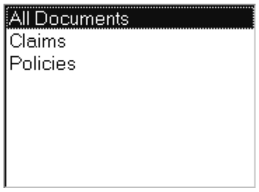


Figure 24. Simple template list style

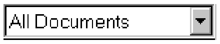


Figure 25. Drop-down template list style

CMBSearchTemplateViewer bean

The CMBSearchTemplateViewer bean (see Figure 26) displays a window where users can specify search criteria according to the search template defined by the system administrator. The CMBSearchTemplateViewer bean launches a search and generates the CMBSearchResults event to return the search results.

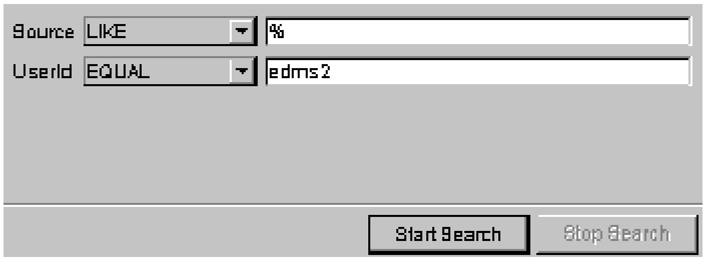

Source LIKE %
UserId EQUAL edms2

Figure 26. CMBSearchTemplateViewer bean

The CMBSearchTemplateViewer bean lists search criteria such as Source or Userid. Each search criteria has a label, an operator drop-down box, and a text field. The BETWEEN or NOTBETWEEN operator display has two text fields. The IN or NOTIN operators have a multi-line text area. Each value should be entered on a separate line.

Text search areas

The CMBSearchTemplateViewer bean can contain areas that allow users to perform a search on full text or index attributes. A full text search area on the template can be as simple as a text field with a label.

Users must match the query syntax for a free or boolean text search when they enter the query string in the text field (see the DKDatastoreTS class). Turn to the online API reference for details.

Validating or editing fields of the CMBSearchTemplateViewer

You can provide validation logic for the CMBSearchTemplateViewer bean to modify search criteria entered by the user. Do this by providing a handler for the CMBTemplateFieldChangedEvent. The current values of the search criteria are stored in the CMBTemplate returned by the getTemplate method prior to this event being called. You can examine and change the criteria. After the event handling is complete, the new values display.

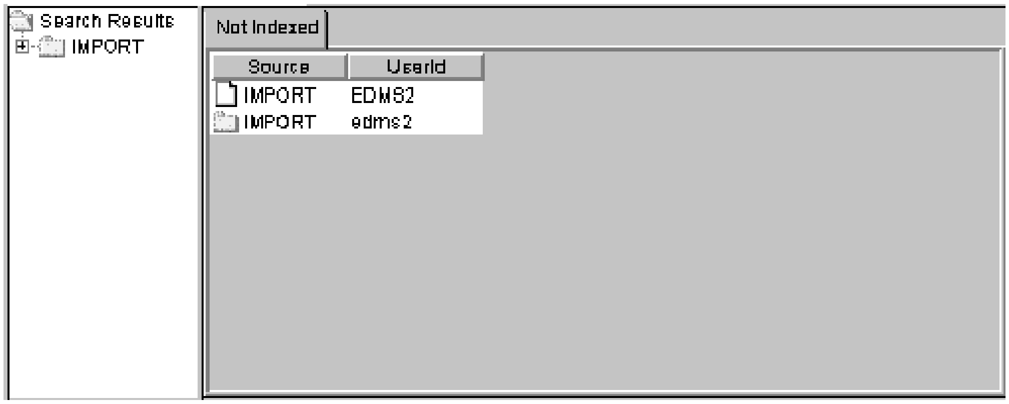
CMBSearchPanel bean

The CMBSearchPanel bean displays a window where users can specify search criteria according to the entities available on the current content server. The CMBSearchPanel bean launches a search and generates the CMBSearchResultsEvent to return the search results. The CMBSearchPanel lists all the available entities in the drop-down list at the top of the window. When an entity is selected, the CMBSearchPanel displays the attributes of the entity. Each attribute has a label, an operator drop-down box, and a text field. A range operator display, such as BETWEEN or NOTBETWEEN, has two text fields. An operator that takes multiple values, such as the IN or NOTIN operator, has a multi-line text area. Each value should be entered on a separate line in the multi-line text area.

CMBSearchResultsViewer bean

The CMBSearchResultsViewer bean displays search results in a window with a tree pane and a details pane. Users can resize the window by clicking and dragging on the line separating the panes.

Figure 27 shows the CMBSearchResultsViewer bean with the **Search Results** folder selected.



Source	UserId
IMPORT	EDMS2
IMPORT	edms2

Figure 27. CMBSearchResultsViewer bean

CMBSearchResultsViewer Tree pane

The tree pane (on the left) contains a main folder labeled **Search Results**. Beneath that folder is each folder found in the search. The tree pane is optional. Remove it by setting the TreePaneVisible property: `setTreePaneVisible(false)`.

CMBSearchResultsViewer Details pane

The details pane displays the contents of the folder selected in the tree pane. When users select the **Search Results** folder, a tab appears on the notebook containing the search template name. When users select a different folder within **Search Results**, one or more tabs display: one for each index class in the folder. The tab names have the form:

index class @ ***server***

where *index class* is the index class or item type name and *server* is the content server name. The table columns change to display the attributes according to the index class or item type. Multiple selection is supported in the details pane. Turn off Multiple selection by setting the MultiSelectEnabled property: `setMultiSelectEnabled(false)`. If an item type is hierarchical, the attribute values of the children are displayed in the table with column headers of the form: child component name/attribute name, where child component name is the name of the child component, and attribute name is the name of the child component's attribute. For example, if an item type called `Journal` has a child component called `Author`, and the `Author` child component has an attribute called `Last Name`, the column header is: `Author/Last Name`.

Pop-up menus
A pop-up menu offering Sort options appears when a user right-clicks on a table column heading. Users click **Sort Ascending** to sort the items in the table in ascending order. Users click **Sort Descending** to sort the items in descending order. Another pop-up menu appears when a user right-clicks a folder other than the **Search Results** folder in the tree pane, or right-clicks a document or folder in the details pane. The pop-up menu lets users View folder details in the tree pane, or Edit Attributes for folders.

Optional: Use the CMBViewFolderEvent rather than show the details of the folder within the CMBSearchResultsViewer bean. Use the event to make the CMBFolderViewer bean display the selected folder's contents.

Double-click action
Double-clicking a folder in the tree pane or an item in the details pane performs the same action as clicking in the **View** pop-up menu item. If you suppress the default item pop-up menu, a CMBItemActionEvent occurs.

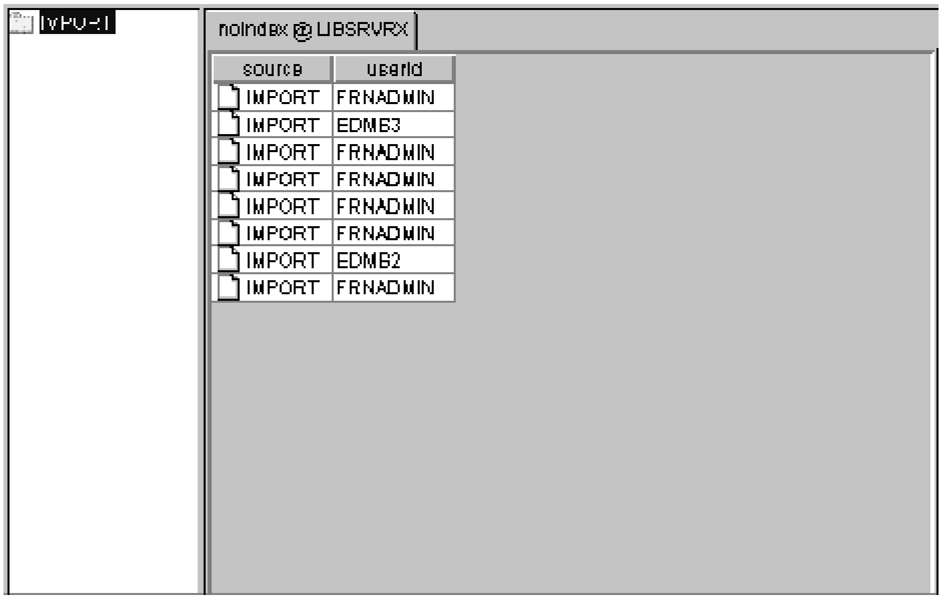
Overriding pop-up menus

You can override the pop-up menus on the CMBSearchResultsViewer and CMBFolderViewer with either a different pop-up menu or no pop-up menu. To turn off the default menus, use `setDefaultPopupMenu(false)`.

When the user right-clicks a folder in the tree pane, a CMBFolderPopupEvent is generated. When the user right-clicks an item in the details pane, a CMBItemPopupEvent is generated. You can use a handler to provide a different pop-up menu.

CMBFolderViewer bean

The CMBFolderViewer bean displays a tree pane that looks like the CMBSearchResultsViewer bean. There is no main **Search Results** folder. Figure 28 on page 409 shows the tree and details panes of the CMBFolderViewer bean.



IVPU-1

noIndex @ LIBSRVRX

source	userid
IMPORT	FRNADMIN
IMPORT	EDMB3
IMPORT	FRNADMIN
IMPORT	FRNADMIN
IMPORT	FRNADMIN
IMPORT	FRNADMIN
IMPORT	EDMB2
IMPORT	FRNADMIN

Figure 28. CMBFolderViewer bean

The CMBFolderViewer bean displays a tree of folders on the left pane. The right pane displays a notebook of tables of the documents contained by folder selected in the tree pane. A resizeable splitter separates the tree and notebook panes.

CMBFolderViewer Tree pane
The tree pane contains folders. Nested folders appear beneath each folder.

CMBFolderViewer Details pane
The details pane contains the contents of the folder that is selected in the tree pane. The contents display in a notebook with a tab for each entity (index class, item type, or other) and server, that the items in the table are indexed under. The tab names have the form: `index class @ server` where *index class* is the name of the index class and *server* is the name of the server. Within each notebook page is a table displaying the documents and folders within the selected folder. The table columns change to display the attributes according to the index class.

Pop-up menus

The behavior of the pop-up menus for the folder viewer is identical to that of the search results viewer.

Double-click action

Double-clicking in the folder viewer is identical to that of the search results viewer.

CMBDocumentViewer bean

The CMBDocumentViewer bean provides capabilities to view documents by either launching or embedding content-type specific document viewers. There are two types of viewers supported:

1. Java-based viewers. These viewers must extend the class CMBJavaDocumentViewer.
2. Non-Java viewers. Any executable may be launched as a viewer for a particular content-type.

If the Visible property is set to false, the viewer is always displayed in a separate window. If the Visible property is true, the viewer will be displayed within the display region of the CMBDocumentViewer bean if possible. (Currently, this is only possible for Java-based viewers.)

CMBJavaDocumentViewer is an abstract class extended by providers of Java-based document viewers that plug into the CMBDocumentViewer bean. These viewers can display the documents in the visible space of the CMBDocumentViewer bean or in separate windows on the screen.

A call to CMBDocumentViewer terminate() waits until all document closed events are processed. If you call terminate() from within the document closed event handler, deadlock may occur and the program hangs. To avoid this problem, when calling terminate() from within the onDocumentClosed(CMBDocumentClosedEvent) event handler, call the CMBDocumentViewer.terminate() method using SwingUtilities.invokeLater(Runnable). This adds the terminate() call to the end of the event queue and continues with the other events in the queue (such as handling the other document closed events) before calling the terminate method.

Viewer specifications

There are two ways to specify viewers:

1. In EIP Administration, specify the viewers using the MIME Type to Application Association Editor. This is selected by choosing **MIME to Appl. Editor** from the **Tools** menu. For Java-based viewers, the application name should be the Java class name, including the **.class** suffix. For executables, the application name should be the name of the executable.
2. Using the Mime2App property on CMBDocumentViewer. This property can be set to an instance of a Properties object that maps the MIME types to application names.

In cases where a viewer is specified for a MIME type in both EIP Administration and using the Mime2App property, the specification using the Mime2App will take precedence.

Default viewers

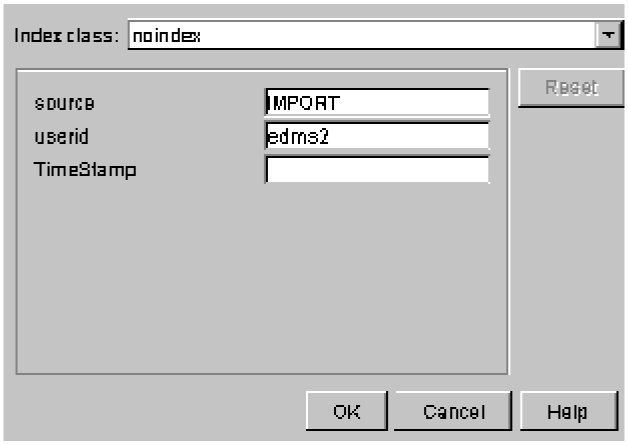
If no viewer is specified for a particular content type, a default viewer will be launched. For documents from OnDemand, the OnDemand client (in view-only mode) is launched. Documents from all other content servers will be viewed using the Content Manager viewer. To edit annotations, select ″Edit Document″ from the ″File″ menu of the viewer.

Launching external viewers

Use the Mime2App property of CMBDocumentViewer to specify applications to launch as document viewers for documents of certain MIME types. Use `setMime2App` with a properties object as the argument that has names of MIME types mapping to values that are executable names.

CMBItemAttributesEditor bean

The CMBItemAttributesEditor bean (see Figure 29 on page 411) displays a window for viewing and modifying the index class and indexing attributes of a folder or document.


Index class: noindex

source: IMPORT

userid: edms2

TimeStamp:

Figure 29. CMBItemAttributesEditor bean

A list containing all available entities appears at the top of the window. The current entity is selected by default. A list of attributes for that entity appears beneath the entity. The text fields (first name, last name, and so forth) initially contain the current values for the item.

If users select a new entity, any attributes with the same names as the previously-selected entity have their values propagated to the like-named attributes in the new entity.

Clicking **Reset** returns the entity and attributes to their original values.

Clicking **OK** updates the entity and attributes and triggers events before and after the update. You can use the event before the update to validate fields or complete missing fields before the update is performed. This event can veto the specified update.

Vetoing changes in the CMBItemAttributesEditor

You can provide additional validation logic to the CMBItemAttributesEditor that verifies attribute values entered by the user and modifies them, or rejects an update, if the values are not valid. Do this by providing a handler for the CMBEditRequestedEvent.

CMBVersionsViewer bean

The CMBVersionsViewer bean displays a table of versioning attributes for a single document or item. The versioning attributes displayed are: the version number, the user ID of the creator, the timestamp when the version was created, the user ID of the latest updater, and the timestamp of the last update. From the versions viewer, you can view the different versions of an item or update the attributes of an item.

General behaviors for visual beans

The following sections describe properties and behaviors that are common among visual beans.

Properties

This section describe three properties shared by visual beans.

Connection

Each bean has a Connection property, which refers to an instance of the

CMBConnection non-visual bean. You must set the Connection property for the visual bean to operate correctly.

CollationStrength
All beans that perform sorting have a CollationStrength property. The values defined for CollationStrength property are the same values defined for the java.text.Collator class of Java.

Hiding/Showing buttons
You can hide or show the buttons that appear on all visual beans. Use the *setnameButtonVisible* property, where *Name* is the name of the button.

Save/restore configuration

The CMBSearchTemplateViewer, CMBSearchResultsViewer, and CMBFolderViewer have two methods - `loadConfiguration` and `saveConfiguration`- that can be used to save and restore field values and column sizes between application sessions. A properties object is an argument for all these methods. You can use the same properties object for all three beans. The names of the saved properties are unique across the beans.

Help events

Each visual bean generates a CMBHelp event when the user requests help, either by clicking the **Help** button or pressing F1. Some beans generate the following help-related events when users press F1 or Help from secondary windows:

CMBChangePasswordHelpEvent
When **Help** is clicked on the Change Password window

CMBUpdateMappingHelpEvent
When **Help** is clicked on the Update Mapping window

CMBLoginFailedHelpEvent
When **Help** is clicked on the Server Logon Failed window

CMBServerUnavailableHelpEvent
When **Help** is clicked on the Server Unavailable window

Tip: One possible method of handling help from all of these sources is to create a single class that implements the listeners for all of these events. Within the `onHelp` method, additional logic might be needed to determine which bean was the source of the event, and display help text appropriate for that bean.

Replacing a visual bean

It is possible to replace one of the visual beans with another bean or with Swing components. To do this, the new bean should implement the handlers for the events of the visual bean it is replacing. It should also generate at least the key events of the bean it is replacing. The key events are described in Table 31.

Table 31. Visual beans and key events

Visual bean	Key events
CMBSearchTemplateList	CMBTemplateSelectedEvent
CMBSearchTemplate Viewer	CMBSearchStartedEventCMBSearchResults Event
CMBSearchResultsViewer	CMBViewDocumentEvent CMBViewFolderEvent-CMBEditItemAttributesEvent
CMBFolderViewer	CMBViewDocumentEvent CMBEditItem AttributesEvent
CMBDocumentViewer	CMBDocumentOpenedEvent CMBDocument Closed Event

Table 31. Visual beans and key events (continued)

Visual bean	Key events
CMBItemAttributesEditor	none

All data needed for implementing the bean function is available either from events that the bean is handling or from the CMBConnection non-visual bean.

Building an application using visual beans

A sample client application that was written using the visual beans is provided for you. The source files for the sample are located in: `<cmbroot>/samples/java/beans/gui`. You should also read the readme.html file in this directory for details about the sample client and setup requirements.

The following sections show how the visual beans fit together when you build an application.

Connecting the visual beans

This section explains one scenario for connecting visual beans to create a simple application. Except for the **Search** button, all beans are connected by adding the target bean as a listener of the indicated event of the source bean. For example, to connect the SearchTemplateList to the SearchTemplateViewer, a single line of code is needed. To add a button for launching searches, use a standard JButton. Create an inner class to cause the action event from the button to invoke the appropriate method.

InFigure 30, the lines from each of the beans to the connection bean indicate that the bean contains a reference to the connection bean. This is created by setting the connection property for each bean. For example, to create a reference from the logon panel bean to the connection bean, a line of code is needed.

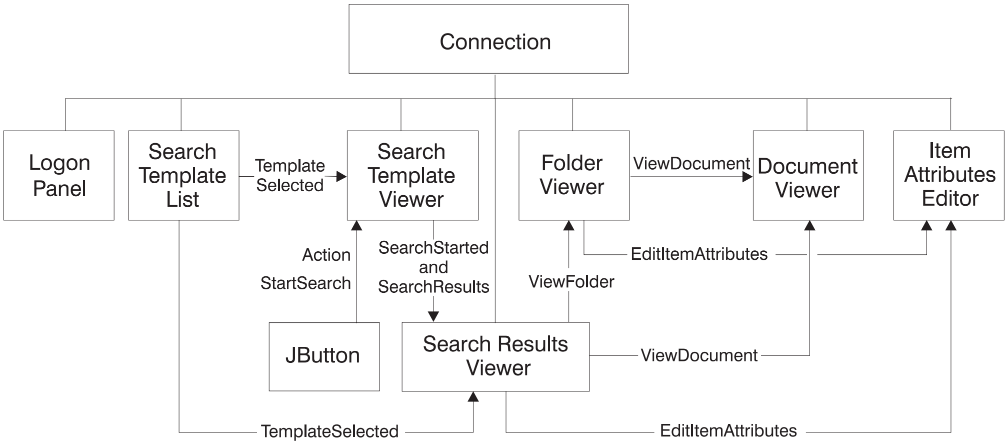


Figure 30. Visual bean connections

Figure 30 shows nine beans. A JFrame or other container bean would be the parent of all of these beans. One possible order of events during run time might be:

1. The user enters a user ID and password into the logon window and clicks **OK**. The CMBLogonPanel bean invokes the `connect` method of the CMBConnection bean to establish the connection to the server.
2. The connection bean establishes the connection. The CMBSearchTemplateList bean retrieves and displays the list of search templates for that user ID. (No

methods need to be invoked to cause this to happen. The CMBSearchTemplateList bean is listening to the appropriate events of the CMBConnection bean. CMBSearchTemplateList sets up the listeners when a CMBConnection bean associated itself with it using the `setConnection` method.)

3. The user selects a search template from the list. The CMBSearchTemplateList bean generates a CMBTemplateSelectedEvent. Both the CMBSearchTemplateViewer and the CMBSearchResultsViewer are listening for the event. The CMBSearchTemplateViewer displays the appropriate template. The CMBSearchResultsViewer clears and displays columns in the details pane as defined by the template.
4. The user completes the template, and either presses Enter or clicks **Search**. If the user clicks **Search**, the action event handler invokes the `startSearch` method. If the user presses Enter, the `startSearch` method is invoked implicitly.
5. The CMBSearchTemplateViewer bean validates the template fields to determine whether a search can begin. If the search can begin, a CMBSearchStartedEvent is generated. CMBSearchResultsViewer listens for a CMBSearchStartedEvent and clears the results in preparation for new search results.
6. As the search progresses, CMBSearchResultsEvents are generated to provide partial search results to the CMBSearchResultsViewer. (When the search is completed the CMBSearchCompleted event is generated. This event can be used to enable the **Search** button again if it was disabled at the start of the search.)
7. The user can expand folders in the Search Results window, then select a document or folder for viewing. When this is done, a CMBViewFolderEvent or CMBViewDocumentEvent is generated. The CMBFolderViewer and CMBDocumentViewer beans are listening to their respective events, and display the folder or document.
8. From the CMBFolderViewer, users can select a document to view. Selecting a document for viewing generates a CMBViewDocumentEvent. The CMBDocumentViewer listens for this event and displays the document in the appropriate viewer.
9. Users can select a document's or folder's attributes for updating from the CMBSearchResultsViewer or CMBFolderViewer. Selecting a document generates a CMBEditItemAttributesEvent.
10. The CMBItemAttributesEditor bean listens for an CMBEditItemAttributesEvent. It displays the entity and attributes for the item. The user can then change the entity and attributes and then click **OK** to apply the changes.

Using beans in more than one window or dialog

You must provide additional code to pass an event from a bean in one window to a bean in another window. Typically, the fact that an event has been sent is usually the reason for displaying a window. The EditAttributesDialog window contains the ItemAttributesEditor. SearchFrame creates the window when a CMBEditItemAttributesEvent launches:

```
// Invoke a secondary dialog for edit attributes
searchResultsViewer.addEditItemAttributesListener(new
CMBEditItemAttributesListener() {
    public void onEditItemAttributes(CMBEditItemAttributesEvent event) {
    EditAttributesDialog editAttributesDialog = new
    EditAttributesDialog(SearchFrame.this,connection,event.getItem());
    editAttributesDialog.setVisible(true);
}
});
```

The information that is normally passed to the CMBItemAttributesEditor bean is passed as arguments to the constructor of the window instead. Within the constructor, the information is passed to the CMBItemAttributesEditor bean by setting the following properties:

```
itemAttributesEditor.setConnection(connection);

itemAttributesEditor.setItem(item);
```

Working with the Java document viewer toolkit

You can use a document viewer to access and annotate documents contained in your content servers. You can create a custom document viewer using the EIP Java viewer toolkit. You can also create custom viewer applets and applications to integrate into EIP or standalone applications.

Important: The *viewdata* options are not supported for the OnDemand backend.

The Java viewer toolkit classes include action objects that provide the following function:

- Page viewing options
  - Rotate documents: 90 degrees clockwise, 90 degrees counter clockwise, and 180 degrees
  - Zoom: in and out
  - Scale: 25%, 50%, 100%, 150%, 200%, and 400%
- Invert
- Enhance
- Print
- Close the current document
- Close all the documents
- Create and edit annotations
  - Write
  - Highlight
  - Draw a box
  - Draw a circle
  - Draw a line
  - Draw an arrow
  - Add text
  - Stamp
  - Add a note
  - Erase
  - Hide or show
  - Default cursor mode
  - Bring annotation to the front
  - Bring annotation to the back
  - Change annotation properties
  - Undo and redo annotation operations
  - Cut, copy, paste, and delete annotations
  - Save document (only annotations are saved)
- Navigate documents or a pages within a document

    Navigate pages: first page, previous page, go to a page, next page, and last page

    Navigate documents: first document, previous document, go to a document, next document, and last document

- Thumbnails
  - Hide or show thumbnails
  - Page navigation using thumbnails
  - Panning and zooming

The Java viewer toolkit provides many GUI classes that you can use to build Swing-based applications. It also provides non-GUI classes that you can use for non Swing-based document viewing applications.

Viewer architecture

The Java viewer toolkit contains a Document Viewer and a Document Services bean. The beans provide a mechanism for integrating the viewer into EIP based applications.

The toolkit also contains the Generic Doc Viewer, Streaming Doc Services, and Annotation Services classes. The classes enable you to use the viewer in applications where you cannot use the beans, such as in standalone viewing or in distributed process situations where the connection to content stores is not local.

Streaming Doc Services manages a set of document processing engines that parse documents, render pages and provide you the capability to manipulate document pages. The engines provide parseddocument images in various formats, such as TIFF and IOCA, as well as text and rich text documents and office formats. You can also write additional document engines and plug them into the viewer toolkit architecture to support additional formats or alternative rendering for document formats.

The Annotation Services class enables you to manipulate annotations. The annotation engine parses Content Manager specific annotation formats. You can write additional annotation engines.

The document engines

There are four document engines provided with EIP:

- MS-Tech Document Engine: This engine handles content types typically found on IBM Content Manager, rendering pages as images. Document types supported include TIFF, MO:DCA. This engine also supports GIF, JPEG, and plain text.
- INSO Document Engine: This engine supports Microsoft Office, Lotus SmartSuite, and other office document formats.
- AFP2Web Document Engine: This engine understands AFP and converts the AFP documents to HTML or PDF.
- Java Document Engine: This engine converts documents that are URL's to HTML with a forwarding link to the URL. This engine also converts XML to HTML by invoking XSLT.

The engines are public interfaces, but you should not program directly to them. Instead, use the interfaces provided by the Document Services bean or the Streaming Doc Services class.

Some of these engines are not pure Java and have portability limitations. This can restrict use of the toolkit on some platforms. The MS-Tech and Java engines are pure Java. The other engines contain platform specific logic that restricts their use to the Windows platform.

The annotations engine

EIP provides the MS-Tech annotation engine to handle Content Manager annotations. The MS-Tech engine supports Content Manager Version 8.2, Content Manager Version 8.1, Content Manager Version 7, and VI/400 annotations.

Creating a generic document viewer

To create a generic document viewer, you work primarily with the CMBGenericDocViewer class, which uses the CMBStreamingDocServices interface. CMBStreamingDocServices loads and renders documents. CMBStreamingDocServices uses a set of document engines to convert different document formats like TIFF and MO:DCA. To load, edit, and save annotations in documents use the CMBAnnotationServices interface.

Customizing the generic document viewer

You can customize the default configuration file, CMBViewerConfiguration.properties, located in the cmbview81.jar file, or you can create a new configuration file. Whether you create a configuration file or customize CMBViewerConfiguration.properties, you must keep the same file name and place it before cmbview81.jar in the class path.

To create a configuration file, complete the following steps:

1. You must specify the following entry for each toolbar specified by the toolbar name. This step specifies the toolbar's position on the main frame. The default position is NORTH. Specify the position for each toolbar that is listed.

```
Toolbars=[<toolbar_name>[,<toolbar2_name>][,<toolbar3_name>]...]
<toolbar_name>.position={NORTH|SOUTH|EAST|WEST}
```

2. Specify the actions that are to be added to the specified toolbar. Use the word 'separator' to insert a separator between the actions on the toolbar.

```
<toolbar_name>.tools=[<action_name>[,<action2_name>]
[,<action3_name>]...]

<action_name>.label=<action_label>
<action_name>.tooltip=<action_tooltip>
<action_name>.icon=<icon_file_name>
<action_name>.key=<key code>
<action_name>.cursor=<cursor_file>
<action_name>.hotspot=<x,y>
```

3. You cannot add new popup menus. However, you can add submenus and menu items to the three pre-defined popup menus.

```
<popup_menu_name>.items=[<menuitem_name>[,<menuitem2_name>]
[,<menuitem3_name>]...]
<popup_menu_name>.submenu=[<submenu_name>[,<submenu2_name>]
[,<submenu3_name>]...]
<submenu_name>.label=<submenu_label>
```

If you have specified the appropriate configuration file, you are ready to build a standalone viewer application or applet. See the *Online API Reference* as you complete the following steps:

1. Create a private class that implements CMBStreamingDocServicesCallbacks.
2. Create CMBStreamingDocServices with the callbacks implemented in the first step.
3. Create a private class that implements CMBAnnotationServicesCallbacks.
4. Create CMBAnnotationServices with the callbacks implemented.

5. Create an instance of the CMBGenericDocViewer and initialize it with the CMBStreamingDocServices, CMBAnnotationServices, and the configuration properties file. Pass null for the properties file if you just want to us the default configuration file.
6. Call loadDocument() in CMBGenericDocViewer to load a document. This returns a CMBDocument instance.
7. Call loadAnnotationSet() on CMBGenericDocViewer to load any annotations. A CMBAnnotationSet object is returned. Use the method setItemHandle(CMBAnnotationSet, CMBItem) in CMBAnnotationServices.
8. Customize the look and feel of the viewer using the different methods provided by the viewer for the position, size of thumbnails, MDI or SDI view, and so forth.
9. Add the viewer to the application's or applet's main frame.
10. Prepare the menu bar by retrieving the actions from the generic document viewer and add the menu to the applications main frame.
11. Call showDocument() in the generic document viewer to display the document.
12. Call saveAnnotations(CMBDocument) on the CMBGenericDocViewer to save annotations on the document.
13. Call closeDocument(CMBDocument) to close a document or call closeAllDocuments() to close all the documents.

Figure 31 is an example of a generic document viewer that uses all of the default settings.

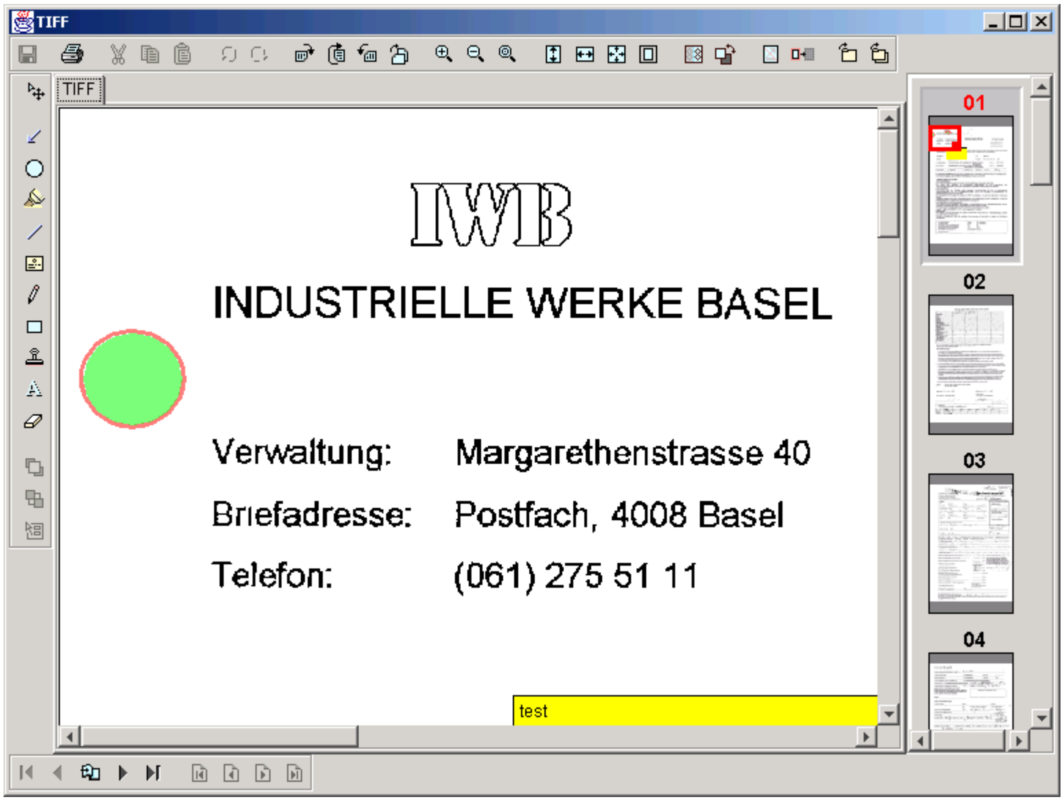


Figure 31. Generic document viewer

Example applications

To help you understand the Java Document Viewer toolkit, this section provides five example applications that you can create. The examples below are not the only ways that you can use the toolkit.

Standalone viewer

You can use the Generic Document Viewer to implement a standalone viewer. You can use the standalone viewer to view files or documents that you retrieve from URL's. You can use the standalone viewers as pallets on web pages or for viewing documents that you obtained through e-mail, for example. Figure 32 illustrates a standalone viewer's architecture.

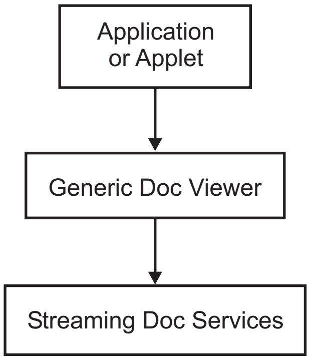


Figure 32. Standalone viewer

Java application

To create a production Java application use the EIP visual beans, which use the CMBDocumentViewer visual bean. This bean uses the Generic Document Viewer internally to display documents. The CMBDocumentViewer bean can also launch other viewers to view documents. Remember, however, that these viewers might have platform dependencies. Figure 33 illustrates the architecture you can use to build a Java application.

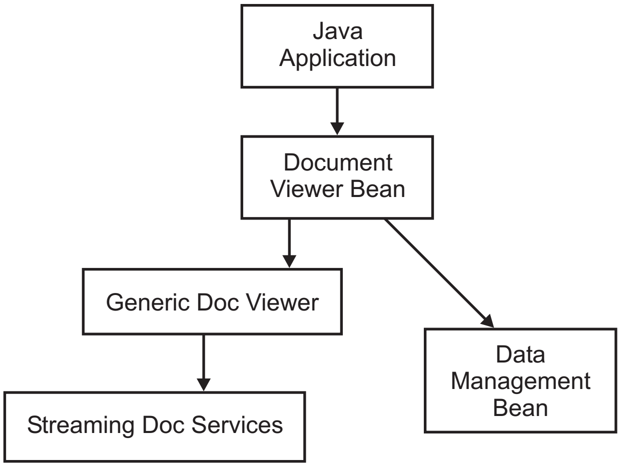


Figure 33. Java application

Thin client

You can use the CMBDocumentServices bean to perform server-side document conversions in a web-based application. You can convert documents from content types that are not handled by the browser (documents that require a plugin or native application launch) to content types that are handled by the browser natively, such as HTML, GIF, JPEG, or for which plugins are readily available, such as PDF. Figure 34 illustrates a thin client architecture.

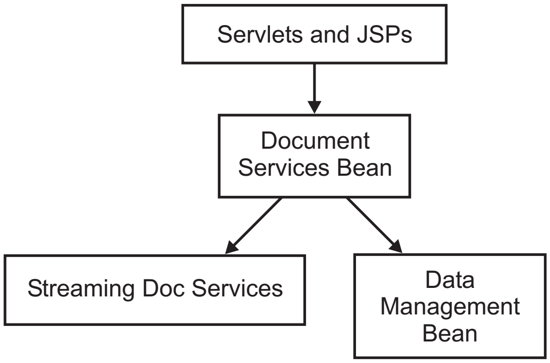


Figure 34. Thin client

Applet or servlet

A web-based application can also provide document viewing and annotation editing capabilities using an applet or servlet approach. The eClient viewer applet uses this architecture. The applet can use the Generic Document Viewer to view the document. Some document types are stored in several parts on the content server to make sharing of common information, such as background forms, more efficient. The additional parts are requested by the Generic Document Viewer when needed. The applet containing the viewer satisfies the requests by sending HTTP requests to the servlet. On the servlet side, the content server using the CMBDataManagement bean obtains the requested information. Figure 35 illustrates an applet or servlet architecture.

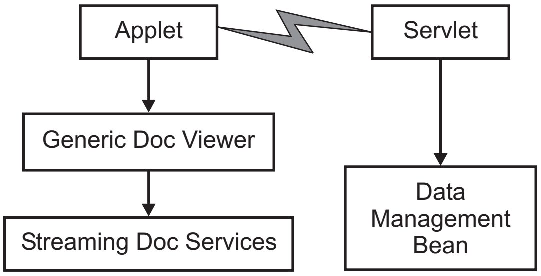


Figure 35. Applet or servlet

Dual-mode and applet or servlet

You can use a variation of the examples provided for web based viewing applications. Use CMBDocumentServices on the server and in the applet. This approach is useful when documents are not rendered in the applet but are converted on the server. This might happen when the underlying document engines on the applet and on the server have different capabilities.

For example, if the server is Windows NT or 2000 and the applet is running on OS/2, the applet might not be able to render all document types. The applet renders document formats that it supports, like TIFF. For types that the applet cannot render, such as Office formats, it requests conversion from the servlet. From the application user's perspective, the same interface is presented with the same functionality. However, server performance of server side converted documents might be slower than for locally rendered documents.

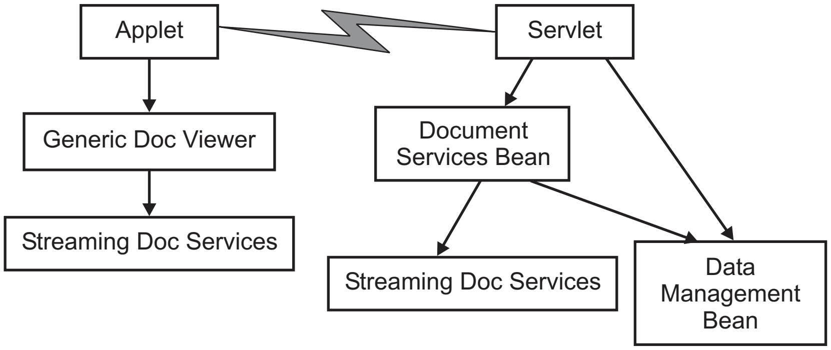


Figure 36. Dual-mode viewer

Working with the annotation services

The Enterprise Information Portal 8.1 Java viewer toolkit provides capabilities for rendering and converting document annotations. Similar to the document services, pluggable annotation engines provide additional facilities that you can use in your applications to interpret different types of annotations. Figure 37 on page 424 illustrates how the generic document viewer and document services and annotation services fit together.

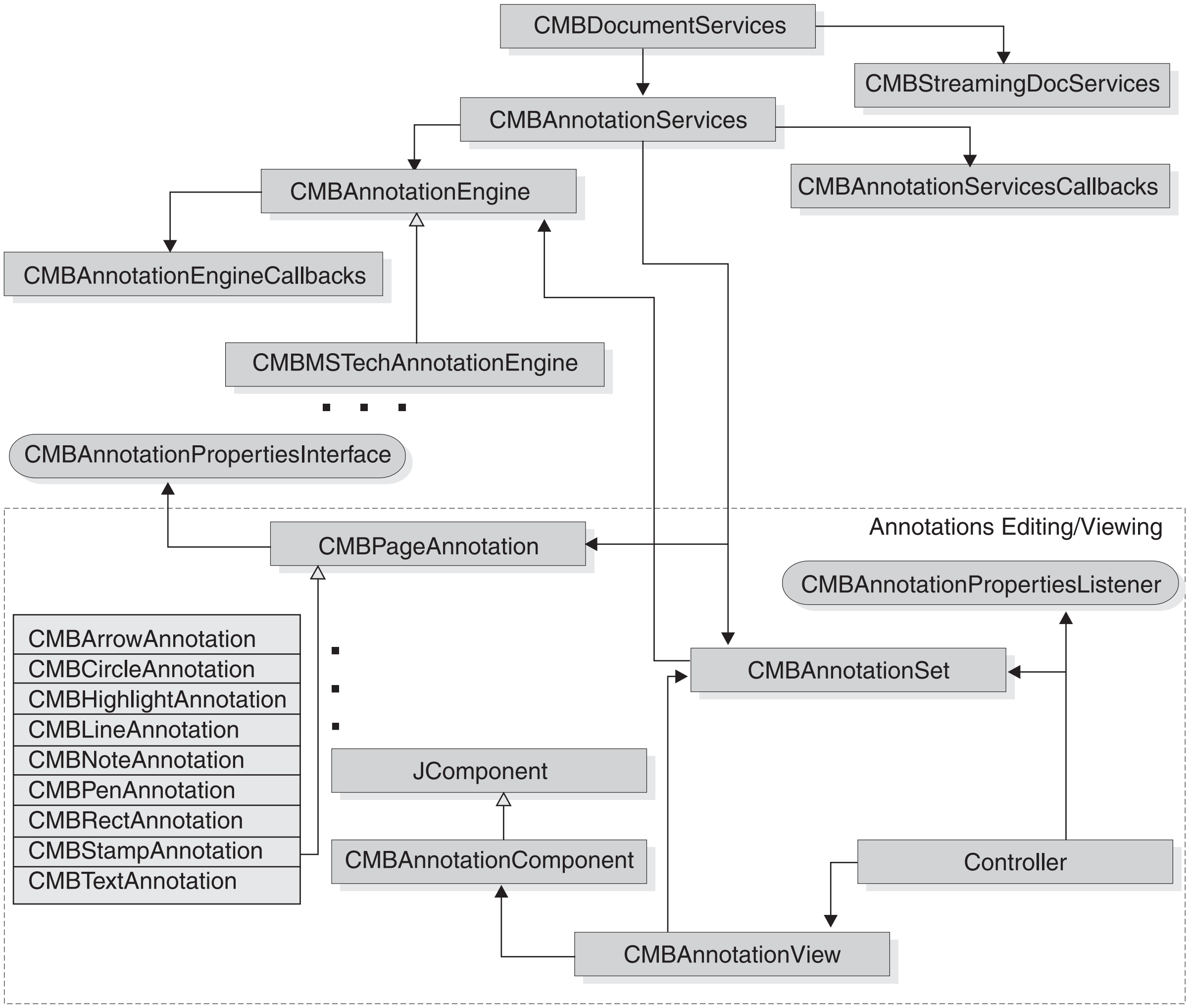


Figure 37. Annotation services and generic document viewer association

Using annotation services interfaces

CMBAnnotationServices provides the main interfaces used in the Java viewer toolkit. The annotation services enables you to load, manipulate, and save annotation objects using the annotation services, independently of backends. To work with annotations, you need to pass in annotation data as a stream and plug in a suitable annotation engine, which converts the annotation objects to CMBPageAnnotation instances. You can then manipulate and edit the annotations and save the annotations back into the original backend in the original format.

Figure 38 on page 425 shows the annotation services class diagram.

To implement the annotation engine, you must extend the abstract class CMBAnnotationEngine. The annotation engine uses CMBAnnotationServicesCallbacks and CMBAnnotationEngineCallbacks interfaces to communicate with an application and annotation services. The annotation engine in EIP 8.1 understands only the Content Manager annotation format. This annotation format is used by Content Manager Version 7, Content Manager Version 8. 1, and VI/400 backends.

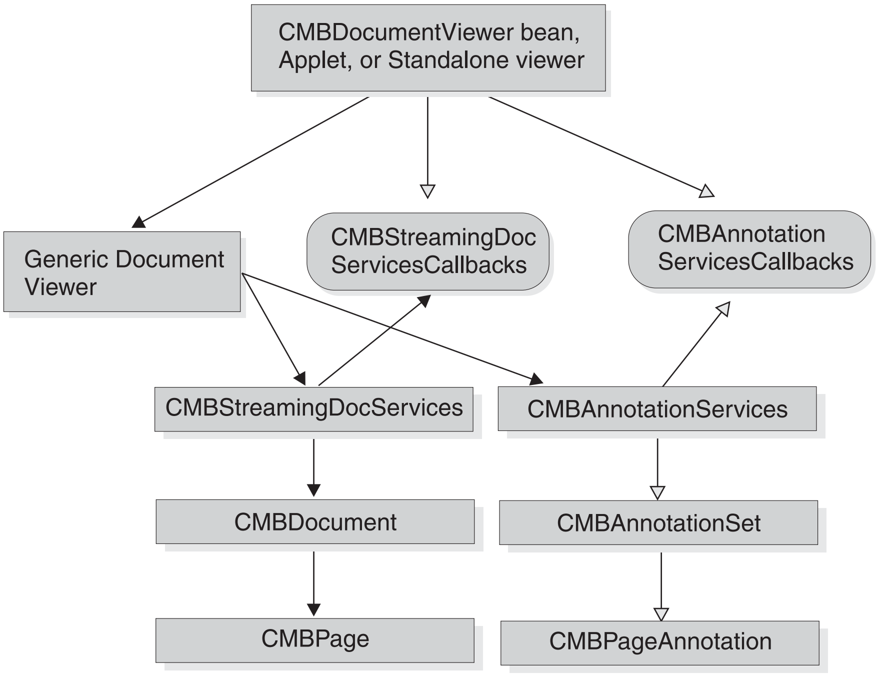


Figure 38. Annotation services class diagram

Understanding annotation editing support

The Model View Controller (MVC) design pattern is used to implement the annotations editing functionality. MVC acts as the model that represents the annotation data. CMBAnnotationSet has methods that operate on the data, but has no user interface. The CMBAnnotationSet class maintains the list of CMBPageAnnotation objects. Each document is associated with a CMBAnnotationSet object that represents its annotations. CMBAnnotationView acts as the view that presents the data from the model to the user. CMBAnnotation handles all the drawing of the annotations on the view component (a JComponent). CMBAnnotationComponent is a helper class that can be used as the view component on which the annotations are drawn. The controller is internal to the viewer toolkit and handles the mouse and keyboard events for annotation creation and editing.

CMBPageAnnotation is the base class that describes a single annotation on a page of a document. If you need to define additional types of graphical annotations, you must extend the CMBPageAnnotation class. There are nine types of Content Manager annotation types that you can create: CMBArrowAnnotation, CMBCirlceAnnotation, CMBHighlightAnnotation, CMBLineAnnotation, CMBNoteAnnotation, CMBPenAnnotation, CMBRectAnnotation, CMBStampAnnotation, and CMBTextAnnotation.

Note: You can use annotation services in non-GUI applications.

Building an application using the annotation services

This section includes the steps to follow and APIs to use to build an annotation services application. Refer to the online API reference for details about API usage.

1. Create a subclass of CMBAnnotationServicesCallbacks to implement the abstract methods to handle annotation callbacks.

2. Create an instance of CMBAnnotationServices.

   ```
   CMBAnnotationServices annoServices = new
   CMBAnnotationServices(annoServicesCallbacks);
   ```

3. Get an instance of CMBAnnotationSet by loading an annotation stream.

   ```
   CMBAnnotationSet annotationSet = annoServices.loadAnnotationSet(annoStream,
    format, documentResolution, annotationPartNumber );
   ```

4. Prepare the annotation view.

   ```
   CMBAnnotationComponent annoComponent = new CMBAnnotationComponent();
   CMBAnnotationViewer annoView = annoServices.prepareAnnotationView(
   annoComponent, annotationSet );
   annoView.refreshEntireDrawingArea();
   ```

5. Add the annotation component to a Swing container like Jframe or Jpanel of the application.
6. You can now use the annotation services API to add new annotations, edit, or delete existing annotations.

   ```
   annoServices.prepareToAddAnnotation();
   annoServices.addAnnotation()
   annoServices.removeAnnotation();
   annoServices.reorderAnnotation();
   ...
   ```

7. Save the modified annotations.

   ```
   annoServices.saveAnnotationset(annotationSet);
   ```

An Annotation Services sample is provided in the `<CMBROOT>\Samples\java\viewer directory(TAnnotationEditor.java)` directory.

Adding a custom annotation type to your application

To add a custom annotation type to the annotation services, complete the steps below. Refer to the online API reference for details about API usage.

1. Create a subclass of CMBPageAnnotation.`public class TImageAnnotation extends CMBPageAnnotation`
2. Define a constant for the custom annotation with a value larger than 100. The range of 1 to 99 is reserved.

   ```
   public static final int ANN_IMAGE = 101;
   ```

3. Override the following methods of CMBPageAnnotation:

   ```
   public void draw(Graphics2D g2);
   public void drawOutline(Graphics2D g2);
   public CMBPropertiesPanel getAnnotationPropertiesPanel();
   ```

4. Implement the CMBAnnotationPropertiesInterface interface to create the properties panel. The properties panel appears when a user chooses to edit the custom annotation properties.
5. Provide set and get methods specific to the custom annotation properties.
6. Add the custom annotation type.

   ```
   annoServices.prepareToAddAnnotation(TImageAnnotation.ANN_IMAGE,
   "TImageAnnotation",1);
   ```

The annotation type sample `TImageAnnotation` is included in the `<CMBROOT>\Samples\java\viewer` directory. The sample demonstrates how to add a custom annotation type to the annotation services.

Working with the Enterprise Information Portal tag library and controller servlet

Enterprise Information Portal includes a Java Server Pages tag library and a servlet that you can use when writing JSP or servlets for Web applications. Using the tag library reduces the need for Java scriptlets in JSP written to the EIP JavaBeans.

This tag library works with the servlet which can act as a controller of a model-view-controller design Web application and performs bean initialization and other actions.

Setting up the tag library and servlet

You must install the tag library and servlet on a Web server with IBM WebSphere® Application Server and configure the Web server to use them. For information about installing the tag library and servlet, configuring them, and for information about building WAR/EAR files, see *Planning and Installing Enterprise Information Portal*.

Using the tag library

The following JSP sample shows using the search templates tag to get a list of search templates:

```
<%@ 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 cellspacing=3 cellpadding=3>
  <tr>
    <td><b>Available Search Templates</b></td>
  </tr>
  <cmb:searchtemplates>
  <tr>
    <td><%= searchtemplate.getName() %></td>
  </tr>
  </cmb:searchtemplates>
</table>
</body>
</html>
```

The `taglib` directive declares that the page uses the EIP tag library and associates the `cmb` prefix with it. Then the `searchtemplates` tag is called and the `getName()` method returns the name of each search template.

Conventions used in the tag library

The JSP tags have attributes to specify the beans that they use or generate. When those attributes are not specified, default values are used. These parameters are optional. Their values are picked up from local variables, and attributes in the request and session scope. When used in conjunction with the servlet toolkit, local variables, request attributes, or session attributes contain appropriate defaults. Table 32 shows the default beans and the scope in which they are assumed or placed. These conventions are also followed by the servlet; follow these conventions in any other servlets that you write to work with the tag library.

Table 32. Tag library conventions

Scope	Name	Type	Description
application	connectionPool	CMBConnectionPool	The connection pool bean that is shared across sessions
session	connection	CMBConnection	The instance of CMBConnection for the session
session	schema	CMBSchemaManagement	Schema management bean
session	data	CMBDataManagement	Data management bean
session	user	CMBUserManagement	User management bean
session	query	CMBQuesryService	Query service bean
session	traceLog	CMBTraceLog	Trace log bean; all of the other beans send their trace to this bean
session	docservices	CMBDocumentServices	Document services bean
request	item	CMBItem	The last item that you operated on
request	items	CMBItem[ ]	Collection of the items that you last operated on
request	searchTemplate	CMBSearchTemplate	The selected search template
request	searchResutls	CMBSearchResults	The results from the last search

Tag summary

The following sections summarize the tag library:

Connection related tags

<cmb:datasources" connection="*connection*">datasource ... </cmb:datasources>

This tag iterates through the available data sources.

connection

Specify the name of a variable of type CMBConnection that contains the connection.

datasource

A string variable to contain the data source name as a string.

Schema related tags

<cmb:searchtemplates searchTemplates="*searchTemplate*"> ... </cmb:searchTemplates>

This tag iterates through the available search templates.

searchTemplates

Specify the name of an array of type CMBSearchTemplate[] to contain the search templates.

searchTemplate

A variable of type CMBSearchTemplate to contain the search template.

<cmb:searchcriteria searchTemplate="*searchtemplate*">criterion ... </cmb:searchcriteria>

This tag iterates through the search criteria of a search template.

searchTemplate

Specify the name of the search template.

criterion

A variable of type CMBSTCriterion to contain the search criterion.

<cmb:displaycriteria searchTemplate="*searchTemplate*">criterion ... </cmb:displaycriteria>

This tag iterates through the search criteria that can be displayed for a search template.

searchTemplate

Specify the name of the search template.

criterion

A variable of type CMBSTCriterion to contain the search criterion.

<cmb:allowedoperators criterion="*criterion*">operator ... </cmb:allowedoperators>

This tag iterates through the operators allowed for a search criterion.

criterion

Specify the name of a variable of type CMBSTCriterion to contain the search criterion.

operator

A string to contain the value of the operator.

<cmb:predefinedvalues criterion="*criterion*"> value... </cmb:predefinedvalues>

This tag iterates through the predefined values of a search criterion.

criterion

Specify the name of a variable of type CMBSTCriterion to contain the search criterion.

value A string to contain the predefined value of the search criterion.

<cmb:entities schema="*schema*">entity ... </cmb:entities>

This tag iterates through the available federated entities.

schema Specify the name of a variable of type CMBSchemaManagement containing the schema.

entity A string to contain the name of the entity.

<cmb:attributes entity="*entity*" schema="*schema*">attribute ... </cmb:attributes>

This tag iterates through the federated attributes of a federated entity.

schema Specify the name of a variable of type CMBSchemaManagement to contain the schema.

entity Specify the name of the entity.

attribute
A string to hold the name of a variable of type CMBAttribute that contains the attribute.

Search related tags

<cmb:searchresults searchresults="*searchResults*">item ... </cmb:searchresults>
This tag iterates through the search results.

searchResults
Specify the name of a variable of type CMBSearchResults that contains the search results.

item A string to contain the name of a variable of type CMBItem to contain the item resulting from the search.

Item related tags

<cmb:itemattributes item="*item*"> attrname ... attrtype... attrvalue... </cmb:itemattributes>
This tag iterates through the attributes of an item.

item Specify the name of a variable of type CMBItem that contains the item.

attrname
A string variable to contain the name of the attribute.

attrtype
A string variable to contain the attribute type.

attrvalue
A string variable to contain the value of the attribute.

<cmb:itemcontents " data="*data*" item="*item*"> content... </cmb:itemcontents>
This tag iterates through the contents of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

content
A variable of type CMBObject for the item's contents.

<cmb:itemnotelogs " data="*data*" item="*item*">notelog ... </cmb:itemnotelogs>
This tag iterates through the note logs of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

notelog
A variable of type CMBObject to contain the item's note log.

<cmb:itemprivileges data="*data*" item="*item*">privilege ... </cmb:itemprivileges>
This tag iterates through the privileges of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

privilege
A variable of type CMBPrivilege to contain the item′s privilege.

<cmb:itemresources data="*datamanagement*" **item=**"*item*"**> resource... </cmb:itemresources>**
This tag iterates through the resources of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

resource
A variable of type CMBResources to contain the item′s resource.

<cmb:unmappeditem data="*data*" **item=**"*item*"**>unmappeditem...</cmb:unmappeditem>**
This tag returns an unmapped item from the given mapped item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the mapped item.

unmappeditem
A variable of type CMBItem to contain the unmapped item.

<cmb:viewdata data="*data* " **item=**"*item*"**>viewdata... </cmb:viewdata>**
This tag returns a view of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

viewdata
A variable of type CMBViewData to contain the viewable data.

Folder related tags

<cmb:folderitems folder="*folder*"**> item... </cmb:folderitems>**
This tag iterates through the contents of a folder.

folder Specify the name of a variable of type CMBItem to contain the folder contents.

item A variable of type CMBItem that represents the folder.

Document related tags

<cmb:viewerdocuments docservices="*docservices*"**>document ... </cmb:viewerdocuments>**
This tag iterates through the documents that are currently loaded.

docservices
Specify the name of a variable of type CMBDocumentServices.

document
A variable of type CMBDocument to contain the document.

<cmb:documentpages document="*document*"**> docpage... </cmb:documentpages>**
This tag iterates through the pages of a document.

document
: Specify the name of a variable of type CMBDocument to contain the document.

docPage
: A variable of type CMBPage to contain the page.

EIP controller servlet

EIP provides a servlet with pluggable actions that can be used when building Web applications. This servlet acts as a controller of a model-view-controller design Web application, performing actions and initializing the beans (the model) which are then accessed in the JSP's (the views) either directly or indirectly by using the JSP tags.

Actions are provided for typical application tasks:

- Log on and log off.
- Search.
- Create, retrieve, modify, and delete documents.
- Create folders, and add documents to or remove documents from folders.
- Launch documents and document pages for viewing.

In addition, the servlet performs common tasks before and after the action, such as management of the connection to the content server. After every action, a JSP is invoked to format the results and send them back to the browser.

You can customize the servlet to add new actions and associate JSPs with the actions.

What the servlet can do

Here are some of the aspects of the servlet that you can use:

Connection pooling

The controller servlet uses EIP connection pooling to provide high performance connection management. The time in which a connection is allocated to a session may be either for the request or for the time the session is logged on. Currently connection pooling is at the application scope.

Logon of Timed-Out Sessions
If a session has timed-out, and a request comes into the servlet, the logon JSP is displayed, allowing the user to logon again. The original request is performed after successful logon.

Clean up on session termination
The servlet cleans up the session properly when a session is terminated, either by logging off or by a time-out. This means that the connection is destroyed or returned to the pool. All other EIP beans created by the servlet are terminated and their resources are freed without waiting for a garbage collection cycle to occur.

Locale The servlet insures that the locale is set correctly on the underlying beans, so messages and character strings are locale sensitive.

Using different JSP sets
A properties file, named `cmbservletjsp.properties`, by default, describes

the JSPs to use for responses to servlet actions. The location of the properties file is an application parameter. Therefore, several different web applications could be written using different sets of JSPs.

Extending the servlet

All actions known to the servlet are defined in a properties file named `cmbservlet.properties` (default). You can add, modify, or delete servlet actions by changing this file. To add a new action, follow these steps:

1. Implement a class to perform the action. The class must extend `com.ibm.mm.servlets.CMBServletAction`.
2. Add the name of the class and the action name to the cmbservlet.properties file. This has the following syntax:

   ```
   actions = list of actionsaction.<action_name>.class = class_name
   ```

   `actions` lists the actions understood by the servlet. For each action, a line within the properties file defines the class for the action. For example, to add an action named `replay`, in a class named `ReplayAction`:

   ```
   actions =... replay
   action.replay.class = ReplayAction
   ```

   You can also replace an action, or provide your own action to precede or follow any predefined action. For example, to precede logon with your own action, to perform additional validation:

   ```
   action.logon.class = MyLogonAction com.ibm.mm.servlets.CMBLogonAction
   ```

   The naming convention used for all predefined actions is `com.ibm.mm.servlets.CMB`*`action`*`Action`, where *action* is the name of the action, with the first letter in uppercase.

Servlet reference

You use a set of application parameters, request parameters, and a properties file to use the controller servlet in your applications.

Conventions

The servlet defines the following session and request values, which can be used in other JSP's or servlets. These conventions are followed by the JSP tag library. These conventions are the same as those for the EIP tag library.

Application parameters

The servlet understands the following application parameters (an alternative is to place these in the cmbservlet.properties file).

Application parameter	Values	Description
servletPropertiesURL	URL	The location of the cmbservlet.properties file
defaultServerType	Fed, ICM, OD, DL, DES, V4, IP, DD, ...	Default logon information. This, along with defaultServer, defaultUserid, and defaultPassword can be used in situations of shared user ID. Rather than prompting with a login page, the default logon information will be used to perform the logon.
defaultServer		Default logon information.
defaultUserid		Default logon information.

Application parameter	Values	Description
defaultPassword		Default logon information.
connectionpool	Boolean: true | false	To enable connection pooling
maxfreeconnection	integer	Maximum number of connections available in a connection pool.
minfreeconnection	integer	Minimum number of connection available in a connection pool
timeout	integer	The time duration (in milliseconds) after which a free connection will be disconnected and destroyed.
noSessionPage	URL	This is a page to display for logon, if the servlet is invoked without an established session or connection. This can be used to prompt for logon and chain back to the original action, allowing bookmarked links into EIP to work even if the user must log on.
timedOutPage	URL	This is a page to display if the session has timed out due to inactivity.
serverErrorPage	URL	This is a page to display if an error has occurred in accessing a server.
connectFailedPage	URL	This is a page to display if an error has occurred in connecting to a server. A prompt could be displayed to enter the correct userid/password for the server and retry can be performed.
tracelevel	0, 1, *or* 2	To indicate the level of tracing, as follows: • **0** - log nothing • **1** - log exceptions (default) • **2** -log exceptions, alert messages, WebSphere Application Server headers and attributes, EIP ini files, JVM system properties, EIP internal trace information
connectiontype	0, 1, *or* 2	The location of the EIP database and content server runtimes: • **0** - local (default) • **1** - remote • **2** -dynamic
cmbclient	URL	Location of `cmbclient.ini`
cmbcs	URL	Location of `cmbcs.ini`
serviceconnectiontype	0, 1, *or* 2	Location of services runtimes • **0** - local (default) • **1** - remote • **2** -dynamic
cmbsvclient	URL	Location of `cmbsvclient.ini`
cmbsvcs	URL	Location of `cmbsvcs.ini`
cmbcc2mime	URL	Location of `cmbcc2mime.ini`
cachedir	name of a directory	Directory to cache documents during document conversion

Application parameter	Values	Description
jnitrace	name of a file	File to which to write the JNI trace information for the JNI logic used in document conversion (for IBM diagnostic purposes)
conversion	Boolean: true *or* false	If `true`, documents are converted to formats that can be displayed in a browser on middle tier, if possible. If `false`, the original document, unconverted, is sent to the browser.
maxresults	integer	Maximum hits returned; -1 (default) means all hits.
valuedelimiter	character	Defines the character that will delimit values in search criteria. The default is locale dependent and is comma (`,`) for US English.
conversion.<mimetype>	<none | document | page >	Conversion options for viewing documents of a specific mimetype. This affects the behavior of the viewDocument servlet. Page means attempt to paginate the document. Document means convert the document to a form readable in a browser. None means perform no conversion -- return the document in its native form.
nameseparator	character	Defines the character that will separate child component attribute from the parent component attribute in qualified names. The default is locale dependent, and is a forward slash (`/`) for US English.

Properties File

The servlet looks for a properties file, `cmbservlet.properties`. This file defines the actions that the servlet can use, including the actions defined here. It also defines the names of the JSP files that are used.

You can also define the servlet properties on the Web application server (servlet engine). The syntax is the same as used in the file.

The content of cmbservlet.properties is stored in a Properties object by the control servlet. It can be accessed through the application attribute ″cmbServletProperties,″ as shown in the following example.

```
// check to see if connection pooling is enabled
String name = "connectionpool";
Properties props = (Properties) application.getAttribute
    ("cmbServletProperties");
String value = props.getProperty(name);
// "true" if enabled, "false" otherwise
```

Request parameters

The servlet understands the following request parameters. Additional parameters can be specified, for use in the reply JSP.

General

action=*action*

The action to be performed. Additional parameters allowed are based on the action and are described below.

This parameter is optional. If it is not specified, the reply page will be executed by the servlet, after performing standard setup, such as checking the connection for time-out and logon.

reply=<URL>

Optional. Forwards to the JSP specified in this parameter rather than the JSP defined as the reply for the action in the cmbservlet.properties file. If the action parameter is not specified, and reply is specified, the reply page is executed by the controller servlet after performing standard setup, such as checking the connection for timeout and logon.

Connection Related

action=logon serverType=<> server=<> userid=<> password=<> [connstring=<>] [configstring=<>]

Login to a server. You must specify the server type, server name, user ID, and password. The connect string and init string are optional and are different depending on the type of server.

action=logoff [endSession=<true|false>]

Logout of the server. The session is also ended by default.

Search related

action=searchTemplate template=<> {<criterianame>.op=<> <criterianame>=<>}

Perform™ a search using the specified search template and criteria values.

action=searchEntity entity=<> {attribute.<*attrname*>.op=<> attribute.<*attrname*>=<>} [conjunction=<and|or>]

Perform a search using an entity. The attributes values and operators can also be specified. Multiple values in the attribute value are separated with the value delimiter as specified in the application parameters. The attributes are combined together to form a query using `and` (default) or `or`, as specified by the conjunction parameter.

action=searchQuery queryString=<> {queryParameter.<parametername>=<>}

Perform a search using the specified query string. The query syntax depends on the server being searched.

Any number of additional query parameters may be specified. These are also server dependent.

Item related

action=lock itemId=<>

Lock an item, typically for exclusive access while updating.

action=unlock itemId=<>

Unlock a locked item.

action=createItem type=<document|folder> entity=<> {attribute.<*attrname*>=<*attrvalue*>}

Create an item. If posted, content may be provided.

action=retrieveItem itemId=<>

Retrieve an item's attributes and content. This is useful to insure the latest content as stored on the server is used.

action=updateItem itemId=<> [entity=<>] {attribute.<*attrname*>=<*attrvalue*>}

Update the attributes of an item. If entity is specified, the item is reindexed. Content is also updated if the servlet is invoked through a post.

action=deleteItem itemId=<>

Delete an item.

action=addContent itemId=<>

Add a content part to an item. The content data is posted.

action=getContent itemId=<> contentIndex=<>

Gets the content part and returns it to the browser.

action=updateContent itemId=<> contentIndex=<>

Update a content part on an item; the content data is posted. If no content exists, a content part is added.

action=deleteContent itemId=<> contentIndex=<>

Delete a content part for the specified item.

action=addNoteLog itemId=<>

Modifies the notelog on an item. The text of the notelog is posted.

action=updateNoteLog itemId=<> notelogIndex=<>

Modifies the note log of an item; the text of the note log is posted. If a note log does not exist, it is added.

action=deleteNoteLog itemId=<> notelogIndex=<>

Deletes the note log text of an item. The text of the note log is posted.

Folder related

action=addItemToFolder itemId=<> folderId=<>

Adds the specified item to the specified folder.

action=removeItemFromFolder itemId=<> folderId=<>

Removes the specified item from the specified folder.

Document related

action=viewDocument itemId=<>

Retrieves the document and views it. If the document is paginated, this action forwards to a JSP which generates the viewer frameset. If the document is not paginated, this action returns the actual content of the document.

action=viewPage itemId=<> page=<> scale=<> rotation=<> annotations=<yes|no>

Retrieves a page of the document.

Servlet toolkit function matrix

Table 33. Servlet function matrix

Action	CMv8	CMv7	VI/400	IP/390	OD/Wkstn	OD/390	DD
logon	Y	Y	Y	Y	Y	Y	Y

Table 33. Servlet function matrix (continued)

Action	CMv8	CMv7	VI/400	IP/390	OD/Wkstn	OD/390	DD
logoff	Y	Y	Y	Y	Y	Y	Y
searchTemplate	N/A	N/A	N/A	N/A	Y	Y	N/A
searchEntity	Y	Y	Y	Y	Y	Y	N/A
searchQuery	Y	Y	Y	Y	Y	Y	Y
lock	Y	Y	Y	Y	N/A	N/A	N/A
unlock	Y	Y	Y	Y	N/A	N/A	N/A
createItem	Y	Y	N/A	N/A	N/A	N/A	N/A
retrieveItem	Y	Y	Y	Y	Y	Y	Y
updateItem	Y	Y	Y	Y	N/A	N/A	N/A
deleteItem	Y	Y	Y	Y	N/A	N/A	N/A
addContent	Y	Y	N/A	N/A	N/A	N/A	N/A
getContent	Y	Y	Y	Y	Y	Y	Y
updateContent	Y	Y	Y	Y	N/A	N/A	N/A
deleteContent	Y	Y	Y	Y	N/A	N/A	N/A
addNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
updateNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
deleteNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
addItemToFolder	Y	Y	Y	N/A	N/A	N/A	N/A
removeItemFromFolder	Y	Y	Y	N/A	N/A	N/A	N/A
viewDocument	Y	Y	Y	Y	Y	Y	Y
viewPage	Y	Y	Y	Y	Y	Y	Y

Y -- Function is supported **N/A** -- Function is not supported **Note:** The logon, logoff, getContent, retrieveitem,viewDocument and viewPage actions are supported on all the content servers.

Table 34. Servlet function matrix Continued

Action	DES	DB2	JDBC	IC	Fed	FileNet
logon	Y	Y	Y	Y	Y	Y
logoff	Y	Y	Y	Y	Y	Y
searchTemplate	N/A	N/A	N/A	N/A	Y	N/A
searchEntity	N/A	Y	Y	N/A	Y	N/A
searchQuery	Y	Y	Y	Y	Y	Y
lock	N/A	N/A	N/A	N/A	Y	N/A
unlock	N/A	N/A	N/A	N/A	Y	N/A
createItem	N/A	Y	Y	N/A	N/A	N/A
retrieveItem	Y	Y	Y	Y	Y	Y
updateItem	N/A	Y	Y	N/A	Y	N/A
deleteItem	N/A	Y	Y	N/A	Y	N/A
addContent	N/A	N/A	N/A	N/A	Y	N/A
getContent	Y	Y	Y	Y	Y	Y
updateContent	N/A	N/A	N/A	N/A	Y	N/A

Table 34. Servlet function matrix Continued (continued)

Action	DES	DB2	JDBC	IC	Fed	FileNet
deleteContent	N/A	N/A	N/A	N/A	Y	N/A
addNoteLog	N/A	N/A	N/A	N/A	Y	N/A
updateNoteLog	N/A	N/A	N/A	N/A	Y	N/A
deleteNoteLog	N/A	N/A	N/A	N/A	Y	N/A
addItemToFolder	N/A	N/A	N/A	N/A	N/A	N/A
removeItemFromFolder	N/A	N/A	N/A	N/A	N/A	N/A
viewDocument	Y	Y	Y	Y	Y	Y
viewPage	Y	Y	Y	Y	Y	Y

Y -- Function is supported **N/A** -- Function is not supported **Note:** The logon, logoff, getContent, retrieveitem,viewDocument and viewPage actions are supported on all the content servers.

Working with information mining

This section begins with a description of how to build an application using the Information Mining beans, followed by sections on building your own content provider, using the Information Mining service API, and working with the JSP sample.

Building an information mining application using the beans

Information Mining enables you to create powerful applications based on state-of-the-art text analysis and mining technology. The Information Mining beans offer:

- Text summarization and categorization, that is, assign a summary to a document
- Categorization, that is, assign a category to a document
- Information extraction, that is, find and extract relevant information units from a document
- Language identification, that is, determine the language a document is written in
- Clustering, that is, group similar documents in a document collection
- Text search that can be restricted to documents of a certain category
- Access to documents that are continually retrieved from the Web using the IBM Web Crawler

Note

Before using the information mining beans, it is important to understand the differences between the service API and the JavaBeans.

- The Information Mining JavaBeans are software components for rapid application development and are JavaBeans conform. Certain elements of the code are ’bolted together’ and cannot be changed by the user.
- The Java service API contains the full Information Mining functionality as individual building blocks for creating applications. For more information, refer to “Using the service API” on page 483.

Before you can use the Information Mining beans, you must install and configure the Enterprise Information Portal Information Mining component on the machine that will run the beans.

The Information Mining beans are:

- **CMBSummarizationService**

  This is a non-visual bean that can be used to create summaries for documents. It can be run in three different modes and you can determine the length of the summary.
- **CMBCategorizationService**

  This is a non-visual bean that can be used to determine document categories. It runs on the results obtained using the Information Structuring Tool, namely the created taxonomy and metadata constituting terms that have been identified as being distinctive of the category and rules that describe each category. A document can be assigned to one or more categories. The categorization result consists of one or more categories and confidence values which shows how well the document fits each category. You can set this confidence value as well as the maximum number of results to be returned.

- **CMBInformationExtractionService**

  This is a non-visual bean that can be used to extract names, terms, and expressions from a document.

- **CMBLanguageIdentificationService**

  This is a non-visual bean that can be used to determine the language in which a document is written.

- **CMBClusteringService**

  This is a non-visual bean that can be used to partition a document collection into similar sets of documents or clusters.

- **CMBAdvancedSearchService**

  The CMBAdvancedSearchService is a non-visual bean that performs a search. An advanced search can only find items that have been stored in the Information Mining data store.

- **CMBCatalogService**

  The CMBCatalogService maintains the persistent Information Mining data store. In order to use this service, you need to create a catalog using the Information Structuring Tool. You can use the bean to store results created by any of the Information Mining services, and of course to look up and delete this information.

- **CMBWebCrawlerService**

  The CMBWebCrawlerService monitors the Web space, that is, a directory where the Web crawler, that is running asynchronously, places all crawled HTML documents. CMBItems (an EIP helper class that represents a document) are created from these files that can then be used with any of the Information Mining services, or simply imported into the Information Mining data store using the CMBCatalogService.

- **CMBInfoMiningAdapter**

  The CMBInfoMiningAdapter is a switch between different Information Mining events. In addition, it supports the CMBResultEvent, which enables the Information Mining beans to work together with other Enterprise Information Portal beans. The CMBInfoMiningAdapter allows you to combine the beans to build various scenarios, as shown in the following diagram.

Figure 39 on page 443 demonstrates the use of the Information Mining beans.

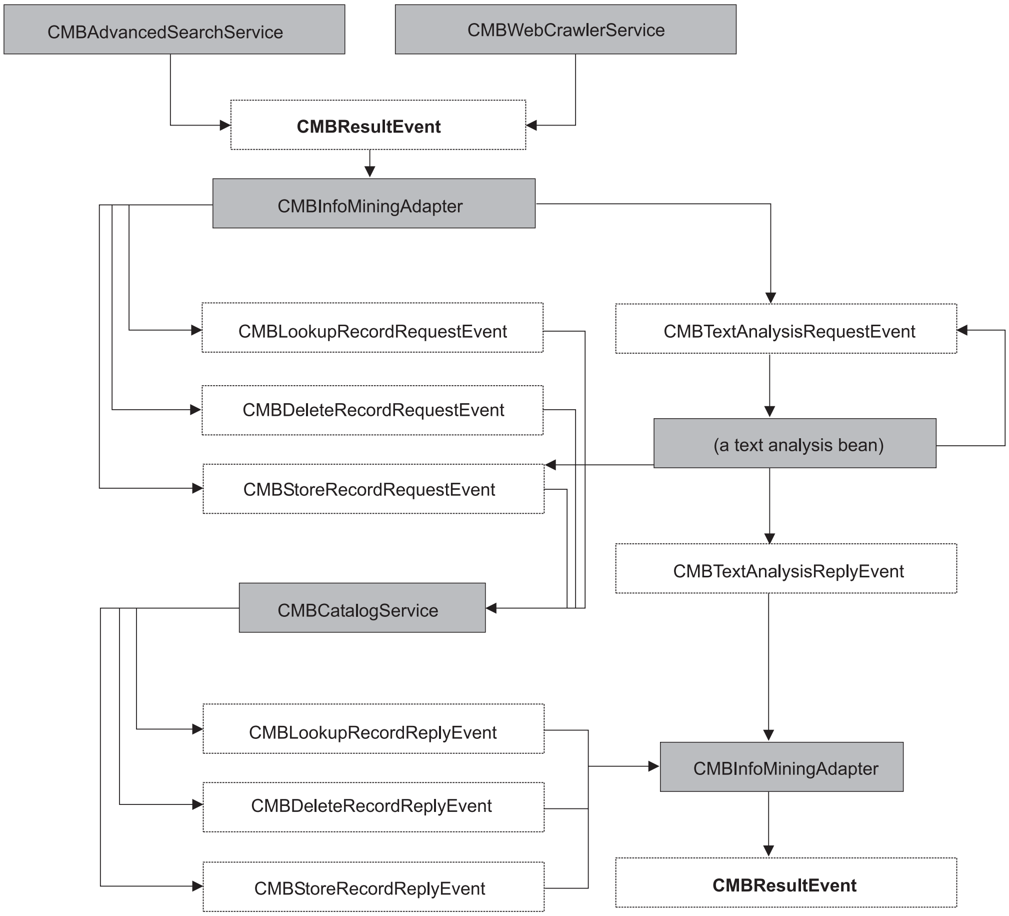

CMBAdvancedSearchService
CMBWebCrawlerService
CMBResultEvent
CMBInfoMiningAdapter
CMBLookupRecordRequestEvent
CMBTextAnalysisRequestEvent
CMBDeleteRecordRequestEvent
(a text analysis bean)
CMBStoreRecordRequestEvent
CMBTextAnalysisReplyEvent
CMBCatalogService
CMBLookupRecordReplyEvent
CMBInfoMiningAdapter
CMBDeleteRecordReplyEvent
CMBStoreRecordReplyEvent
CMBResultEvent

Figure 39. The Information Mining beans

Figure 40 lists the text analysis beans.

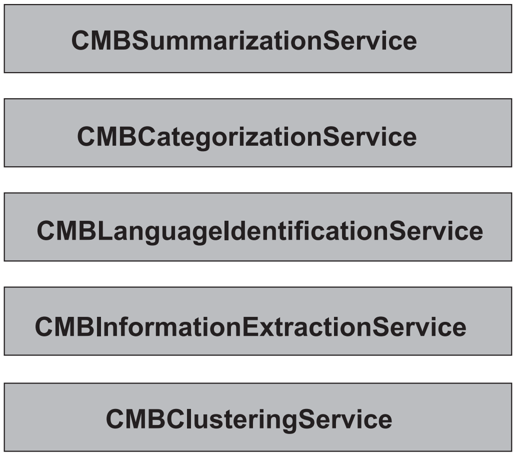


Figure 40. The text analysis beans

The following examples demonstrate how you can use the information mining beans to build applications:

- **Categorization sample:** Gathering information in preparation for Information Mining is a typical step for a librarian. See Figure 41 on page 445. In this sample, you begin by making a standard Enterprise Information Portal (EIP) search to find documents in the EIP content server. (Gathering documents from the Web is in the Web Crawler sample.) Then the language of the documents is determined and the English documents are categorized. The category information is stored in the metadata store.

- **The summarization sample:** This is another typical step for a librarian. As in the categorization sample, you begin by making a standard EIP search to find documents in the EIP content server, identify the English documents, and then summarize these documents, storing the summaries in the metadata store. See Figure 43 on page 453.
- **The information extraction sample:** This is an information mining step. In this sample, you begin by making a standard EIP search to find documents in the EIP content server. Then names and terms extracted from the English documents. The extracted information is stored in the metadata store.

  This is shown in Figure 45 on page 458.
- **The clustering sample:** This is an information mining step. You begin by making a standard EIP search to find documents in the EIP content server. Then the English documents are determined and the clustering service is triggered manually. The results are displayed on the screen. Clustering information is not stored in the metadata store.

  This is shown in "Clustering" on page 464.
- **The Web Crawler sample:** Get documents using the Web Crawler and make the information available for search within the categories (the advanced search sample). This is also a librarian step, similar to the categorization and summarization sample, except that you gather the documents from the Internet or an intranet rather than from the EIP content server. To restore the category and summary information from the catalog to the search results, you retrieve them using the catalog service.

  See Figure 49 on page 469.
- **The advanced search sample:** This is an information mining step. You make an advanced search, which allows you to search for documents using a flexible query limited to specific categories. To restore the category and summary information from the catalog to the search results, you retrieve them using the catalog service.

Location of the sample files

The samples described in this chapter are provided in the following directories:

Sample	**Location**
Categorization application	`<CMBROOT>\samples\java\beans\infomining\categorization`
Summarization application	`<CMBROOT>\samples\java\beans\infomining\summarization`
Information extraction application	`<CMBROOT>\samples\java\beans\infomining\informationExtraction`
Clustering application	`<CMBROOT>\samples\java\beans\infomining\clustering`
Advanced search application	`<CMBROOT>\samples\java\beans\infomining\advancedSearch`
Web Crawler application	`<CMBROOT>\samples\java\beans\infomining\webcrawler`

Each of these directories contains a `compile.bat` file to enable you to compile the source code. The application sample directories also contain a `run.bat` file to enable you to run the sample applications.

Categorizing documents

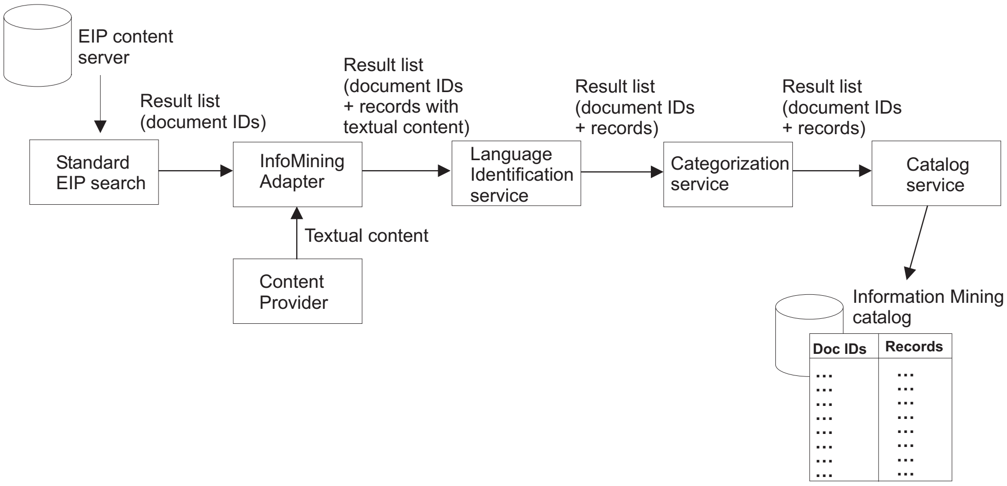

EIP content server
Result list (document IDs)
Result list (document IDs + records with textual content)
Result list (document IDs + records)
Result list (document IDs + records)
Standard EIP search
InfoMining Adapter
Language Identification service
Categorization service
Catalog service
Textual content
Content Provider
Information Mining catalog
Doc IDs
Records

Figure 41. The categorization sample

This sample demonstrates how you can categorize documents that have been retrieved by a standard EIP search. The analysis results are stored, and the appropriate documents are made available for search within the categories.

In Figure 41, a standard EIP search is made on documents held in the EIP content server. The language identification service bean determines the language in which the documents are written. The bean takes the document IDs to retrieve the document content and then analyses the content to determine the language. The categorization service bean categorizes the documents. This information, the document IDs and the category information, is then made available for further processing.

The following beans are used in this sample:

- CMBConnection
- CMBQueryService (to perform standard EIP search)
- CMBSearchResults (to perform standard EIP search)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBCategorizationService
- CMBCatalogService

For this sample, the application:

1. Creates the beans.
2. Customizes the beans.
3. The categorization service analyses the English documents. The results get stored in the catalog.
4. Runs the standard EIP query.
5. Displays the search results (lists of documents) and categories for verification.

The Java source is as follows.

Complete source for 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       = "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 Categorization() throws Throwable
  {
   // creating beans
   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();

   // reading parameters
   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');

// customizing beans
samplConnection.setServerName(sampDbName);
samplConnection.setUserid(sampUserName);
samplConnection.setPassword(sampPasswd);
samplConnection.setConnectionType
                (CMBConnection.CMB_CONNTYPE_DYNAMIC);
samplConnection.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);
```

```
  // connecting beans
  samplConnection.addCMBConnectionReplyListener(queryService);
  samplConnection.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);

  samplConnection.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);

  // running query
  samplConnection.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();
 }

// implementing 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();
     }
  }

 //implementing 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();
    }
  }
}
```

Creating the beans

You need a connection to a content server to perform a search. The connection can be established using the CMBConnection bean. The beans CMBQueryService and CMBSearchResults are required to perform a standard EIP search. The language of the documents is determined using the CMBLanguageIdentificationService and stored in the attribute IKF_LANGUAGE of the corresponding record. The documents are assigned to one or more categories using the CMBCategorizationService, which also stores the category information in the appropriate records. The CMBCatalogService is used to store the item's category information in the catalog and make the appropriate documents available for advanced search. The two adapters are required to convert search result events to text analysis request events and then store item reply events back to search result events.

The code that creates the required beans is:

```
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();
```

Customizing the beans

The code shown in the customizing section has to be adapted according to your installation. On the connection bean you need to specify the name of the content server to connect, a user ID, and the appropriate password.

Before you can run the categorization service bean and the catalog service bean, you have to associate them with an existing catalog. In addition, the catalog service bean needs a default category to which items are assigned that do not contain any category information.

The code that does the customization for the sample is:

```
samplConnection.setServerName(sampDbName);
samplConnection.setUserid(sampUserName);
samplConnection.setPassword(sampPasswd);
samplConnection.setConnectionType
          (CMBConnection.CMB_CONNTYPE_DYNAMIC);
samplConnection.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);
```

Connecting the beans

Figure 42 on page 451 illustrates the flow of events among the beans.

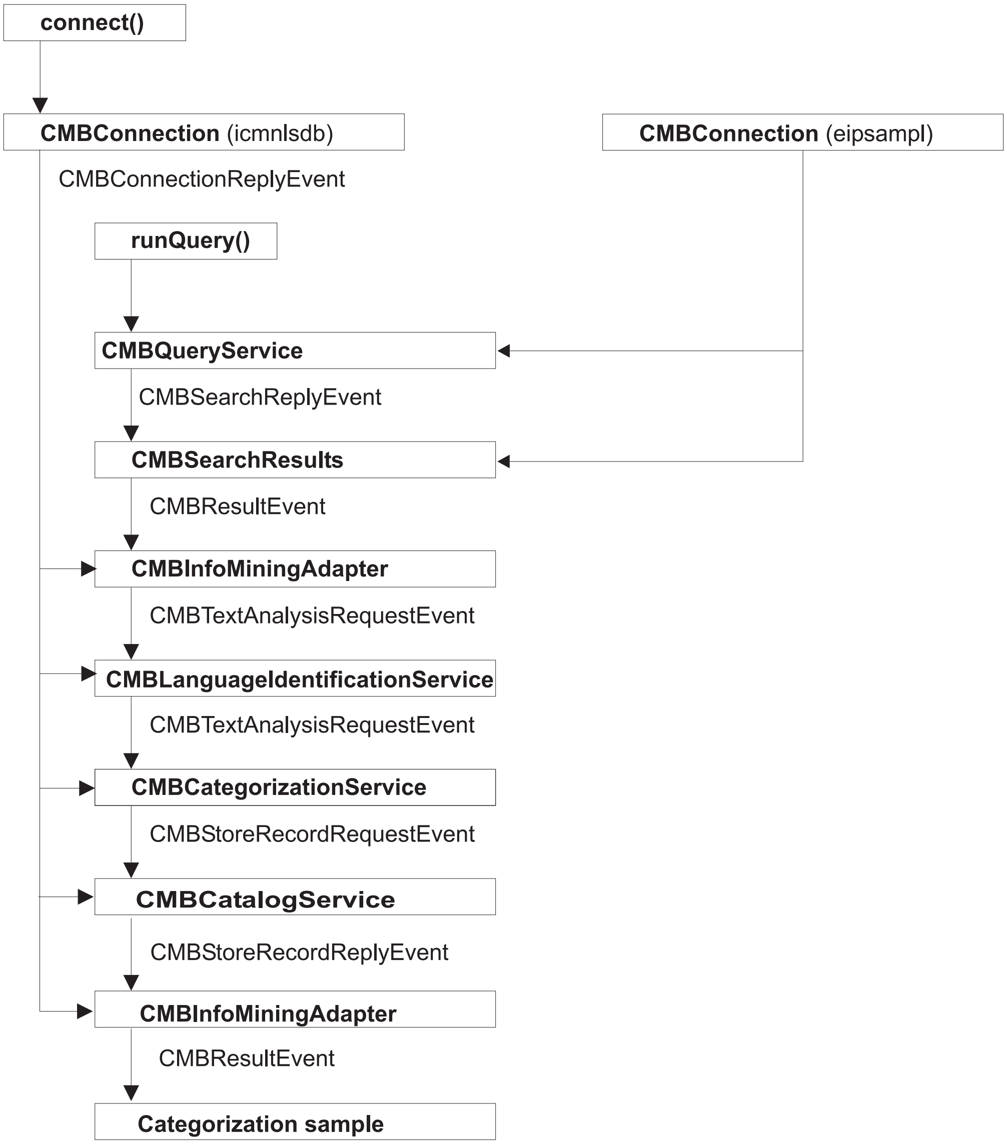


Figure 42. The categorization sample: Event flow

Five of the beans used in this sample listen to the CMBConnectionReplyEvent to get the connection handle. The CMBQueryService bean initiates a search that results in an event which then starts the event flow through the other beans.

The code that connects the beans is:

```
samplConnection.addCMBConnectionReplyListener(queryService);
samplConnection.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);

   samplConnection.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);
```

Because exceptions are also sent as events, the Categorization class has to handle the appropriate event by implementing the CMBExceptionListener interface and is connected to the beans to be notified about exceptions.

> **Tip**
>
> You can combine the use of the categorization service bean and the summarization service bean, one after the other, in any sequence, to analyze documents. Refer to “Importing documents from a web space” on page 469 for information on how to do this.

Running the query

Before you can run the query you need to establish the connection to the content server by calling the connect methods on the CMBConnection bean:

```
   samplConnection.connect();
   eipConnection.connect();
```

To start the standard EIP search, you need to specify a search template, a criterion of the template, the compare operator, and, of course, a search value.

You need to adapt the following code according to your configuration:

```
 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);
```

To close the current connection, call the disconnect methods:

```
   samplConnection.disconnect();
   eipConnection.disconnect();
```

Displaying text analysis results

The Categorization class implements the CMBResultListener interface to be able to list the documents that have been found during the search and to display the category information created for each document. The CMBResultEvent, received as argument to the CMBResult method, contains a vector of CMBItem objects where each CMBItem object represents a document:

```
   Vector cmbItemVector = (Vector)e.getData();
```

A CMBItem object encapsulates the PID of the document:

```
     CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);
     CMBRecord currentRecord = currentItem.getInfoMiningRecord();
     System.out.println("\n\nPID       : " + currentRecord.getPID());
```

as well as the results of text analysis:

```
System.out.println("Categories : " + currentRecord.getValue("IKF_CATEGORIES"));
```

Summarizing documents

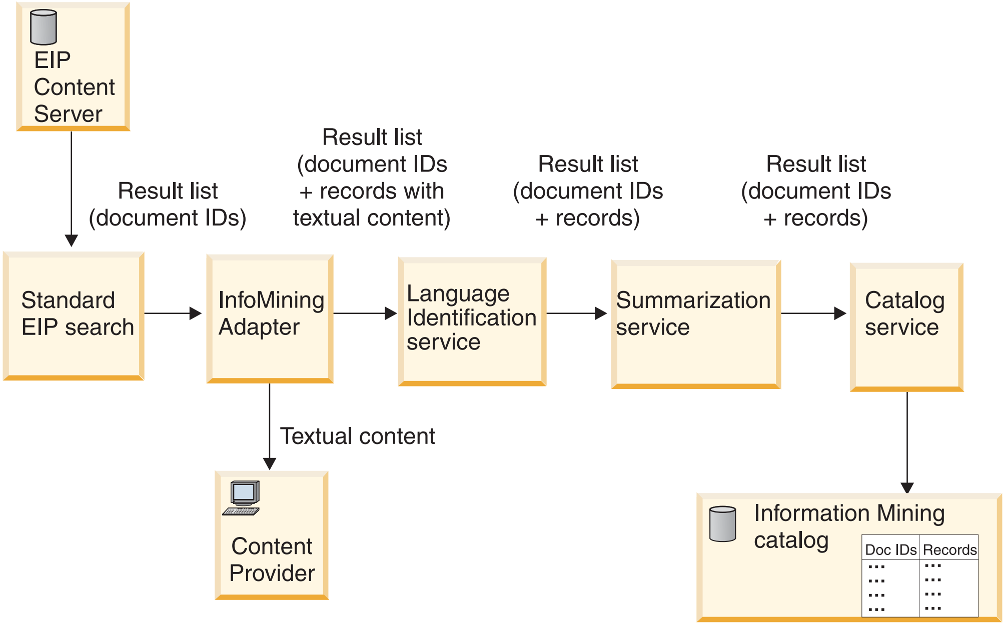

EIP Content Server
Result list (document IDs)
Result list (document IDs + records with textual content)
Result list (document IDs + records)
Result list (document IDs + records)
Standard EIP search
InfoMining Adapter
Language Identification service
Summarization service
Catalog service
Textual content
Content Provider
Information Mining catalog
Doc IDs
Records

Figure 43. The summarization sample

This sample demonstrates how you can summarize documents that have been retrieved by a standard EIP search. The analysis results are stored, and the appropriate documents are made available for advanced search.

In Figure 43, a standard EIP search is made on documents held in the EIP content server. The language identification service bean determines the language in which the documents are written. The bean takes the document IDs to retrieve the document content and then analyses the content to determine the language. The summarization service bean takes the English document IDs to retrieve the content of the found documents using the content provider and to create summaries of them. The catalog service bean then stores the summarization information in the metadata store. This information, the document IDs and the summaries, is then made available for further processing. The following beans are used in this sample:

- CMBConnection
- CMBQueryService (to perform standard EIP search)
- CMBSearchResults (to perform standard EIP search)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService

- CMBSummarizationService
- CMBCatalogService

Complete source for Summarization.java

Here is the complete source of the summarization sample. As the summarization and categorization beans operate in a similar fashion, see “Categorizing documents” on page 445 for further detail.

```
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       = "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 Summarization() throws Throwable
  {
   // creating beans
   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();

   // reading parameters
   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');

// customizing beans
samplConnection.setServerName(sampDbName);
samplConnection.setUserid(sampUserName);
samplConnection.setPassword(sampPasswd);
samplConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samplConnection.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);
```

```
  // connecting beans
  samplConnection.addCMBConnectionReplyListener(queryService);
  samplConnection.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);

  samplConnection.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);

  // running query
  samplConnection.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();
 }

// implementing 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();
     }
 }

 //implementing 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();
    }
  }
}
```

The following illustrates the flow of events among the beans.

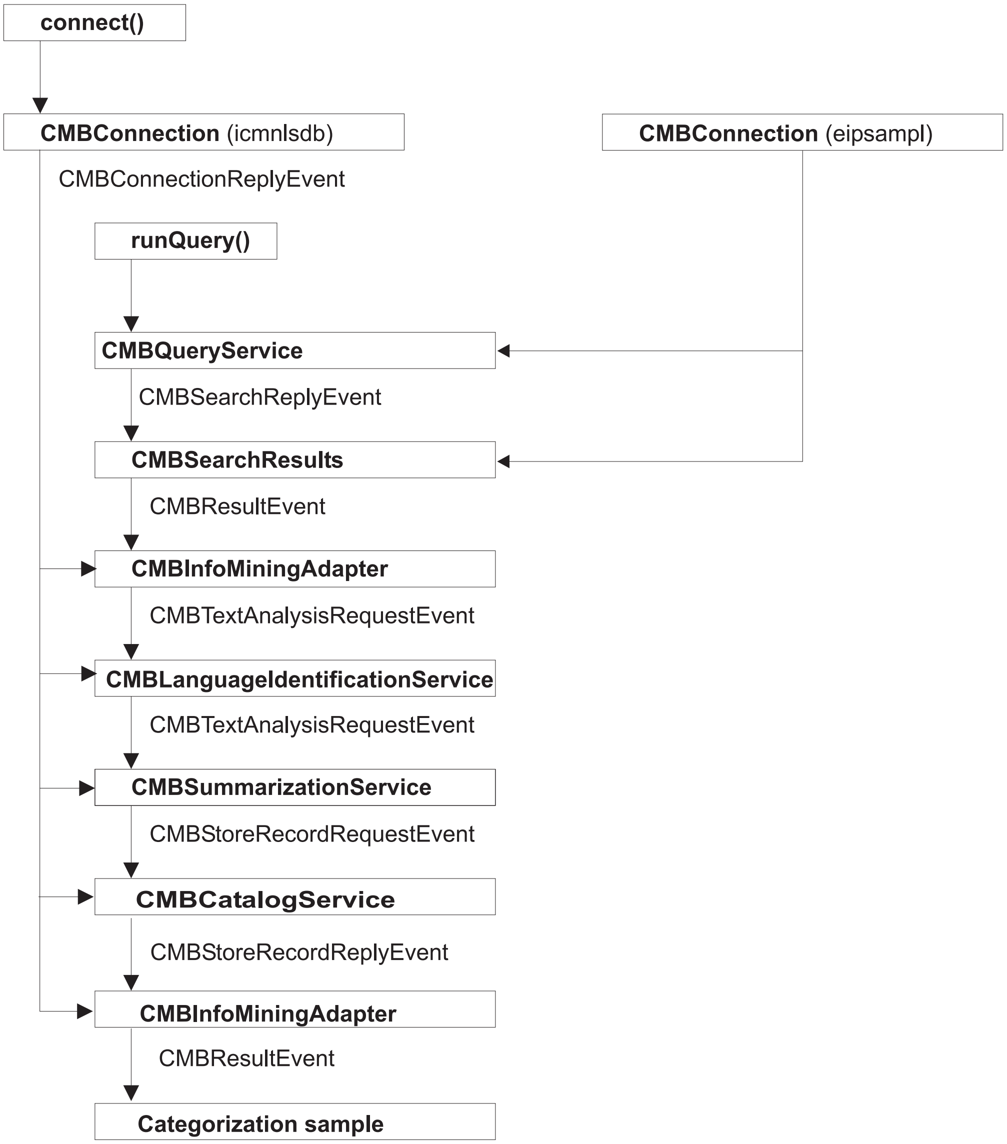


Figure 44. The summarization sample: Event flow

Extracting information

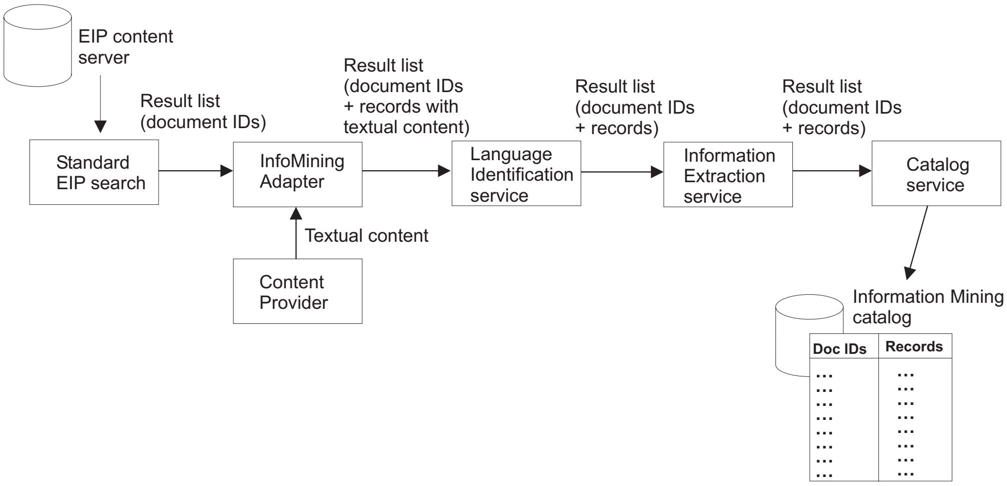

EIP content server
Result list (document IDs)
Result list (document IDs + records with textual content)
Result list (document IDs + records)
Result list (document IDs + records)
Standard EIP search
InfoMining Adapter
Language Identification service
Information Extraction service
Catalog service
Textual content
Content Provider
Information Mining catalog
Doc IDs
Records

Figure 45. The extracting information sample

This sample demonstrates how you can extract information, like names, terms and expressions, from documents that have been retrieved by a standard EIP search. The analysis results are stored, and can be used to retrieve related documents or extract important information from individual documents.

In Figure 45 on page 458, a standard EIP search is made on documents held in the EIP content server. The language identification service bean determines the language in which the documents are written. The bean takes the document IDs to retrieve the document content and then analyses the content to determine the language. The information extraction service bean takes the English documents, extracts information from these documents and stores the results in the metadata store. This information, the document IDs and the extracted information, is then made available for further processing.

The following beans are used in this sample:

- CMBConnection
- CMBQueryService (to perform standard EIP search)
- CMBSearchResults (to perform standard EIP search)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBInformationExtractionService
- CMBCatalogService

For this sample, the application:

1. Creates the beans.
2. Customizes the beans.
3. Connects the beans so that the language identification service can analyze the documents. The results get stored in the catalog.
4. The information extraction service analyzes the documents. The results get stored in the catalog.
5. Displays the search results (lists of documents) and extracted information for verification.

The Java source is as follows. As the information extraction and categorization beans operate in a similar fashion, see “Categorizing documents” on page 445 for further detail.

Complete source for 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
  {
   // creating beans
   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();

   // reading parameters
   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');

// customizing beans
samplConnection.setServerName(sampDbName);
samplConnection.setUserid(sampUserName);
samplConnection.setPassword(sampPasswd);
samplConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samplConnection.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);

// connecting beans
samplConnection.addCMBConnectionReplyListener(queryService);
samplConnection.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);
```

```
  samplConnection.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);

  // running query
  samplConnection.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();
 }

// implementing 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();
    }
 }

// implementing 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();
     }
  }
}
```

Figure 46 illustrates the flow of events among the beans.

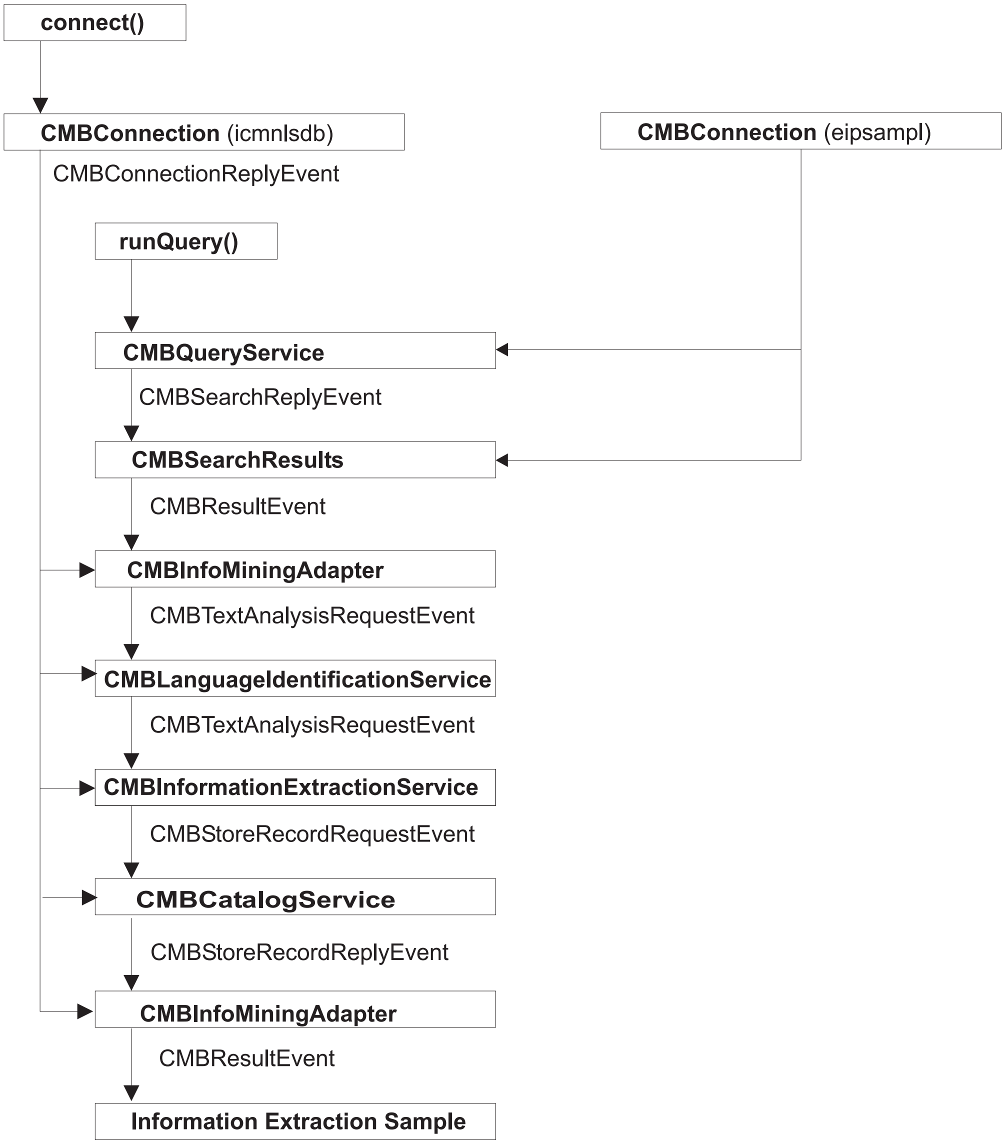


Figure 46. The information extraction sample: Event flow

Clustering

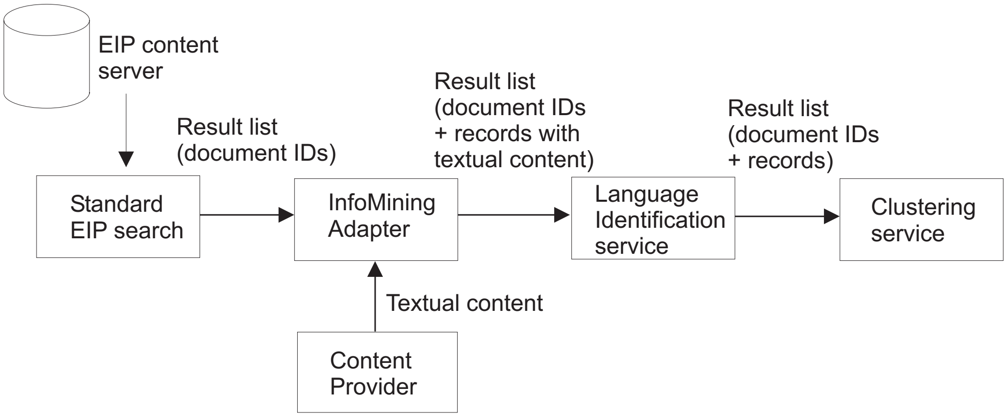

EIP content server
Result list (document IDs)
Result list (document IDs + records with textual content)
Result list (document IDs + records)
Standard EIP search
InfoMining Adapter
Language Identification service
Clustering service
Textual content
Content Provider

Figure 47. The clustering sample

This sample demonstrates how you can cluster documents that have been retrieved by a standard EIP search. Before clustering, it is necessary to identify the language of the documents. The clustering service only automatically collects documents returned from an EIP search. The actual clustering process has to be triggered manually by calling the cluster() method.

The following beans are used in this sample:
- CMBConnection
- CMBQueryService (to perform standard EIP search)
- CMBSearchResults (to perform standard EIP search)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBClusteringService

For this sample, the application:
1. Creates the beans.
2. Customizes the beans.
3. Connects the beans so that the Language Identification service can analyze the documents and identify the document language, and the Clustering service can collect the documents ready for subsequent clustering.
4. Triggers clustering by calling the cluster() method. The result (of class CMBClusterResult) is then printed out on the screen in a readable form.

The beans for the Clustering service are similar to the other Information Mining beans except that:
- The clustering service only collects documents returned from an EIP search. The actual clustering process has to be triggered manually.
- The catalog is not designated to store clustering results which means that the catalog service is not needed in this scenario.

Refer to “Categorizing documents” on page 445 for details on how the beans operate.

The Java source is as follows.

Complete source for 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       = "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 Clustering() throws Throwable
  {
   // creating beans
   CMBConnection samplConnection = new CMBConnection();
   CMBConnection eipConnection = new CMBConnection();
   CMBQueryService queryService = new CMBQueryService();
   CMBSearchResults searchResults = new CMBSearchResults();
   CMBLanguageIdentificationService languageIdentificationService =
                                new CMBLanguageIdentificationService();
   CMBClusteringService clusteringService = new CMBClusteringService();
   CMBInfoMiningAdapter adapter = new CMBInfoMiningAdapter();

   // reading parameters
   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');

// customizing beans
samplConnection.setServerName(sampDbName);
samplConnection.setUserid(sampUserName);
samplConnection.setPassword(sampPasswd);
samplConnection.setConnectionType(CMBConnection.CMB_CONNTYPE_DYNAMIC);
samplConnection.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);

// connecting beans
samplConnection.addCMBConnectionReplyListener(queryService);
samplConnection.addCMBConnectionReplyListener(searchResults);

eipConnection.addCMBConnectionReplyListener(adapter);
eipConnection.addCMBConnectionReplyListener(languageIdentificationService);
eipConnection.addCMBConnectionReplyListener(clusteringService);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter);
adapter.addCMBTextAnalysisRequestListener(languageIdentificationService);
```

```
languageIdentificationService.addCMBTextAnalysisRequestListener(clusteringService);

samplConnection.addCMBExceptionListener(this);
eipConnection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
clusteringService.addCMBExceptionListener(this);
adapter.addCMBExceptionListener(this);

// running query
samplConnection.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);

   // running clustering and printing results
   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]);
   }

   // disconnecting
   samplConnection.disconnect();
   eipConnection.disconnect();
  }

 // implementing com.ibm.mm.beans.CMBResultListener
 public void onCMBResult(CMBResultEvent e)
  {
   return;
  }

 // implementing 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();
    }
  }
}
```

Figure 48 illustrates the flow of events among the beans.

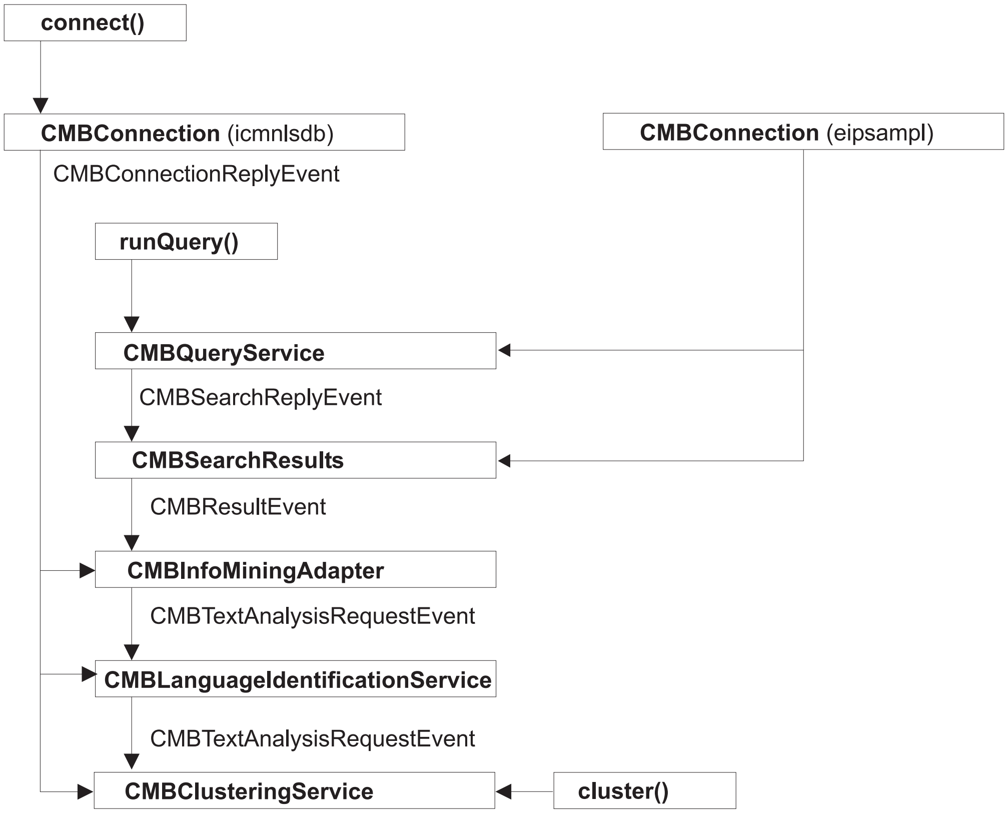


Figure 48. The clustering sample: Event flow

Importing documents from a web space

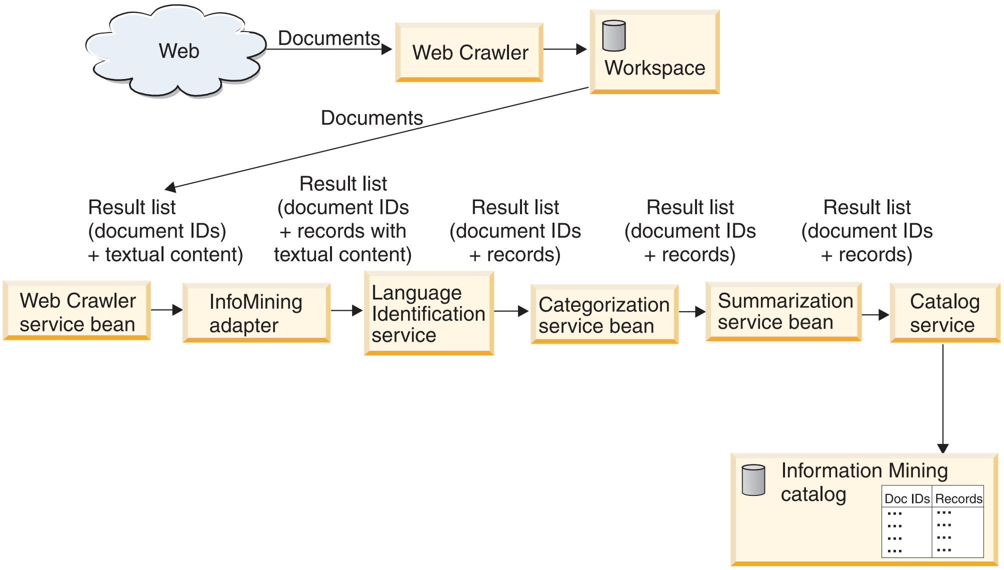

Web
Documents
Web Crawler
Workspace
Documents
Result list (document IDs) + textual content)
Result list (document IDs + records with textual content)
Result list (document IDs + records)
Result list (document IDs + records)
Result list (document IDs + records)
Web Crawler service bean
InfoMining adapter
Language Identification service
Categorization service bean
Summarization service bean
Catalog service
Information Mining catalog
Doc IDs
Records

Figure 49. The Web Crawler sample

This sample demonstrates how to import crawled documents into the Information Mining component and conduct text analysis (language identification, categorization and summarization) on the documents that have been imported. This can only be done if the Web Crawler has run or is running, and new or changed objects are saved within the specified Web space directory, defined by the `summaries-dir` attribute in the Web Crawler configuration.

When using the Web Crawler for information mining, make sure that you run it with the information mining configuration settings described in the section on managing information mining in *Managing Enterprise Information Portal*. These settings are part of a sample configuration file for the information mining component `im-crawler-config-sample.xml` located in the following directory:

- On Windows:

  `<CMBROOT>\samples\java\beans\infomining\webcrawler\`
- On UNIX (AIX and Solaris):

  `<CMBROOT>/samples/java/beans/infomining/webcrawler/`

Once the Web Crawler has placed a copy of the web-page documents that it has been monitoring into the specified Web space directory, you can use the CMBWebCrawlerService bean to import the crawled documents into Enterprise Information Portal Information Mining.

The Web Crawler and CMBWebCrawlerService bean can be run as persistent processes to monitor and import documents as they change. A CMBResultEvent can be triggered when the number of imported documents exceeds a certain value or when all monitored files have been imported. The CMBWebCrawlerService bean notifies all the attached listeners. The event contains the imported files as a vector of CMBItems.

Access rights for the CMBWebCrawlerService on AIX and Solaris. When importing documents using the CMBWebCrawlerService, the corresponding files are either deleted from the file system or, if the archiving option is turned on using

archiveEnabled, the files are moved from the disks directory to an archives directory. The archives directory is located at the same level as the disks directory (in `.../webspaces/ikf`) and has the same subdirectory structure as the disks directory. To use the CMBWebCrawlerService, make sure that the corresponding user ID has the following permissions:

- Write permission for the disks directory, and all subdirectories and files below this directory. The location of the disks directory is `.../webspaces/ikf/disks` as defined by `summaries-dir` in the Web Crawler configuration file. This permission is required to delete or move crawled documents and files.
- When archiving is turned on, write permission in directory `.../webspaces/ikf`, so that the archives directory can be created the first time archiving is used.
- When archiving is turned on, write permission in the archives directory and its subdirectories, and for all files, so that the files that are moved from the disks directory can be created in the archives directory.

Modifying the import operation. You can change some aspects of the import operation on the CMBWebCrawlerService bean. Initially, the program looks for crawled documents every 30 minutes, but you can change this value. It also throws a CMBResultEvent notification to all listeners after every 100 imported documents. You can change this to a different number of documents, or cause a notification to be thrown when all crawled files are imported to the Enterprise Information Portal set.

Properties of CMBWebCrawlerService Java bean:

pollCycles
Number of times to poll.

pollMinutes
Number of minutes to sleep before next poll. The default is 30.

rootDir
To crawl %IMY_WEBSPACE% on local host.

archiveEnabled
Keep the crawled files in *.../webspaces/ikf/archives*. The default is false which means that the CMBWebCrawlerService deletes the crawled files from `../webspaces/ikf/disks` during processing without backing them up somewhere else.

pageSize
Number of imported items until a CMBResultEvent is thrown to the CMBResultListeners.

webSpace
The Web space that is monitored by the Web Crawler.

As with the other text analysis beans, the results of the Web Crawler search bean can be made available for a subsequent advanced search.

The following beans are used in this sample:

- CMBConnection
- CMBWebCrawlerService
- CMBInfoMiningAdapter
- CMBLanguageIdentification
- CMBCategorizationService
- CMBSummarizationService

- CMBCatalogService

For this sample the application:

1. Creates the beans
2. Customizes the beans
3. Connects the beans
4. Starts the Web Crawler service
5. Displays the crawler results and text analysis results

An explanation of each of the preceding steps follows in the source for WebCrawler.java.

Complete source for 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      = ""; // set to the dir where the web spaces reside
 String WEBSPACE_NAME     = ""; // set this to the name of your web space

public WebCrawler() throws Throwable
  {
   // creating beans
   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();

   // reading parameters
   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("Webspace directory = " + webspaceDir);
System.out.println("Webspace 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');

// customizing beans
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);

// connecting beans
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);
```

```
   // running query
   connection.connect();
   catalogService.setDefaultCategoryPath
     (catalogService.getTaxonomy().getRootCategory().getPathAsString());

   crawlerService.start();
   connection.disconnect();
  }

 // implementing 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();
     }
  }

 //implementing 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();
    }
  }
}
```

Creating the beans

You build a connection to a content server for security reasons; information mining needs to know who is working with the system. This ensures that only those who are registered in the system are allowed to use it. The connection can be established using the CMBConnection bean. The bean CMBWebCrawlerService is used to import the previously crawled documents into the information mining component and to throw a CMBResultEvent notification. Language identification, summary and category information are created using the beans CMBLanguageIdentification, CMBSummarizationService and CMBCategorizationService. The two adapters convert result events to text analysis request events, and then store record reply events back to search result events.

The code that creates the beans is:

```
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();
```

Customizing the beans

The code shown in the customizing section has to be adapted according to your installation. You need to specify the name of the content server to connect, a user ID and the appropriate password.

On the Web Crawler service bean, the root directory of your local host, where the Web Crawler saved or changed the crawled objects within the Web space, must be specified together with the previously defined Web space name. Before you can run the text analysis service beans and the catalog service bean, you need to associate them with an existing catalog.

The code that does the customization for the sample is:

```
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);
```

Connecting the beans

Figure 50 on page 475 illustrates the flow of events among the beans in this sample.

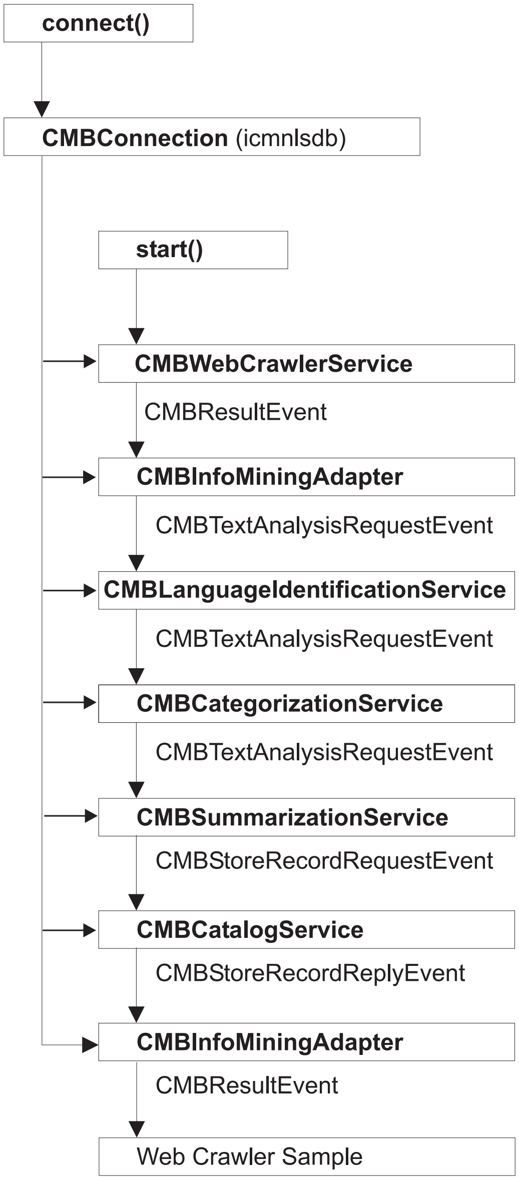


Figure 50. The Web Crawler sample: Event flow

Five of the beans used in this sample listen to the CMBConnectionReplyEvent to get the connection handle. The CMBWebCrawlerService initiates the crawl service that results in an event which then starts the event flow through the other beans.

The code that connects the beans is:

```
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);
```

Because exceptions are also sent as events, the WebCrawler class has to handle the appropriate event by implementing the CMBExceptionListener interface, and is connected to the beans to be notified about exceptions.

Starting the Web Crawler service

Before you can start the Web Crawler service, you need to establish the connection to the content server by calling the connect method on the CMBConnection bean:

```
connection.connect();
```

The code to start the Web Crawler service is:

```
catalogService.setDefaultCategoryPath
               (catalogService.getTaxonomy().getRootCategory().getPathAsString());
crawlerService.start();
```

To close the current connection, call the disconnect() method:

```
connection.disconnect();
```

Displaying text analysis results

The WebCrawler class implements the CMBResultListener interface to be able to list the documents that have been found during the search and to display the category and summary information created for each document. The CMBResultEvent, received as argument in the onCMBResult method, contains a vector of CMBItem objects, where each CMBItem object represents a document:

```
Vector cmbItemVector = (Vector)e.getData();
```

A CMBItem object encapsulates the PID of the document:

```
     CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

     CMBRecord currentRecord = currentItem.getInfoMiningRecord();

     System.out.println("\n\nPID       : " + currentRecord.getPID());
```

as well as the results of text analysis.

Then you get the summary and category information:

```
     System.out.println("Categories   : " +
                           currentRecord.getValue("IKF_CATEGORIES"));
     System.out.println("Summary    : " +
                           currentRecord.getValue("IKF_SUMMARY"));
```

Searching for documents by category

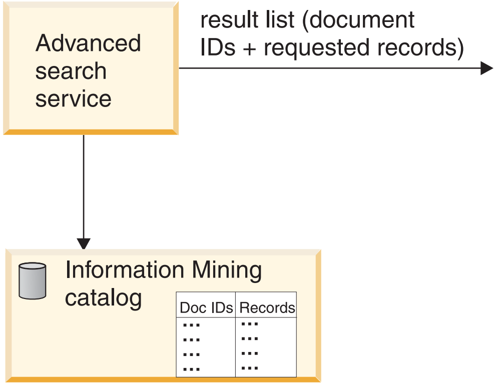

Advanced search service
result list (document IDs + requested records)
Information Mining catalog
Doc IDs
Records
...
...

Figure 51. The advanced search sample

This sample demonstrates how to run an advanced search and look up category information in the documents that have been found. An advanced search can only find documents that have been made available for that kind of search as in the categorization sample. See "Location of the sample files" on page 444. In contrast to the previous samples where a standard EIP search is carried out on the entire EIP content server, this sample searches metadata in the data store.

If you experience performance problems when submitting complex queries, refer to "Performance tuning" on page 492 for more information.

When the advanced search has produced a result list, the IDs of the found documents can be used by the catalog service bean to retrieve the metadata previously stored in the data store.

The following beans are used in this sample:

- CMBConnection
- CMBAdvancedSearchService
- CMBInfoMiningAdapter
- CMBCatalogService

For this sample the application:

1. Creates the beans
2. Customizes the beans
3. Connects the beans
4. Runs the advanced search
5. Displays the search and text analysis results for verification

An explanation of each of the preceding steps follows the source for AdvancedSearch.java.

Complete source for 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
  {
   // creating beans
   CMBConnection connection = new CMBConnection();
   CMBAdvancedSearchService advancedSearchService = new
                           CMBAdvancedSearchService();
   catalogService = new CMBCatalogService();

   // reading parameters
   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');

  // customizing beans
  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 + "'\")");

  // connecting beans
  connection.addCMBConnectionReplyListener(catalogService);
  connection.addCMBConnectionReplyListener(advancedSearchService);
  advancedSearchService.addCMBResultListener(this);

  connection.addCMBExceptionListener(this);
  advancedSearchService.addCMBExceptionListener(this);
  catalogService.addCMBExceptionListener(this);

  // running query
  connection.connect();
  advancedSearchService.runQuery();
  connection.disconnect();
 }

// implementing 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();
      }
  }

 // implementing 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();
    }
  }
}
```

Creating the beans

You need a connection to a content server to perform a search. The connection can be established using the CMBConnection bean. The CMBAdvancedSearchService bean is required to perform an advanced search. Category information can be retrieved using the CMBCatalogService bean.

The code that creates the beans is:

```
CMBConnection connection = new CMBConnection();
CMBAdvancedSearchService advancedSearchService =
   new CMBAdvancedSearchService();
catalogService = new CMBCatalogService();
```

Customizing the beans

The code shown in the customizing section has to be adapted according to your installation. You need to specify the name of the content server to connect, a user ID, and the appropriate password.

Before you can run the advanced search service and the catalog service bean, you need to associate them with an existing catalog.

The code that does the customization for the sample is:

```
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 + "'\")");
```

Connecting the beans

Figure 52 illustrates the flow of events among the beans in this sample.

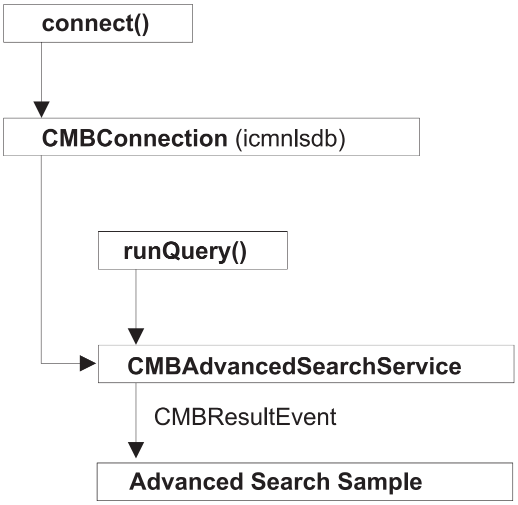


Figure 52. The advanced search sample: Event flow

Two of the beans used in this sample listen to the CMBConnectionReplyEvent to get the connection handle. The CMBAdvancedSearchService bean initiates a search that results in an event which then starts the event flow through the other beans.

The code that connects the beans is:

```
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(advancedSearchService);
advancedSearchService.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
advancedSearchService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
```

Because exceptions are also sent as events, the AdvancedSearch class has to handle the appropriate event by implementing the CMBExceptionListener interface and is connected to the beans to be notified about exceptions.

Running the query

Before you can run the query you need to establish the connection to the content server by calling the connect method on the CMBConnection bean:

```
connection.connect();
```

To start the advanced search, you need to specify a catalog, a search query string and the metadata to search in.

You start the advanced search:

```
advancedSearchService.runQuery();
```

To close the current connection, call the disconnect() method:

```
connection.disconnect();
```

Displaying text analysis results

The AdvancedSearch class implements the CMBResultListener interface to be able to list the documents that have been found during the search and to display the category information retrieved for each document. The CMBResultEvent, received as argument in the onCMBResult method, contains a vector of CMBItem objects where each CMBItem object represents a document:

```
   Vector cmbItemVector = (Vector)e.getData();
```

A CMBItem object encapsulates the PID of the document:

```
     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());
```

as well as the results of text analysis beans if there were some in the event flow.

Then you get the category information:

```
     System.out.println("Categories    :
                        " + currentRecord.getValue("IKF_CATEGORIES"));
```

The vector returned by the getCategories() method of class CMBItem contains objects of class CMBCategory that can be used to determine the absolute path to the current category.

Building your own content provider

You may want to build your own content provider when you need to apply proprietary filters to retrieve text from the binary part of a proprietary format.

This section describes how you can write your own content provider that can deal with a user-defined object model, or with proprietary formats inside the parts of a CMBItem. To support you in this work, Information Mining provides:

- Interface **CMBContentProvider** which defines the interface for classes that know how to determine the text to be used for text analysis.
- Method **setContentProvider(CMBContentProvider)** in CMBInfoMiningUtilities which sets a ContentProvider.

The CMBContentProvider interface defines one method `getContent()` which returns the text of the specified item to be used for text analysis. For example:

```
public CMBTextAnalysisDocument getContent(CMBConnection connection, CMBItem item )
throws CMBContentProviderException;
```

- Parameter `connection`: an open connection to the server.
- Parameter `item`: the current item to be processed.
- Exception `CMBContentProviderException`: if an error occurs while processing the current item.
- Returns the text as an object of class CMBTextAnalysisDocument.

To tell the system which ContentProvider to use, use the method `setContentProvider(CMBContentProvider)` in the `CMBInfoMiningUtilities` class to specify an object that has this interface. Here's an example:

```
CMBInfoMiningUtilities.setContentProvider
         (new MyCompaniesLatestGreatestContentProvider());
```

To develop your own content provider, you can use the sample content provider (SimpleContentProvider) as a starting point. The content provider sample is located at: `<CMBROOT>\samples\java\beans\infomining\contentprovider`

A default content provider is provided with Information Mining. It extends the CMBContentProvider interface to allow selecting individual parts for processing, and offers a mechanism for preventing processing objects that are too large for in-memory processing, such as video streams.

To register the default content provider, use:

```
CMBInfoMiningUtilities.setContentProvider(new CMBDefaultContentProvider());
```

Using the service API

The information mining service API is a Java API that integrates information mining functionality as an EIP service. Using the information mining service API you can write applications that can:

- Filter the textual content from different document formats, for example, pdf, Microsoft Word and HTML
- Perform text analysis on the text documents to create metadata, such as summarization and categorization
- Store metadata for a document in persistent records
- Search for records

Note

Before using the information mining beans, it is important to understand the differences between the service API and the JavaBeans.

- The JavaBeans provide a facility for rapid application development. Certain elements of the code are ’bolted together’ and cannot be changed by the user. For more information, refer to “Building an information mining application using the beans” on page 441.
- The information mining service API provides more flexibility for building information mining applications. The code can be seen more as individual ’building blocks’ which can be put together to meet your specific requirements.

Connecting to the information mining service API

Before you can use the information mining service API, you need to create a service object using class `DKIKFServiceFed`. Depending on what type of application you want, you have to import the class from one of the following three packages:

- If you want to create an application on a server host, use the following import statement: `import com.ibm.mm.sdk.server.DKIKFServiceFed;`
- If you want to create an application on the client host, use the following import statement: `import com.ibm.mm.sdk.client.DKIKFServiceFed;`
- If you want to remain flexible and create an application that can run on the server as well as the client, use the following import statement: `import com.ibm.mm.sdk.cs.DKIKFServiceFed;`

After using the appropriate import statement, create the `ikfservice` object by using:

```
DKIKFService ikfService = DKIKFServiceFed.create();
```

After creating the service object, you can connect to the database using name, user ID and password.

```
ikfService.connect("databaseName", "userID", "password", null);
```

You now have a connection to the information mining service API and can use classes from the following two packages:

- **com.ibm.mm.sdk.common.infomining** to manage library, catalog and record tasks.
- **com.ibm.mm.sdk.common.infomining.analysis** to use the different text analysis tools.

For an explanation of these key functions, see the following sections.

To disconnect, use:

```
ikfservice.disconnect();
```

Managing the library, taxonomies, and catalogs

The classes to manage the library, catalog, and taxonomy tasks are in the **com.ibm.mm.sdk.common.infomining** package.

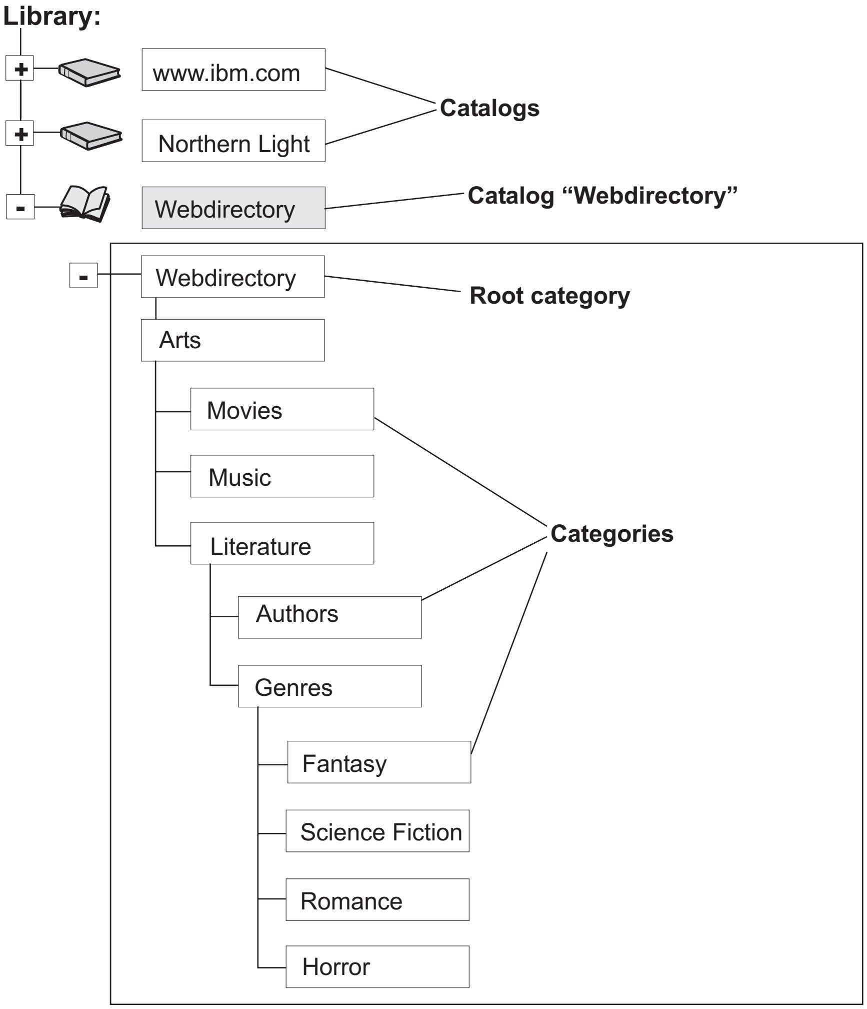


Figure 53. An example of a catalog

In the catalog example, the Library contains three catalogs namely, www.ibm.com, Northern Light, and Webdirectory. Webdirectory has the category tree structure shown in the diagram.

Catalogs can only be created by the Information Structuring Tool (IST).

Returning a library

To obtain a library object, use:

```
DKIKFLibrary library = ikfService.getLibrary();
```

Listing all the catalogs in the library

To get the names of all the catalogs in the library, use:

```
String[] catalogNames = library.getCatalogNames();
```

This returns an array listing all the catalog names. The example returns the following catalog names in no specific order: `Webdirectory`, `www.ibm.com` and `Northern Light`.

Returning a specific catalog

To obtain a specific catalog, use:

```
DKIKFCatalog catalog = library.getCatalog(catalogNames[0]);
```

This sample returns the catalog with the first name of the list returned in the previous step. In this case, the Webdirectory catalog returns.

Returning the taxonomy of a catalog

To get the taxonomy of a catalog, use:

```
DKIKFTaxonomy taxonomy = catalog.getTaxonomy();
```

The taxonomy object has various methods to work with the categories.

Note

The taxonomy object is read from the database, and all further actions are carried out on this copy.

Listing all the categories in a taxonomy

To get all the categories in a taxonomy, use:

```
DKIKFCategory[] categories = taxonomy.getCategories();
```

This returns an array listing all the categories, including the root category. The example returns the following categories in no specific order: Horror, Author, Music, Literature, Webdirectory, Genres, Science Fiction, Fantasy, Movies, Romance, and Arts.

Note

The category objects are read from the database, and all further actions are carried out on the copy.

Returning the root category

To get the root category, use:

```
DKIKFCategory rootCategory = taxonomy.getRootCategory();
```

This sample returns the root category from the list returned in the previous step. In this case, the Webdirectory root category returns.

Returning a specific category

To return a specific category, use:

```
DKIKFCategory category = taxonomy.getCategory("Webdirectory/Literature/Genres");
```

Using the example, the category Genres returns.

Also use the same separator character ″/″ that is currently set in the taxonomy object.

Note

Three getCategory methods are available in the taxonomy object, each requiring different parameters:

- The path is a string with category names separated using a specified character
- The path is an array of category names
- Another category object

See the *online API Reference* for further information.

Ensuring that the current taxonomy is the most recent

To ensure that the current taxonomy is the most recent, use:

```
if(taxonomy.getTimestamp().before(catalog.getTaxonomyLastModified()))
  {
  taxonomy = catalog.getTaxonomy();
  }
```

Note

The taxonomy object is read from the database, and all further actions are carried out on this copy. Therefore, ensure that the taxonomy object is the most recent.

Returning the children of a category

To return the children of a category, use:

```
DKIKFCategory[] children = category.getChildren();
```

This returns an array listing all the children of the category. The example based on `Genres` category, returns the following children in no specific order: `Fantasy`, `Science Fiction`, `Romance`, and `Horror`.

Note

There can be multiple children for a category, but does not return any further sub-categories from these children.

Returning the parent of a category

To return the parent of a category, use:

```
DKIKFCategory parent = category.getParent();
```

The example based on the `Genres` category, only returns the `Literature` category.

Returning a specific catalog schema and list all the attributes in the schema

To get the specific catalog schema, use:

```
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());
  }
```

The example returns the schema of the `Webdirectory` catalog.

You can then return the keys (attribute names) from the schema. See the *online API Reference* for additional information.

Note

The schema object is read from the database, and all further actions are carried out on this copy.

Using the Information Mining tools

The classes to use the Information Mining tools are in the **com.ibm.mm.sdk.common.infomining.analysis** package. Use the tools to:

- Generate a document summary
- Determine the category of the document
- Determine the language of the document
- Extract information, such as names, terms, and expressions from the documents
- Cluster sets of documents

The tools process text documents, which are objects of class `DKIKFTextDocument`. There are two methods of creating a text document object, namely:

- Create methods of the class `DKIKFTextDocument` if the document content already exists as a Java string
- `DKIKFDocumentFilter` class, if the document content is in a formatted form

Both classes are in the **com.ibm.mm.sdk.common.infomining** package.

Text Document

To create a text document object if the document content exists as a Java string, use:

```
DKIKFTextDocument document = DKIKFTextDocument.create("the document content");
```

Note

Three different `create` methods are available to:

- Create a text document with specified content
- Create a text document with specified content and name
- Create a text document with specified content, name, and language

See the *online API Reference* for further information.

Filtering the document

To create a text document object from a formatted document, use:

```
DKIKFDocumentFilter documentFilter = new DKIKFDocumentFilter(ikfService);
byte[] documentBytes = ... //set the bytes of the document from a data source
DKIKFTextDocument document2 = documentFilter.getTextDocument(documentBytes);
```

Note that the original text document can be in any one of the supported format, for example, a pdf or Microsoft Word document.

Use the method of the document filter object to set filter encoding. See the *online API Reference* for further information.

Determining the document language

To determine the language of a document, use:

```
DKIKFLanguageIdentifier languageIdentifier =
  new DKIKFLanguageIdentifier(ikfService);
DKIKFLanguageIdentificationResult[] languageResults =
  languageIdentifier.analyze(document);
document.setLanguage(languageResults[0].getLanguage());
```

This returns an array listing all the objects containing language and confidence value results, with the highest confidence value listed first.

Note

In order to use the other Information Mining tools, such as the summarization and categorization service, the language must be set on the text document.

Use the methods of the language result objects to return confidence values and language.

```
string language = languageResults[0].getLanguage();
float  confidence = languageResults[0].getConfidence();
```

For further information on the Language Identification service and the default settings, see the *online API Reference*.

Generating a document summary

To create a document summary, use:

```
DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);
DKIKFSummarizationResult summarizationResult = summarizer.analyze(document);
```

To return the summary, use:

```
string summary = summarizationResult.getSummary();
```

For further information on the Summarization service and the default settings, see the *online API Reference*.

Extracting document information

To extract information from a document, use:

```
DKIKFInformationExtractor extractor = new DKIKFInformationExtractor(ikfService);
DKIKFInformationExtractionResult information = extractor.analyze(document);
```

Use the methods of the information extraction result objects to explore additional analytical functionality.

```
DKIKFFeature[] features = information.getFeatures();
```

For further information on the Information Extraction service and the default settings, see the *online API Reference*.

Clustering

You can cluster documents, so that similar documents are grouped together by using:

```
DKIKFClusterer clusterer = new DKIKFClusterer();
cluster.analyze(doc1);
cluster.analyze(doc2);
cluster.analyze(doc3);
DKIKFClusterResult clusterResult = clusterer.cluster();
```

Use the methods of the cluster result object to return the cluster nodes, the number of clusters, and the total number of documents.

```
int clusterCount = clusterResult.getClusterCount();
DKIKFClusterNode[]nodes = clusterResult.getClusterNodes();
int documentCount = clusterResult.getDocumentCount()
```

For further information on the Clustering service and the default settings, see the *online API Reference*.

Categorizing

To assign categories to documents, use:

```
DKIKFCategorizer categorizer = new DKIKFCategorizer(catalog);
DKIKFCategorizationResult[] categorizationResults = categorizer.analyze(document);
```

This returns an array listing all the categorization result objects containing categories and confidence values, with the highest confidence value listed first.

Use the methods of the categorizer result objects to return confidence values and the categories.

```
DKIKFCategory bestCategory = categorizationResults[0].getCategory();
```

Note

To use the categorization service, you must first create and train the catalog using the Information Structuring Tool (IST). The categorization service uses one catalog.

Also note that the returned category objects do not belong to the taxonomy object. Use the taxonomy methods to traverse parent and child categories.

For further information on the Categorization service and the default settings, see the *online API Reference*.

Creating records and storing metadata in catalogs

The classes to create records and store metadata are in the **com.ibm.mm.sdk.common.infomining** package. Use these to:

- Create a new record in a catalog
- Retrieve a record from a catalog
- Return the categories for a record
- Update a record value
- Update record category assignment
- Delete a record

Note that the information mining service API can only manage catalogs created by the Information Structuring Tool (IST). Once created, records and metadata can then be added.

Creating a new record in a catalog

A record is created using a PID and a catalog schema:

```
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);
```

In the example, the title, summary and content values of the record are set. The record is created using the values set above and can be assigned to one or more categories.

Retrieving a record from a catalog

To retrieve a record from a catalog using the PID, enter:

```
DKIKFRecord retrievedRecord = catalog.getRecord("PID");
```

Returning the categories for a record

To return the categories of a record, use:

```
DKIKFCategory[] recordCategories = catalog.getCategoriesForRecord("PID");
```

Updating a record value in a catalog

To update a record value, for example the title of a record, use:

```
record.setValue("IKF_TITLE", "This is the new title");
catalog.updateRecord(record);
```

Note

Three different update methods are available to:

- Update the record with record values only (see the above title example)
- Update the record with record values and assigned categories
- Update the record with assigned categories only (see the example below where a record in one category is added to another category as well)

See the *online API Reference* for further information.

Adding a record in a category to another category

To add a record to another category, use:

```
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[categoryList.size()]));
```

In the example, the record is added to a new Movies category. If you want to keep your original category assignments, you must include the original categories in the update record call, as shown in the above example. An update record call on a new category only will remove all original category assignments.

There are three methods to update a record, see “Updating a record value in a catalog” for further information.

The record can also be added to the root category and to more than one category. See the *online API Reference* for details.

Deleting a stored record

To delete a record, use:

```
catalog.deleteRecord("PID");
```

Searching for documents

The classes to search the records are in the **com.ibm.mm.sdk.common.infomining** package.

Finding records containing a specific word

For example, to find a record containing the word ″Bach″ in the category ″Music″, use:

```
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();
```

To run the searchRecords method on the catalog object you need:

- A query string, and
- A search configuration object

The search configuration object specifies search properties, for example, the maximum number of search results. If you are using categories in the query string, the taxonomy object is also necessary.

The search results can be PID strings or records, and can be specified in the DKIKFSearchConfiguration call.

Performance tuning

You may experience performance problems when submitting complex queries, especially if the query string contains several OR operators. To increase performance, avoid complex queries containing OR operators by applying deMorgan's rules to convert OR expressions. Build complex text-search queries using unary + and - operators. The classes to convert these expressions into the Information Mining query language are in com.ibm.mm.sdk.common.infomining.DKIKFWebQueryConverter for applications using the Service API. For applications using the beans, use the CMBAdvancedSearchService.convertWebQuery method.

Running a server task

If you want to execute a defined sequence of tasks from a client application, you can bundle these calls in a server task, send it across to the server once, and then call this server task as often as is needed from the client. This keeps the level of network traffic as low as possible and improves performance significantly.

A server task is an object that implements the interface com.ibm.mm.sdk.common.infomining.DKIKFServerTask. The object is instantiated from the client application, set on the service object using the setServerTask method and then executed on the server using the runServerTask method.

In the example below, an object of class AnalysisTask, which implements the server task interface is set on the service and then executed:

```
...
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"));
  }
...
```

A map of documents is passed to the server task for processing. The server task returns a map of the records for the specified documents containing the created metadata (document language and summary in this case).

The source for the sample server task is as follows:

```
  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();
      }

      //preparing the argument, tools, and the map to be returned
      Map documentMap = (Map)argument;
      DKIKFLanguageIdentifier languageIdentifier = new DKIKFLanguageIdentifier(ikfService);
      DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);
      HashMap recordMap = new HashMap();
      Iterator pids = documentMap.keySet().iterator();

      //creating a record for each 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);
      }

      //returning the results
      return recordMap;
    }
    catch(Exception e) {
      throw new DKIKFServerTaskException(e);
     }
  }
}
```

The server task is instantiated with the name of the catalog that contains the schema required for record creation. The method `runServerTask` retrieves the schema only once. For each of the documents, it creates a record, runs the tools to analyze the document, and stores the created metadata in the record. Finally, all created records are returned to the caller (the client application).

Note

Only the metadata is returned in the above example code, no records are created in the catalog.

Example of an Information Mining application based on JSPs

The Information Mining Java Server Page (JSP) application searches within the Information Mining component for documents within categories. It displays the found documents in a category structure.

The JSP sample application is located in the following directory:

Windows	`<CMBROOT>\samples\jsp\infomining\`
AIX	`/usr/lpp/cmb/samples/jsp/infomining/`
Solaris	`/opt/IBMcmb/samples/jsp/infomining/`

The directory contains these files:

advSearch.jsp The top-level file of the sample.

This part provides the advanced search-specific form-handling code and form-formatting instructions (HTML). It also acts as the controller for selecting views of the data. This file contains initialization instructions specific to the advanced search, in particular the availability of categories that can be searched in.

catView.jsp This part provides the view-specific code and formatting instructions (HTML) for the category view of returned results. It contains loops for iterating through returned results, but it mostly contains formatting instructions.

classes.jsp This part provides the logic and bean-connection code. It contains only Java code. There is an implementation of a simple data structure class used for the viewing of returned results. There is also an implementation of an event handler to retrieve and manipulate returned results. This is where the core of the work is done after a results list has been returned from the advanced search.

logon.html The account and catalog input required to run the sample.

The account and catalog input comprises server name, user name, password and catalog name.

The source code is a sample of using the information mining beans. It contains a description of how the code works, for example, instantiating the beans, connecting the beans together, processing the return of documents using an event handler, and so on.

For the JSPs to run, you must have Enterprise Information Portal and the Information Mining component installed. You also need a Web server that is capable of running JSPs.

For more details on JSP applications, go to:
`http://java.sun.com/products/jsp/index.html`

Installing the JSPs

Before deploying the JSPs, ensure that the IBM WebSphere Application Server (WAS) is installed and running.

The user access rights required for the deployment of the JSPs are:

- For Windows: Administrator authority
- For AIX: Root user privileges
- For Solaris: Root user privileges

The JSPs are deployed as a Web application in the directory `<WAS_Home>\installedApps\JSP.ear\JSP.war` If you make changes to a sample JSP, replace the JSP in the directory just mentioned with the JSP you have created. The WAS will re-compile it automatically.

For information on how to configure the WebSphere Application Server for the JSPS, refer to *Planning and Installing Enterprise Information Portal*.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM
400
Advanced Peer-to-Peer Networking
AIX
AIXwindows
APPN
AS/400
C Set ++
CICS
DATABASE 2
DataJoiner
DB2
DB2 Universal Database
DisplayWrite
e-business
HotMedia
Hummingbird
ImagePlus
IMS
Micro Channel
MQSeries
MVS/ESA
NetView
OS/2
OS/390
PAL
PowerPC
PTX
QBIC
RS/6000
SecureWay
SP
VideoCharger
Visual Warehouse
VisualAge
VisualInfo
WebSphere

Approach, Domino, Lotus, Lotus 1-2-3, Lotus Notes and SmartSuite are trademarks or registered trademarks of the Lotus Development Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines terms and abbreviations specific to this system. Terms shown in *italics* are defined elsewhere in this glossary.

A

abstract class. An object-oriented programming *class* that represents a concept; classes derived from it represent implementations of the concept. You cannot construct an object of an abstract class; that is, it cannot be instantiated.

access control. The process of ensuring that certain functions and stored *objects* can be accessed only by authorized users in authorized ways.

access control list. A list consisting of one or more user IDs or user groups and their associated *privileges.* You use access control lists to control user access to *items* and *objects* in the Content Manager system. You use access control lists to control user access to *search templates* in the Enterprise Information Portal system.

action list. An approved list of the actions, defined by a system administrator or some other *workflow coordinator,* that a user can perform in a *workflow* or document routing process.

ADSM. See *Tivoli® Storage Manager.*

API. See *application programming interface.*

application programming interface (API). A software interface that enables applications to communicate with each other. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by the underlying licensed program.

archive. Persistent storage used for long-term information retention, typically very inexpensive for each stored unit and slow to access, and often in a different geographic location to protect against equipment failures and natural disasters.

attribute. A unit of data that describes a certain characteristic or property (for example, name, address, age, and so forth) of an item, and which can be used to locate that item. An attribute has a type, which indicates the range of information stored by that attribute, and a value, which is within that range. For example, information about a file in a multimedia file system, such as title, running time, or encoding type (MPEG1, H.263, and so forth). For Enterprise Information Portal, see also *federated attribute* and *native attribute.*

attribute group. Convenience grouping of one or more *attributes.* For example, Address might include the attributes Street, City, State, and Zip.

Audio/Video Interleaved (AVI). A RIFF (*Resource Interchange File Format*) file specification that permits audio and video data to be interleaved in a file. The separate tracks can be accessed in alternate chunks for playback or recording while maintaining sequential access on the file device.

AVI. See *Audio/Video Interleaved.*

B

base attributes. A set of indexes that is assigned to each *object*. All Content Manager objects have base *attributes*.

binary large object (BLOB). A sequence of bytes with a size ranging from 0 bytes to 2 gigabytes. This string does not have an associated code page and character set. Image, audio, and video objects are stored in BLOBs.

BLOB. See *binary large object.*

C

cache. A special-purpose buffer, smaller and faster than main storage, used to hold a copy of data that can be accessed frequently. Use of a cache reduces access time, but might increase memory requirements. See also *resource manager cache* and *LAN cache*.

cardinality. The number of rows in a database table.

category. See *item type.*

CGI. See *Common Gateway Interface.*

CGI script. A computer program that runs on a Web server and uses the *Common Gateway Interface (CGI)* to perform tasks that are not usually done by a Web server (for example, database access and form processing). A CGI script is a CGI program that is written in a scripting language such as Perl.

child component. Optional second or lower level of a hierarchical *item type.* Each child component is directly associated with the level above it.

CIF. See *common interchange file.*

CIU. See *common interchange unit.*

class. In object-oriented design or programming, a model or template that can be instantiated to create objects with a common definition and therefore, common properties, operations, and behavior. An object is an instance of a class.

client application. An application written with the Content Manager APIs to customize a user interface. An application written with the object-oriented or Internet APIs to access *content servers* from Enterprise Information Portal.

Client Application for Windows. A complete object management system provided with Content Manager and written with Content Manager APIs. It supports document and folder creation, storage, and presentation, processing, and access control. You can customize it with user exit routines and partially invoke it with APIs.

client/server. In communications, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server.

collection. A group of objects with a similar set of management rules.

combined search. A query that combines one or more of the following types of searches: *parametric,* text, or image.

Common Gateway Interface (CGI). A standard for the exchange of information between a Web server and programs that are external to it. The external programs can be written in any programming language that is supported by the operating system on which the Web server is running. See *CGI script.*

common interchange file (CIF). A file that contains one ImagePlus Interchange Architecture (IPIA) data stream.

common interchange unit (CIU). The independent unit of transfer for a common interchange file (CIF). It is the part of the CIF that identifies the relationship to the receiving database. A CIF can contain multiple CIUs.

component. Generic term for a *root component* or a *child component.*

connection manager. A Content Manager component that helps maintain connections to the library server, rather than starting a new connection for each query. The connection manager has an application programming interface.

connector class. Object-oriented programming *class* that provides standard access to APIs that are native to specific *content servers.*

constructor. In programming languages, a method that has the same name as a class and is used to create and initialize objects of that class.

container. An element of the user interface that holds objects. In the *folder manager,* an *object* that can contain other folders or documents.

content class. See *MIME type.*

content server. A software system that stores multimedia and business data and the related metadata required for users to work with that data. Content Manager and Content Manager ImagePlus for OS/390 are examples of content servers.

cursor. A named control structure used by an application program to point to a specific row within some ordered set of rows. The cursor is used to retrieve rows from the set.

D

data format. See *MIME type.*

datastore. (1) Generic term for a place (such as a database system, file, or directory) where data is stored. (2) In an application program, a virtual representation of a *content server.*

DCA. See *document content architecture.*

DDO. See *dynamic data object.*

destager. A function of the Content Manager *resource manager* that moves objects from the *staging area* to the first step in the object's *migration policy*.

device manager. In a Content Manager system, the interface between the *resource manager* and one or more physical devices.

document. An *item* that can be stored, retrieved, and exchanged among Content Manager systems and users as a separate unit. An item with the document *semantic type* is expected to contain information that forms a document, but does not necessarily imply that it is an implementation of the Content Manager document model.

An item created from a document classified item type (a specific implementation of the Content Manager document model), must contain document parts. You can use document classified item types to create items with either the document or folder semantic type.

Document parts can include varied types of content, including for example, text, images, and spreadsheets.

document content architecture (DCA). An architecture that guarantees information integrity for a document being interchanged in an office system network. DCA provides the rule for specifying form and meaning of a document. It defines revisable form text (changeable) and final form text (unchangeable).

document routing process. In Content Manager a sequence of *work steps,* and the rules governing those steps, through which a *document* or *folder* travels while it is being processed.

document type definition (DTD). The rules that specify the structure for a particular class of XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation can be used within the particular class of documents. A DTD is analogous to a database schema in that the DTD completely describes the structure for a particular markup language.

DTD. See *document type definition.*

dynamic data object (DDO). In an application program, a generic representation of a stored object that is used to move that object in to, and out of, storage.

E

element. An *object* that the *list manager* allocates for an application.

extended data object (XDO). In an application program, a generic representation of a stored complex multimedia *object* that is used to move that object in to, and out of, storage. XDOs are most often contained within *DDOs.*

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that was derived from, and is a subset of, SGML. XML omits the more complex and less-used parts of SGML and makes it much easier to write applications to handle document types, author and manage structured information, and transmit and share structured information across diverse computing systems. The use of XML does not require the robust applications and processing that is necessary for SGML. XML is being developed under the auspices of the World Wide Web Consortium (W3C).

F

feature. The visual content information that is stored in the image search server. Also, the visual traits that image search applications use to determine matches. The four *QBIC* features are average color, histogram color, positional color, and texture.

federated attribute. An Enterprise Information Portal metadata category that is mapped to *native attributes* in one or more *content servers.* For example, the federated attribute, `policy number`, can be mapped to an *attribute,* `policy num`, in Content Manager and to an attribute, `policy ID`, in Content Manager ImagePlus for OS/390.

federated collection. A grouping of objects that results from a *federated search.*

federated datastore. Virtual representation of any number of specific *content servers,* such as Content Manager.

federated entity. An Enterprise Information Portal metadata object that is comprised of *federated attributes* and optionally associated with one or more *federated text indexes.*

federated search. A query issued from Enterprise Information Portal that simultaneously searches for data in one or more *content servers,* which can be heterogeneous.

federated text index. An Enterprise Information Portal metadata object that is mapped to one or more *native text indexes* in one or more *content servers.*

file system. In AIX, the method of partitioning a hard drive for storage.

folder. An *item* of any *item type,* regardless of classification, with the folder *semantic type.* Any item with the folder semantic type contains specific folder functionality that is provided by Content Manager, in addition to all non-resource item capabilities and any additional fuctionality available from an item type classification, such as *document* or resource item. Folders can contain any number of items of any type, including documents and subfolders. A folder is indexed by *attributes.*

folder manager. The Content Manager model for managing data as online documents and folders. You can use the folder manager APIs as the primary interface between your applications and the Content Manager content servers.

H

handle. A character string that represents an object, and is used to retrieve the object.

history log. A file that keeps a record of activities for a *workflow.*

HTML. See *Hypertext Markup Language.*

Hypertext Markup Language (HTML). A markup language that conforms to the SGML standard and was designed primarily to support the online display of textual and graphical information that includes hypertext links.

I

Image Object Content Architecture (IOCA). A collection of constructs used to interchange and present images.

index. To add or edit the attribute values that identify a specific *item* or *object* so that it can be retrieved later.

index class. See *item type.*

index class subset. In earlier Content Manager, a view of an *index class* that an application uses to store, retrieve, and display folders and objects.

index class view. In earlier Content Manager, the term used in the APIs for *index class subset.*

information mining. The automated process of extracting key information from text (summarization), finding predominant themes in a collection of documents (categorization), and searching for relevant documents using powerful and flexible queries.

inline. In Content Manager, an object that is online and in a drive, but has no active *mounts.* Contrast with *mounted*.

interchange. The capability to import or export an image with its index from one Content Manager ImagePlus for OS/390 system to another ImagePlus system using a *common interchange file* or *common interchange unit.*

IOCA. See *Image Object Content Architecture.*

item. In Content Manager, generic term for an instance of an *item type.* For example, an item might be a *folder, document*, video, or image. Generic term for the smallest unit of information that Enterprise Information Portal administers. Each item has an identifier. For example, an item might be a *folder* or a *document.*

item type. A template for defining and later locating like *items*, consisting of a *root component,* zero or more *child components,* and a classification.

item type classification. A categorization within an *item type* that further identifies the *items* of that item type. All items of the same item type have the same item type classification.

Content Manager supplies the following item type classifications: *folder, document*, object, video, image, and text; users can also define their own item type classifications.

iterator. A class or construct that you use to step through a collection of objects one at a time.

J

JavaBeans. A platform-independent, software component technology for building reusable Java components called "beans." After they are built, these beans can be made available for use by other software engineers or can be used in Java applications. Using JavaBeans, software engineers can manipulate and assemble beans in a graphical drag-and-drop development environment.

Joint Photographic Experts Group (JPEG). (1) A group that worked to establish the standard for the compression of digitized continuous-tone images. (2) The standard for still pictures developed by this group.

JPEG. See *Joint Photographic Experts Group.*

K

key field. See *attribute.*

L

LAN. See *local area network.*

LAN cache. An area of temporary storage on a local *resource manager* that contains a copy of objects stored on a remote resource manager.

library client. The component of a Content Manager system that provides a low-level programming interface for the library system. The library client includes APIs that are part of the software developer's kit.

library object. See *item.*

library server. The component of a Content Manager system that stores, manages, and handles queries on *items*.

link. A directional relationship between two *items:* the source and the target. You can use a set of links to model one-to-many associations. Contrast with *reference.*

local area network (LAN). A network in which a set of devices are connected to one another for communication and that can be connected to a larger network.

M

machine-generated data structure (MGDS). (1) An IBM structured data format protocol for passing character data among the various Content Manager ImagePlus for OS/390 programs. (2) Data extracted from an image and put into general data stream (GDS) format.

management class. The term used in the APIs for *migration policy*.

media archiver. A physical device that is used for storing audio and video stream data. The VideoCharger is a type of media archiver.

media server. An AIX-based component of the Content Manager system that is used for storing and accessing video files.

method. In Java design or programming, the software that implements the behavior specified by an operation. Synonymous with member function in C++.

MGDS. See *machine-generated data structure.*

migration. (1) The process of moving data and source from one computer system to another computer system without converting the data, such as when moving to a new operating environment. (2) Installation of a new version or release of a program to replace an earlier version or release.

migration policy. A user-defined schedule for moving *objects* from one *storage class* to the next. It describes the retention and class transition characteristics for a group of objects in a storage hierarchy.

migrator. A function of the *resource manager* that checks *migration policies* and moves objects to the next *storage class* when they are scheduled to move.

MIME type. An Internet standard for identifying the type of object being transferred across the Internet. MIME types include several variants of audio, image, and video. Each object has a MIME type.

Mixed Object Document Content Architecture (MO:DCA). An IBM architecture developed to allow the interchange of object data among applications within the interchange environment and among environments.

Mixed Object Document Content Architecture–Presentation (MO:DCA–P). A subset architecture of MO:DCA that is used as an envelope to contain documents that are sent to the Content Manager ImagePlus for OS/390 workstation for displaying or printing.

MO:DCA. *Mixed Object Document Content Architecture*

MO:DCA–P. *Mixed Object Document Content Architecture—Presentation*

mount. To place a data medium in a position to operate.

mounted. In Content Manager, an object that is online and in a drive, with active *mounts*. Contrast with *inline*.

multimedia. Combining different media elements (text, graphics, audio, still image, video, animation) for display and control from a computer.

multimedia file system. A *file system* that is optimized for the storage and delivery of video and audio.

Multipurpose Internet Mail Extensions (MIME) . See *MIME type.*

N

native attribute. A characteristic of an object that is managed on a specific *content server* and that is specific to that content server. For example, the *key field* `policy num` might be a native attribute in a Content Manager content server, whereas the field `policy ID` might be a native attribute in an Content Manager OnDemand content server.

native entity. An *object* that is managed on a specific *content server* and that is comprised of *native attributes.* For example, Content Manager *index classes* are native entities comprised of Content Manager *key fields.*

native text index. An index of the text *items* that are managed on a specific *content server.* For example, a single text search index on a Content Manager content server.

network table file. A text file that contains the system-specific configuration information for each node in a Content Manager system. Each node in the system must have a network table file that identifies the node and lists the nodes that it needs to connect to.

The name of a network table is FRNOLINT.TBL.

O

object. Any digital content that a user can store, retrieve and manipulate as a single unit, for example, *JPEG* images, MP3 audio, *AVI* video, and a text block from a book.

Object Linking and Embedding (OLE). A Microsoft® specification for both linking and embedding applications so that they can be activated from within other applications.

object server. See *resource manager.*

object server cache. See *resource manager cache.*

OLE. See *Object Linking and Embedding.*

overlay. A collection of predefined data such as lines, shading, text, boxes, or logos, that can be merged with variable data on a page during printing.

P

package. A collection of related *classes* and interfaces that provides access protection and namespace management.

parametric search. A query for *objects* that is based on the *properties* of the objects.

part. See *object.*

patron. The term used in the Content Manager APIs for *user*.

persistent identifier (PID). An identifier that uniquely identifies an *object,* regardless of where it is stored. The PID consists of both an item ID and a location.

PID. See *persistent identifier.*

privilege. The right to access a specific *object* in a specific way. Privileges includes rights such as creating, deleting, and selecting objects stored in the system. Privileges are assigned by the administrator.

privilege set. A collection of *privileges* for working with system components and functions. The administrator assigns privilege sets to users (user IDs) and *user groups*.

property. A characteristic of an *object* that describes the object. A property can be changed or modified. Type style is an example of a property.

purger. A function of the *resource manager* that removes *objects* from the system.

Q

QBIC. See *query by image content.*

query by image content (QBIC). A query technology that enables searches based on visual content, called features, rather than plain text. Using QBIC, you can search for objects based on their visual characteristics, such as color and texture.

query string. A character string that specifies the properties and property values for a query. You can create the query string in an application and pass it to the query.

R

rank. An integer value that signifies the relevance of a given part to the results of a query. A higher rank signifies a closer match.

README file. A file that should be viewed before the program associated with it is installed or run. A README file typically contains last-minute product information, installation information, or tips for using the product.

reference. Single direction, one-to-one association between a root or *child component* and another *root component.* Contrast with *link.*

release. To remove suspend criteria from an *item.* A suspended item is released when the criteria have been met, or when a user with proper authority overrides the criteria and manually releases it.

Remote Method Invocation (RMI). A set of APIs that enables distributed programming. An object in one Java Virtual Machine (JVM) can invoke methods on objects in other JVMs.

render. To take data that is not typically image-oriented and depict or display it as an image. In Content Manager, word-processing documents can be rendered as images for display purposes.

Resource Interchange File Format (RIFF) . Used for storing sound or graphics for playback on different types of computer equipment.

resource manager. The component of a Content Manager system that manages *objects.* These objects are referred to by *items* stored on the *library server.*

resource manager cache. The working storage area for the *resource manager*. Also called the *staging area.*

RIFF. See *Resource Interchange File Format.*

RMI server. A server that implements the Java *Remote Method Invocation (RMI)* distributed object model.

root component. The first or only level of a hierarchical *item type,* consisting of related system- and user-defined *attributes.*

S

search criteria. In Content Manager, *attribute* values that are used to retrieve a stored *item*. In Enterprise Information Portal, specific fields that an administrator defines for a *search template* that limit or further define choices available to the *users*.

search template. A form, consisting of *search criteria* designed by an administrator, for a specific type of federated search. The administrator also identifies the *users* and *user groups* who can access each search template.

semantic type. The usage or rules for an *item.* Base, annotation, and note are semantic types supplied by Content Manager; users can also define their own semantic types.

server definition. The characteristics of a specific *content server* that uniquely identify it to Enterprise Information Portal.

server inventory. The comprehensive list of *native entities* and *native attributes* from specified *content servers.*

server type definition. The list of characteristics, as identified by the administrator, required to uniquely identify a custom server of a certain type to Enterprise Information Portal.

SMS. See *system-managed storage.*

staging. The process of moving a stored *object* from an offline or low-priority device back to an online or higher priority device, usually on demand of the system or on request of a user. When a user requests an object stored in permanent storage, a working copy is written to the *staging area*.

staging area. The working storage area for the *resource manager*. Also referred to as *resource manager cache.*

stand-alone system. A preconfigured Content Manager system that installs all of the components of a Content Manager system on a single personal computer.

storage class. Identifies the type of media that an object is stored on. It is not directly associated with a physical location; however, it is directly associated with the *device manager*. Types of storage classes include:

DASD
Fixed Disk
Optical
Stream
Tape
TSM

storage group. Associates a storage system to a storage class.

storage system. A generic term for storage in the Content Manager system. See *TSM volume*, *media archiver*, and *volume.*

streamed data. Any data sent over a network connection at a specified rate. A stream can be one data type or a combination of types. Data rates, which are expressed in bits per second, vary for different types of streams and networks.

subclass. A *class* that is derived from another class. One or more classes might be between the class and subclass.

superclass. A *class* from which a class is derived. One or more classes might be between the class and superclass.

suspend. To remove an *object* from its *workflow* and define the suspension criteria needed to activate it. Later activating the object enables it to continue processing.

system-managed storage (SMS). The Content Manager approach to storage management. The system determines object placement, and automatically manages object backup, movement, space, and security.

T

table of contents (TOC). The list of *documents* and *folders* that are contained in a folder or *workbasket.* Search results are displayed as a folder table of contents.

thin client. A client that has little or no installed software but has access to software that is managed and delivered by network servers that are attached to it. A thin client is an alternative to a full-function client such as a workstation.

Tivoli Storage Manager (TSM). A *client/server* product that provides storage management and data access services in a heterogeneous environment. It supports various communication methods, provides administrative facilities to manage the backup and storage of files, and provides facilities for scheduling backup operations.

TOC. See *table of contents.*

TSM. See *Tivoli Storage Manager.*

TSM volume. A logical area of storage that is managed by *Tivoli Storage Manager.*

U

uniform resource locator (URL). A sequence of characters that represent information resources on a computer or in a network such as the Internet. This sequence of characters includes the abbreviated name of the protocol used to access the information resource and the information used by the protocol to locate the information resource. For example, in the context of the Internet, these are abbreviated names of some protocols used to access various information resources: http, ftp, gopher, telnet, and news.

user. A person who requires the services of Content Manager. This term generally refers to users of client applications, rather than the developers of applications, who use the Content Manager APIs. In Enterprise Information Portal, anyone who is identified in the Enterprise Information Portal administration program.

user exit. A point in an IBM-supplied program at which a user exit routine can be given control.

user exit routine. A user-written routine that receives control at predefined *user exits*.

user group. A group consisting of one or more defined individual *users,* identified by a single group name.

user mapping. Associating Enterprise Information Portal user IDs and passwords to corresponding user IDs and passwords in one or more content servers. User mapping enables single logon to Enterprise Information Portal and multiple *content servers*.

utility server. A Content Manager component that is used by the database utilities for scheduling purposes. You configure a utility server when you configure a *resource manager* or *library server*. There is one utility server for each resource manager and each library server.

V

volume. A representation of an actual physical storage device or unit on which the objects in your system are stored.

W

wildcard character. A special character such as an asterisk (*) or a question mark (?) that can be used to represent one or more characters. Any character or set of characters can replace a wildcard character.

workbasket. A collection of *documents* or *folders* that are either in process or waiting to be processed. A workbasket definition includes the rules that govern the presentation, status, and security of its contents.

workflow. In earlier Content Manager, a sequence of *workbaskets* through which a *document* or *folder* travels while it is being processed. In Enterprise Information Portal, a sequence of *work steps,* and the rules governing those steps, through which a *work packet, document,* or *folder* travels while it is being processed.

For example, `claims approval` would describe the process that an individual insurance claim must follow for approval.

workflow coordinator. In earlier Content Manager workflow, a user who receives notification that a *work item* in the *workflow* has not been processed in some specified time. The user is selected for a specific *user group* or upon creation of the workflow.

workflow state. The status of an entire *workflow*.

work item. In earlier Content Manager workflow and Enterprise Information Portal advanced workflow, any work activity that is active within a *workflow.*

worklist. A collection of *work items, documents,* or *folders* that are assigned to a user.

work packet. In Enterprise Information Portal Version 7.1, a collection of *documents* that is routed from one location to another. Users access and work with work packets through *worklists.*

work state. The status of an individual *work item*, *document*, or *folder*.

work step. A discrete point in a *workflow* or *document routing process* through which an individual *work item, document,* or *folder* must pass.

X

XDO. See *extended data object.*

XML. See *Extensible Markup Language.*

Index

Program Number: 5724-B43

Printed in U.S.A.

SC27-1347-01

Spine information:

IBM Content
Manager for Multiplatforms/IBM
Information
Integrator for Content

Workstation Application Programming Guide

Version 8 Release 2