

IBM[®] Enterprise Information Portal for Multiplatforms



Application Programming Guide

Version 7.1

IBM[®] Enterprise Information Portal for Multiplatforms



Application Programming Guide

Version 7.1

Note

Before using this information and the product it supports, read the information in "Notices" on page 413.

First Edition (March 2001)

This edition applies to Version 7.1 of IBM Enterprise Information Portal and IBM Enterprise Information Portal Client Kit (product numbers 5697-G29, 5697-G31) and to all subsequent releases and modifications until otherwise indicated in new editions. This edition replaces SC27-0877-00.

Portions of this product are: Copyright © 1990–2000 ActionPoint, Inc. and/or its licensors, 1299 Parkmoor Drive, San Jose, CA 95126 U.S.A. All rights reserved.

Outside In[®] Viewer Technology © 1992–2000 Inso Corporation. All rights reserved.

© Copyright International Business Machines Corporation 1996, 2001. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this guide vii

Who should read this guide.	vii
Where to find more information	vii
Information included in your product package	vii
Support available on the Web	viii
How to send your comments	viii

Chapter 1. Introducing Enterprise

Information Portal 1

Searching for customer information.	1
The need	1
The solution	2
The Enterprise Information Portal solution	2
Enterprise Information Portal components	4
What's new in Version 7.1	6

Chapter 2. Enterprise Information Portal application programming concepts . . . 7

Understanding data access through content servers	7
Understanding data concepts	8
Items	8
Item attributes and index classes.	8
Item parts	8
Documents and folders	9
Understanding dynamic data object concepts	9
Dynamic data objects (DDO)	9
Extended data objects (XDO)	10
Representing multimedia content	10
Understanding datastores and DDOs	11
Comparing DDO/XDOs with attribute values and item parts	11
Understanding persistent identifiers (PID)	11
Understanding federated searching	11
Federated schema mapping	14
Using federated datastore mapping components	14
Running federated queries	14

Chapter 3. Using the Java application programming interfaces (APIs) 17

Client/server architecture.	17
Differences between the Java and C++ APIs	18
Packaging for the Java environment	18
Programming tips	18
Setting up the Windows® and AIX environment	18
Setting environment variables	20
Using Remote Method Invocation (RMI) with content servers	21
Multiple search facilities	22
Tracing and diagnostic information	22
For text queries using Text Search Engine	22
For parametric queries.	22
Exception handling.	22
Constants	23
Connecting to content servers	23

Establishing a connection.	23
Connecting and disconnecting from a content server in a client.	24
Setting and getting content server options	24
Listing servers	25
Listing the entities and entity attributes for a content server	25
Working with DDOs	26
Creating a DKDDO.	26
Adding properties to a DDO	27
Creating a persistent identifier (PID)	27
Working with data items and properties.	27
Getting properties	28
Displaying the whole DDO	29
Working with XDOs	29
Using an XDO PID	29
Understanding XDO properties.	29
Programming tips	30
Using XDO as a part of a DDO or stand-alone	30
Examples of working with an XDO	32
Importing XML documents	41
The XML Document Type Definition (DTD)	42
Storing content in XML documents	43
Extracting content from different XML sources.	44
Importing XML content into Content Manager.	44
Creating and using the DKPARTS attribute.	44
Creating and using the DKFOLDER attribute	45
Using collections and iterators	46
Using sequential collection methods	46
Using the sequential iterator.	46
Sorting the collection	47
Understanding federated collection and iterator	48
Querying a content server	49
Differences between dkResultSetCursor and DKResults	49
Using parametric queries.	49
Using text query.	51
Using the result set cursor	57
Opening and closing the result set cursor to rerun the query	57
Setting and getting positions in a result set cursor	57
Creating a collection from a result set cursor	58
Querying collections	58
Getting the result of a query.	58
Evaluating a new query	59
Using queryable collection instead of combined query	59
Working with specific content servers	59
Working with Content Manager	62
Working with OnDemand	96
Working with ImagePlus for OS/390	100
Working with VisualInfo for AS/400.	103
Working with Domino.Doc	106
Working with Domino Extended Search (DES)	110
Working with relational databases	116

Working with DB2 Warehouse Manager	
Information Catalog Manager	119
Using Enterprise Information Portal workflow	122
Creating custom content server connectors	125

Chapter 4. Using non-visual and visual JavaBeans. 141

Using JavaBeans in builders	141
Using IBM VisualAge for Java	141
Non-visual beans	142
Non-visual bean configurations	143
Understanding properties and events for non-visual beans	143
Building an application using non-visual beans	144
Working with visual beans	151
CMBLogonPanel bean	151
CMBSearchTemplateList bean	153
CMBSearchTemplateViewer bean	154
Validating or editing fields of the CMBSearchTemplateViewer.	154
CMBSearchResultsViewer bean	155
Overriding pop-up menus	156
CMBFolderViewer bean	156
CMBDocumentViewer bean	157
Viewer specifications	157
Default viewers	157
Launching external viewers	158
CMBItemAttributesEditor bean	158
Vetoing changes in the CMBItemAttributesEditor	159
General behaviors for visual beans	159
Replacing a visual bean	160
Building an application using visual beans	161

Chapter 5. Using the sample Java applets and servlet 169

Understanding the connect applet	169
Understanding the query applet	170
Understanding the view applet	173
Understanding the retrieve servlet	174
Running applets in a Java application	175
Accessing local applets	175
Accessing remote applets	176
Displaying part information	177
Indexing a part.	179
Setting the content viewer	181
Loading video streams	182
Displaying video stream parts information	184
Playing video streams	185
Working in conjunction with Dynamic Page Builder	185

Chapter 6. Using the sample thin client 187

Sample JavaServer Pages	187
-----------------------------------	-----

Chapter 7. Working with information mining 191

Building an application using the Information Mining beans	191
Location of the sample files	191
The categorization sample: Categorizing information found by a standard EIP search	192
Scenario 2: Import documents and metadata using federated search	197
The advanced search sample: Make an advanced search and analyze the results	200
The Web Crawler sample: Getting information from crawling the Web	206
Building your own content provider.	210
Understanding the Information Mining JSP applications	212

Chapter 8. Using the C++ application programming interfaces 215

Setting up the Windows and AIX environment	215
Setting AIX environment variables	216
Setting Windows environment variables	216
Building C++ programs on Windows	217
Setting console subsystem for code page conversion on Windows.	217
Multiple search facilities.	218
Tracing information	218
For text queries using Text Search Engine	218
For parametric queries	218
Catching a DKException.	219
Connecting to content servers	219
Establishing a connection	219
Setting and getting content server options.	219
Listing content servers	220
Listing a content server's schema.	220
Using DDOs.	221
Creating a DKDDO	222
Creating a persistent identifier (PID)	222
Adding data items and properties	223
Adding properties to a DDO	223
Setting and getting data item values.	224
Getting the DDKDO and attribute properties	224
Displaying the DDO	225
Deleting a DDO	225
Using XDOs.	225
Using an XDO PID	225
Understanding XDO data members	225
DB2, ODBC and DataJoiner configuration strings for C++.	226
Programming tips	226
Using XDO as a part of DDO instead of a stand-alone XDO	227
Creating and using the DKPARTS attribute	238
Creating and using the DKFOLDER attribute.	239
Using DKAny	240
Using type code	241
Managing memory in DKAny.	241
Using constructors.	241
Getting the type code.	241
Assigning a new value to DKAny	241
Assigning a value from DKAny	242
Displaying DKAny	242
Destroying DKAny	242

Programming tips	243
Using collections and iterators.	243
Using sequential collection methods.	243
Using the sequential iterator	243
Managing memory in collections	244
Sorting the collection	245
Programming tips	245
Understanding federated collection and iterator	245
Querying a content server	247
Differences between dkResultSetCursor and	
DKResults	247
Using parametric query	247
Using text query	250
Using the result set cursor	255
Opening and closing the result set cursor to	
re-execute the query	255
Setting and getting positions in a result set	
cursor	255
Creating a collection from a result set	256
Querying collections	257
Getting query results	257
Evaluating a new query	257
Using queryable collection instead of combined	
query	258
Using specific content servers	258
Working with Content Manager	260
Working with ImagePlus for OS/390	295
Working with VisualInfo for AS/400.	299
Working with Domino.Doc	304
Working with Domino Extended Search (DES)	308
Working with custom content servers	316

Chapter 9. Using the ActiveX (OLE) application programming interface . . . 331

Running in client/server mode	332
Updating the registry using regedit or	
DCOMCnfg	332
Updating the registry using OLEView	332
Setting up the Windows environment	333
Setting Windows environment variables	333
Using DXInstallDL, DXInstallDES, and	
DXInstallFed, DXInstallDD, DXInstallIP, and	
DXInstallV4	333
Multiple search facilities.	334
Connecting to content servers	334
Establishing a connection	334
Setting and getting datastore options	335
Listing servers	335
Listing schema and schema attributes	335
Using DDOs.	336
Creating a persistent identifier (PID)	336
Adding data items and properties	337
Adding properties to a DDO	337
Setting and getting data item values.	338
Getting the properties	338
Displaying the DDO	338
Deleting a DDO	339
Using XDOs.	339

Using an XDO PID	339
Understanding XDO data members	339
Using XDO in a datastore	340
Programming tips	340
Using XDO as a part of DDO instead of	
stand-alone XDO	340
Creating and using the DX_DL_DKPARTS attribute	344
Creating and using the DX_DL_DKFOLDER	
attribute	344
Using collections and iterators.	345
Querying a content server	346
Using parametric query	346
Using text query	349
Using result set cursor	353
Opening and closing the result set cursor to	
re-execute the query	353
Setting and getting positions in a result set	
cursor	353
Creating a collection from a result set	354
Querying collections	354
Getting the result of a query	354
Evaluating a new query	355
Using queryable collection instead of combined	
query	355
Using specific content servers	356
Working with Content Manager	357
Working with Domino Extended Search (DES)	385

Chapter 10. Using the Dynamic Page Builder 393

Configuring the Dynamic Page Builder with	
Net.Data	393
The ENVIRONMENT statement (DTW_DLDPB)	393
The MACRO_PATH statement.	394
The INCLUDE_PATH statement	394
The HTML_PATH statement	394
Dynamic Page Builder functions	394
API functions	394
Input parameters	395
Inline data	396
Variable definition.	399
Special output variable	399
Developing a Net.Data macro for the Dynamic	
Page Builder	400
Sample macro 1	400
Sample macro 2	404
Improving performance	408
Invoking the wizard	409
Starting the Dynamic Page Builder sample	409
Web server configuration	409
Connection manager setup	410

Notices	413
Trademarks	415

Glossary	417
---------------------------	------------

Index	423
------------------------	------------

About this guide

This guide describes how to use and modify the Java, JavaBeans, C++, Active/X, and Dynamic Page Builder portions of IBM Enterprise Information Portal. Enterprise Information Portal lets you create multiple search query applications that search for documents in various content servers and have a World Wide Web interface.

This guide includes:

- Descriptions of how the various components work.
- Tips for identifying application requirements as you create a query application.
- Details of how the sample code works and how to tailor it to your needs.

The C++ API and Java API chapters contain similar information placed within a consistent structure; this consistency between chapters unites the information contained in the C++ and Java API sets. The consistent structure also helps application developers coordinate the development of their applications, regardless of the programming language they choose.

For additional information about programming with other IBM Content Manager components in the C language, see the *C Application Programming Guide* for the platform you are using.

Who should read this guide

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

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

Where to find more information

IBM® Enterprise Information Portal 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 IBM Enterprise Information Portal publication library is available online in HTML (Hypertext Markup Language) format. On Windows operating systems, you can view the publications from the taskbar by clicking **Start** → **Programs** → **IBM Enterprise Information Portal for Multiplatforms 7.1** → **Information** and then the title of the publication that you want to view.

The *IBM Enterprise Information Portal* CD-ROM contains each publication in the Adobe Acrobat portable document format (PDF).

You can view the PDF files online using the provided Adobe Acrobat Reader for the following operating systems: Windows® or AIX®. (Adobe provides Acrobat Readers for additional operating systems on their Web site, <http://www.adobe.com>.)

Table 1 shows the Enterprise Information Portal publications included on the *IBM Enterprise Information Portal* CD-ROM.

Table 1. Enterprise Information Portal publications

File name	Title	Publication number
apgoo	<i>Application Programming Guide</i>	SC27-0877-01
eipinst	<i>Planning and Installing Enterprise Information Portal¹</i>	GC27-0873-00
eipmc	<i>Messages and Codes</i>	SC27-0874-00
eipmng	<i>Managing Enterprise Information Portal</i>	SC27-0875-00
txtse	<i>Text Search Engine Application Programming Reference</i>	SC27-0876-00

Notes:

1. You receive a printed copy of the *Planning and Installing Enterprise Information Portal* with the product.

Copying the PDF files: To copy the PDF files from the CD-ROM to your hard drive:

1. Change to the directory for the language that you are using (for example, ENU for English).
2. Copy *.PDF files to your designated hard drive directory

Installing the PDF reader: The Adobe Acrobat Reader is available in the ACROBAT directory of the CD-ROM and from <http://www.adobe.com>. To install Acrobat Reader, decompress the program files for your platform and follow the instructions in the Acrobat installation program or the installation text file.

On AIX, untar the .tar file and read INSTGUID.TXT.

On Windows, run the executable file.

Support available on the Web

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

<http://www.ibm.com/software/data/eip/>

The documentation is included in softcopy on the product CD-ROM. 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 reference information about the object-oriented and Internet APIs. 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 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:
<http://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, please include the location of the text (for example, a chapter and section title, a table number, a page number, or a help topic title).

Chapter 1. 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. Enterprise Information Portal can help you maximize the value of your information and multimedia assets by connecting disparate content servers through a single client. With an Enterprise Information Portal 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 Enterprise Information Portal, you can customize applications for your enterprise. Using the Enterprise Information Portal toolkits, application programmers can write both desktop and Web-based applications.

This chapter provides an overview of Enterprise Information Portal. A scenario about a fictitious insurance company, AA Insurance, demonstrates Enterprise Information Portal’s features and functionality.

Searching for customer information

AA Insurance (AA), 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.

AA keeps all memos that are sent to policy holders, along with medical and appraisal electronic forms in Lotus® Domino™.Doc file cabinets. AA archives all policy declarations, notices, and invoices in a Content Manager OnDemand server for long-term storage and quick access. AA stores all claim forms, photographs, and letters received from policy holders in a Content Manager for AS/400® system folder. AA keeps reports from experts in a DB2 Universal Database™ (DB2 UDB) Data Warehouse Center Information Catalog Manager. AA 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, AA 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. AA 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. AA must restrict access to certain items, while providing unlimited access to others. AA also wants an easy-to-use interface to reduce the need for training.

The solution

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

AA 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

Enterprise Information Portal is a comprehensive product; its components work together to provide a solution uniquely suited to your enterprise. Centered on a multiple-tier architecture, Enterprise Information Portal 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 Enterprise Information Portal.

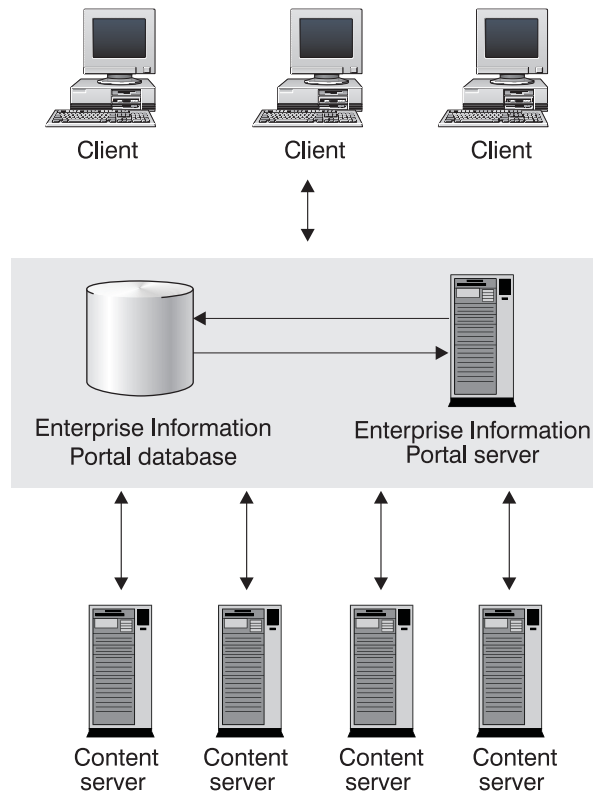


Figure 1. Multiple-tier architecture

The Enterprise Information Portal 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 Enterprise Information Portal administrator.

Using search templates, the client runs a *federated search*, a search that runs simultaneously across content servers whose native attributes have been mapped with the federated attributes used in the search template. The Enterprise Information Portal search templates contain search criteria, which reference federated attributes that are mapped to native attributes on each of the content servers. The Enterprise Information Portal administrator creates the search templates. Enterprise Information Portal provides connectors to make compatible the heterogeneous interfaces of content servers. The content servers then return data objects to the client.

The Enterprise Information Portal 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.

Enterprise Information Portal components

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

Administration database

The Enterprise Information Portal database is a DB2 UDB database that stores the information managed by the administration client. This data is used to perform federated searching, information mining, and workflow processes.

Migrating your Enterprise Information Portal Version 6.1 database: You must migrate your data from Enterprise Information Portal Version 6.1 *before* using your Enterprise Information Portal Version 7.1 administration database.

Administration client

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

- Define each content server for federated searching.
- Identify native entities and attributes on content servers and map them to federated entities.
- Maintain an inventory of the search criteria for all content servers.
- Create search templates.
- Identify and manage users who can access search templates, the information mining feature, and workflow processes.
- Define business workflow processes.
- Administer the information mining server: stopping, starting, and testing the information mining server connection and setting up trace and log levels.

This information is stored in the Enterprise Information Portal database.

It is recommended that you install the administration client on the same workstation, or server, as the Enterprise Information Portal 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 Application Enabler installed and configured 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 Enterprise Information Portal database is installed. You need to make sure that your `\CMBROOT\cmbclient.ini` file points to this server.

Connectors

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

- Relational database connectors
- Federated connector (to the Enterprise Information Portal database)
- Content Manager 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

- DB2 Data Warehouse Manager Information Catalog Manager connector

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

Connectors are either local or remote. Local connectors are a set of connector classes you use for directly connecting to various content servers. Local connectors can reside on an Enterprise Information Portal desktop client or on an Enterprise Information Portal RMI server. Remote connectors, which connect to a content server through an RMI server or an RMI server pool member, are always installed.

Enterprise Information Portal sample client application

The sample client application, built by IBM from the JavaBeans™ in the federated access toolkit, provides an interface for performing a single search to retrieve and display documents from multiple content servers.

When a user runs the Enterprise Information Portal sample client application, the search templates available to that user are retrieved from the Enterprise Information Portal database. The user then selects a template and enters values for the search criteria. A list of documents satisfying the search criteria is returned to the user. Documents can then be selected from the search results list, viewed, and updated.

Enterprise Information Portal thin client samples

The Enterprise Information Portal thin client samples run only in a Web browser. Written in HTML and JavaServer Pages™ (JSP), the thin client samples are easily changed and deployed through a Web browser interface. The thin client samples are a set of files that you install on your Web server.

Users use the thin client samples for performing a single search to retrieve and display documents and folders from multiple content servers. Users can also use the thin client samples for performing tasks in a workflow process.

Connector toolkits and samples

Use the connector toolkits and samples to build your own Internet or desktop client applications that access data and content in content servers. The toolkits provide:

- Java®, C++, and ActiveX classes
- Content server-specific samples

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. Enterprise Information Portal 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.

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 text search server and client

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

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

Content Manager 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: The Content Manager image search server and client is an optional Content Manager feature that you can configure and run with Content Manager servers only. If you do not use Content Manager servers, do not install this feature.

What's new in Version 7.1

Enterprise Information Portal Version 7.1 provides unprecedented access to disparate content servers. The new features and components include:

- XML import capabilities
You can now import XML content into Content Manager as 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

Chapter 2. 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, including 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 shows how the C++, Java (including JavaBeans), and ActiveX APIs access content servers through the connectors.

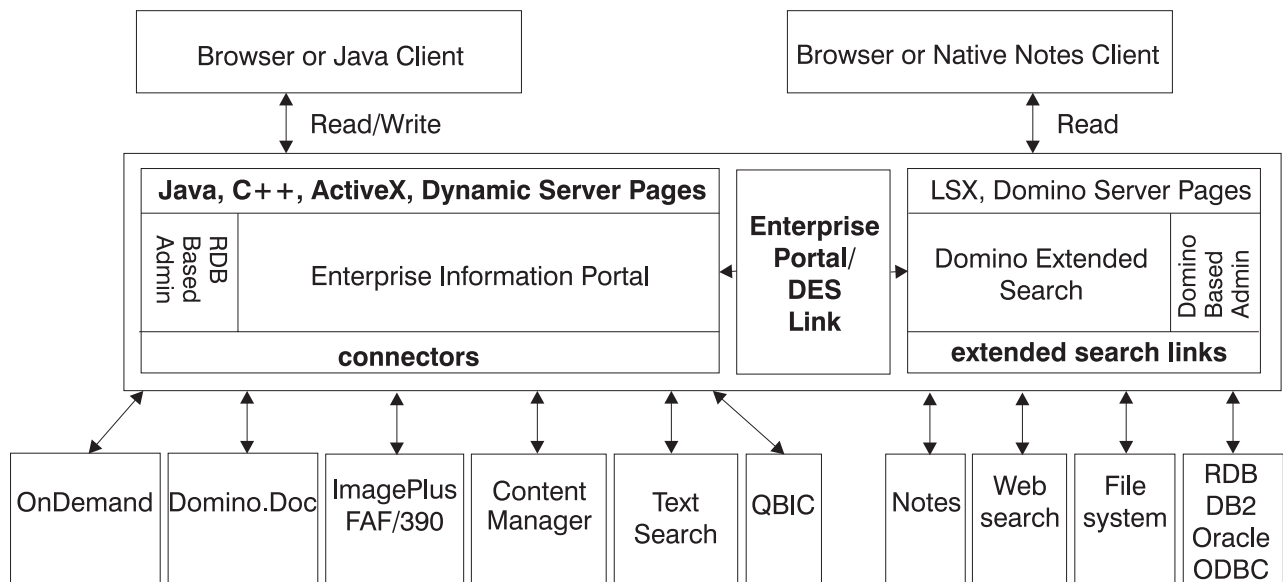


Figure 2. Enterprise Information Portal organization

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
- Domino.Doc
- Domino Extended Search
- ImagePlus for OS/390
- Content Manager OnDemand
- VisualInfo™ for AS/400
- DB2
- DB2 DataJoiner

- DB2 Warehouse Manager Information Catalog Manager
- JDBC/ODBC server

Applications that use Enterprise Information Portal can create a federated datastore, 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 datastore 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 EIP system administration. Enterprise Information Portal keeps the schema and mappings, as well as other administration information in its administration database.

Understanding data concepts

Enterprise Information Portal provides a set of data objects that you can use to organize and store your data. This section describes the following concepts:

- Items
- Item attributes and index classes
- Item parts
- Documents and folders

Items

Items are the building blocks in Enterprise Information Portal. An item can represent anything from a multimedia document to a folder. Each item is uniquely identified by an item ID. The item ID is persistent and stored in a content server. EIP uses the item ID to access the item's associated data.

Item attributes and index classes

A native entity is a conceptual object on a content server; for example, index classes are the native entities on a Content Manager server. The items belong to an entity. Each native entity defines a set of attributes that describe the items. An attribute is a characteristic; for example, title and duration might be attributes of a video object. The characteristic can be determined, and possibly changed, through operations on the managed object.

The native entity is a category for storing and retrieving objects, consisting of a named set of attributes. When you create an object in the Content Manager system for example, your application must assign an index class, the native entity, and supply the *key field* values required by that class, the native attributes.

Item parts

An item can contain one or more parts; the number, order, and type of the parts is specific to the content server and the applications that manage the content. Each item part corresponds to one multimedia data object, such as an image, video, or text.

Documents and folders

A document can be any multimedia item that has digital content.

In Enterprise Information Portal, a document can contain text, images, video clips, or a combination of these. For example, a movie document, which can be considered a document that represents movie images, can also contain text, still images, and an image of a movie poster.

A document can be composed of parts. For example, a paper document consists of a set of closely related subsections, such as chapters.

A folder is a container that holds one or more items, such as documents or other folders. A folder is an item itself. Figure 3 shows how documents, parts, and folders work together in the Enterprise Information Portal framework.

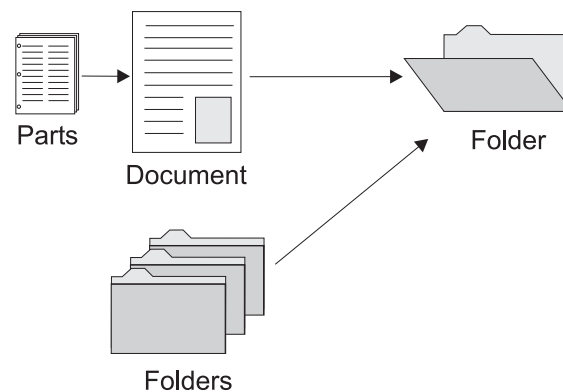


Figure 3. Documents, folders, and parts

Understanding dynamic data object concepts

In compliance with Object Management Groups's (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 datastore, 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 to move data in and out of a datastore. DDOs exist in the application and do not exist after an application terminates.

Dynamic data objects (DDO)

A DDO is a datastore-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 datastore.

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 4 shows Dynamic data objects and data items.

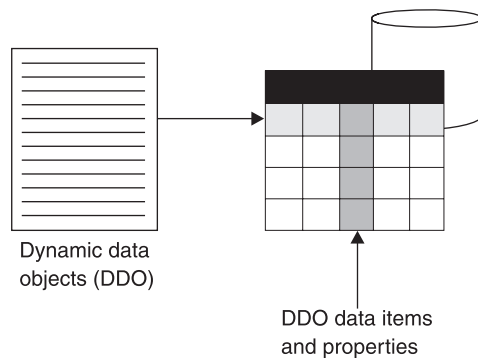


Figure 4. Dynamic data objects and data items

Extended data objects (XDO)

An XDO is a representation of complex multimedia data, for example a part 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 5 shows an example of XDOs.

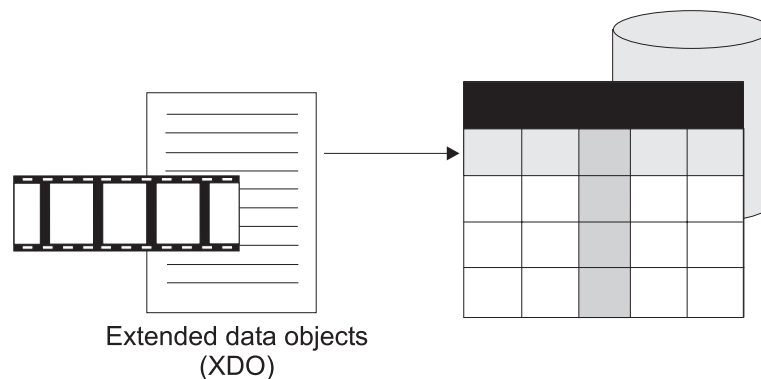


Figure 5. Extended data objects (XDOs)

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. An another example, for a DB2 database, a table is represented by a DDO and a large object, such as a CLOB or BLOB, by an XDO.

Understanding datastores and DDOs

DDOs are created and dynamically associated with a datastore. The association between a DDO and a datastore 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 datastore:

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

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 index class. The data items of a DDO correspond to an item's attributes. For example, in Content Manager an index class is created using a set of attributes, and an item is always indexed by an index class.

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 datastore. A DDO's PID consists of an item ID, a datastore name, and other related information. When a DDO is added to a datastore, a PID needs to be created for it. For example, a PID must be assigned to a DDO that is created by getting persistent data out of the datastore with a query result list.

Because a DDO is a dynamic interface to persistent data that is moved in or out of datastores, 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 datastore to store data persistently, and another DDO can be created to hold the same data entity that is checked out from the same datastore for modification. In this case, these two DDOs share the same PID value.

Understanding 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 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 6 on page 12.

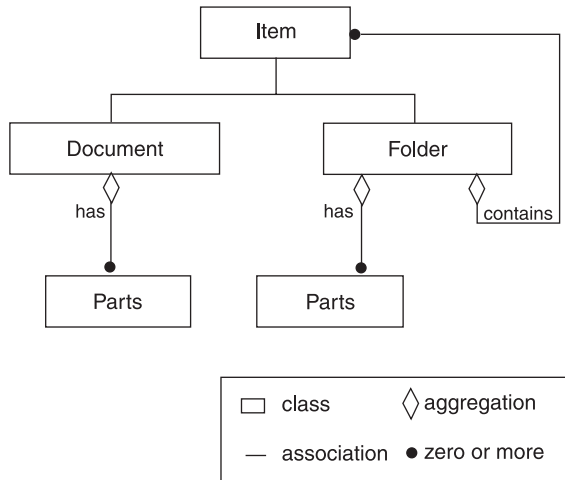


Figure 6. Federated document view

The lines in the Figure 6 indicate that a folder can have parts; however, in the Enterprise Information Portal, parts for folders is not formally supported.

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. A federated document maps to a row in a DB2 table or other relational database.

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, key fields are attributes in Domino.Doc content servers.

Each datastore has its own terminology to explain the model it is supporting. Table 2 relates the terminology used for various content servers to the federated model:

Table 2. Mapping terminology for each datastore

Content server	Data Source	Entity	Attribute	View
Content Manager	Library server	index class	<ul style="list-style-type: none"> • attribute • key attribute 	index class view
OnDemand	OnDemand server	<ul style="list-style-type: none"> • application group • search template 	<ul style="list-style-type: none"> • field • criteria 	N/A
ImagePlus	ImagePlus for OS/390 server	entity	attribute	N/A

Table 2. Mapping terminology for each datastore (continued)

Content server	Data Source	Entity	Attribute	View
VisualInfo	VisualInfo for AS/400 server	index class	attribute	index class view
Domino.Doc	Domino server	<ul style="list-style-type: none"> • library • cabinet • binder 	key field	N/A
Domino Extended Search	Domino 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	
Enterprise Information Portal federated datastore	mapping server	mapped federated entity	mapped federated attribute	search template

Figure 7 illustrates a federated search. The federated search uses the Enterprise Information Portal federated datastore, working through search templates. The federated datastore then calls the searches for the individual datastores to perform the actual search on the content servers. This association is established by schema mapping.

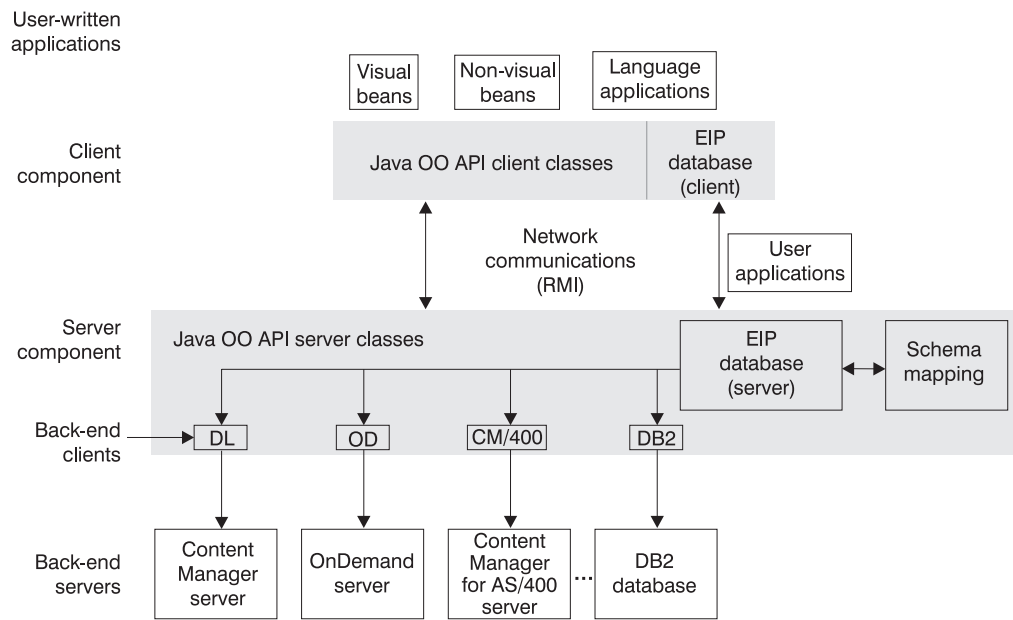


Figure 7. Structure of federated searches

RIM is currently used for the communication between the Enterprise Information Portal federated datastore and the content servers. The applications you develop sit on top of the Java API classes. You can develop application programs, either in the form of Java applications, applets, or beans.

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 datastore and defines a mapping between the concepts in the federated datastore 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 datastore mapping components

In addition to schema mapping information for mapping the entities and attributes, a federated datastore 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, first you create 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 datastore to process the query directly.

The query string is parsed into a federated query form, which is essentially a datastore 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 datastore 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 datastore 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 8.

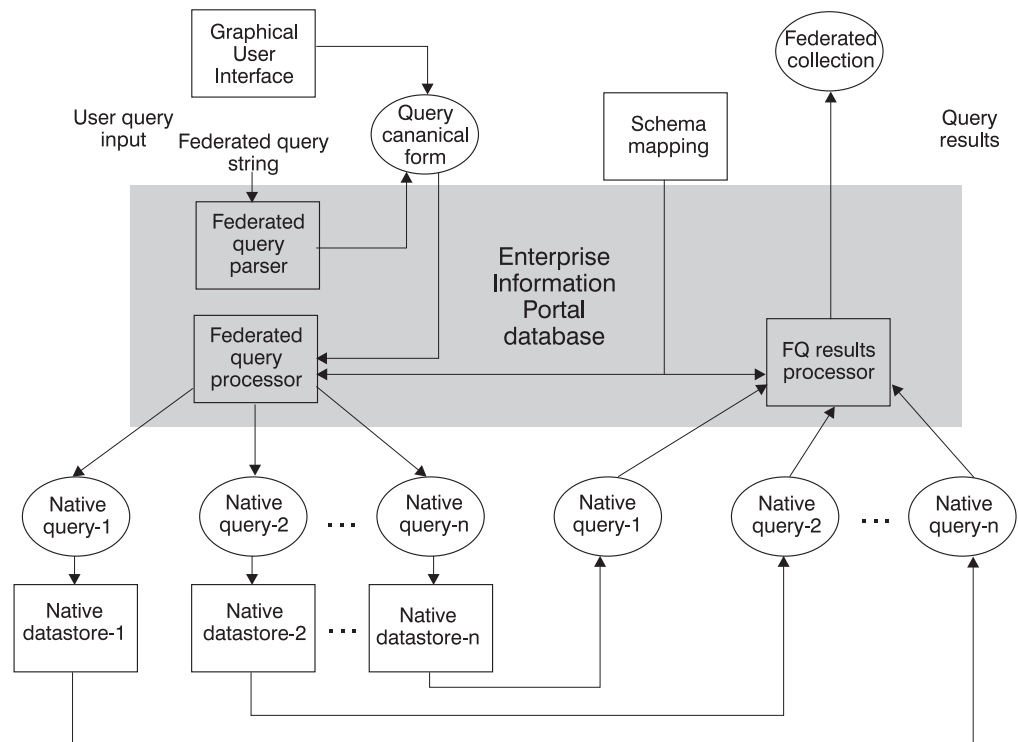


Figure 8. Federated query processing

Federated query syntax

When you create a federated query, it must be in the proper syntax, as shown below. The federated datastore does not support image query. For more information about image query, see “Understanding image search terms and concepts” on page 79).

```
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) };]
                       [MAX_RESULTS=maximum_results;]
                       [TIME_LIMIT=time_limit]
                       )]
]
```

Chapter 3. Using the Java application programming interfaces (APIs)

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

The Java 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, Text Search Engine and query by image content (QBIC)
- Client/server implementation for Java application users
- Workflow capability
- Administration functions

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

Client/server architecture

The APIs provide a convenient programming interface for application users. APIs can reside on both the server and the client (both provide the same interface), and the applications can be located locally or remotely. The client API communicates with the server to access data through the network. Communication between the client and the server is performed by classes; it is not necessary to add any additional programs.

API classes consist of three packages: server, client 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 are related mainly to Enterprise Information Portal and connect directly with it.
- The client package is `com.ibm.mm.sdk.client`. The classes in this package are not directly connected to Enterprise Information Portal; these client classes communicate with the server classes through the network by invoking the server classes to execute and retrieve the results
-
- 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, so that the application can move to wherever the content resides.

The client application must import the client 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.

Differences between the Java and C++ APIs

The list below describes differences between the Enterprise Information Portal 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 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 APIs.
- The Java APIs use Java's garbage collector.

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).

Use the `client` package for Web applications. The `client` package is created with pure Java programs; the `server` package includes C programs.

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 Windows[®] and AIX environment

When you set up your Windows or AIX environment you must establish the following settings:

server package

Import when a datastore and application are on the server side

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

client package

Import when a datastore and application are on the client side.

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

cs package

Import when a datastore location is different from the application location.

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

Library files

- cmbcm71.jar
- cmbdl71.jar
- cmbdlc71.jar
- cmbdd71.jar
- cmbddc71.jar
- cmbip71.jar
- cmbipcv.jar
- cmbod71.jar
- cmbv471.jar
- cmbv4c71.jar
- cmbfed71.jar
- cmbfedc71.jar
- cmbjdbc71.jar
- cmbjdbcc71.jar
- cmbdes71.jar
- cmbdesc71.jar
- cmbdb271.jar
- cmbdb2c71.jar
- cmbdj71.jar
- cmbdj71.jar
- DKConstantDL.txt: constants used in the Content Manager class library
- DKConstantFed.txt: constants used in the Federated class library
- DKConstantIP.txt: constants used in the ImagePlus for OS/390 class library
- DKConstantOD.txt: constants used in the OnDemand class library
- DKConstantV4.txt: constants used in the VisualInfo for AS/400 class library
- DKConstantDES.txt constants used in the Domino Extended Search class library
- DKConstantDB2.txt constants used in the DB2 class library
- DKConstantJDBC.txt constants used in the JDBC class library
- DKConstantDJ.txt constants used in the Data Joiner class library

Shared objects for AIX

- libcmbdsdl71.so
- libcmbdsfed71.so
- libcmbdsdb271.so

- libcmbdsdj71.so

DLLs for Windows

- cmbdsdl71.dll
- cmbdsdb271.dll
- cmbdsdj71.dll
- cmbdsdd71.dll
- cmbdsip71.dll
- cmbdsod71.dll
- cmbdsv471.dll
- cmbdsdes71.dll
- cmbdsfed71.dll
- de_db2.dll
- de_ora.dll

Setting environment variables

When developing an application for the Enterprise Information Portal, you must set up your environment.

For Windows

You can open a DOS command prompt with the environment set up for developing EIP applications by selecting **Start** → **Programs** → **IBM Enterprise Information Portal for Multiplatforms 7.1** → **Development Window**. As an alternative, you can run `cmbenv71.bat` in a DOS 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, `cmbfed71.jar`).

For AIX

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

1. For a Bourne shell, use `cmbenv71.sh`
2. For a C shell, use `cmenv71.csh`
3. For a Korn shell, use `cmenv71.ksh`

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, `cmbfed71.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 port number of the client and server must be same. Figure 9 shows an example of the RMI hierarchy.

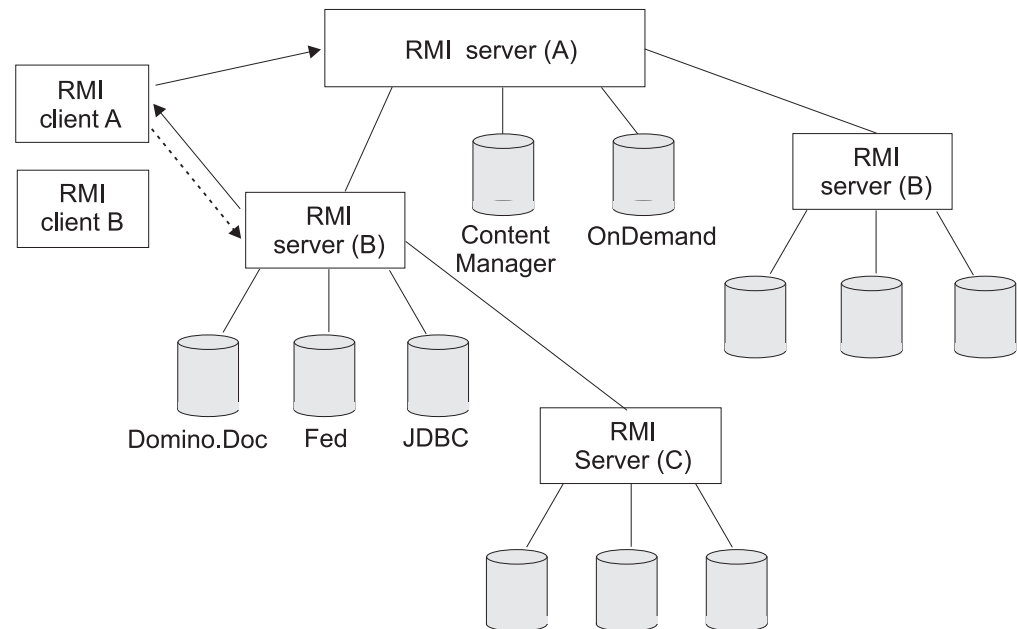


Figure 9. RMI hierarchy

Figure 9 represents a typical Remote Method Invocation (RMI) client/server hierarchy with multiple RMI servers. An RMI server can connect to an infinite number of datastores, but each server must be connected to at least one datastore. The master RMI server (A) can reference other RMI servers in the server pool. When an RMI client first searches for a datastore, it starts at RMI server (A). If the datastore is not found there, the RMI pool servers (B), and then (C), are searched next.

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

To start the RMI server, use `cmbregist71.bat` on Windows or `cmbregist71.sh` on AIX. 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*.

Multiple search facilities

Use the multiple search facilities to search within a given datastore, 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 datastores support multiple search facilities. For more information about specific datastores and multiple search see “Working with specific content servers” on page 59.

Parametric query

Query requiring an exact match on the condition specified in the query predicate and the data values stored in the datastore.

Text query

Query 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.

Image query

Query 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.

Tracing and diagnostic information

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

For text queries using Text Search Engine

The Text Search Engine and all of its functions can only be used with the Content Manager server.

The following environment variable setting write 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:

```
CMBMTTERM=fileName
```

For parametric queries

Use the following environment variable setting to write the parametric query passed to the folder manager to the specified file:

```
CMBDLQRYTRACE=fileName
```

Exception handling

When the Java 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 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. The error ID and exception ID are also given. The code below shows an example of the throw and catch process:

```
try {
    DKDatastoreDL dsDL = new DKDatastoreDL();
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
    dsDL.disconnect();
}
catch (DKException exc) {
    System.out.println("Exception name " + exc.name());
    System.out.println("Exception message " + exc.getMessage());
    exc.printStackTrace();
}
```

Constants

The constants provided for use with the Enterprise Information Portal APIs are defined in DKConstant.java. You can also review a text version of the constants in DKConstant.txt.

The constants specified are in the form of DK_CM_ (Common constants) or DK_XX_ (where the XX indicates different datastores, for example, DL for Content Manager). 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).

Connecting to content servers

An object of the class DKDatastoreXX (where XX indicates a specific content server, for example, Content Manager (DL), OnDemand (OD), or ImagePlus for OS/390 (IP)) represents and manages a connection to a content server, provides transaction support, and runs server commands.

Establishing a connection

Each DKDatastorexx class provides methods for connecting to it and disconnecting from it. The following example uses a Content Manager library server named LIBSRVRN, the user ID FRNADMIN and password PASSWORD. The example creates a DKDatastoreDL object for the content server, connects to it, works with it, then disconnects from it.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect(libSrv,userid,pw,"");
System.out.println("datastore connected " + libSrv + " userid - " + userid);
userName = dsDL.userName();
dsName = dsDL.datastoreName();
System.out.println("user " + userName + " dsName " + dsName);
dsDL.disconnect();
```

The complete sample application from which this example was taken (TConnectDL.java), is available in the CMBROOT\Samples\java\dl 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 a similar procedure to access a content server from a client application. Your client application must handle any communications errors incurred.

The following example of connecting to a content server from a client application uses the Content Manager library server LIBSRVRN with the user ID FRNADMIN and password PASSWORD, then disconnects from the library server.

```
import com.ibm.mm.sdk.common.*;
import com.ibm.mm.sdk.client.*;
import java.io.*;

public class TConnectDL implements DKConstant
{
    // Main method
    public static void main(String argv[])
    {
        DKDatastoreDL dsDL = null;
        try {
            dsDL = new DKDatastoreDL();
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
            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();
        }
    }
}
```

After you are finished with a datastore, call the destroy method to free it.

Setting and getting content server options

You can access or set the processing options on a content server using the methods in `DKDatastorexx`. 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.

```
Integer input_option = new Integer(DK_DL_SS_CONFIG);
Integer output_option = null;
dsDL.setOption(DK_DL_OPT_DL_ACCESS,input_option);
output_option = (Integer)dsDL.getOption(DK_DL_OPT_ACCESS);
```

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

Listing servers

`DKDatastorexx` provides a method to list the servers that it can connect to. The list of servers are returned in a `DKSequentialCollection` of `DKServerInfoxx` objects (where `xx` identifies the specific content server, for example, DL for a Content Manager server, OD for an OnDemand server, and so forth).

Restriction: The `Domino.Doc` datastore does not provide a method that lists the servers.

After you obtain a `DKServerDefxx` object you can retrieve the server name and server type, and use the server name to establish a connection to it.

The following example shows how to retrieve the list of servers:

```
DKDatastoreDL dsDL = new DKDatastoreDL();
// ----- List the servers
pCol = (DKSequentialCollection)dsDL.listDataSources();
pIter = pCol.createIterator();
int i = 0;
while (pIter.more())
{
    i++;
    pSV = (DKServerDefDL)pIter.next();
    strServerName = pSV.getName();
    strServerType = pSV.getServerType();
    System.out.println("Server Name [" + i + "] - " + strServerName +
        " Server Type - " + strServerType);
}
```

Refer to `TListCatalogDL.java` in the `CMBROOT\Samples\java\dl` directory for a complete sample.

Listing the entities and entity attributes for a content server

`DKDatastoreXX` provides methods for listing the entities or index classes and their attributes, for a content server.

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 index classes as well as the list of attributes from a Content Manager server.

```
int i, j;
DKIndexClassDefDL icDef;
DKDatastoreDL dsDL = new DKDatastoreDL();
....
// ----- Connect to the datastore
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD","");
// ----- List the entities (index classes in Content Manager)
pCol = (DKSequentialCollection) dsDL.listEntities();
pIter = pCol.createIterator();
i = 0;
```

```

while (pIter.more()) {
    i++;
    icDef = (DKIndexClassDefDL)pIter.next();
    strIndexClass = icDef.getName();
    System.out.println("index class name [" + i + "] - " + strIndexClass);
    System.out.println(" list attributes for " + strIndexClass + " index class");
    pCol2 = (DKSequentialCollection) dsDL.listEntityAttrs(strIndexClass);
    pIter2 = pCol2.createIterator();
    j = 0;
    while (pIter2.more()) {
        j++;
        attrDef = (DKAttrDefDL)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 + " attributes listed for " + strIndexClass
        + " index class");
}

```

The complete sample application from which this example was taken (TListCatalogDL.java) is available in the CMBROOT\Samples\java\d1 directory.

Working with DDOs

You use the DKDDO class for dynamic data objects (DDOs) in your Enterprise Information Portal applications. A DKDDO object represents an item, which, for example, could be a Content Manager document or a folder. 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 index classes 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.

```
DKDDO addo = new DKDDO();
```

You can use other constructors, passing the number or attributes or the content server and object type; for example, to create a DKDDO by supplying content server and object type:

```

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

```

Which constructor you use depends on your application.

Adding properties to a DDO

After you create a DKDDO object to represent a DDO, you must set its type property. It must be either a document or a folder. This information is recorded as a DKDDO property. For example, the following line sets the type of DDO to be a document:

```

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

```

Creating a persistent identifier (PID)

Each DDO must have a persistent identifier (PID). The PID contains information about the content server name, type, ID, and object type. The ID identifies the location of the DDO's persistent data. For example, in a Content Manager content server, this ID is the item ID. The item ID is used for the retrieve, update, and delete methods. For the add method, the item ID will be created and returned by the datastore.

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

Note: The text used in the item ID is provided as an example only.

```

DKDatastoreDL dsDL = new DKDatastoreDL(); // create a Content Manager datastore
DKPid pid = new DKPid(); // create a PID object
pid.setObjectType("GRANDPA"); // set the index class name it belongs to
pid.setPrimaryId("LN#U5K6ARLGM3DB4"); // set the item ID
DKDDO ddo = new DKDDO(dsDL, pid); // create a DDO with PID and associate
// it with dsDL

```

After you create a DDO to retrieve a known item, connect to the content server and call retrieve to retrieve this DDO.

Working with data items and properties

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

Suppose you have an entity, such as an index class, DK_CM_DOCUMENT and the attributes: Name, Type, and Nullable. 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 table shows the relationship between the attributes and the data items:

Table 3. Attribute and data item information

Attribute	data item 1	data item 2
Name	TITLE	Subject
Type	String	String
Nullable	No	Yes

You create these attributes and set their values in a DKDDO as follows:

```

// ----- create a Content Manager content server
DKDatastoreDL dsDL = new DKDatastoreDL();
// -----create a DDO to hold an object type document in dsDL

```

```

DKDDO cddo = new DKDDO(dsDL, new short(DK_CM_DOCUMENT));

Short vstring = new Short(DK_CM_DATAITEM_TYPE_STRING);
Integer no = new Integer(DK_CM_FALSE);
Integer yes = new Integer(DK_CM_TRUE);

// ----- Add the first attribute
short data_id = cddo.addData("TITLE"); // add a new data item named "TITLE"
// ----- Add a property Type and set to variable length string
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstring);
// ----- Add a property Nullable and set to false
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, no);
// ----- Set the value of the data item
cddo.setData(data_id, "One dark and stormy night");

// ----- Add the second attribute
data_id = cddo.addData("SUBJECT"); // add a new attribute named "SUBJECT"
// ----- Add a properties and set the value
cddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, vstring);
cddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, yes);
cddo.setData(data_id, "Mystery");

```

You must set the property Type for an attribute; Nullable and other properties are optional.

Use the `getData` method to get the values back for TITLE and SUBJECT:

```

Object val;
int data_num = 2;
val = cddo.getData(1);
// ----- Get the value of data item 1; displays "One dark and stormy night"
System.out.println(" TITLE = " + val );
// ----- Get the value of data item 2; displays "Mystery"
System.out.println(" SUBJECT = " + cddo.getData(data_num));

```

Getting properties

When processing a DKDDO, the first thing you want to know is its type: document or folder. The following sample code determines the type of a DDO:

```

short prop_id = cddo.propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    short type = ((Short) cddo.getProperty(prop_id)).shortValue();
    switch(type) {
        case DK_CM_DOCUMENT:
            // process document
            ....
            break;
        case DK_CM_FOLDER:
            // process folder
            ....
            break;
    }
}

```

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

```

data_id = cddo.dataId("Title"); // get data_id of Title
// ----- Get the number of properties for the attribute
short number_of_data_prop = cddo.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 = " + cddo.getDataPropertyName(data_id,i)
        + " value = " + cddo.getDataProperty(data_id,i));
}

```


Both the `data_id` and `property_id` start from 1.

Displaying the whole DDO

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

```
short number_of_attribute = cddo.dataCount();
short number_of_prop      = cddo.propertyCount();
short number_of_data_prop;
// list DDO properties
for (short k = 1; k <= number_of_prop; k++) {
    System.out.println( k + " Property Name = " + cddo.getPropertyName(k) +
        ",\t value = " + cddo.getProperty(k));
}
// list data-items and their properties
for (short i = 1; i <= number_of_attribute; i++) {
    System.out.println( i + " Attr. Name = " + cddo.getDataName(i) +
        ",\t value = " + cddo.getData(i));
    number_of_data_prop = cddo.dataPropertyCount(i);
    for (short j = 1; j <= number_of_data_prop; j++) {
        System.out.println( "\t" + j + " Data Prop. Name = " +
            cddo.getDataPropertyName(i,j) +
            ",\t value = " +
            cddo.getDataProperty(i,j));
    }
}
```

Working with XDOs

An XDO represents a single part in Enterprise Information Portal. To create an XDO for binary objects you use `DKBlobxx`, where `xx` is the suffix representing the specific server; for example, use `DKBlobDL` for Content Manager, `DKBlobOD` for OnDemand, or `DKBlobIP` for ImagePlus for OS/390. When you create a `DKBlobxx` object, you must pass it the datastore `DKDatastorexx`.

Using an XDO PID

To use an XDO to locate and store data, you must supply a PID for the `DKBlobxx`, using a `DKPidXDOxx`. An item ID and part ID are required for the `DKPidXDOxx`. For Relational Databases (RDB), the table name, column name and datapredicate are required for `DKPidXDOxx`.

Understanding XDO properties

Use the methods of the `DKBlobxx` 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, with Content Manager the following defaults are used:

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 Content Manager correctly, you must set SearchEngine, SearchIndex, and SearchInfo in the extension object DKSearchEngineInfoDL; refer to “Adding an XDO to be indexed by Text Search Engine” on page 73.

Programming tips

For Content Manager, VI400 and IP, 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 datapredicate 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 DKBlobxx to add the current content to a datastore. If you set part ID to 0, the system assigns an available part ID for it. You can retrieve the part ID value after add if you want to do some other operation with that object later.

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

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

Important: There are two situations where a valid part ID is required and you cannot set part ID to 0 for a Content Manager server:

1. Adding a part to be indexed by search manager
2. Adding a large object that will be divided into MAXPIECE size pieces

Using XDO as a part of a DDO or stand-alone

An XDO represents a single part object, if you have a DDO representing a document, which is a collection of part 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.

XDO as a part of DDO

The following example creates a DDO and an XDO as part of it:

```
// ----- create the DDO
DKPid pid = new DKPid();
pid.setObjectType(indexClassName);
DKDDO ddo = new DKDDO(dsDL, pid);
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE,
                new Short(DK_CM_DOCUMENT));
...
DKParts parts = new DKParts();

// ----- create the XDO
DKBlobDL axdo = new DKBlobDL(dsDL);
DKPidXDODL apid = new DKPidXDODL();
apid.setPartId(partId);
axdo.setPidObject(apid);
axdo.setContentClass(DK_DL_CC_GIF);
axdo.setAffiliatedType(DK_DL_BASE);
axdo.setContentFromClientFile(imageNames[i]);

// ----- add XDO to the DKParts collection
parts.addElement(axdo);
```

```

...
// ----- add DDO
dataId = ddo.addData(DKPARTS);
ddo.addDataProperty(dataId, DK_CM_PROPERTY_TYPE,
    new Short(DK_CM_COLLECTION_XDO));
ddo.setData(dataId, parts);
ddo.add();

```

The complete sample application from which this example was taken (TLoadSampleDL.java) is available in the CMBROOT\Samples\java\d1 directory. TLoadSampleDL.java shows more examples of XDO use.

Stand-alone XDO

All of the following examples are specific to Content Manager. For examples for RDBs and other connectors, please refer to the sample programs in the CMBROOT\Samples directory.

Examples of working with an XDO: The following examples illustrate using a stand-alone XDO.

Adding an XDO from the buffer: This example shows how to add an XDO from a buffer.

Requirement: You must know the existing XDO item ID to use this sample.

```

public class txdoaddDL
    implements DKConstantDL {

    public static void main(String[] args)
    {
        int    partId = 0;                //let system decide the partId
        String itemId = "CPPIORH4JBIXWIY0"; //existing itemId
        String fileName = "g:\\test\\cheetah.gif"; //a Windows file for add
        try
        {
            DKDatastoreDL dsDL = new DKDatastoreDL(); //required datastore
            dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD",""); //connect to datastore
            DKBlobDL axdo = new DKBlobDL(dsDL); //create XDO
            DKPidXDODL apid = new DKPidXDODL(); //create Pid
            apid.setPartId(partId); //set partId
            apid.setItemId(itemId); //set itemId
            axdo.setPidObject(apid); //setPid to XDO
            axdo.setContentClass(DK_DL_CC_GIF); //set ContentClass
            axdo.setContentFromClientFile(fileName); //set file content to buffer area
            axdo.add(); //add from buffer
            System.out.println("after add partId = " + ((DKPidXDODL)
                (axdo.getPidObject())).getPartId()); //display the partId after add
            dsDL.disconnect(); //disconnect from datastore and destroy
            dsDL.destroy(); }
        catch (DKException exc)
        {
            .... Handle exceptions
        }
    }
}

```

Adding an XDO from a file: This example adds an XDO from a file using the DKBlobDL class:

```

public class txdoaddfDL
    implements DKConstantDL {

    public static void main(String[] args)
    {

        int    partId = 19;                //partId 19 is not used yet

```



```

{
    int    partId = 17;                //partId of object
    String itemId = "CPPIORH4JBIXWIY0"; //existing itemId
    String fileName = "g:\\test\\choice.gif"; //file content to update
    try {
        DKDatastoreDL dsDL = new DKDatastoreDL();//required datastore
        ...
        // ---- connection 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.retrieve(); //retrieve the object
        //---- Set the file content to the buffer area
        axdo.setContentFromClientFile(fileName);
        axdo.update(); //update the object with buffer data
        axdo.retrieve("new.gif"); //retrieve content to a file
        axdo.del(); //delete object from datastore
        dsDL.disconnect(); //disconnect from datastore
        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();
    } }
}

```

Invoking an XDO function

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

```

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 will be 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());

    // ----- 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
}
}

```

Adding an XDO media object

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.

```

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());

    // ----- 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();
    }
}
}
}

```

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.

```

public class txdoDelVSDL implements DKConstantDL
{
    public static void main(String args[])
    {
        int partId = 45;
        String repType = "";
        String itemId = "K1A04EWBHVHJAV1D7";
        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);
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();
}
}
}
}

```

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.

```
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));
            }
            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 +
}
}
}

```

Adding an XDO to a storage collection

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

The following example uses `DKStorageManageInfoDL`, for a Content Manager server.

```

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 datastore
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

```

Refer to the following code samples in the CMBROOT\Samples\java\d1 directory for examples of adding search indexed objects and media objects to Content Manager:

- TxdoAddBsmsDL.java
- TxdosAddBsmsDL.java
- TxdoAddFsmsDL.java
- TxdosAddFsmsDL.java
- TxdomAddsmsDL.java

Changing the storage collection of an XDO

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

```

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

```

The complete sample application from which this example was taken (TxdoChgSmsCM.java) is available in the CMBROOT\Samples\java\d1 directory.

Importing XML documents

Enterprise Information Portal supports importing content from XML documents into Content Manager as DDOs and XDOs using the Java APIs. This feature makes it possible to import and store 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 EIP’s Java APIs. Once in Content Manager, you can do anything with the object that you could with any other Content Manager object.

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 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 two constructors: `public DKDDO(DKNPair xmlSource, int options)` and `Public DKDDO(DKNVPair xmlSource)`.

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

1. Remember you can only import into Content Manager.
2. XML files containing content for import must conform to the XML document type definition, shown below.
3. XML import is 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, you must store the content in XML documents that conform to ddo.dtd, which is located at CMROOT\samples\java\d1\ddo.dtd.

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

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

<!ELEMENT propertyCount (#PCDATA)>

<!ELEMENT property EMPTY>
<!ATTLIST property propertyId CDATA #IMPLIED
    propertyName CDATA #IMPLIED
    propertyValue CDATA #IMPLIED>

<!ELEMENT dataCount (#PCDATA)>

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

<!ELEMENT dataPropertyCount (#PCDATA)>

<!ELEMENT dataProperty EMPTY>
<!ATTLIST dataProperty propertyId CDATA #IMPLIED
    propertyName CDATA #IMPLIED
    propertyValue CDATA #IMPLIED>

<!ELEMENT dataValues (dataValueCount?, dataValue+)>

<!ELEMENT dataValueCount (#PCDATA)>

<!ELEMENT dataValue (#PCDATA | ddo | xdoRef)*>

<!ELEMENT xdoRef (xdoPid, xdoValue)>

<!ELEMENT xdoPid EMPTY>
<!ATTLIST xdoPid dsType CDATA #REQUIRED
    dsName CDATA #IMPLIED
    xdoType CDATA #REQUIRED
    partId CDATA #IMPLIED
    repType CDATA #IMPLIED
    pidString CDATA #IMPLIED>

<!ELEMENT xdoValue (contentType?, 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 searchEngineInfo EMPTY>
<!-- ATTENTION: searchEngineInfo searchEngine CDATA #REQUIRED
searchIndex CDATA #REQUIRED
searchInfo CDATA #REQUIRED -->

<!ELEMENT smsInfo EMPTY>
<!-- ATTENTION: 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.

```

<?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, see the following code samples in the `DK\Samples\java\d1\nt\` directory:

- `d1samp01.xml`
- `d1samp02.xml`
- `d1samp03.xml`
- `d1samp04.xml`
- `d1samp05.xml`

- dltypes01.xml.

Extracting content from different XML sources

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

Here are examples of each XML source:

XML from a file

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

XML from a buffer

```
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)

```
xmlSource = new DKNVPair("URL", "file:///d://myxml//dlsamp01.xml");
Int importOptions=0;
```

Importing XML content into Content Manager

The following example follows these basic steps:

1. Create a DKDDO and specify an XML source.
2. Create and connect to a datastore, in this case Content Manager.
3. Add the new DKDDO to the datastore, again, in this case Content Manager.

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

```
// ----- Construct a DDO by importing the XML document
xmlSource = new DKNVPair("FILE", "dlsamp01.xml");
int importOptions = DK_CM_XML_VALIDATION;
DKDDO ddo = new DKDDO (xmlSource, importOptions);
ds = new DKDatastoreDL();
// ..... connect to the datastore
ddo.setDatastore (ds);
// ----- Add the DDO to the datasure
ddo.add()
```

Creating 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 XDOs. You set the DKPARTS attribute when you create a DDO, as shown in the following example. The example text applies only to Content Manager.

```
DKDatastoreDL dsDL = new DKDatastoreDL();
DKParts parts = new DKParts(); // create new DKParts, collection of parts

DKBlobDL blob = new DKBlobDL(dsDL); // create new XDO BLOB
DKPidXDODL pid = new DKPidXDODL(); // create PID for this XDO object

pid.setPartId(5); // set part number to 5
```



```

pid.setPrimaryId("LN#U5K6ARLGM3DB4");// the item id this part belongs to
blob.setPidObject(pid);                // set the PID for the XDO BLOB
blob.setContentClass(DK_CC_GIF);       // set content class type GIF
blob.setRepType(DK_REP_NULL);          // set RepType 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 DKParts collection
//      collection as necessary
....
DKDDO ddo = new DKDDO();                // create a ddo
....                                  // sets some of its attributes

Object obj = new Short(DK_CM_DOCUMENT); // set the type to document DDO
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, obj);

// ----- 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_DATAITEM_TYPE_COLLECTION_XDO); // add type property
ddo.addDataProperty(data_id, DK_CM_PROPERTY_TYPE, obj);
obj = new Boolean(true);                // add nullable property
ddo.addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE, obj);
ddo.setData(data_id, parts);            // sets the attribute value

```

After you set DKPARTS as an attribute value of a DDO, the DDO owns it.

To get the parts from a DDO, use the following example:

```

data_id = ddo.dataId(DKPARTS); // get DKPARTS data ID
if (data_id == 0)              // handle parts not found
    throw new DKException(" parts data item not found");

DKParts pCol = (DKParts) ddo.getData(data_id); // get the parts collection

// ----- Create an iterator and process the part collection member one by one
if (pCol != null) {
    DKBlobDL blob;
    dkIterator iter = pCol.createIterator();
    while (iter.more()) {
        blob = (DKBlobDL) iter.next();
        if (blob != null) {
            blob.retrieve(); // retrieve the BLOB
            blob.open();     // display the BLOB using the viewer
            ....            // other processing
        }
    }
}

```

Creating 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 with DKPARTS, you set DKFOLDER when you create a DDO, as shown below:

```

DKDatastoreDL dsDL = new DKDatastoreDL();
...
// ----- Create a new DKFolder, collection of DDO
DKFolder folder = new DKFolder();

DKDDO member = new DKDDO(); // create the first member of this folder
...                          // set member DDO attributes and properties
folder.addElement(member);   // add member to the folder collection
....                          // create and add some more member DDO to the
....                          // DDO collection as necessary
DKDDO ddo = new DKDDO();     // create a folder DDO

```

```

.... // sets some of its attributes

Object obj = new Short(DK_CM_FOLDER); // set the type to folder DDO
ddo.addProperty(DK_CM_PROPERTY_ITEM_TYPE, obj);

// create DKFOLDER attribute and set it to refer to the DKFolder object
short data_id = ddo.addData(DKFOLDER); // add attribute "DKFolder"
obj = new Short(DK_CM_DATAITEM_TYPE_COLLECTION_DDO); // add type property
ddo.addDataProperty(data_id,DK_CM_PROPERTY_TYPE,obj);
obj = new Boolean(true); // add nullable property
ddo.addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE,obj);
ddo.setData(data_id, folder); // sets the attribute value

```

After you set DKFOLDER as an attribute of a DDO, the DDO owns it.

To get the folder from a DDO, use the following example:

```

data_id = ddo.dataId(DKFOLDER); // get DKFOLDER data id
if (data_id == 0) // handle folder not found
    throw new DKException(" folder data item not found");

DKFolder fCol = (DKFolder) ddo.getData(data_id); // get the parts collection

// create iterator and process the DDO collection member one by one
if (fCol != null) {
    DKDDO item;
    dkIterator iter = fCol.createIterator();
    while (iter.more()) {
        item = (DKDDO) iter.next();
        if (item != null) {
            item.retrieve(); // process the member DDO
            .... // other processing
        }
    }
}

```

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.

Using sequential collection methods

DKSequentialCollection provides methods for adding, retrieving, removing, and replacing its members. In addition, it also has a sort method, see “Sorting the collection” on page 47 for more information. The following example illustrates how to add a new member to a collection:

```

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

```

The addElement method takes an object as the parameter.

Using the sequential iterator

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

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. The following example shows using an iterator:

```
dkIterator iter = sq.createIterator(); // create an iterator for sq
Object member;
while(iter.more()) {                // While there are more members
    member = iter.next();            // get the current member and
                                    // advance iter to the next member
    System.out.println(member);     // display it, if you want to
    ....                             // do other processing
}
```

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

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

Using DKSequentialIterator allows you to perform some operations on the current member before moving to the next member, such as replacing a member with a new one, or removing it.

```
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
....
```

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:

```
....
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
```

Check the removal condition to avoid skipping the next member after removing the current one.

Sorting the collection

Use the sort method to sort collection members based on a specified key in either ascending or descending order. You control the actual sort function by creating a sort object containing the function and the desired order and passing it to sort. The interface for sort objects is defined in dkSort.java. The following example illustrates how to sort a collection of DDOs based on each DDO's item ID:

```
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
....
```

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 grouping of objects that results from a *federated search*. Each DKResults object contains the results of the search submitted to a specific content server. A federated collection can contain other nested collections.

To iterate over the DKResults in a federated collection, use dkIterator or DKSequentialIterator. Then create another dkIterator for each DKResults object to process the data objects as appropriate for that content server. Alternatively, you can use a federated iterator, dkFederatedIterator, to iterate over all the members in the collection, regardless of which content server the result came from.

You cannot query a federated collection.

Figure 10 illustrates the structure and behavior of DKFederatedCollection.

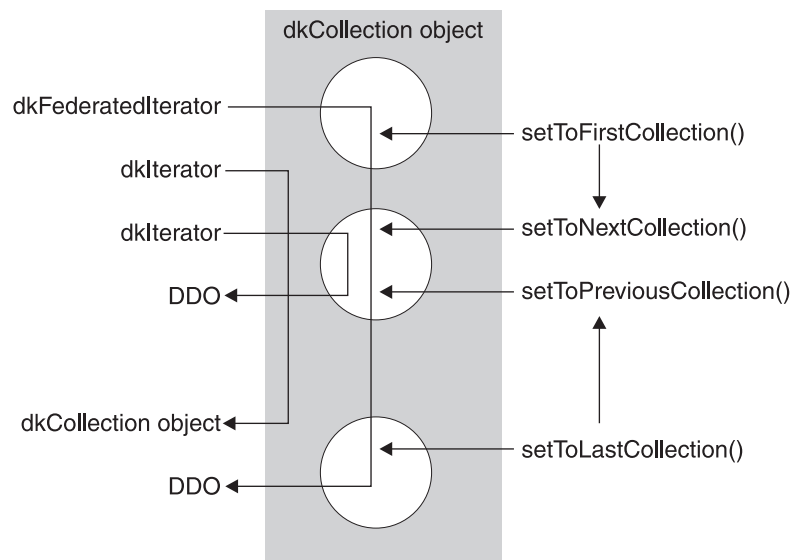


Figure 10. DKFederatedCollection structure and behavior

In Figure 10, the oval represents the DKFederatedCollection containing the DKResults objects, represented as circles. The dkFederatedIterator traverses across member boundaries and returns a DDO for each data object.

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 method in dkFederatedIterator sets the position to the first DDO object of DKFederatedCollection. In this case, it is the first element of the first DKResults collection object. If the setToNextCollection method is invoked, it sets the iterator position to the first DDO object of the second DKResults collection.

The `setToLastCollection` method 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. If you call the `setToPreviousCollection` method, it sets the iterator position to the last DDO of the previous `DKResults` collection.

Querying a content server

You query a content server datastore and receive results in a `dkResultSetCursor` or `DKResults` object. First you create a query object to represent your query; then you call the `execute` or `evaluate` method of the query object. Using the APIs of the content server, 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.

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. Content Manager supports image query.

A content server uses two methods for running a query: `execute` and `evaluate`. The `execute` method returns a `dkResultSetCursor` object; `evaluate` returns a `DKResults` object. The `dkResultSetCursor` object is used to handle large result sets, as well as to delete and update the current position of the result set cursor. Use the `fetchNext` method to fetch 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 57.

`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 57 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 datastore cursor. You can use it for large result sets because the `DKDDOs` it contains are fetched one at a time. It can also be used to rerun a query 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.

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 datastore.

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`.

The condition of the query is to search for all documents or folders where the attribute DLSEARCH_DocType is greater than 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.

```
String cmd = "SEARCH=(INDEX_CLASS=GP2DLS2," +
             "MAX_RESULTS=5," +
             "COND=(DLSEARCH_DocType <> null));" +
             "OPTION=(CONTENT=YES;" +
             "TYPE_QUERY=DYNAMIC;" +
             "TYPE_FILTER=FOLDERDOC)";
```

The example specifies that a Content Manager server use dynamic SQL for this query and that all folders and documents be searched. 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.

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.

```
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)";
```

Executing a parametric query

After you have a query string you create the query object. The DKDatastorexx 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 a Content Manager server. Once the query is executed, the results are returned in a DKResults collection.

```
// ----- Create the datastore, the query object, and the results set
DKDatastoreDL dsDL = new DKDatastoreDL();
dkQuery pQry = null;
DKResults pResults = null;
DKNameValuePair 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\d1 directory.

Executing a parametric query from a content server

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

```
// ----- Create the datastore and cursor
DKDatastoreDL dsDL = new DKDatastoreDL();
dkResultSetCursor pCur = null;
DKNameValuePair 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\d1` directory.

Evaluating a parametric query from a content server

The `DKDatastorexx` 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 a Content Manager content server.

```
// ----- 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)";

DKNameValuePair parms[] = null;
DKDDO item = null;
// ----- Create the datastore and connect
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();
```

Using text query

On a Content Manager content server, 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.

```
String cmd = "SEARCH=(COND=(UNIX OR member));" +
            "OPTION=(SEARCH_INDEX=TMINDEX; MAX_RESULTS=5)";
```

You can specify that a search is performed against more than one index. The following example shows how to specify a query on two indexes.

```
String cmd = "SEARCH=(COND=(UNIX OR member));" +
            "OPTION=(SEARCH_INDEX=(TMINDEX, INDEX2); MAX_RESULTS=5)";
```

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.

Executing a text query

After you have a text query string you create the query object. The `DKDatastorexx` 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:

```
// ----- Create the datastore; declare query and the results
DKDatastoreTS dsTS = new DKDatastoreTS();
dkQuery pQry = null;
DKResults pResults = null;
DKNameValuePair 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\d1` directory.

Executing a text query from the datastore

The `DKDatastorexx` used to represent a content server provides a method to execute a query. The results are returned in a `dkResultSetCursor` object. The following example shows how to execute a text query against the Content Manager datastore:

```
// ----- Create the datastore; declare query and the results
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNameValuePair 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\d1 directory.

Evaluating a text query from the datastore

The DKDatastorexx 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 the Content Manager datastore:

```
// ----- 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();
```

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.

```

// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNameValuePair parms[] = null;
// ----- Connect to the content server
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)
    {
        // ----- 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();

```

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.

```

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();
            }
            ...
        }
        // 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();

```

Using the result set cursor

The `dkResultSetCursor` is a datastore 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 query submitted to the datastore.

Important: When you are finished using the cursor, call the `destroy` method to free the memory it used.

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.

```

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

```

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.

```

// ----- 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();
    }
}

```

Another way to do this is:

```

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();
    }
}

```

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

```

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 position
    if (item != null) { // (relative)
        i = pCur.getPosition();
    }
}

```

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 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.

```

DKSequentialCollection seqColl = new DKSequentialCollection();
boolean fItems = false;
int 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:

```

// ----- Create 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 result
DKResult rs = (DKResults) pq.result();

```

The results are in `rs`, which is a `DKResults` object.

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:

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

```
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);
```

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 specific content servers

You use the dkDatastore classes to define an appropriate datastore for the content servers in your application. The datastore is the primary interface to the Enterprise Information Portal. Each content server has a separate datastore class.

Use the `DKDatastorexx` classes, where `xx` is the suffix of the specific content server, to create a datastore. Table 4 shows these classes.

Table 4. Server type and class name terminology

Server type	Class name
Content Manager	<code>DKDatastoreDL</code>
OnDemand	<code>DKDatastoreOD</code>
VisualInfo for AS/400	<code>DKDatastoreV4</code>
ImagePlus for OS/390	<code>DKDatastoreIP</code>
Domino.Doc	<code>DKDatastoreDD</code>
Domino Extended Search	<code>DKDatastoreDES</code>
Relational Database	<code>DKDatastoreDB2</code> , <code>DKDatastoreJDBC</code> (for Java) <code>DKDatastoreDJ</code>
Information Catalog	<code>DKDatastoreIC</code>

When creating a datastore for a content server, implement each of the following classes and interfaces:

dkDatastore

To represent the content server and manage the connection, communications, and execution of datastore commands. `dkDatastore` is an abstract version of the query manager class. It supports the `evaluate` method.

dkDatastoreDef

To use the methods to access items stored in the content server, as well as to create, list, and delete its entities. It maintains a collection of `dkEntityDefs`. Examples of concrete classes for this interface are:

- `DKDatastoreDefDL`
- `DKDatastoreDefOD`

dkEntityDef

To use the methods to access entity information and to create and delete entities and attributes. The methods of this class support accessing multiple-level entities. If a datastore does not support subentities, they generate `DKUsageError` objects. If a datastore supports multiple-level entities, you must implement methods to overwrite the exceptions for subclasses for these datastores. Examples of concrete classes for the `dkEntityDef` interface are:

- `DKIndexClassDefDL`
- `DKAppGrpDefOD`

The class hierarchy for an entity definition is illustrated in Figure 11 on page 61:

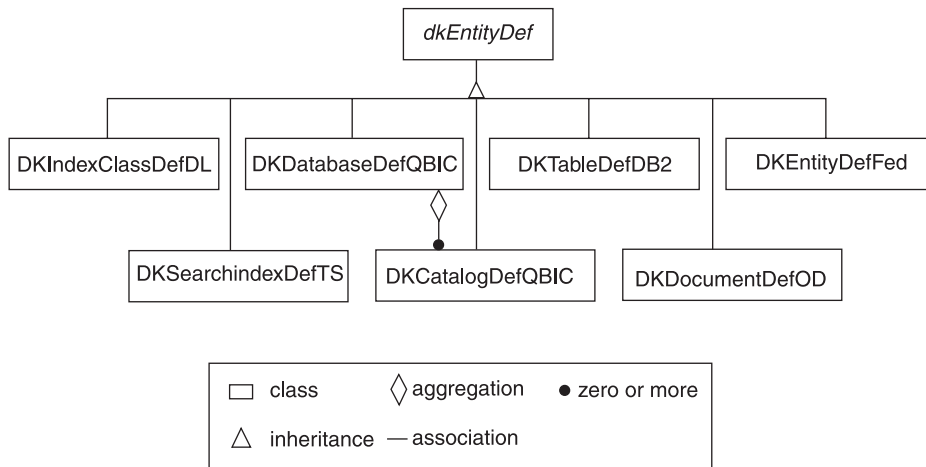


Figure 11. Class hierarchy

dkAttrDef

To define methods to access attribute information and to create and delete attributes. Examples of concrete classes for dkAttrDef are:

- DKAttributeDefDL
- DKFieldDefOD

dkServerDef

To defines methods to access server information. Examples of concrete classes for dkServerDef are:

- DKServerDefDL
- DKServerDefOD

dkResultSetCursor

To create a datastore cursor that manages a collection of DDO objects. To use the addObject, deleteObject, and updateObject methods, set the datastore option DK_CM_OPT_ACCESS_MODE to DK_CM_READWRITE.

dkBlob

To declare 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 12 on page 62:

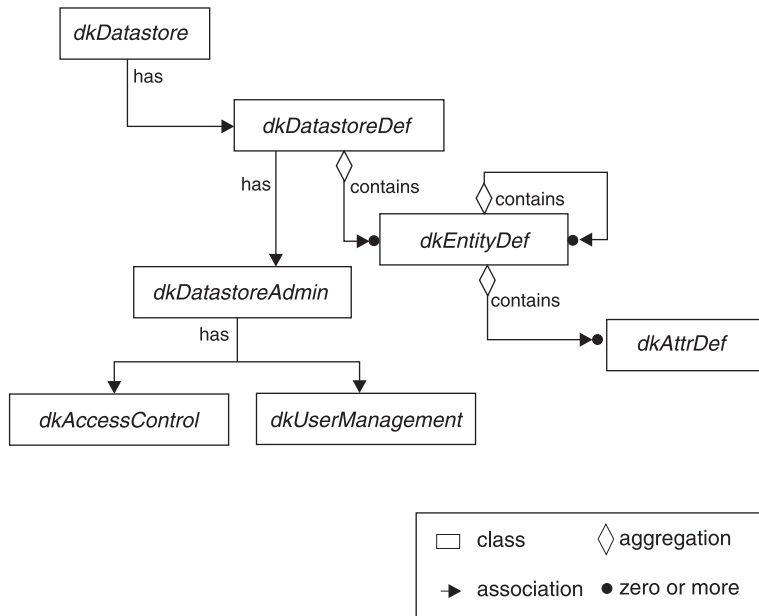


Figure 12. Data definition class hierarchy

For more information on dkDatastore and other common classes, see “Developing custom content server connectors” on page 125.

Working with 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

When working with large objects in Content Manager, you must set the MAXPIECE variable and Java heap size.

MAXPIECE environment variable: Content Manager uses the MAXPIECE variable to define the largest object to be processed as a whole in megabytes. When MAXPIECE is set, Content Manager stores an object larger than the setting as a sequence of objects whose size is equal to or less than MAXPIECE. If you do not set the variable, the object is always treated as a single large object.

In Windows, set the MAXPIECE environment variable as a user environment variable in the system properties. Use an integer value to indicate the size in megabytes.

In AIX, set the size in your profile; for example export MAXPIECE=4 sets MAXPIECE to 4 megabytes.

Setting Java heap size: 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.

Using DDOs to represent Content Manager's data

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:

```
DKDDO addo = new DKDDO(dsDL, pid);
Object obj = addo.getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
if (obj != null) {
    short item_type = ((Short) obj).shortValue();
}
```

`item_type` is set 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 27 and “Getting properties” on page 28.

PID: The PID contains some information specific to Enterprise Information Portal. The object type indicates the index class of the DDO; the item ID holds the ID of the associated item from the datastore. See “Creating a persistent identifier (PID)” on page 27.

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 DKParts 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.

For more information on creating and processing a DKParts object see, “Creating, updating, and deleting documents or folders” on page 64, “Retrieving a document or folder” on page 67, and “Creating and using the DKPARTS attribute” on page 44.

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.

Representing folders: A DDO representing a folder has the property `DK_CM_PROPERTY_ITEM_TYPE` set to `DK_CM_FOLDER`. The PID contains the index class name as the object type. The PID ID is the same as the item 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 datastore.

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 67, and “Creating and using the DKFOLDER attribute” on page 45.

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 the datastore, you create a DDO, setting all of its attributes and other information, except its item ID. The item ID is assigned and returned by the datastore. Some of the previous examples are combined in the following example:

```
// ----- 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
....                             // 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 property
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

```

In the preceding example, the last step created a document in the datastore with the information. When a document DDO is added to a datastore, 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 datastore as a component of a folder. The folder contains the table of contents of its members, which are existing documents and folders, so create all folder members in the datastore before adding a folder DDO.

You can add the same document to a different Content Manager content server. 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 indexclass
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

```

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

For example:

```

// ----- Update the value of attribute Title
String newTitle = "Accident Report";
short data_id = ddo.getDataByName("Title");
ddo.setData(data_id, newTitle);
ddo.update();

```

After the call to the update method, the value of the attribute Title in the datastore 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 datastore.

The document must already exist in the content server.

Adding and removing a member: The following example illustrates adding parts:

```
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
```

To remove newPart from the collection and the datastore, you would use:

```
parts.removeMember(addo, newPart);
```

The removeMember method in DKParts actually deletes the persistent copy of the part from the datastore.

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 datastore. 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:

```
....
// ----- 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
....
```

Updating folders: You represent the collection of documents and folders within a folder using a DKFolder object. In the datastore, 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 datastore.

Adding and removing a member: The following example illustrates adding another document or folder DDO to a folder DDO:

```
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);

```

Both newMember and folderDDO must exist in the datastore for another document or folder to be added to it.

Similarly, to remove newMember from the collection and the datastore use the following example:

```
folder.removeMember(folderDDO, newMember);
```

Important: Removing a member from a folder only removes that member from the folder table of contents.

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 datastore. 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.

```
ddo.del();
```

The DDO must have its item ID and object type set, and have a valid connection to a datastore.

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 datastore later, in the application. See “Creating a document” on page 64 for more information.

Retrieving a document or folder

To retrieve a document from a DKDatastoreDL representing a Content Manager content server, you must know its index class name and item ID. You also must associate the DDO with a datastore and establish a connection.

```

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

```

After a call to retrieve, attribute values of the DDO are set to the values 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, it is not desirable to retrieve all of them at once. Parts are explicitly retrieved as needed.

If the DDO is the result set of a parametric query with the option CONTENT=NO, the DDO is empty (does not have the attribute values). However, all information needed to retrieve it is already set.

Retrieving parts: After you retrieve a DDO, you can retrieve its parts stored in the DKPARTS attribute, as follows:

```
DKParts parts = (DKParts) ddo.getDataByName(DKPARTS);
```

This example assumes that the DKPARTS attribute exists; an exception is thrown if it does not exist. See “Retrieving a folder” 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 and using the DKPARTS attribute” on page 44.

```
// ----- 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
        }
    }
}
```

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:

```
blob.retrieve();
```

Retrieving a folder: You retrieve a folder DDO in the same way as you retrieve a document DDO. After being retrieved, the folder DDO has all of its attributes set, including the attribute, DKFOLDER, which 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 by using:

```
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
        }
    }
}
```


For more information, see “Creating and using the DKFOLDER attribute” on page 45 .

Understanding text searching (Text Search Engine)

You can use a variety of query types with the Text Search Engine. Using the query results returned from the text search, the item ID, part number, and ranking information you can create an XDO to retrieve the document from a 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 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 methods. You can query this datastore. Refer to “Loading data to be indexed by Text Search Engine” on page 74 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:

```
String cmd = "SEARCH=(COND=(www OR 'web site'));" +  
            "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Free text query: A free text query is made up of words, phrases, or sentences enclosed in braces (`{b}`). 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:

```
String cmd = "SEARCH=(COND={web site});" +  
            "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:

```
String cmd = "SEARCH=(COND=(IBM AND UNIX {web site}));" +  
            "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:

```
String cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}));" +  
            "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Proximity queries require 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, but also supports single-byte character set (SBCS) languages. Enclose all double-byte characters in single quotes ('). Be 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.

```
String cmd = "SEARCH=(COND=($CCSID=850,LANG=6011,MATCH=1$ " +  
            "'IBM marketing'))"; +  
            "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Make sure that the text search datastore 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 with 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.

DKDatastoreTS does not have a property item type as a DKDatastoreDL object. 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 attributes:

DKDLITEMID

The item ID for the item containing this text part. 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 part from the content server.

DKREPTYPE

The RepType (representation type) 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 indicating the relevance of this part to the results of a text query. A higher rank means a better match. See Text Search Engine Application Programming Reference for further information.

DKDSIZE

An integer number representing word occurrences; this is returned only for boolean queries. See Text Search Engine Application Programming Reference for further information.

DKRCNT

An integer number representing number of boolean search matches. See Text Search Engine Application Programming Reference for further information.

The PID for a text search DDO contains the following information:

datastore type

TS

datastore 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 methods for connecting and disconnecting. Normally, you create a DKDatastoreTS object, connect to it, run a query, then disconnect when done. The connect method has several signatures, providing different ways to connect. The following example shows the one way to connect using the text search server TM as the datastore name.

```
// ----- 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\d1 directory.

The following example shows another way to connect 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 connect method 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:

```
dsTS.connect("TM", "", "", "LIBACCESS=(LIBSRVR2, FRNADMIN, PASSWORD)");
```

The final parameter includes the connect string and can be used to pass a sequence of parameters in one string.

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 for longer than necessary.

Getting and setting text search options: Text search provides some options that you can set or get using its methods. The following example shows how to set and get the option for a text search character code set. See the online API reference for the list of options and their descriptions.

```
DKDatastoreTS dsTS = new DKDatastoreTS();
Integer input_option = new Integer(DK_TS_CCSSID_00850);
Integer output_option = null;

dsTS.setOption(DK_TS_OPT_CCSSID, input_option);
output_option = (Integer) dsTS.getOption(DK_OPT_TS_CCSSID);
```

The output_option is an object, but is usually cast to an Integer. .

Tips:

- You must use the search options CCSID and LANG together. The default CCSID and LANG are specified by the DKDatastoreTS options, DK_OPT_TS_CCSSID and DK_OPT_TS_LANG. Refer to the online API reference for the list of the datastore 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 69.
- 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 method to list the text search servers that it can connect to. The following example shows how to retrieve the list of servers.

```
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 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, and use them to establish a connection.

The complete sample application from which this example was taken (TListCatalogTS.java) is available in the CMBROOT\Samples\java\d1 directory.

Listing schema: You can use methods in DKDatastoreTS to list the schema. The following example shows how to retrieve the list of indexes. Because you are using text search, the indexes are text search indexes.

```
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 list of indexes returned as a DKSequentialCollection object containing DKSearchIndexDefTS objects. After you get an individual DKIndexTS object, you can retrieve information about the index, such as its name and library ID; then you can use the name to form a query.

The complete sample application from which this example was taken (TListCatalogTS.java) is available in the CMBROOT\Samples\java\d1 directory.

Indexing XDOs by search engine: Before you can search data item using the Text Search Engine, you must index them, that is, have the Text Search Engine create an index of the words in them. In a similar way, you must index data item before searching them using image search. If you want to index content using Text Search Engine, the values of SearchEngine, SearchIndex and SearchInfo are required.

The value of the SearchIndex property is a combination of two names: the search service name and search index name. For example, if you have defined a text search server named TM in the EIP administration 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 79).

Refer to LoadSampleTSQBICDL.java and LoadFolderTSQBICDL.java the CMBROOT\Samples\java\d1 directory for examples of how to load data, or create a folder and load data.

Adding an XDO to be indexed by Text Search Engine: The following example illustrates add an XDO that is to be indexed:

```
// ----- 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 (...)
```

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.

Refer to TListCatalogDL.java and TListCatalogTS.java in the CMBROOT\Samples\java\d1 directory for examples of setting up the environment. Before running the samples, update them with your server, user ID, and so forth.

To create parts in Content Manager that are indexed by the Text Search Engine, refer to "Working with XDOs" on page 29.

After the data is loaded into Content Manager, place the documents on the document queue by calling the wakeUpService method in the DKDatastoreDL. This method takes a search engine name as a parameter. Then use the Content Manager text search administration window to perform the indexing. After the indexing is complete, you can perform queries against Text Search Engine.

For more information on text search administration, refer to the *System Administration Guide*.

Using text structured document support: Text structured documents are composed using a text structure. A document model defines the text structure. For example, an HTML file contains tags that denote a certain structure. Text Search Engine can perform searches on words or phrases between the structure elements, such as HTML tags.

You can perform text queries on structured documents as follows:

1. Create a document model. The document model contains sections; each section contains the section name and the document tag used; for example, the model for HTML would look like the following:

```
<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 the text search index that uses the document model in Content Manager.
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.

The following example shows how to list the document models defined in the system:

```
// ----- 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\d1 directory.

The following example shows how to create a document model:

```
// ----- 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("SAMPITITLE");
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPORPBODY");
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\d1 directory for more examples.

The following example shows how to create and set the indexing rules for a text search index that uses a document model:

```

// ----- 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("SAMPITITLE");
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);

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\d1 directory.

The following example shows how to start the indexing process, which is asynchronous. You can start the indexing process and check the indexing status using system administration.

```

// ----- 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\d1 directory.

Refer to TCheckStatusTS.java in the CMBROOT\Samples\java\d1 directory for an example of checking status, including checking whether a queued request has been moved from the scheduled document queue to the primary or secondary queues. If an indexing error occurs you can check the imldiag.log file in the text search instance directory. For more information on the imldiag.log, see the Text Search Engine Application Programming Reference.

The following example shows how to execute a structure document text query based on the document model and the text search index defined above.

```
// ----- Create the datastore
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNameValuePair 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\d1 directory.

Searching images by content

Users can use IBM Image Search server to search for images in a database by specifying the type of image or by providing an example image.

Figure 13 on page 79 shows a window in a sample application that connects to the image search server. Users enter fuzzy search criteria, for example: colors, layout, and patterns. The application searches for matching images in the database. The image search server uses query by image content (QBIC) technology to support searches based on similar colors, layouts, and patterns.

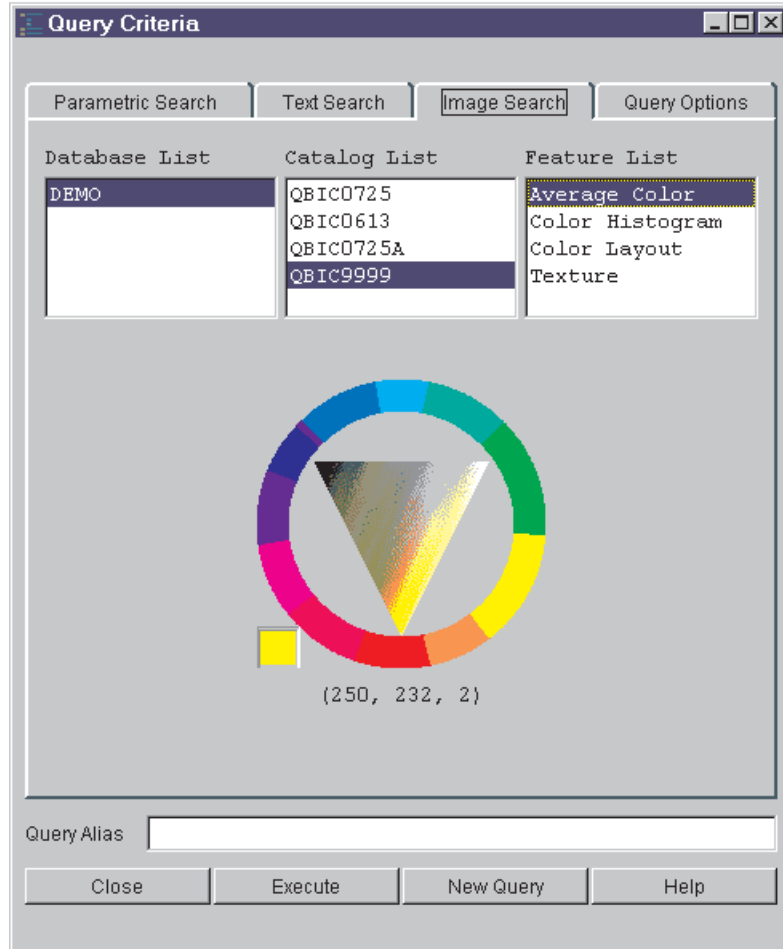


Figure 13. Image search sample client

Understanding image search terms and concepts: This section describes the image search components: the server, databases, and catalogs, as well as the *features*, the searchable visual characteristics of images, and the relationship of the image search server to Content Manager.

Understanding image search servers, datastores, and catalogs: Content Manager uses an image search server to query images. Content Manager applications store image data items in the object server; the image search server analyzes images and stores the image information.

A datastore defined by a DKDatastoreQBIC object represents the image search server. The image search server does not actually store images. It indexes the images that are stored in the Content Manager server to support searches on that image. 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 datastore. However, the datastore for image search does not support add, update, retrieve, and delete operations. Figure 14 on page 80 shows an example of an image search server.

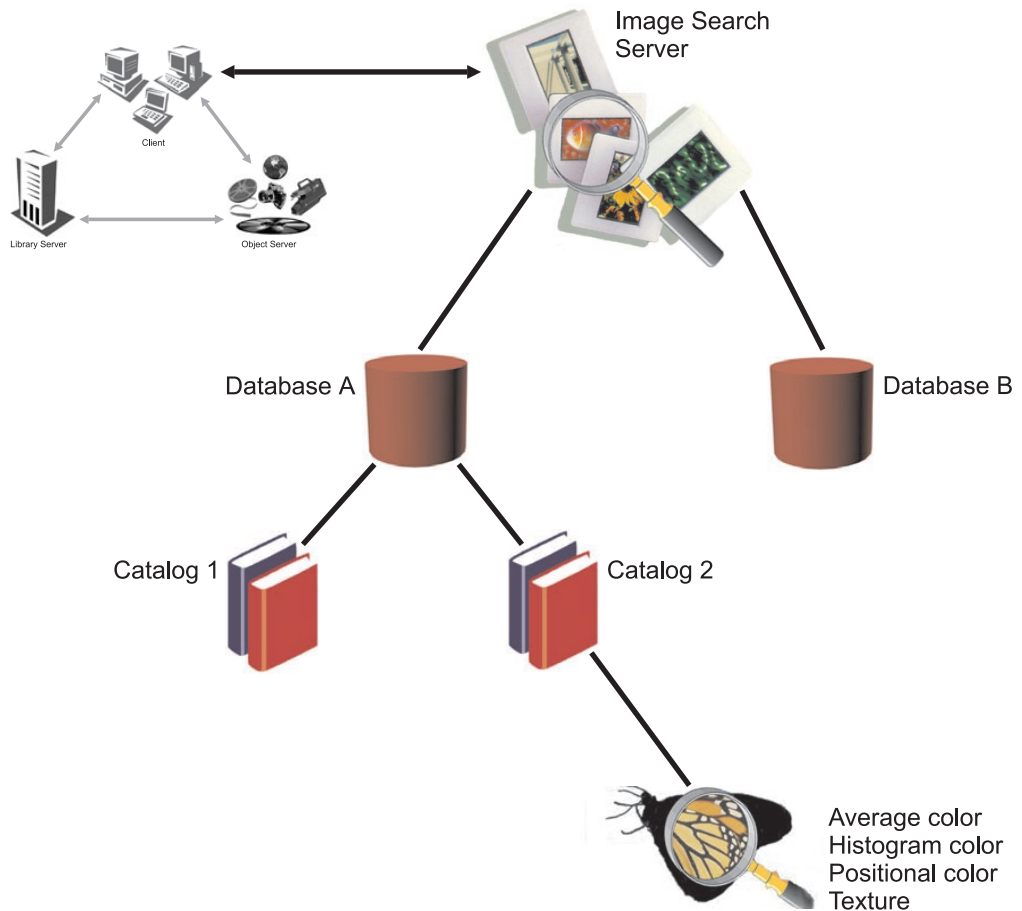


Figure 14. An image search server in a Content Manager system. The image search server communicates with the other Content Manager components through the clients.

The image search server uses a database to hold one or more catalogs, which contain information about the visual features of images. A single server can contain one or more databases. Each catalog stores information about one or more of the four image search features:

- Average color
- Histogram color
- Positional color
- Texture

Understanding image search features: The four image search features and their purposes are defined in this section:

Average color This is the sum of the color values for all pixels in an image divided by the number of pixels in the image. Images with similar predominant colors have similar average colors. For example, images that contain equal portions of red and yellow will have an average color of orange.

You use average color when you want to search for images with a predominant color. You specify the average color using the feature name `QbColorFeatureClass`.

The following example shows a query string to search for all images based on a histogram of three colors: 10% red, 50% blue, and 40% green.

```
String cmd = "QbColorHistogramFeatureClass ";  
cmd += "histogram=<(10, 255, 0, 0), (50, 0, 255, 0),  
(40, 0, 0, 255)>";
```

Histogram color

This is the percentage of color distribution in 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.

You use histogram color when you want to search for images that contain a specific variety of colors. You specify the histogram color using the feature name `QbColorHistogramFeatureClass`.

Example: 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)>
```

Positional color (color layout)

This is 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.

You use positional color when you want to search for images that has a particular average color in a particular area. You specify the positional color using the feature name `QbDrawFeatureClass`. For example, you can search for average color red and limit the returned matches to five:

```
QbColorFeatureClass color=<255,0,0> and max=5
```

The following example shows a query string to search for all images based on color layout described by an image in a file on the client:

```
String cmd = "QbDrawFeatureClass file=<client,  
\"\\patterns\\pattern1.gif\">";
```

Texture

This is a measure of the coarseness, contrast, and direction of an image. Coarseness indicates the size of repeating items in an image. Contrast identifies the brightness variations in an image. Direction 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.

You use texture when you want to search for images that have a particular pattern. You specify the texture using the feature name `QbTextureFeatureClass`.

A query based on texture consists of a feature name and its value. The following example shows a query string to search for all images based on the texture value provided by an image in a file on the client:

```
String cmd = "QbTextureFeatureClass file=<client,  
\"\\patterns\\texture2.gif\">";
```

You can specify a query with multiple features. The following example shows a query string to search for all images based on an

average color and a texture value. The average color is red and is weighted twice that of the texture. The texture value is provided by an image in a file on the client.

```
String cmd = "QbColorFeatureClass color=<255, 0, 0> weight=2.0 and ";  
cmd += "QbTextureFeatureClass file=<client,  
  \\\"patterns\\texture2.gif\\\">";
```

Using image search applications

Image search clients 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 15 shows an example of the client and retrieve images.

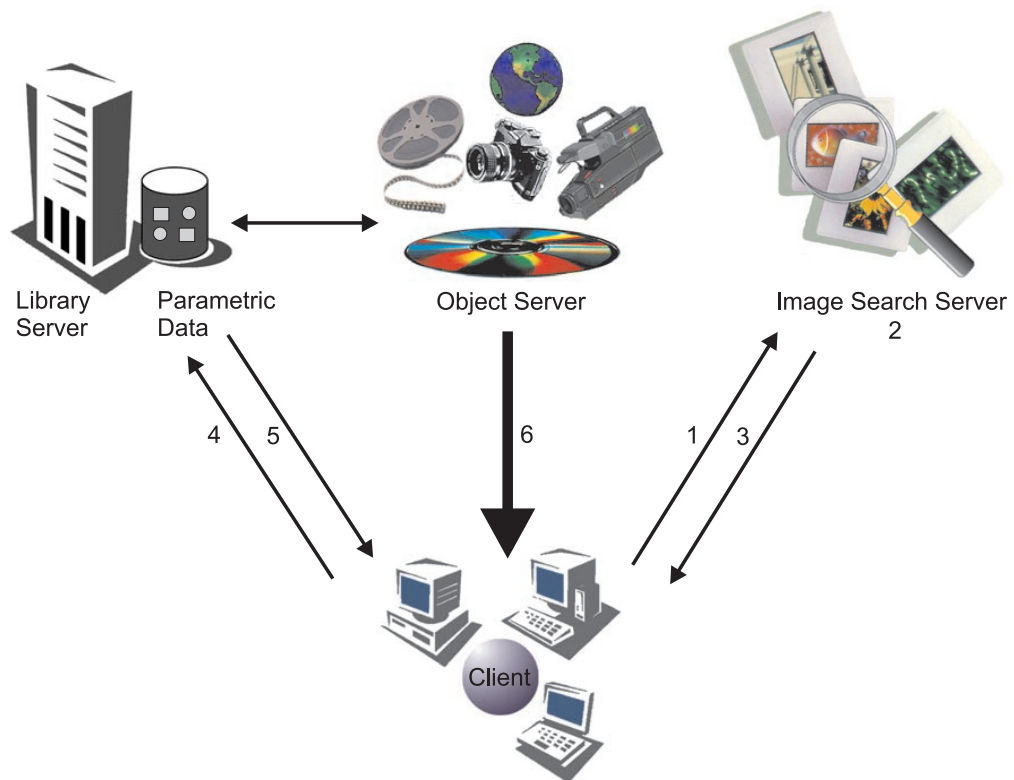


Figure 15. How image search clients search and retrieve images

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, and 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. Table 5 describes image search query feature names and values.

Table 5. Image search query valid feature names and values

Feature name	Values
QbColorFeatureClass or QbColor	<p>color = < rgbValue , rgbValue , rgbValue > where rgbValue is an integer from 0 to 255.</p> <p>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.</p>
QbColorHistogramFeatureClass or QbHistogram	<p>histogram = < histList > where histList consists of one or more histClause separated by a comma (,).</p> <p>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.</p> <p>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.</p>
QbDrawFeatureClass or QbDraw	<p>description = < " descString " > where descString is a special encoded string describing a picker file. Format of the description string:</p> <ol style="list-style-type: none"> 1. D,w,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. The colon character (:) is used as a separator. <p>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.</p>

Table 5. Image search query valid feature names and values (continued)

Feature name	Values
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.

The query string: 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 6 on page 85 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:

```

Image search query 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 6 on page 85 describes feature names and values.

Table 6. Image search query: valid feature values

Feature name	Values
QbColorFeatureClass or QbColor	<p>color = < rgbValue , rgbValue , rgbValue > where rgbValue is an integer from 0 to 255.</p> <p>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</p>
QbColorHistogramFeatureClass or QbHistogram	<p>histogram = < histList > where histList consists of one or more histClause separated by a comma (,).</p> <p>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.</p> <p>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.</p>

Table 6. Image search query: valid feature values (continued)

Feature name	Values
QbDrawFeatureClass or QbDraw	<p>description = < " descString " > where descString is a special encoded string describing a picker file. Format of the description string:</p> <ol style="list-style-type: none"> 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. <p>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"</p> <p>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.</p>
QbTextureFeatureClass or QbTexture	<p>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.</p>

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 methods for connecting and disconnecting to the datastore. After creating a datastore, you connect to it. After you finish working with the datastore, you disconnect from it. The following example shows how to connect to an image search server named QBICSRV using the user ID QBICUSER and the password PASSWORD.

```
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\d1 directory

Your application uses the image search connection to connect to an image search server.

After connecting, call methods that access the image search datastore. To use the image catalogs, you must first call the `openCatalog` method to open a catalog for processing. Call the `closeCatalog` method after processing is done. The following example shows how to connect, open a catalog, close the catalog, and disconnect:

```
// ----- 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();
```

Listing image search servers

`DKDatastoreQBIC` provides a method for listing the image search servers that it can connect to. It returns a `DKSequentialCollection` object that contains `DKServerInfoQBIC` objects. After you retrieve an individual `DKServerInfoQBIC` object, you can get the server name, the host name, and the port number. The following example shows how to retrieve the list of servers:

```
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\d1` directory.

Listing image search databases, catalogs, and features

`DKDatastoreQBIC` provides a method for listing all the image search databases, catalogs and features on an image search server. The list is returned as a `DKSequentialCollection` object containing `DKServerDefQBIC` objects. The database is `DKDatabaseDefQBIC` objects, catalogs are `DKCatalogDefQBIC` objects and features are `DKFeatureDefQBIC` objects. The following example shows how to retrieve the list of databases, catalogs, and features

```
// ----- 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\d1 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:

datastore type

QBIC

datastore name

The server name used to connect to the datastore

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: DKDatastoreQBIC provides a method to create a query object, an instance of dkQuery. You can then use the execute method of dkQuery to

run the query and obtain the results. After you run a query, the results are returned in a DKResults collection. The following example shows how to create an image query object and run a query:

```
// ----- Generate a query string; then create the datastore and connect
String cmd = "QbColor color=<255, 0, 0>";
DKNameValuePair 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 (SampleIQueryQBIC.java) is available in the CMBROOT\Samples\java\d1 directory.

Running an image query from the datastore: As an alternative you can use the execute method of DKDatastoreQBIC to run a query. The results are returned in a dkResultSetCursor object. The following example shows how to run an image query.

```
// ----- Generate a query string; then create the datastore and connect
String cmd = "QbColorFeatureClass color=<255, 0, 0>";

DKNameValuePair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");
// ----- 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\d1 directory.

Evaluating an image query from the datastore: DKDatastoreQBIC also provides an evaluate method you can use to run a query. You cast the results to a DKResults collection. The following example shows how to evaluate an image query from the datastore:

```
// ----- Generate a query string; then create the datastore and connect
String cmd = "QbColorFeatureClass color=<255, 0, 0>";
DKNameValuePair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");

// ----- 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();
```

Using the image search engine

As described in the previous section, you can use the image search engine to specify a query based on one of the following features: average color, color percentages, color layout, and textures. You can specify multiple features in a query. The query results contain the item ID, part number, representation type, and ranking information. This information can be used to create an XDO that you use to retrieve the image contents.

Loading data to be indexed for image search: To load data into Content Manager 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 holds a collection of image search catalogs. A catalog holds data about the visual features of images. The image search features must be added to the catalog for indexing. It is recommended that you add all of the 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 you load data into Content Manager, select **Process Image Queue** in the system administration program to place the images in the 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 method of the DKBlobDL class.

```
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
}

```

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 from a previous parametric or text query. The final result is an intersection between the scopes and the results of each query. Therefore, be careful in formulating the query and including scopes so that the individual query results intersect and the result of the combined query is useful.

If there is at least one parametric and one text query, the resulting DDO returned in DKResults has the attribute DKRANK, which signifies the highest rank of the matching part belonging to the document.

Important: 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 with one parametric and one text query, without scope, create a combined query object and pass the two queries as parameters to be executed by the combined query. For example:

```

// ----- Create a 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
}

```

The final if statement ensures 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 one of the query parameters. The following example illustrates using a DKResults object to function as a scope for a combined query: :

```

// ----- 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 of 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);
....

```

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 DDO in the result set has the attribute DKRANK. This attribute is not stored in the datastore but is computed each time by the Text Search Engine. The value of the DKRANK is the highest rank of the parts 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 parametric query. This is also true for text queries.
- The query option "MAX_RESULTS=nn" limits the number of results returned. If you set it to 10, for example, it means you only want the 10 results of highest rank. Usually, this option is more applicable to text queries, because the result is sorted in descending order by rank.

The meaning of "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 a parametric and text query, do not specify this option for the parametric query.

Understanding the Content Manager workflow and Content Manager workbasket functions

This section describes the Content Manager workflow and workbasket functions.

Understanding the Content Manager workflow service: A Content Manager *workbasket* is a container that holds Content Manager documents and folders that you want to process. A Content Manager *workflow* is an ordered set of Content Manager workbaskets that represent a specific business process. Folders and documents move between workbaskets within a Content Manager workflow, allowing your applications to create simple business models and route work through the process until completion.

The Content Manager workflow model used in the Content Manager folder manager follows these rules:

- A workbasket does not need to be located in a Content Manager workflow
- A workbasket can be located in one or more Content Manager 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 Content Manager workflow

The Enterprise Information Portal APIs provide classes to work with Content Manager workflow.

You use `DKWorkflowServiceDL` to represent the workflow service of Content Manager. This class provides the capability to start, change, remove, route, and complete a document or folder in a Content Manager workflow. In addition, you can use the `DKWorkflowServiceDL` class to retrieve information about workbaskets and workflows. The `DKWorkflowDL` and `DKWorkBasketDL` classes are the 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 its workflow service. The datastore provides connection and disconnection methods.

The following example shows how to connect to a Content Manager server named `LIBSRVRN`, using the user ID `FRNADMIN` and the password `PASSWORD`.

```
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\d1` directory.

Creating a workflow: Use `DKWorkflowServiceDL` to create a Content Manager workflow, which typically consists of the following six steps:

1. Create an instance of `DKWorkflowDL`.
2. Set the workflow name.
3. Set the workbasket sequence to `NULL` to indicate that this workflow contains no workbaskets.
4. Set the privilege.

5. Set the disposition.
6. Call the add method.

The following example follows steps 1-6 to create a Content Manager workflow.

Note: This sample uses DK_SS_CONFIG because, unless you connect to the datastore with administrator privileges, you do not get the workflow defined after you connect.

```
// ----- 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\d1 directory.

Listing workflows: DKWorkflowServiceDL provides a method to list the Content Manager workflows. It returns a sequential collection of DKWorkflowDL objects. The following example demonstrates the retrieval of the list of Content Manager workflows:

```
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\d1 directory.

Creating a Content Manager workbasket: You can use DKWorkflowServiceDL to create a Content Manager workbasket. The Content Manager workbasket creation process typically consists of the following steps:

1. Create an instance of DKWorkBasketDL.

2. Set the workbasket name.
3. Set the privilege.
4. Call the add method.

The following example follows these steps to create a Content Manager workbasket. This example sets the access option to DK_SS_CONFIG, since user level access (DK_SS_NORMAL) does not provide for workbasket creation.

```
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\d1 directory.

Listing workbaskets: DKWorkflowServiceDL provides a method that lists the workbaskets in the system. The following example demonstrates the retrieval of the list of workbaskets; this list is returned in a sequential collection of DKWorkBasketDL objects.

The complete sample application from which this example was taken (TListWorkBasketWFS.java) is available in the CMBROOT\Samples\java\d1 directory.

```
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 (pIter.hasMore())
    {
        pwb1 = (DKWorkBasketDL)pIter.next();
        pwb1->retrieve();
        ... // do some work
    }
}
dsDL.disconnect();
dsDL.destroy();
```

Listing items in a Content Manager workflow: DKWorkflowServiceDL provides a method that lists the item IDs of the items in a Content Manager workflow. It returns the list as a sequential collection of DKString objects. The following example demonstrates retrieving a list of the item IDs for the items in a Content Manager workflow:

```
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\d1 directory.

Executing a Content Manager workflow: DKWorkflowServiceDL provides methods that allow you to execute a Content Manager workflow. The following example demonstrates how to start an item in a Content Manager workflow, how to route an item to a workbasket, and how to complete an item in the workflow:

```

DKDatastoreDL dsDL = new DKDatastoreDL();
DKWorkFlowServiceDL wfDL = new DKWorkFlowServiceDL(dsDL);
DKString itemID = new String("EP8L80R9MHH#QES");
DKString itemIDWF = new String("HI7MOPALUPFQ1U47");
DKString itemIDWB = new String("E3PP1UZ0ZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkFlowItem(itemID,           // itemID
                       itemIDWF,       // itemIDWB
                       NULL,            // default (the first workbasket)
                       TRUE,            // overload
                       DK_WIP_DEFAULT_PRIORITY // initial_priority
                       );
...
wfDL.routeWipItem(itemID,           // do some work
                  itemIDWF,         // itemID
                  TRUE,              // itemIDWB
                  DK_NO_PRIORITY_CHANGE // overload
                  );
...
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\d1 directory.

Working with OnDemand

Enterprise Information Portal APIs support the following features when working with OnDemand content servers:

- Connecting and disconnecting from one or more OnDemand servers
- Listing users on an OnDemand server
- Listing application groups
- Listing application group fields

- Listing search template folders
- Querying an application group
- Retrieving an OnDemand document
- Retrieving the logical view data of a given OnDemand document
- Retrieving the resource group for a given OnDemand document
- Retrieving annotation data for a given OnDemand document
- Retrieving OnDemand segments

Restriction: OnDemand does not support Text Search Engine and QBIC search or Combined query.

You represent an 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:

DKViewDataOD

A collection of logical views

DKFixedViewDataOD

Contains fixed view information

DKResourceGrpOD

Contains resource group data

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:

```
...
pCol = (DKSequentialCollection) dsOD.listEntities(); // gets 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 - ");
    ...
}
```

The following example illustrates getting the attribute information for each application group:

```

...
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");
...

```

Listing search template folders: The following example shows how to list folders in OnDemand:

```

...
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();

```

Retrieving an OnDemand document

In OnDemand you can retrieve folders and documents. You can also display documents with their parts and attributes.

Retrieving a particular document or folder: The following example shows the query string used to retrieve a particular document or folder from an OnDemand server:

```

// ----- Generate the query string
String cmd = "SEARCH=(APPL_GROUP=CREDIT," +
    "MAX_RESULTS=5,COND=(where name='ADRIAN CYCLERY'));" +
    "OPTION=(CONTENT=YES)";

DKNVPair[] parms = new DKNVPair[1];
String cmd = "where ( account = '000-000-001' )";
parms[0] = new DKNVPair("APPL_GROUP", "CREDIT");

parms[1] = new DKNVPair("CONTENT", "YES");
parms[1] = new DKNVPair("MAX_RESULTS", new Integer(5));
...

```

Displaying documents and their parts and attributes: The following example displays the documents found by the query with their parts and attributes:

```
// ----- 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_CM_PROPERTY_TYPE));
    System.out.println("nullable: " + p.getDataPropertyByName
        (j,DK_CM_PROPERTY_NULLABLE));

    if (strDataName.equals(DKPARTS) == false &&
        strDataName.equals("DKResourceOD") == false &&
        strDataName.equals("DKViewDataOD") == false &&
        strDataName.equals("DKFixedViewDataOD") == 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;

            if (pDO.protocol() == DK_XDO)
            {
                System.out.println("dkXDO object " + i + " in collection");
                pXDO = (dkXDO)pDO;
                DKPidXDO pid2 = pXDO.getPid();
                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("DKResourceOD") ||
        strDataName.equals("DKFixedViewDataOD"))
    {
        pDO = (dkDataObjectBase)a;

        if (pDO.protocol() == DK_XDO)
        {
            System.out.println("dkXDO object ");
            pXDO = (dkXDO)pDO;
            DKPidXDO pid2 = pXDO.getPid();
            System.out.println("XDO PID string: " + pid2.pidString());
            // Retrieve and open instance handler for an XDO
            pXDO.retrieve();
            // pXDO.open();
        }
    }
}

```

Working with ImagePlus for OS/390

Enterprise Information Portal APIs support the following features when working with 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 or combined queries.

Listing entities and attributes

After creating a datastore for an ImagePlus for OS/390 content server as a DKDatastoreIP object and connecting, you can check the entities and attributes for the datastore. The following example illustrates listing the entities found in a ImagePlus for OS/390 content server:

```

// ----- 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\d1 directory.

The following example illustrates listing the attributes associated with each entity using the getAttr and listAttrNames methods of DKEntityDefIP.

```
// ----- 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(" ");
    }
}
}
```

The following example illustrates an alternative way to list the attributes associated with each entity by using the listEntityAttrs method of DKDatastoreIP.

```
// --- 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(" ");
}
}
}

```

ImagePlus for OS/390 query syntax

The following illustrates the query syntax for ImagePlus for OS/390:

```

SEARCH = (COND=(search_expression), ENTITY={entity_name | mapped_entity_name}
[, MAX_RESULTS = maximum_results]);
[OPTION=( [CONTENT={YES | ATTRONLY | NO};] [PENDING={YES | NO};])]

```

The query consists of the following parameters:

search_expression

Each search expression consists of one or more search criteria. The operator AND is used between search criteria.

The search criteria have the form:

```
{attr_name | mapped_attr_name} operator literal
```

where

attr_name

the name of the entity attribute on which to base the search

mapped_attr_name

the mapped attribute name associated with the attribute on which to base the search

operator

Equality (==) is supported for all attributes; 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 literals, no quotes should be used.

For date, time, timestamp, and string literals, no quotes are necessary, although you can use either double and single quotes. If a string literal contains a single quote, use single quote for the string literal and two single quotes for the quote in the string.

Examples of literals include:

```

ReceiveDate == 1999-03-08
ReceiveDate == '1999-03-08'
FolderId == "Folder number 1"
FolderID == 'John''s Folder'

```

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 be returned

The option keywords are:

CONTENT

Controls the amount of information returned in the results

YES (default)

Set 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

Set only the document or folder PIDs.

ATTRONLY

Set 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 an entity mapped to DOCUMENT.

YES

Include pending documents

NO (default)

Do not include pending documents in the results

Working with VisualInfo for AS/400

The Enterprise Information Portal API classes provided for VisualInfo for AS/400 are similar to those provided for Content Manager.

Restriction: VisualInfo for AS/400 does not support Text Search Engine and QBIC search or combined queries.

Listing index classes and attributes

You represent a VisualInfo for AS/400 content server as a DKDatastoreV4. After creating the datastore and connecting to it, you can list the index classes and attributes for the VisualInfo for AS/400 server. The following example illustrates doing this:

```
// ----- 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\d1 directory.

Executing a query

The following example runs a query in VisualInfo for AS/400 and processes the results.

```

// ----- 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));
    }

    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\d1 directory.

Executing a parametric query

The following example runs a parametric query.

```

// ----- 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.

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.

The Enterprise Information Portal APIs support the following features for working with a Domino.Doc content server:

- 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 Text Search Engine and QBIC search or combined queries.

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 API and how to build the expression. Figure 16 on page 107 represents the Domino.Doc object model.

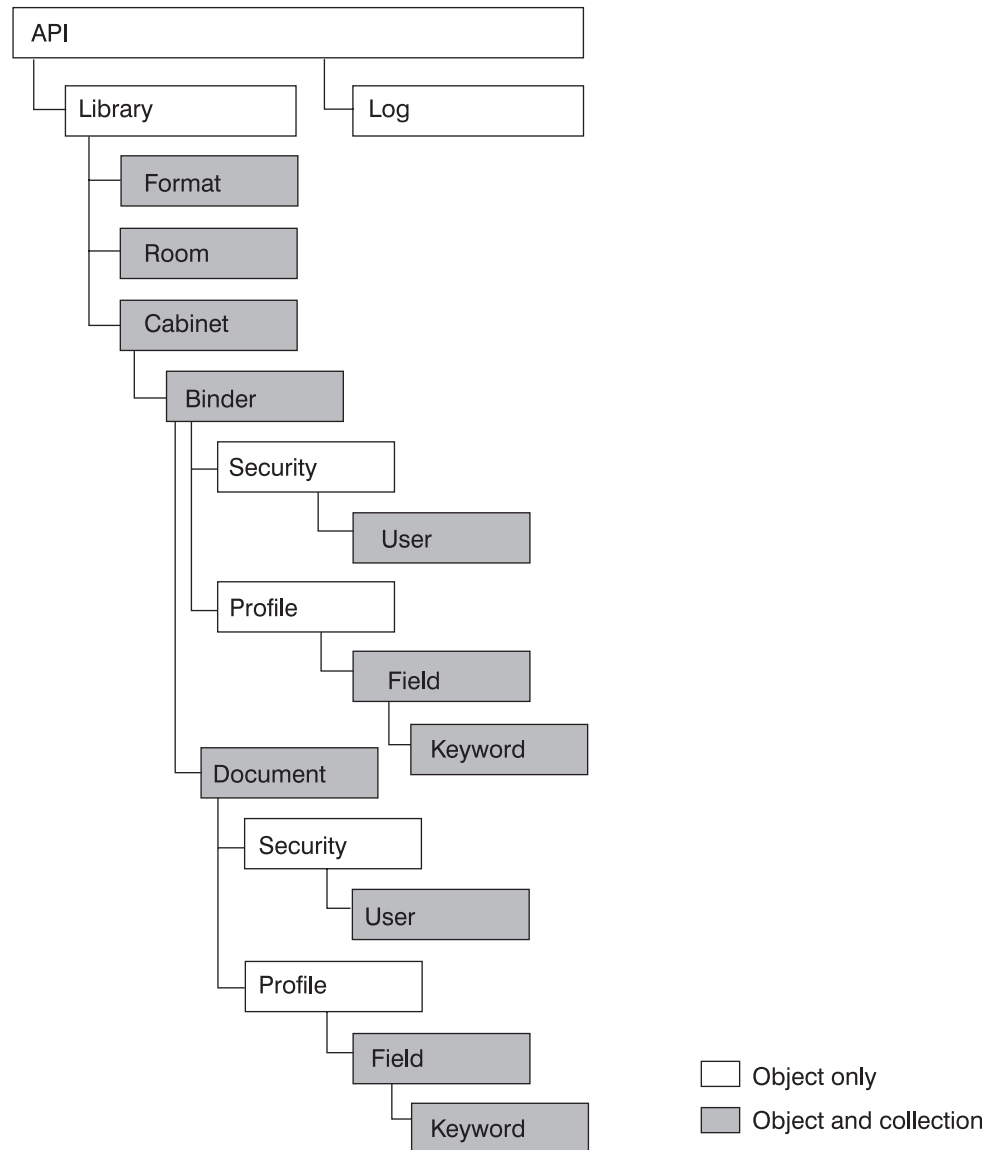


Figure 16. Domino.Doc object model

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 subtentities

The following example lists the rooms in a Domino.Doc content server:

```

...
// ----- 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\d1 directory.

The following example lists document attributes and keywords:

```

...
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
                }
            }
        }
    }
}
...

```

Listing cabinet attributes

Cabinets are the only items that contain useful attributes for application developers. DKDatastoreDD lists cabinets as searchable entities. The following example lists cabinets and their attributes.

```

...
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\d1 directory.

Using 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 double quotation marks ("), for example, "Diane Cabinet".

QUERY= is a required parameter.

So, in Domino.Doc a query string looks like this:
"ENTITY=<"cabinetTitle"> QUERY=<"lotusQueryString">"

Use the `FTSearch` method to query the Domino.Doc datastore. The Domino.Doc datastore needs to be fully text indexed for this method to work efficiently. To test for an index, use the `IsFTIndexed` property. To create an index, use the `UpdateFTIndex` method.

The `FTSearch` method searches all of the documents in a datastore — to search documents found in a particular view, use the `FTSearch` method in `NotesView`, to search documents found in a particular document collection, use the `FTSearch` method in `NotesDocumentCollection`.

If you don't specify a sort option, you get the documents sorted according to a relevance score. 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, it formats its doclink report with 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

To search for a word or phrase, enter the word or phrase you are searching for. Enclose search keywords and symbols in apostrophes ('). In a LotusScript literal, remember to use quotation marks (").

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 precedence 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`, and `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 Domino Extended Search (DES)

Domino Extended Search (DES) allows you to query and retrieve documents from:

- Lotus Notes® databases
- NotesPump databases
- File systems
- Web search engines

DES includes the following features:

- Connecting and disconnecting from one or more DES servers
- Listing DES servers
- Listing databases and fields
- Ability to use Generalized Query Language (GQL) to perform searches
- Ability to retrieve documents

Restriction: DES does not support:

- Adding, updating, and deleting documents
- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

All DES features are accessed and controlled by the DES 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 DES servers

In order to provide access to multiple DES servers, you can create a file named `cmbdes.ini` that contains the server information. This file must be located in `x:\CMBROOT` (where x 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
```

TCP/IP is the TCP/IP address of the DES server and the *port number* is the port number defined in order to access the server.

Listing databases and fields

When you build a query to search a DES server, you must know the database names and the field names that are available. The `DKDatastoreDES` object provides the `listEntities` method to list the databases and the `listEntityAttrs` method to list the fields for each database. The following example retrieves a list of databases and their fields:

```
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(" 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("    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");
}
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\d1 directory.

Using Generalized Query Language (GQL)

DES uses the Generalized Query Language (GQL) to perform searches. Table 7 contains examples of valid GQL expressions.

Table 7. GQL expressions

GQL expression	Description
----------------	-------------

Table 7. GQL expressions (continued)

"software"	search for documents which contain the word software
(TOKEN:WILD "exec*")	search for documents which contain any word beginning with exec
(AND "software" "IBM")	search for documents which 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

DES 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])]
```

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 executes a GQL query against DES. After the query is executed, the results are returned in a dkResultSetCursor object.

```
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\d1 directory.

DDO properties in DES

A DDO in DES will always have the type DK_CM_DOCUMENT. To get the item type of the DDO, you call:

```
Object obj = ddo.getPropertyByName(DK_CM_PROPERTY_ITEM_TYPE);
short type = ((Short) obj).shortValue();
```

Creating PIDs in DES

The persistent identifier (PID) contains specific information about a document. The object type indicates the database where the document was found. For more information on PIDs see "Understanding persistent identifiers (PID)" on page 11 and "Creating a persistent identifier (PID)" on page 27. A PID is created by using the database name, followed by the | character and the document ID, for example: database name|documentId ()

Contents of a DES document

Each item in the DDO represents either a field, a collection, or a DKParts object.

Field A single field has the field name inside the item name. The value of the field is inside the item value. The property of the field 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 be used to store the address information about a document.

This example processes the contents of a DDO:

```
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\d1 directory.

Retrieving a document

To retrieve a document from a DKDatastoreDES object, you must know the name of the database where the document resides and the document ID. You must also associate the DDO to a datastore and establish a connection. This example retrieves a document from a database:

```

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 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\d1 directory.

Retrieving a BLOB

To retrieve a BLOB from a DKDatastoreDES object, you must know the name of the database where the document resides, the document ID which contains the BLOB, and the field name that has the contents of the BLOB. You must also associate the DDO to a datastore and establish a connection.

In the following example, the file system database named DES files contains an HTML file named D:\desdoc\README.html. The field that has the contents of the HTML file is named Doc\$Content. The HTML file is retrieved and saved as D:\DESReadme.html.

```
DKDatastoreDES dsDES = new DKDatastoreDES();
dsDES.connect(libSrv, userid, pw, connect_string);
DKBlobDES xdo = new DKBlobDES(dsDES);
DKPidXDOES pid = new DKPidXDOES();
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 (TRetrieveXDOES.java) is available in the CMBROOT\Samples\java\d1 directory.

Associating MIME types with documents

DES does not directly support identification of MIME types. However, you must know the MIME type of an XDO that you want to display within a viewer. The CMBCC2MIME.INI file is used to determine the MIME type of a document. When a retrieved query from DES returns a BLOB, the CMBCC2MIME.INI file is searched to determine if a MIME type can be assigned to the BLOB. This file is searched for NotesPump and FileSystem databases only. The default MIME type is text/html, when there is no type match found inside the CMBCC2MIME.INI file.

Using federated searching in DES

When you create query expressions for federated searching, the syntax used in DES is similar to SQL syntax. The federated query expressions are converted to GQL syntax before they are submitted to DES. Because SQL and GQL grammar have many differences, only a subset of the SQL grammar is supported by Enterprise Information Portal.

Table 8 summarizes the SQL to GQL conversion of the supported comparison and logical operators.

Table 8. 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 relational databases

The Enterprise Information Portal API classes support IBM DB2 Universal Database, and other relational databases using Java Database Connectivity (JDBC) and IBM DB2 DataJoiner.

Connecting to relational databases

To represent a relational database, create a `DKDatastorexx` object, where the `xx` is DB2 for a DB2 database, DJ for DataJoiner, or JDBC for Java Database Connectivity. The following sample connects to the DB2 sample database:

```
dsDB2 = new DKDatastoreDB2();
dsDB2.connect("sample","db2admin","password","");
.....
dsDB2.disconnect();
dsDB2.destroy();
```

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=(datastore name)`

Specifies the name of the datastore. 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 datastore 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.

CC2MIMEFILE=(filename)

Specifies the cmbcc2mime.ini file by name.

JDBCSEVERURL=(URL)

Specifies the cmbjdbcsrvs.ini file in a uniform resource locator address. This file contains the list of JDBC servers.

JDBCSEVERFILE=(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 EIP administration client program.

DSNAME=(datastore name)

Specifies the name of the datastore. 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 datastore is used for federated queries and other federated functions, autocommit is on by default.

Listing entities and entity attributes

After creating the datastore for the relational database and connecting to it, you can list the entity and entity attributes. The following example illustrates retrieving the list and stepping through it:

```
// ----- 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\d1 directory for complete examples.

Executing a query

To run a query you first create the query string and then execute the query. The following example runs a query and processes the results.

```

// ----- 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;

// ----- 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\d1 directory for complete examples.

Working with DB2 Warehouse Manager Information Catalog Manager

Enterprise Information Portal APIs support the following features when working with DB2 Warehouse Manager Information Catalog Manager content servers:

- Connecting and disconnecting from one or more DB2 servers with DB2 Warehouse Manager Information Catalog Manager
- Listing the data sources available
- Listing entities and attributes

- Mapping information in the Information Catalog
- Retrieving an DB2 Warehouse Manager Information Catalog Manager reports
- Setting and getting access options

Restriction: DB2 Warehouse Manager Information Catalog Manager does not support Text Search Engine and QBIC search or Combined query.

Connecting to a DB2 Warehouse Manager Information Catalog Manager server

You represent a DB2 Warehouse Manager Information Catalog Manager content server using DKDatastoreIC in an Enterprise Information Portal application.

To access an Information Catalog content server, you create a DKDatastoreIC object in your application and connect to it. The following example illustrates connecting to a DKdatastoreIC datastore, getting the datastore name, and disconnecting:

```
// ----- Create the datastore and connect
DKDatastoreIC icDs = new DKDatastoreIC();
icDs.connect(database, userid, pw, "");

// ----- Get the datastore name
String dsName = icDs.datastoreName();
....
icDs.disconnect();
icDs.destroy();
```

Listing entities and attributes

You can list the entities and attributes for a DB2 Warehouse Manager Information Catalog Manager content server. When mapping the Information Catalog objects to federated entities, you must map Instance identifier attribute to retrieve the objects. Do not use Instance identifier or INSTIDNT as the names for federated entities or attributes when querying a DB2 Warehouse Manager Information Catalog Manager content server.

The following example illustrates getting a list of entities and attributes for an Information Catalog content server; the entities and attributes are mapped to the objects and properties on the content server.

```
// ----- Create the datastore and connect
DKDatastoreIC dsIC = new DKDatastoreIC();
dsIC.connect(db, userid, pw, "");

// ----- Create an iterator and get a list of entities
dkIterator itr = dsIC.datastoreDef().listEntities().createIterator();
while ( itr.more() ) {
    dkEntityDef entity = (dkEntityDef)itr.next();
    .... // --- Process as appropriate ;
}

// ----- Get a list of the entity names
String entityNames[] = dsIC.datastoreDef().listEntityNames();
int numEntities = entityNames.length;
for ( i = 0; i < numEntities; i++ ) {
    .... // Process the names as appropriate
}
// ----- Get a list of the attributes
String eName = "IMAGES";
itr = dsIC.datastoreDef().listEntityAttrs(eName).createIterator();
while ( itr.more() ) {
    dkAttrDef entityAttr = (dkAttrDef)itr.next();
    .... // Process as you want or print out a list
    System.out.println("Entity Attr = " + entityAttr.getName());
    System.out.println("Entity Attr Desc= " + entityAttr.getDescription());
}
```

```

}
// ----- Get a list of the attribute names
eName = "CHARTS";
String entityAttrNames[] = dsIC.datastoreDef().listEntityAttrNames(eName);
int numEntityAttrs = entityAttrNames.length;
for ( i = 0; i < numEntityAttrs; i++) {
    .... // Process as you want or print out the names
    System.out.println("Entity Attr Name = " + entityAttrNames[i]);
}
...
dsIC.disconnect();
dsIC.destroy();

```

Executing a query

To query a DB2 Warehouse Manager Information Catalog Manager content server, you first create a query string. You can use a parametric query string for a parametric query, or an SQL query string.

Parametric query string: You can use a parametric query by creating a parametric query string. The following example shows a parametric query string, that searches specified dimensions in the table using ENTITY, the Name, and the short description:

```

String parqry =
    "SEARCH=(ENTITY=Dimensions within a multi-dimensional database," +
    "MAX_RESULTS=10, COND=( ('Name' LIKE 'M%')" +
    " OR ('Short description' = 'Product dimension')));" +
    "OPTION=(CONTENT=YES)";

```

A parametric query specifies the attributes to be searched.

SQL query string: You can create an SQL query by creating an SQL query string. The following example shows an SQL query string:

```

String sqlqry = "Select * from dimension where name like 'M%' +
    "or shrtdesc = 'Product dimension'";

```

An SQL query uses SQL syntax.

Running the query: After you have the query string, use the execute method on DKDatastoreIC to run the query. The execute method returns a result set cursor you can use to access the collection. The following sample illustrates running a parametric query:

```

DKDatastoreIC dsIC = new DKDatastoreIC();
.... // create the query string

dkResultSetCursor rs = dsIC.execute( qry, DK_CM_PARAMETRIC_QL_TYPE, null );
if( rs != null ) {
    while( rs.isValid() ) {
        DKDDO ddo = rs.fetchNext();
        if( ddo != null ) {
            for( short i = 1; i <= ddo.dataCount(); i++ ) {
                String dName = ddo.getDataName( i ).trim();
                Object dValue = ddo.getData( i );
                .... // ----- Process the data as you want
            }
        }
    }
}
rs.close();

```

For an SQL query, the method call would be like the following:

```

rs = dsIC.execute( cmd, DK_CM_SQL_QL_TYPE, null );

```

You can also use the evaluate method on DKDatastoreIC to run a query. This method returns a collection as a Java Object; you can cast the Object to a DKResults collection. The following example illustrates using the evaluate for a parametric query.

```
DKDatastoreIC dsIC = new DKDatastoreIC();
....
// ----- Create the query string
String qry = "SEARCH=(ENTITY=Dimensions within a multi-dimensional database," +
    "MAX_RESULTS=10,COND=( ('Name' LIKE 'M%')" +
    " OR ('Short description' = 'Product dimension')));" +
    "OPTION=(CONTENT=YES)";

pResults = (DKResults)dsIC.evaluate(qry, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Process the results
processResults((dkCollection)pResults);
System.out.println("--- Number of items = " + pResults.cardinality());
....
```

Using Enterprise Information Portal workflow

You can use the Java APIs to create or extend your own applications to use the workflow support available in Enterprise Information Portal. Typically, you first perform a federated search, then create a work packet, assign at least one content item to the workpacket, then start the workflow. You use the APIs to access a worklist and then to display the worklist contents. As the activities are completed, the workflow moves from one activity to the next in the workflow.

Connecting to workflow services

To use EIP workflow in your applications, first create a DKWorkFlowServicesFed object and connect to it. The following example starts workflow services:

```
// ----- Create the strings for the name of the service, user ID and Password
String wfsrv = "cmbdb";
String userid = "cmbadmin";
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, "");
```

When you are finished using the workflow service, you must disconnect by calling the disconnect and the destroy methods.

```
svWF.disconnect();
dsFed.disconnect();
svWF.destroy();
dsFed.destroy();
```

Creating a workpacket

You create a workpacket by constructing a DKWorkPacketFed object, specifying the name of the workpacket, a description, and the workflow template name. Add the PID string of each document in the workpacket into the workpacket by calling the addItem method. You get the PID string from the federated search results, and it contains the references to the actual document in the content server. The following is a simple example.

```
DKWorkPacketFed wp1 = new DKWorkPacketFed(dsFed);
// ----- Set the workpacket name, descriptions and priority
wp1.setName("wp1");
```

```

wp1.setDescription("This is a test1");
wp1.setPriority(100);
// ----- Set workflow template name
wp1.setWFTemplateName("Batman");
// ----- Add a PID string referring to a DES document
wp1.addItem("45 3 DES4 rossi0 Notes Help18 15 Notes Help|23fa");
// ----- Add the new workpacket into database
wp1.add();
// ----- Retrieve the ID of the workpacket
String wpID = wp1.getID();

```

Starting a workflow

Once you have created the workflow you have to start it. Starting a workflow consists of the following steps:

1. Create a DKWorkflowFed object and set the workflow name
2. Create a workflow instance using a valid workflow template, that is a workflow definition defined in the EIP workflow builder.
3. Set the workpacket ID and priority in the container.
4. Start the workflow.

In the following example these steps are used to start a workflow:

```

// ----- Create the DKWorkflowFed object and set the name and workpacket ID
DKWorkflowFed WF = new DKWorkflowFed(svWF);
WF.setName(wp1.getID());
// ----- Create an instance of a workflow with the workflow template name
//           in the workpacket
WF.add(wp1.getWFTemplateName());
// ----- Refresh the workflow object
WF.retrieve();
// ----- Construct the container object for the workflow
DKWorkflowContainerFed con = WF.inContainer();
// ----- Retrieve the container data
con.retrieve();
// ----- Set the workpacket ID and priority
con.setWorkPacketID(wp1.getID());
con.setPriority(wp1.getPriority());
// ----- Update the container
con.update();
// ----- Start the workflow
WF.start(con);

```

Terminating a workflow

You can terminate a workflow by calling the terminate or del method as shown in the following example:

```

// ----- 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_NOTSET ||
    state == DKConstantFed.DK_FED_FMC_PS_RUNNING ||
    state == DKConstantFed.DK_FED_FMC_PS_SUSPENDED ||
    state == DKConstantFed.DK_FED_FMC_PS_SUSPENDED)
{
    WF.terminate();
}

if (state == DKConstantFed.DK_FED_FMC_PS_READY ||
    state == DKConstantFed.DK_FED_FMC_PS_FINISHED ||
    state == DKConstantFed.DK_FED_FMC_PS_TERMINATING ||
    state == DKConstantFed.DK_FED_FMC_PS_TERMINATED ||
    state == DKConstantFed.DK_FED_FMC_PS_TERMINATED)

```

```

{
    WF.del();
}

```

Accessing a worklist

You can access a worklist by creating a `DKWorkListFed` object that refers to the worklist you created using EIP administration. The following example accesses a worklist named `WL0712` and displays the information contained in that worklist:

```

// ----- 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());

```

Accessing work items

After you have created the `DKWorkListFed`, you can retrieve the work items as a collection. The following example illustrates retrieving the work items:

```

// ----- 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 ();
        DKWorkPacketFed wp = new DKWorkPacketFed(dsFed, workflowname);
        System.out.println ("workitem node = " + nodename +
            " workflow name = " + workflowname);
    }
}
iter = null;

```

Moving items in the workflow

As a workflow progresses, you move work items from one activity to the next by using the `checkOut` and `checkIn` methods. The following examples illustrates moving workitems:

```

DKWorkItemFed item =new DKWorkItemFed(svWF, "wf1", "node1", cmbadmin);
item.retrieve();
// ----- Call the checkOut method to lock the workitem
item.checkOut();
// ----- Call the checkIn method
item.checkIn(null);

```

Creating your own actions

You can create your own actions that you can use in a workflow. You defined the actions and add them to actions lists in EIP Administration.

When you create an action you must supply a Java class to perform the action. The Java class you create must implement the interface `dkWorkflowUserExit` and implement the `doAction` method. In this method you actually perform the action.

As an alternative you can create a concrete subclass of `dkAbstractWorkflowUserExit`, which implements `dkWorkflowUserExit`.

Creating custom content server connectors

You can create your own connectors for custom content servers not currently included in Enterprise Information Portal. If you integrate a custom server into Enterprise Information Portal you must provide your own Java classes to support the definition of any additional servers.

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 seamlessly
- 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
- link to the `cmbcm71.jar` file in order to access the common framework

Enterprise Information Portal database infrastructure: The primary interface of each content server to a Enterprise Information Portal database is found in the `dkDatastore` classes. Each content server has a separate datastore class which implements this class to provide the implementation information for a specific content server. Each content server type is represented by a class called `DKDatastorexx`, where `xx` could be the name or type of the specific content server. Table 9 lists the content servers provided in Enterprise Information Portal.

Table 9. Content servers provided in Enterprise Information Portal

Server type	Class name
Content Manager	<code>DKDatastoreDL</code>
OnDemand	<code>DKDatastoreOD</code>
VisualInfo for AS/400	<code>DKDatastoreV4</code>
ImagePlus for OS/390	<code>DKDatastoreIP</code>
Domino.Doc	<code>DKDatastoreDD</code>
Domino Extended Search	<code>DKDatastoreDES</code>
DB2 Universal Database	<code>DKDatastoreDB2</code>
JDBC	<code>DKDatastoreJDBC</code>
IBM DB2 DataJoiner	<code>DKDatastoreDJ</code>
DB2 Warehouse Manager Information Catalog Manager	<code>DKDatastoreIC</code>

Common classes in Enterprise Information Portal:

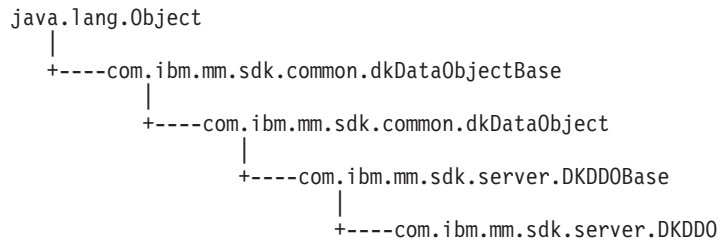
dkDDO

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 methods to define, add, and access each data item of an object. This protocol allows a client to create an object dynamically and get its persistent data in and out of the datastore, independent of the datastore's type.

When implementing a datastore you can utilize schema mapping information registered in the datastore class. The schema maps each individual persistent data item to its underlying representation in the datastore. A DDO has a set of attributes, each attribute has a type, value, and properties associated with it. The DDO itself can have properties belonging to the whole DDO instead of to a particular attribute. For example, this class could be mapped to an item for the Content Manager datastore, and mapped to a document for the OnDemand datastore.

For information on mapping terminology for each datastore see Table 2 on page 12.

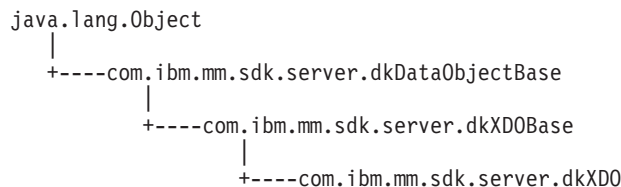
This diagram represents the hierarchy for the dkDDO class:



dkXDO

The dkXDO class represents complex user defined types (UDTs) or large objects (LOB). This object can exist stand-alone or as a part of a DDO. Therefore, it has a persistent ID (PID) and create, retrieve, update, and delete methods. The dkXDO class extends the public interface of dkXDObase by defining independent datastore access create, retrieve, update, and delete methods. These methods enable an application to store and retrieve the object's data to and from a datastore without the existence of an associated DDO class object or stand-alone XDO.

A stand-alone XDO must have its PID set in order to locate its position in the datastore. If the XDO is used in conjunction with the DDO the PID is set automatically. For example, this class could be mapped to a part in a document for the Content Manager datastores, and mapped to notes for the OnDemand datastores. Here is the class hierarchy for the dkXDO class:



dkCollection

The `dkCollection` class is a base public interface for a collection of any objects. `dkCollection` can not evaluate a query. A collection might have a name (the default name is an empty string), but it could be set to anything. 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 can be queried. The element members of a `DKResults` class are objects, instances of the `DKDDO` class, which represent query results. The iterator created by this class is `dkSequentialIterator`.

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 one specific datastore. Objects which implement this interface are created by datastore classes. The result of a query is usually a `DKResults` object. Examples of a concrete implementation of the `dkQuery` interface are `DKParametricQuery`, `DKTextQuery`, `DKImageQuery`, and so forth which are created by their associated datastore.

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 one parametric, text, or image query. In order for each content server to be able to engage in the federated query, the server needs to be able to process this `DKCQExpr` object.

dkSchemaMapping

`dkSchemaMapping` is an interface that defines an associative mapping between a mapped or federated entity and a map-to or native entity in content server. The content server needs to understand this mapping class in order to unmap and remap federated entities and attributes to its own native entities and attributes in 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 a `OnDemand` server, the concrete class that implements the `dkDatastore` interface will be `DKDatastoreOD`.

dkDatastore

`dkDatastore` represents and manages a connection to the datastore, its transactions, and the execution of datastore commands. It supports the `evaluate` method, so it can be considered a subclass of the query manager.

The main methods in the `dkDatastore` interface are:

connect()

Connects to the datastore

disconnect()

Disconnects from the datastore

evaluate(), execute(), executeWithCallback()

Queries the datastore

commit(), rollback()

Performs transactions in the datastore

Restriction: Some content servers do not support these methods.

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 objects to use for logon. This method in dkDatastore class does not need to be connected to the content server.

listDataSourceNames()

Returns an array of content server names.

getExtension(String)

Gets the dkExtension object from the datastore. If the given extension does not already exist and it is supported by the datastore, a newly constructed object will be returned, otherwise, a null value will be returned.

addExtension(String, dkExtension)

Adds a new extension object to this datastore

createDDO(String,int)

Creates a new data object based on the given object type and flag. It returns a new DKDDO object with all the properties and attributes set. The caller must fill in 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

The schema mapping related methods in the dkDatastore interface are:

registerMapping(DKNVPair)

Registers the mapping information to this datastore

unRegisterMapping(String)

Removes the mapping information

listMappingNames()

Returns an array of mapping names

getMapping(String)

Returns a dkSchemMapping object

dkDatastoreDef

The dkDatastoreDef interface defines methods to access datastore information as well as to create, list, and delete its entities. It maintains a collection of dkEntityDef objects.

Table 10 contains examples of concrete classes for the dkDatastoreDef interface.

Table 10. Concrete classes for dkDatastoreDef

Server type	Class name
Content Manager	DKDatastoreDefDL
OnDemand	DKDatastoreDefOD
VisualInfo for AS/400	DKDatastoreDefV4
ImagePlus for OS/390	DKDatastoreDefIP
Domino.Doc	DKDatastoreDefDD
Domino Extended Search	DKDatastoreDefDES
IBM DB2 Universal Database	DKTableDefDB2
JDBC	DKTableDefJDBC
ODBC	DKTableDefODBC
DB2 DataJoiner	DKTableDefDJ
DB2 Warehouse Manager Information Catalog Manager	DKDatastoreDefIC

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 datastore specific methods with method names that are familiar to that datastore. For example, the DKDatastoreDefDL class contains these specific methods:

- listIndexClassNames()
- listIndexClasses()

The DKDatastoreDefOD class contains these specific methods:

- listAppGrps()
- listAppGrpNames()

dkEntityDef

The dkEntityDef class defines methods to:

- Access entity information
- Create and delete attributes

- Create and delete the entity

In the `dkEntityDef` class all methods that are related to subentities generate a `DKUsageError` indicating that the default datastore does not support subentities. However, if the datastore can support this kind of multiple level entity, like the `Domino.Doc` datastore for example, the subclass for this datastore needs to implement the proper methods to overwrite the exceptions.

Table 11 contains examples of concrete classes for the `dkEntityDef` class.

Table 11. Concrete classes for dkEntityDef

Server type	Class name
Content Manager	DKIndexClassDefDL
OnDemand	DKAppGrpDefOD
VisualInfo for AS/400	DKIndexClassDefV4
ImagePlus for OS/390	DKEntityDefIP
Domino.Doc	DKCabinetDefDD
Domino Extended Search	DKDatabaseDefDES
DB2 Universal Database	DKTableDefDB2
JDBC	DKTableDefJDBC
ODBC	DKTableDefODBC
DB2 DataJoiner	DKTableDefDJ
DB2 Warehouse Manager Information Catalog Manager	DKEntityDefIC

The main methods in the `dkEntityDef` class are:

ListAttrs()

Lists the entity attributes

getAttr(String attrName)

Gets an 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()

Checks to see if the entity contains subentities

getSubEntity(String)

Gets the subentity

addSubEntity(dkEntityDef)

Adds a subentity to an entity

listSubEntities()

Lists the subentities

removeAttr(String)

Removes a subentity

add() Adds the entity to the datastore

update()
Updates the entity in the datastore

retrieve()
Retrieves the entity values from the datastore

del() Deletes the entity from the datastore

dkAttrDef

The dkAttrDef class defines methods to access attribute information and create and delete attributes. Table 12 contains examples of concrete classes for the dkAttrDef class.

Table 12. Concrete classes for dkAttrDef

Server type	Class name
Content Manager	DKAttrDefDL
OnDemand	DKFieldDefOD
VisualInfo for AS/400	DKAttrDefV4
ImagePlus for OS/390	DKAttrDefIP
Domino.Doc	DKAttrDefDD
Domino Extended Search	DKFieldDefDES
DB2 Universal Database	DKColumnDefDB2
JDBC	DKColumnDefJDBC
ODBC	DKColumnDefODBC
DB2 DataJoiner	DKColumnDefDJ
DB2 Warehouse Manager Information Catalog Manager	DKAttrDefIC

The main methods in the dkAttrDef class are:

ListAttrs()
Lists the attributes

getAttr(String attrName)
Gets an attribute

getName()
Gets the name of the attribute

getDescription()
Gets the description of the attribute

add() Adds the entity to the datastore

dkServerDef

The dkServerDef class provides the server definition information for each content server. Table 13 contains examples of concrete classes for the dkServerDef class.

Table 13. Concrete classes for dkServerDef

Server type	Class name
Content Manager	DKServerDefDL
OnDemand	DKServerDefOD
VisualInfo for AS/400	DKServerDefV4

Table 13. Concrete classes for dkServerDef (continued)

Domino.Doc	DKServerDefDD
Domino Extended Search	DKServerDefDES
DB2 Universal Database	DKServerDefDB2
JDBC	DKServerDefJDBC
ODBC	DKServerDefODBC
DB2 DataJoiner	DKServerDefDJ
DB2 Warehouse Manager Information Catalog Manager	DKServerDefIC

The main methods in the dkServerDef class are:

setDatastore(dkDatastore ds)

Sets the reference to the datastore object

getDatastore()

Gets the reference to the datastore object

getName()

Gets the name of the datastore

setName(String name)

Sets the name of the datastore

datastoreType()

Gets the datastore type

dkResultSetCursor

dkResultSetCursor is a datastore cursor which manages a virtual collection of DDO objects. The collection is a result set of a query submitted to the datastore. Each element of the collection is not created until the datastore fetches the element.

The main methods 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

isEnd()

Returns TRUE if the cursor is positioned at the end

isInBetween()

Returns TRUE if cursor is positioned between data objects in the result set

getPosition()

Gets the current position of the cursor

setPosition(int position, Object value)

Sets the cursor to the specific position

setToNext()

Sets the cursor to point to the position of the next data object in the result set

fetchObject()

Fetches the element of the result set at the current position and returns it as a DDO

fetchNext()

Fetches the next element of the result set and returns it as a DDO

fetchNextN(int how_many, dkCollection collection)

Fetches as the next *n* elements of the result set and inserts them into the specific collection

findObject(int position, String predicate)

Finds the data object that satisfies the specific predicate, moves the cursor to that position, and then fetches and returns the data object

addObject(DKDDO ddo)

Adds a new element of the same type, represented by the specific DDO, to the datastore

deleteObject()

Deletes the element at the current position from the datastore

updateObject(DKDDO ddo)

Updates the element at the current position in the datastore, using the specific DDO

newObject()

Constructs a new element of the same type and returns it as a DDO

open() Opens the cursor and, if necessary, executes the query to get the result set

close() Closes the cursor and invalidates the result set

isOpen()

Returns TRUE if the cursor is currently open

destroy()

Deletes the resultSetCursor. This allows for cleanup before the resultSetCursor is garbage-collected.

datastoreName()

Gets the datastore name associated with the definition to which the resultSetCursor belongs

datastoreType()

Gets the datastore type associated with the definition to which the resultSetCursor 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` methods you must set the `datastore` 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 datastores. There is also a dkClob and a dkDBClob class which can have concrete classes.

Table 14 contains examples of concrete classes for the dkBlob class.

Table 14. Concrete classes for dkBlob

Server type	Class name
Content Manager	DKBlobDL
OnDemand	DKBlobOD
VisualInfo for AS/400	DKBlobV4
ImagePlus for OS/390	DKBlobIP
Domino.Doc	DKBlobDD
Domino Extended Search	DKBlobDES
DB2 Universal Database	DBBlobDB2, DKBlobDB2
JDBC	DKBlobJDBC, DKBlobJDBC
ODBC	DKBlobODBC, DKBlobODBC
DB2 DataJoiner	DKBlobDJ, DKBlobDJ
DB2 Warehouse Manager Information Catalog Manager	DKBlobIC

The main methods in the dkBlob class are:

getContent()

Returns a byte array stream containing the BLOB data of this object

getContentToClientFile(String afileName, int fileOption)

Copies the BLOB data from this object to the specific file

setContent(byte[] aByteArr)

Sets the LOB data of this object with the content of the byte array argument

setContentFromClientFile(String afileName)

Replaces the LOB data of this object with the contents of the file afileName

add(String afileName)

Adds the content of a file to the persistent datastore

retrieve(String afileName)

Retrieves the content of the persistent datastore into the specified file

update(String afileName)

Updates the object and the persistent datastore with the content of the specified file

del(boolean flush)

Deletes the object' data from the persistent datastore. If flush is TRUE, this method clears the content, otherwise the current content is preserved.

concatReplace(dkBlob aBlob), concatReplace(byte[] aByteArr)

Concatenates this object with another dkBlob object or a byte array stream

length()

Returns the length of the LOB

indexOf(String aString, int startPos), indexOf(dkBlob aBlob, int startPos)

Returns the byte offset of the first occurrence of the search argument within this object, starting the search at offset startPos

substring(int startPos, int length)

Returns a string object containing a substring of the LOB data of this object

remove(int startPos, int aLength)

Deletes the portion of the LOB data of this object starting at startPos for aLength bytes

insert(String aString, int startPos), insert(dkBlob aBlob, int startPos)

Inserts the argument data, following position startPos in the LOB data of this object

open(String fileName)

Unloads the object contents to the file fileName and then invokes a default file handler

setClassOpenHandler(String aHandler, boolean newSynchronousFlag)

Sets the executable program name for a class file handler

setInstanceOpenHandler(String aHandler, boolean newSynchronousFlag)

Identifies the executable file handler program name and indicates whether this file handler should be invoked synchronously or asynchronously for this object

getOpenHandler()

Sets the executable program name handler for the file handler for an entire class

isOpenSynchronous()

Returns the current synchronization property for this handler

dkAnnotationExt

dkAnnotationExt is the interface class for all annotation objects. If your content server supports annotation data, you need to implement this interface. This annotation object is an extension to your DKBlobxx class, where the dkBlob object is the representation of the binary annotation data and the DKParts collection.

dkDatastoreExt

The dkDatastoreExt class defines the interface for datastore standard extension classes.

Table 15 contains examples of concrete classes for the dkDatastoreExt class.

Table 15. Concrete classes for dkDatastoreExt

Server type	Class name
Content Manager	DKDatastoreExtDL

Table 15. Concrete classes for *dkDatastoreExt* (continued)

OnDemand	DKDatastoreExtOD
VisualInfo for AS/400	DKDatastoreExtV4
ImagePlus for OS/390	DKDatastoreExtIP
Domino.Doc	DKDatastoreExtDD
Domino Extended Search	DKDatastoreExtDES
DB2 Universal Database	DKDatastoreExtDB2
JDBC	DKDatastoreExtJDBC
DB2 DataJoiner	DKDatastoreExtDJ

The main methods in the *dkDatastoreExt* class are:

getDatastore()

Gets the reference to the owner datastore object

setDatastore(dkDatastore ds)

Sets the reference to the owner datastore object

isSupported(String functionName)

Queries if the given function name is supported by this extension

listFunctions()

Lists all supported function-names from this extension

addToFolder(dkDataObject folder, dkDataObject member)

Adds a new member item to this folder and reflects the results immediately in the datastore

removeFromFolder(dkDataObject folder, dkDataObject member)

Removes a member from this folder and reflects the results immediately in the datastore

checkout(dkDataObject item)

Check out a document or folder item from the datastore. You have exclusive updating privileges to the item, while other users are allowed read access only, until you check it back in.

checkin(dkDataObject item)

Check in a document or folder item previously checked out from the datastore. You release all write privileges with this method.

getCommonPrivilege()

Gets the common privilege of a specific datastore

isCheckedOut(dkDataObject item)

Checks if a document or folder item was checked out from the datastore

checkedOutUserId(dkDataObject item)

Gets the user ID that checked out the item from the datastore

unlockCheckedOut(dkDataObject item)

Unlocks the item from the datastore

changePassword (String userId, String oldPwd, String newPwd)

Changes password on the server for the given user ID

moveObject (dkDataObject dataObj, String entityName)

Moves the current data 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 16 contains examples of concrete classes for the DKPidXDO class.

Table 16. Concrete classes for DKPidXDO

Server type	Class name
Content Manager	DKPidXDODL
OnDemand	DKPidXDOOD
VisualInfo for AS/400	DKPidXDOV4
ImagePlus for OS/390	DKPidXDOIP
Domino.Doc	DKPidXDODD
Domino Extended Search	DKPidXDODES
DB2 Universal Database	DKPidXDODB2
JDBC	DKPidXDOJDBC
ODBC	DKPidXDOODBC
DB2 DataJoiner	DKPidXDODJ
DB2 Warehouse Manager Information Catalog Manager	DKPidXDOIC

dkUserManagement

The dkUserManagement class represents and processes all of the datastore's user management methods.

Table 17 contains examples of concrete classes for the dkUserManagement class.

Table 17. Concrete classes for dkUserManagement

Server type	Class name
Content Manager	DKUserMgmtDL
VisualInfo for AS/400	DKUserMgmtV4
ImagePlus for OS/390	DKUserMgmtIP

DKConstant

All common constants are defined in the DKConstant class. Each content server will have its own DKConstantxx class for defining the content server specific constants. It is recommended that all content servers use the common constants whenever possible.

DKMessageId

All common message IDs are defined in this class. Each content server will have its own DKMessageIdxx class for defining its own message IDs. It is recommended that all content servers use the common messages whenever possible.

These property files contain common warning and error messages:

- DKMessage_en.properties
- DKMessage_es.properties

Each content server will have its own DKMessage xx_yy_zz .properties files for its warning and error messages.

Figure 17 uses the OnDemand content server as a model and is an example of the datastore structure used in Enterprise Information Portal.

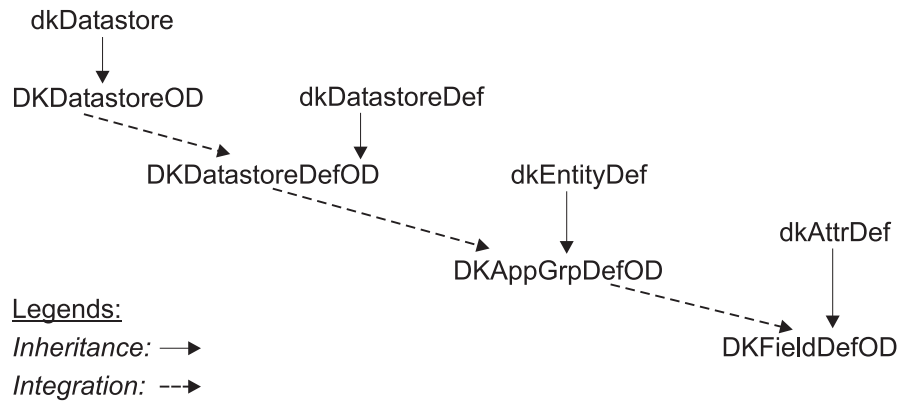


Figure 17. Example of datastore structure

Figure 18 uses the OnDemand content server as a model for showing the Enterprise Information Portal data structure.

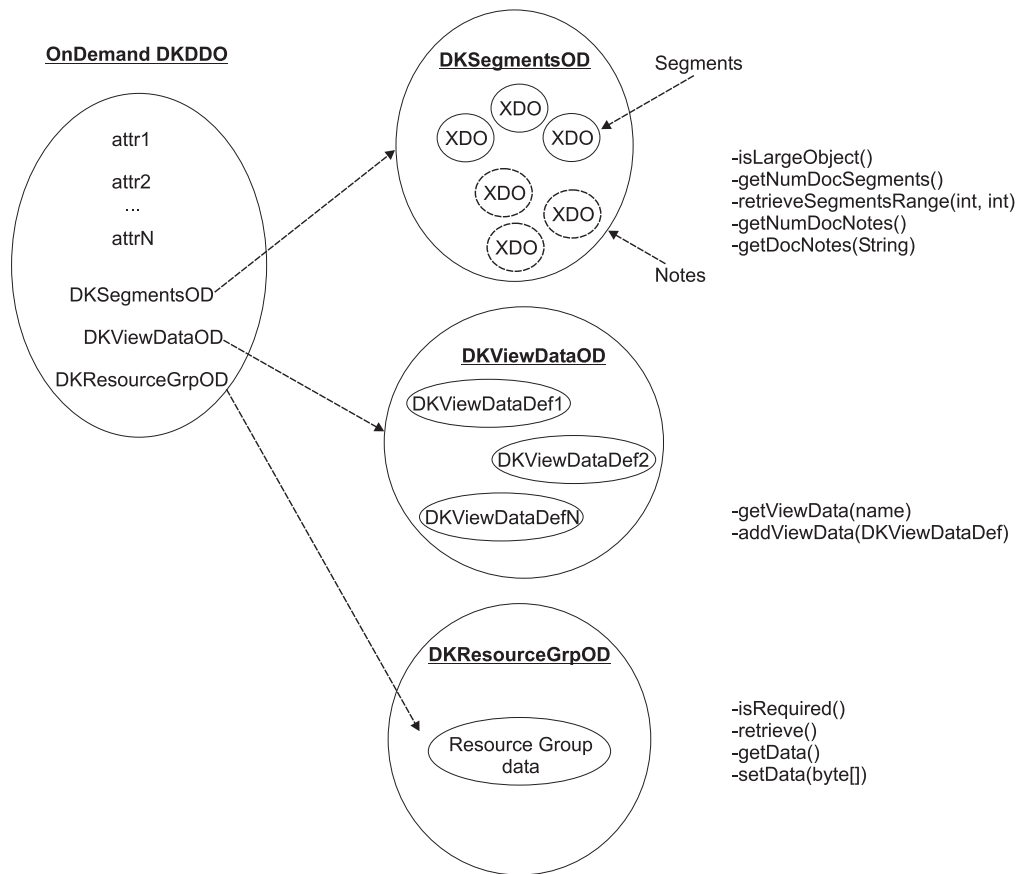


Figure 18. Example of data structure

Using the FeServerDefBase class

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.

Chapter 4. Using non-visual and visual JavaBeans

This chapter describes the non-visual and visual JavaBeans provided in Enterprise Information Portal .

Enterprise Information Portal provides 13 non-visual JavaBeans for use in Java applications, such as building servlets or JavaServer Pages (JSPs). Use non-visual beans in command line or windowed applications and as data models for the visual beans. Non-visual beans provide programming interfaces and components for applications created in a visual builder environment. The `com.ibm.mm.beans` package contains the non-visual beans.

Enterprise Information Portal provides seven visual JavaBeans to build Swing-based Java applications using the Enterprise Information Portal federated search capabilities. The `com.ibm.mm.beans.gui` package contains the classes for the visual beans.

Using JavaBeans in builders

This section explains how to use JavaBeans in IBM VisualAge for Java and in other builders.

Using IBM VisualAge for Java

You must be using a version of VisualAge for Java that supports Java 2. To add the JavaBeans to VisualAge for Java:

1. Create a new project called "EIP".
2. From the Workbench window, choose Selected | Add | Project.
3. Import the necessary jar files into the EIP project and click on EIP in the Projects tree of the Workbench window.
4. choose File | Import... and follow the instructions in the Smartguide to import each of the following:
 - \CMBROOT\lib\cmb71.jar
 - \CMBROOT\lib\cmbview71.jar
 - \CMBROOT\lib\cmbsdk71.jar
 - \CMBROOT\lib\lotuskms.jar
 - \CMBROOT\lib\eclisrv.jar
 - \SQLLIB\java\db2java.zip

The `cmb71.jar` file contains the visual and non-visual beans. The other jar files are used by the beans.

5. Replace `\SQLLIB` with the DB2 path and `\CMBROOT` with Enterprise Portal path.
6. Add the beans to a new category of the tools palette of the Visual Composition.
7. Open a class in the Visual Composition Editor.
8. Choose Bean | Modify Palette.
9. Create a new category called "EIP".
10. Select this category and press Browse.. to look for the EIP beans.

11. The beans classes begin with "CMB" and have associated BeanInfo class. Add each bean class to the category.

Using other builders

The Enterprise Portal JavaBeans can be used with other Java builders that support Java 2. Follow the builder's instructions for adding new jar files to add the jars specified in the instructions for VisualAge for Java above. Then, follow the builder's instructions for adding beans from a jar to add the EIP beans, which are in cmb71.jar.

Non-visual beans

EIP provides a set of non-visual JavaBeans for use in all Java applications. A typical use is in building Servlets or Java Server Pages (JSPs), although they can also be used in command line or windowed applications. The next section describes each non-visual bean.

The CMBROOT\Samples\java directory contains examples of the non-visual and utility beans.

CMBCConnection

This bean maintains the connection to the federated database and content servers. By default, a connection is made to a federated database, but a direct connection can also be made to a content server.

CMBCConnectionPool

This bean maintains a pool of CMBCConnection instances. It provides a performance optimization for server applications in situations where the same user ID is used by multiple users by avoiding a disconnect and reconnect when the a CMBCConnection instance for the same user ID is reused.

CMBDDataManagement

This bean provides EIP data manipulation functions, including creating, retrieving, updating, and deleting document and folder content and associated annotations. The bean provides checkin/checkout/unlock functions on data items. You can obtain an instance of this bean from CMBCConnection.

CMBDDocumentServices

This bean provides rendering and conversion of documents for thick and thin clients (see Document Viewing Services below). Conversion is not supported for AIX.

CMBExceptionHandler

This utility bean can be used to provide common exception handling for exception events generated by other nonvisual beans.

CMBQueryService

This bean provide query capabilities, either by search template or by using a query string.

CMBSchemaManagement

This bean provides access to available search templates, mapped entities and attributes. An instance of this bean can be obtained from CMBCConnection, in which case it returns schema information about the server or database connected to by CMBCConnection. Alternatively, you can create an instance separate from CMBCConnection, to gain access to schema information for any content server.

CMBSearchResults

This bean maintains search results that are generated by searches performed using CMBQueryService or CMBSearchTemplate.

CMBSearchTemplate

This bean provides a search template and methods to perform a search using the template. Instances of CMBSearchTemplate can be obtained from CMBSchemaManagement.

CMBTraceLog

This utility bean can be used to provide common trace event handling for the nonvisual beans. It can write the trace messages to a log file or display them in a window.

CMBUserManagement

This bean provides capabilities to view and modify the content server user ID mapping associated with an EIP user ID.

CMBWorkFlowDataManagement

This bean makes it possible to retrieve workflow data. You can obtain an instance of this bean from CMBCConnection.

CMBWorkFlowQueryService

This bean makes workflow queries possible. You can obtain an instance of this bean from CMBCConnection.

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 `cmbs.ini` file. The `cmbs.ini` file specifies whether the content server is local or remote.

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 (DemoSimpleApp1.java) is available in the Cmbroot/Samples/java/beans directory. The sample application shows how to:

1. Connect to the Enterprise Information Portal 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

```
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.util.*;
import com.ibm.mm.beans.gui.*;

import java.beans.*;
import java.util.*;

public class DemoSimpleApp1
{
    int docCounter = 0;
    TUtilPersistent per = new TUtilPersistent();
    TUtilSimpleUI ui = new TUtilSimpleUI();
    CMBSchemaManagement schemaBean = null;
    CMBSearchResults resultBean = null;
    CMBConnection connBean = null;
    CMBQueryService queryBean = null;
    CMBSearchTemplate stBean = null;
    CMBDataManagement dataBean = null;
    CMBTraceLog traceBean = null;
    CMBDocumentViewer viewerBean = null;

    public DemoSimpleApp1()
    {
        try {
            System.out.println("Creating CMBUserManagement bean using " +
                "ser/class/jar specified in the classpath.");
            connBean = (com.ibm.mm.beans.CMBConnection) Beans.instantiate(null,
                "com.ibm.mm.beans.CMBConnection");

            System.out.println("Creating CMBSchemaManagement Bean from " +
                "CMBConnection bean...");
            schemaBean = connBean.getSchemaManagement();

            System.out.println("Creating CMBQueryService Bean from " +
                "CMBConnection Bean...");
            queryBean = connBean.getQueryService();
            schemaBean.setCacheEnabled(true);

            System.out.println("Creating CMBDataManagement Bean from " +
                "CMBConnection Bean...");
            dataBean = connBean.getDataManagement();

            resultBean = (com.ibm.mm.beans.CMBSearchResults) Beans.instantiate(null,
                "com.ibm.mm.beans.CMBSearchResults");
            resultBean.setConnection(connBean);
```

In the constructor for the DemoSimpleAppl class, we create the beans that we will need to use later in our application. First, the connection bean is created. Then the beans that are created by the connection bean, schema and data management, and query service, are retrieved. The result bean is created, and its connection property is set.

```

        traceBean = (com.ibm.mm.beans.util.CMBTraceLog) Beans.instantiate(null,
            "com.ibm.mm.beans.util.CMBTraceLog");

        connBean .addCMBTraceListener(traceBean);
        schemaBean.addCMBTraceListener(traceBean);
        resultBean.addCMBTraceListener(traceBean);
        queryBean .addCMBTraceListener(traceBean);
        dataBean .addCMBTraceListener(traceBean);

        connBean .setTraceEnabled(true);
        schemaBean.setTraceEnabled(true);
        resultBean.setTraceEnabled(true);
        queryBean .setTraceEnabled(true);
        dataBean .setTraceEnabled(true);

        traceBean.setDebugLogEnabled(true);
        traceBean.setProgressLogEnabled(true);
    
```

In the above code, the trace log bean is created. Then tracing is enabled in all the non-visual beans, and the tracing bean added as a listener to the trace events generated by the beans. Then properties on the trace bean are set to show debug and progress events, and to display the debug messages in a window.

```

        viewerBean = new CMBDocumentViewer();
        viewerBean.setConnection(connBean);
    
```

Create the bean for viewing the documents which the application will retrieve in its search results. The CMBDocumentViewer visual bean will launch the appropriate viewer for the document, depending on its MIME type.

```

        }
        catch (Exception exc) {
            exc.printStackTrace();
        }
    
```

checkConnBeanProperty sets the server name, the name of the EIP federated database, that the connection uses, and the user ID and password for that database. These values are set as properties on the connection bean.

```

/**
 * Checks CMBCConnection bean property before connect.
 */
void
checkConnBeanProperty()
{
    try {
        if (connBean.getServerName().length() == 0) {
            String srv = ui.getString("Server name is empty, " +
                "please enter the server name:");
            connBean.setServerName(srv);
        }
        if (connBean.getUserid().length() == 0) {
            String uid = ui.getString
                ("User ID is empty, please enter the user ID:");
            connBean.setUserid(uid);
        }
        if (connBean.getPassword().length() == 0) {
            String pwd = ui.getString
                ("Password is empty, please enter the password:");
        }
    }
}
    
```

```

        connBean.setPassword(pwd);
    }
}
catch (Exception exc) {
    exc.printStackTrace();
}
}

void
connect()
{
    checkConnBeanProperty();

    try {
        System.out.println("Connecting...");
        connBean.connect();
        System.out.println("OK, connected\n");
    }
    catch (Exception exc) {
        exc.printStackTrace();
    }
}
}

```

The connect function connects to the EIP federated database, using the database(server) name, userid, and password properties that have already been set on the connection bean.

```

public void listTemplates()
{
    try {
        String[] stNames = schemaBean.getSearchTemplateName();
        System.out.println("There are " + stNames.length + " search templates:");
        for(int i=0; i<stNames.length; i++) {
            System.out.println("\t" + stNames[i]);
        }
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}

public void listSTCriteria()
{
    try {
        String stName = ui.getString("Please enter the search template name:");
        CMBSearchTemplate stObj = schemaBean.getSearchTemplate(stName);
        System.out.println("Search template '" + stName + "' retrieved...");
        String[] critNames = stObj.listSearchCriteriaNames();
        System.out.println("There are " + critNames.length + " criteria: ");
        for (int i=0; i<critNames.length; i++) {
            System.out.println("\t" + critNames[i]);
            CMBSTCriterion critObj = stObj.getSearchCriterion(i);
            System.out.println("\t\tdisplayName: " + critObj.getDisplayName());
            System.out.println("\t\tattrName: " + critObj.getAttrName());
            System.out.println("\t\tdisplayWidth: " + critObj.getDisplayWidth());
            System.out.println("\t\ttype: " + critObj.getType());
            System.out.println("\t\tdefaultValue: " + critObj.getDefaultValue());
            System.out.println("\t\tdefaultOp: " + critObj.getDefaultOperator());
            short[] validOps = critObj.getValidOperators();
            System.out.println("\t\tvalidOps:");
            for (int j=0; j<validOps.length; j++) {
                switch((int)validOps[j]) {
                    case CMBBaseConstant.CMB_OP_EQUAL:
                        System.out.println("\t\t\t.EQ."); break;
                    case CMBBaseConstant.CMB_OP_NOT_EQUAL:
                        System.out.println("\t\t\t.NEQ."); break;
                    case CMBBaseConstant.CMB_OP_LESS_EQUAL:
                        System.out.println("\t\t\t.LEQ."); break;
                }
            }
        }
    }
}

```



```

switch (critOp) {
    case 1:
        critObj.setOperator(CMBBaseConstant.CMB_OP_EQUAL); break;
    case 2:
        critObj.setOperator(CMBBaseConstant.CMB_OP_NOT_EQUAL); break;
    case 3:
        critObj.setOperator(CMBBaseConstant.CMB_OP_LESS_EQUAL); break;
    case 4:
        critObj.setOperator(CMBBaseConstant.CMB_OP_LESS); break;
    case 5:
        critObj.setOperator(CMBBaseConstant.CMB_OP_GREATER_EQUAL); break;
    case 6:
        critObj.setOperator(CMBBaseConstant.CMB_OP_GREATER); break;
    case 7:
        critObj.setOperator(CMBBaseConstant.CMB_OP_BETWEEN); break;
    case 8:
        critObj.setOperator(CMBBaseConstant.CMB_OP_NOT_BETWEEN); break;
    case 9:
        critObj.setOperator(CMBBaseConstant.CMB_OP_IN); break;
    case 10:
        critObj.setOperator(CMBBaseConstant.CMB_OP_NOT_IN); break;
}

public void runQuery()
{
    try {
        boolean runQry = false;
        String stName = ui.getString("Please enter the search template name:");
        CMBSearchTemplate stObj = schemaBean.getSearchTemplate(stName);
        if (stObj == null) {
            System.out.println("Unknown template name, please try again");
            return;
        }

        String[] critNames = stObj.listSearchCriteriaNames();
        System.out.println("There are " + critNames.length + " criteria: ");
        for (int i=0; i<critNames.length; i++) {
            System.out.println("\t" + critNames[i]);
        }

        while (!runQry) {
            System.out.println("Selections:");
            System.out.println("1. Specify a search criterion");
            System.out.println("2. Run query ...");
            switch (ui.getInteger("Enter your selection:")) {
                case 2: runQry = true;          break;
                case 1: getSearchCriterion(stObj);  break;
            }
        }
    }
}

```

The runQuery function requests the user to input the name of the search template with which they wish to perform their search. Then, as long as the user does not choose to run the query, it will allow them to specify values for more search criteria.

```

stObj.addCMBSearchReplyListener(resultBean);
stObj.setAsynchSearch(false);
queryBean.setAsynchSearch(false);
queryBean.runQueryWithCursor(stObj);
resultBean.newResults(queryBean.getResults());
CMBItem[] itemList = resultBean.getItem();
for (int k=0; k<itemList.length; k++) {
    CMBItem hit = (CMBItem)itemList[k];
    System.out.println("Hit item #" + k + ", pidString=" + hit.getPidString());
    try {
        String[] values = hit.getAttrValue();
        String[] names = hit.getAttrName();
        for(int I=0; I<values.length; I++) {

```



```

        System.out.println("\t" + names[1] + ": " + values[1]);
    if (docCounter < 5) {
        docCounter++;
        String dsType = hit.getServerType();
        System.out.println("Found an item of dsType '" + dsType +
            "' - about to bring up " +
            dsType + " viewer ...");
        dataBean.setDataObject(hit);
        viewerBean.showDocument(hit); //borrowed from visual bean
    System.out.println("Viewer bean called ...");
    }
}

```

In the code above, the search template is wired to send its search results to the result bean. Then the query is run synchronously, and once it is completed the results are retrieved from the results bean. For each result, the document viewer bean will launch the correct viewer for the document type.

```

        catch (CMBException ex) {
            ex.printStackTrace();
            Object data = ex.getErrorData();
            if (data != null && data instanceof Exception)
                ((Exception)data).printStackTrace();
        }
    }
}

case 11:
    critObj.setOperator(CMBBaseConstant.CMB_OP_LIKE); break;
case 12:
    critObj.setOperator(CMBBaseConstant.CMB_OP_NOT_LIKE); break;
default:
    break;
}
if (critOp == 7 || critOp == 8) {
    String critVal1 = ui.getString("Please enter 1st search value: ");
    String critVal2 = ui.getString("Please enter 2nd search value: ");
    String[] critVals = new String[2];
    critVals[0] = critVal1;
    critVals[1] = critVal2;
    critObj.setValues(critVals);
}
else if (critOp == 9 || critOp == 10) {
    Vector vals = new Vector();
    boolean notDone = true;
    while (notDone) {
        String val =
            ui.getString("Please enter value: (-1 if done entering)");
        if (val.equals("-1"))
            notDone = false;
        else
            vals.addElement(val);
    }
    String[] critVals = new String[vals.size()];
    vals.copyInto(critVals);
    critObj.setValues(critVals);
}
else {
    String critVal = ui.getString("Please enter search value: ");
    if (critVal.equals("null"))
        critObj.setValue(critVal);
    else
        critObj.setValue("'" + critVal + "'");
}
}
}

```

```

        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

```

These catch blocks will handle most errors that occurred during runQuery.

```

public static void main(String argv[])
{
    String testName = "IBM CMB Demonstration program: DemoSimpleAppl";
    String copyright = "C) Copyright IBM Corp. 1994,1999. All Rights Reserved.";

    System.out.println("");
    System.out.println(testName);
    System.out.println(copyright);
    System.out.println("");
    System.out.println("");

    TUtilSimpleUI ui          = new TUtilSimpleUI();
    DemoSimpleAppl psObj      = new DemoSimpleAppl();
    boolean        loopControl = true;

    System.out.println("Going to connect to the server...");
    psObj.connect();
    psObj.viewerBean.setConnection(psObj.connBean);

    while (loopControl) {
        System.out.println("");
        System.out.println("Selections:");
        System.out.println("0. Exit");
        System.out.println("1. List Search Templates");
        System.out.println("2. List Search Criteria");
        System.out.println("3. Run query ...");
        switch (ui.getInteger("Enter your selection:")) {
            case 0: loopControl = false;    break;
            case 1: psObj.listTemplates();  break;
            case 2: psObj.listSTCriteria();  break;
            case 3: psObj.runQuery();       break;
        }
    }
    try {
        psObj.connBean.disconnect();
        psObj.traceBean.finalize();
        System.out.println("Bye.");
        System.exit(0);
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
}

```

This is the main function of the DemoSimpleAppl class. It instantiates the user interface object, UtilSimpleUI, and the DemoSimpleAppl class itself. It then connects to the EIP federated database, and enters a loop. In this loop, the user can choose one of four functions: 1. Exit the application 2. List the Search Templates that exist in the EIP database 3. List the Search Criteria in a given search template 4. Run a query Once the user chooses to exit the application, the function disconnects from the EIP federated database and exits.

JavaServer Pages (JSP) and non-visual beans

For information about JavaServer Pages and non-visual beans, see "Chapter 6. Using the sample thin client" on page 187. For more information about JSPs and

Information Mining beans, see “Building an application using the Information Mining beans” on page 191 and “Understanding the Information Mining JSP applications” on page 212.

Working with visual beans

Visual beans allow you to integrate the functionality of the Enterprise Information Portal client into applications based on Swing. 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 the federated database. It also provides the window where 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

This bean displays a list of available search templates and allows selection of a template.

CMBSearchTemplateViewer

This bean displays a search template and provides fields for users to enter search criteria. It also performs a search.

CMBFolderViewer

Displays 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

CMBLogonPanel bean

The CMBLogonPanel bean (see Figure 19 on page 152) displays a window that lets users login to a content server and change a password.

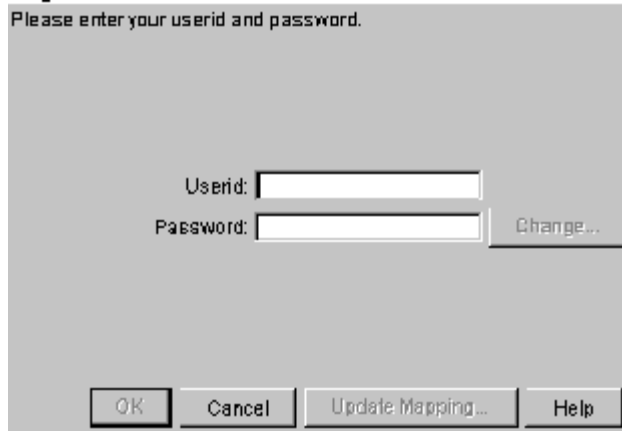


Figure 19. CMBLogonPanel bean

In the CMBLogonPanel bean, when a user clicks **Change**, the **Change Password** window appears (see Figure 20.) The user enters the old password, and enters the new password twice.

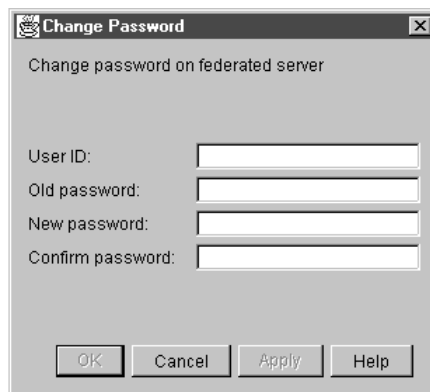


Figure 20. Change Password window

In the CMBLogonPanel bean, when a user clicks **Update Mapping** in the CMBLogonPanel bean, the **Update Userid Mapping** window opens (see Figure 21 on page 153). When you Update Mapping, you update the user ID and password specified for a server.

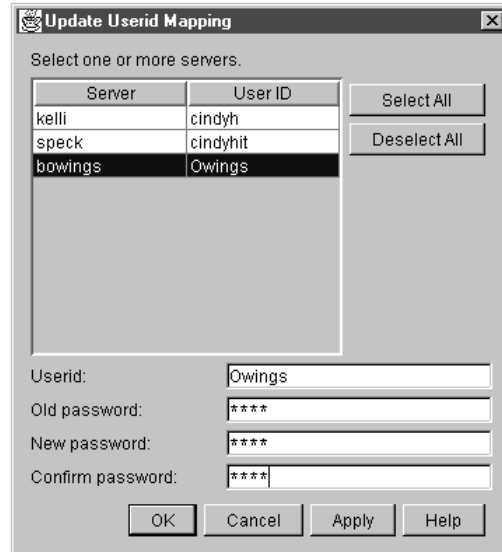


Figure 21. Update Userid Mapping window

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 22, uses one image for the backgrounds of the selected items and another for the unselected items. Figure 23 on page 154 shows the simple template list style. Figure 24 on page 154 shows the drop-down template list style.

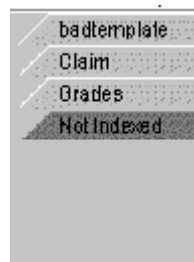


Figure 22. Image template list style

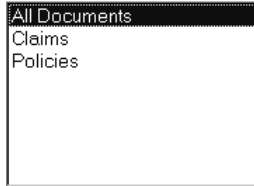


Figure 23. Simple template list style



Figure 24. Drop-down template list style

CMBSearchTemplateViewer bean

The CMBSearchTemplateViewer bean (see Figure 25) 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.

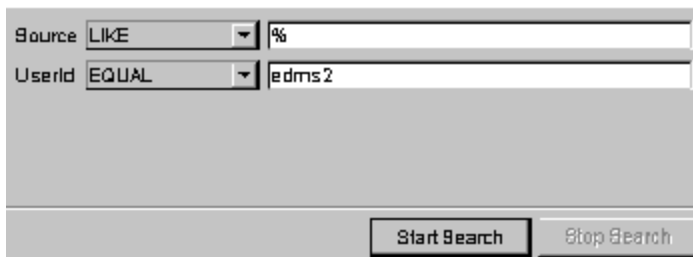


Figure 25. CMBSearchTemplateViewer bean

The CMBSearchTemplateViewer bean lists search criteria such as Source or UserId. Each search criteria has a label, an operator, 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.

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.

CMBSearchResultsViewer bean

The CMBSearchResultsViewer bean displays search results in a window that has a tree pane and a details pane. Users can resize the window by clicking and dragging on the line separating the panes.

Figure 26 shows the CMBSearchResultsViewer bean with the **Search Results** folder selected.

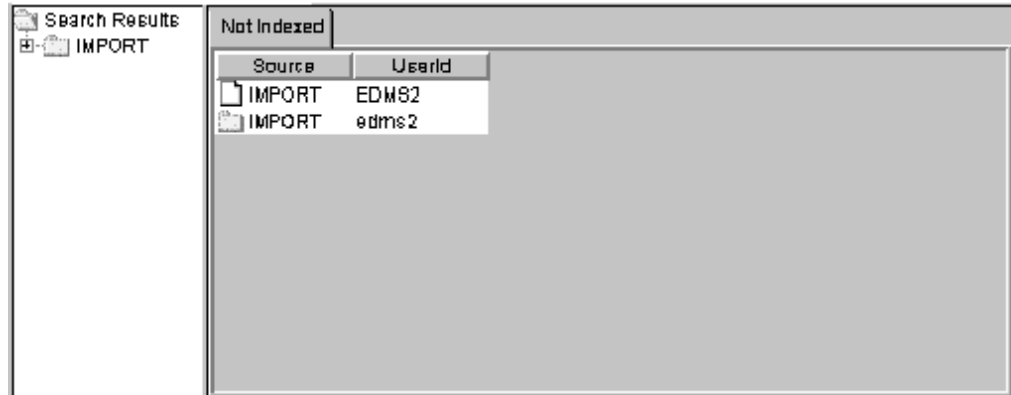


Figure 26. CMBSearchResultsViewer bean

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 name and *server* is the content server name. The table columns change to display the attributes according to the index class. Multiple selection is supported in the details pane. Turn off Multiple selection by setting the `MultiSelectEnabled` property:
`setMultiSelectEnabled(false)`.

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 27 shows the tree and details panes of the `CMBFolderViewer` bean.

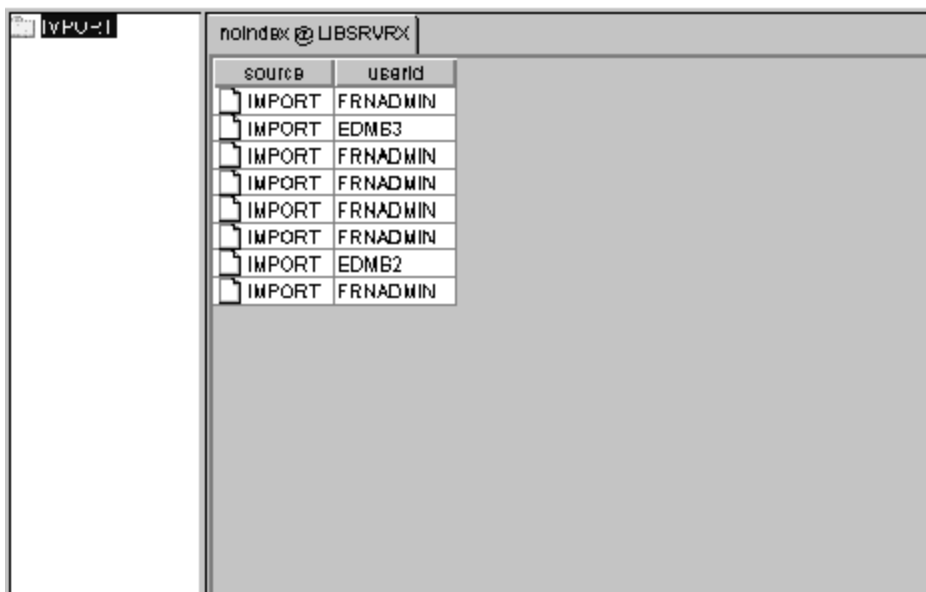


Figure 27. `CMBFolderViewer` bean

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 resizable 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 index class 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.

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. The Content Manager viewer also provides display and editing of annotations. 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. The sample application contains the following code, which launches wanging.exe, for viewing TIFF documents on Windows:

```
Properties apps = new Properties();
apps.put("image/tiff", "C:/Program Files/Windows/Accessories/ImageVue/wanging.exe");
documentViewer.setMime2App(apps);
```

CMBItemAttributesEditor bean

The CMBItemAttributesEditor bean (see Figure 28) displays a window for viewing and modifying the index class and indexing attributes of a folder or document.

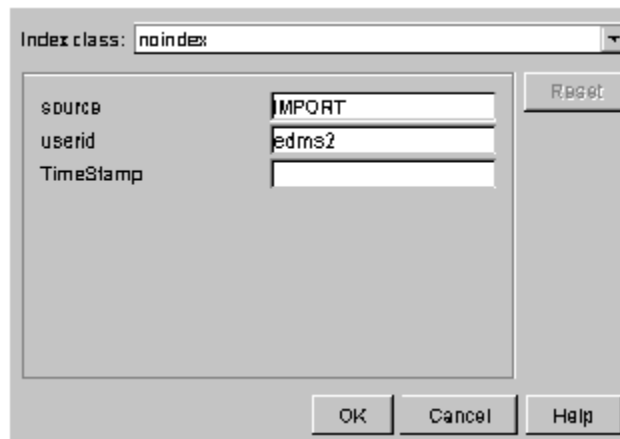


Figure 28. CMBItemAttributesEditor bean

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.

The following example shows how to reject a requested update if any attribute is blank:

```
itemAttributesEditor.addEditRequestedListener(new CMBEditRequestedListener() {
    public void onEditRequested(CMBEditRequestedEvent event) {
        String[] attributes = event.getAttributes();
        for (int i = 0; i < attributes.length; i++) {
            if (attributes[i].length() == 0)
                event.denyUpdate();
                JOptionPane.showMessageDialog(EditAttributesDialog.this,
                    "Blank attribute value", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
        }
    });
```

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 Push buttons that appear on all visual beans. Use the set<name> Button Visible property, where <name> is the name of the push 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.

The following code saves the properties of CMBSearchTemplateViewer and CMBSearchResultsViewer:

```
Properties properties = new Properties();
searchTemplateViewer.saveConfiguration(properties);
searchResultsViewer.saveConfiguration(properties);
try {
```

```

        FileOutputStream outStream = new FileOutputStream("SearchFrame.properties");
        properties.save(outStream,"MyApp");
    } catch (IOException e) {
    }
}

```

The following code restores the properties:

```

try {
    FileInputStream inStream = new FileInputStream("SearchFrame.properties");
    Properties properties = new Properties();
    properties.load(inStream);
    searchTemplateViewer.restoreConfiguration(properties);
    searchResultsViewer.restoreConfiguration(properties);
} catch (FileNotFoundException e) {
} catch (IOException e) {
}
}

```

Help events

Each visual bean generates a CMBHelp event when the user requests help, either by clicking **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 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 18.

Table 18. Visual beans and key events

Visual bean	Key events
CMBSearchTemplateList	CMBTemplateSelectedEvent
CMBSearchTemplate Viewer	CMBSearchStartedEventCMBSearchResults Event
CMBSearchResultsViewer	CMBViewDocumentEvent CMBViewFolder Event-CMBEditItemAttributesEvent
CMBFolderViewer	CMBViewDocumentEvent CMBEditItem AttributesEvent
CMBDocumentViewer	CMBDocumentOpenedEvent CMBDocument Closed Even
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.

The following code examples shows a `TabbedPane` replacing the `CMBSearchTemplateList` bean, where each tab is the name of a search template and the component associated with the tab is the `CMBSearchTemplateViewer` bean. This example can be added to the `SearchFrame.java` class:

```

topPanel.remove(searchTemplateList);
topPanel.remove(searchTemplateViewer);
topPanel.add(templateNotebook, "Center");
connection.addCMBCConnectionReplyListener(new
CMBCConnectionReplyListener() {
    public void onCMBCConnectionReply(CMBCConnectionReplyEvent evt) {
        templateNotebook.removeAll();
        if (connection.isConnected()) {
            CMBSchemaManagement mgt = connection.getSchemaManagement();
            try {
                mgt.clearSchemaCache();
                String[] names = mgt.getSearchTemplateName();
                for (int i = 0; i < names.length; i++) {
                    if (i == 0) {
                        templateNotebook.add(names[i],searchTemplateViewer);
                        CMBSearchTemplate template = mgt.getSearchTemplate(names[i]);
                        searchTemplateViewer.setTemplate(template);
                        searchResultsViewer.setTemplate(template);
                    } else
                        templateNotebook.add(names[i],new JPanel());
                }
            } catch (Exception e) {}
        }
    }
});
templateNotebook.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent event) {
        templateNotebook.setComponentAt(templateNotebook.
            indexOfComponent(searchTemplateViewer),
            new JPanel());
        templateNotebook.setComponentAt(templateNotebook.getSelectedIndex(),
            searchTemplateViewer);
        try {
            CMBSearchTemplate template =
                connection.getSchemaManagement().getSearchTemplate(templateNotebook.getTitleAt
                    (templateNotebook.getSelectedIndex()));
            searchTemplateViewer.setTemplate(template);
            searchResultsViewer.setTemplate(template);
        } catch (Exception e) {}
    }
});

```

Building an application using visual beans

The following sections show how the visual beans fit together when you build an application.

A sample application

The following sample application consists of three classes and uses every bean except `CMBFolderViewer`.

SearchFrame

Display the main application window

LogonDialog

Displays the logon panel

EditAttributesDialog

Displays the item attributes editor

SearchFrame: SearchFrame is the main application window. SearchFrame uses CMBSearchTemplateList, CMBSearchTemplateViewer, CMBSearchResultsViewer, and CMBDocumentViewer beans.

```
import java.util.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.gui.*;

/** Sample application showing the use of Enterprise Information Portal
    visual JavaBeans. */
public class SearchFrame extends JFrame
{
    CMBConnection connection = new CMBConnection();
    CMBSearchTemplateList searchTemplateList = new CMBSearchTemplateList();
    CMBSearchTemplateViewer searchTemplateViewer = new
    CMBSearchTemplateViewer();
    CMBSearchResultsViewer searchResultsViewer = new CMBSearchResultsViewer();
    CMBDocumentViewer documentViewer = new CMBDocumentViewer();

    // Set look and feel to windows
    static {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public static void main(String[] args) {

        // Create the frame and display it
        SearchFrame searchFrame = new SearchFrame();
        searchFrame.setVisible(true);

        // Display modal logon dialog
        LogonDialog logonDialog =
            new LogonDialog(searchFrame,searchFrame.connection);
        logonDialog.setVisible(true);
    }

    public SearchFrame() {

        // add cc beans to main frame
        getContentPane().setLayout(new BorderLayout());
        JPanel topPanel = new JPanel(new BorderLayout());
        topPanel.add(searchTemplateList, "West");
        topPanel.add(searchTemplateViewer, "Center");
        getContentPane().add(topPanel, "North");
        getContentPane().add(searchResultsViewer, "Center");
        getContentPane().add(documentViewer, "East");

        // set connection property on all visual beans
        searchTemplateList.setConnection(connection);
        searchTemplateViewer.setConnection(connection);
        searchResultsViewer.setConnection(connection);
        documentViewer.setConnection(connection);

        // set other properties on cc beans
        try {
            connection.setServerName("cmbddb");
        } catch (PropertyVetoException e) {
        }
    }
}
```

```

searchTemplateViewer.setStartSearchButtonVisible(true);
searchTemplateViewer.setStopSearchButtonVisible(true);
searchResultsViewer.setDetailsHorizontalLinesVisible(false);
searchResultsViewer.setDetailsVerticalLinesVisible(false);
searchResultsViewer.setSaveResultsPerTemplate(true);
documentViewer.setVisible(false);

// Uncomment to launch Wang viewer for TIFF documents on Windows
/* Properties apps = new Properties(); apps.put("image/tiff","C:/Program
Files/Windows NT/Accessories/ImageVue/wangimg.exe");
documentViewer.setMime2App(apps); */

// Restore saved session configuration of beans
try {
FileInputStream inStream = new FileInputStream("SearchFrame.properties");
Properties properties = new Properties();
properties.load(inStream);
searchTemplateViewer.restoreConfiguration(properties);
searchResultsViewer.restoreConfiguration(properties);
} catch (FileNotFoundException e) {
} catch (IOException e) {
}

// "wire" the beans together
searchTemplateList.addTemplateSelectedListener(searchTemplateViewer);
searchTemplateList.addTemplateSelectedListener(searchResultsViewer);
searchTemplateViewer.addSearchStartedListener(searchResultsViewer);
searchTemplateViewer.addSearchResultsListener(searchResultsViewer);
searchResultsViewer.addViewDocumentListener(documentViewer);

// 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);
    }
});

// Event to disconnect and shutdown on window close
addWindowListener(new WindowAdapter() {
public void windowClosing(WindowEvent event) {

// Save configuration of beans
Properties properties = new Properties();
searchTemplateViewer.saveConfiguration(properties);
searchResultsViewer.saveConfiguration(properties);
try {
    FileOutputStream outStream = new FileOutputStream("SearchFrame.properties");
    properties.save(outStream,"SearchFrame Properties");
} catch (IOException e) {
}
documentViewer.terminate();
try {
    connection.disconnect();
} catch (Exception e) {
    JOptionPane.showMessageDialog(SearchFrame.this,"Logoff
failed","Error",JOptionPane.ERROR_MESSAGE);
}
System.exit(0);
}
});
}

/** called when frame is created. Size and position window. */
public void addNotify() {

```

```

        super.addNotify();
        setSize(640,440);
        Dimension size = getSize();
        Dimension screenSize = getToolkit().getScreenSize();
        setLocation((screenSize.width-size.width)/2, (screenSize.height-size.height)/2);
    }
}

```

LogonDialog: The code example that follows shows LogonDialog, a window for logging on. It uses the CMBLogonPanel bean.

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.gui.*;

/** Dialog to logon using the Enterprise Information Portal CMBLogonPanel bean. */
class LogonDialog extends JDialog {

    CMBLogonPanel logonPanel = new CMBLogonPanel();

    LogonDialog(SearchFrame searchFrame, CMBCConnection connection) {
        super(searchFrame,"Logon",true);
        logonPanel.setConnection(connection);
        getContentPane().add(logonPanel);
        setSize(400,300);
        Dimension size = getSize();
        Dimension screenSize = getToolkit().getScreenSize();
        setLocation((screenSize.width-size.width)/2, (screenSize.height-size.height)/2);
        logonPanel.addLogonCompletedListener(new CMBLogonCompletedListener() {
            public void onLogonCompleted(CMBLogonCompletedEvent event) {
                dispose();
            }
        });
        logonPanel.addLogonCancelledListener(new CMBLogonCancelledListener() {
            public void onLogonCancelled(CMBLogonCancelledEvent event) {
                System.exit(0);
            }
        });
    }
}

```

EditAttributesDialog: EditAttributesDialog is a window where users can view and change document or folder attributes. It uses the CMBItemAttributesEditor bean.

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.gui.*;

/** Dialog to edit attributes using the EIP CMBItemAttributesEditor bean.*/
class EditAttributesDialog extends JDialog {

    CMBItemAttributesEditor itemAttributesEditor = new CMBItemAttributesEditor();

    EditAttributesDialog(SearchFrame searchFrame, CMBCConnection connection,
        CMBItem item) {
        super(searchFrame,"Edit Attributes",true);
        itemAttributesEditor.setConnection(connection);
        itemAttributesEditor.setItem(item);
        getContentPane().add(itemAttributesEditor);
        setSize(400,300);
        Dimension size = getSize();
    }
}

```



```

Point parentLocation = searchFrame.getLocation();
Dimension parentSize = searchFrame.getSize();
setLocation(
    parentLocation.x+(parentSize.width-size.width)/2,
    parentLocation.y+(parentSize.height-size.height)/2);
itemAttributesEditor.addEditCompletedListener(
    (new CMBEditCompletedListener() {
        public void onEditCompleted(CMBEditCompletedEvent event) {
            dispose();
        }
    }));
itemAttributesEditor.addEditCancelledListener(new CMBEditCancelledListener()
{
    public void onEditCancelled(CMBEditCancelledEvent event) {
        dispose();
    }
});
}
}

```

Connecting the visual beans

This section explains one scenario for connecting visual beans to create a simple application. Except for the **Search** push 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, similar to:

```
CMBSearchTemplateList1.addItemSelectedListener(CMBSearchTemplateViewer1);
```

To add a push button for launching searches, use a standard JButton. Create an inner class to cause the action event from the push button to invoke the appropriate method, for example:

```

JButton1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        CMBSearchTemplateViewer1.startSearch();
    }
}
}

```

In Figure 29 on page 166, 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, similar to:

```
CMBLogonPanel1.setConnection(CMBConnection1);
```

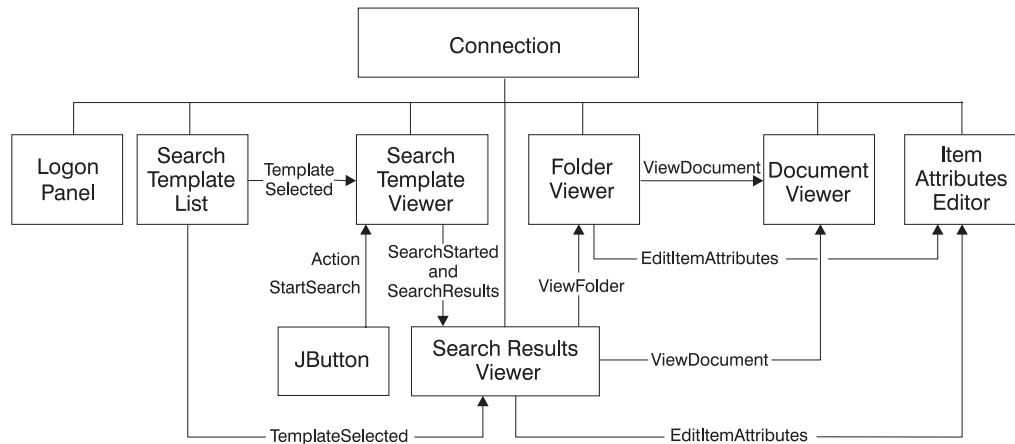


Figure 29. Visual bean connections

Figure 29 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. Although not shown in Figure 29, this event can be used to enable the **Search** push 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);
```

Chapter 5. Using the sample Java applets and servlet

Web administrators use the Java applets to quickly create a Web site for connecting, querying, and accessing Enterprise Information Portal. The Java applets, also known as the Internet application toolkit, are built using the Java API, the Java Development Toolkit 1.1 using Remote Method Invocation (RMI) for remote access, and the new Delegation Event Model. The applets only work with Content Manager.

The Internet application toolkit consists of the connect, query, and view applets, and the retrieve servlet. The applets work in conjunction to give easy access to Enterprise Information Portal. The applet works in conjunction to give easy access to Content Manager. The retrieve servlet provides flexibility for returning and displaying various data types that can be returned and displayed in a Web browser.

The Internet application toolkit has the following three variations:

DTApp

Use this application to browse the data in accessible Enterprise Information Portal servers.

To run on AIX enter:

```
. DTApp.ksh
```

To run on Windows NT: Click **Start Programs** → **Content Manager** → **Internet Application Toolkit**.

dtappsrvr.html

Use this sample file to run the Internet application toolkit locally in a Web browser.

To run, open dtappsrvr.html in a Web browser that fully supports JDK 1.1, such as the HotJava Browser 1.0 FCS.

Requirement: You must install the client toolkit on the workstation machine.

dtappclt.html

Use this sample file to access the Internet application toolkit from a workstation that does not have any Content Manager components installed.

To use dtappclt.html, you must set up a Web site and other configurations that are explained in detail in the following sections.

Understanding the connect applet

The connect applet provides a simple window for users to connect to different servers and check on connection status. As shown in Figure 30 on page 170, it has fields for the user ID and password, and lists of the accessible content servers and their types.

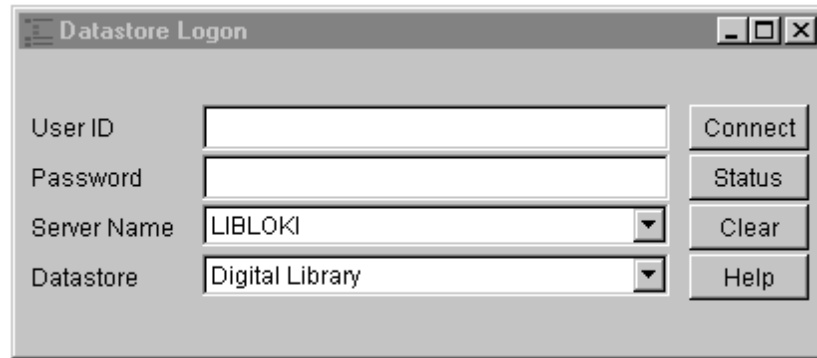


Figure 30. Datastore Logon window

The connect applet has the following parameters:

Country

National language and formats that supersede LanguageCode and RegionCode (for example, Japan).

LanguageCode

Language code, which must have the corresponding region code (for example, ja).

RegionCode

Region code which must have the corresponding language code (for example, JP).

RemoteHost

RMI server host name. Used by remote clients to access the correct RMI server.

RemotePort

RMI server Port number. Used by remote clients to access the correct RMI port number.

Understanding the query applet

The query applet lists the entities and their attributes of the connected datastore and provides a window where users can construct queries. Users can search for attribute values and text strings, or combine them to enrich the search capability. The search results display in the view applet while the query runs. The query applet has no parameters. Figure 31 on page 171 shows an example of a Parametric Search page.

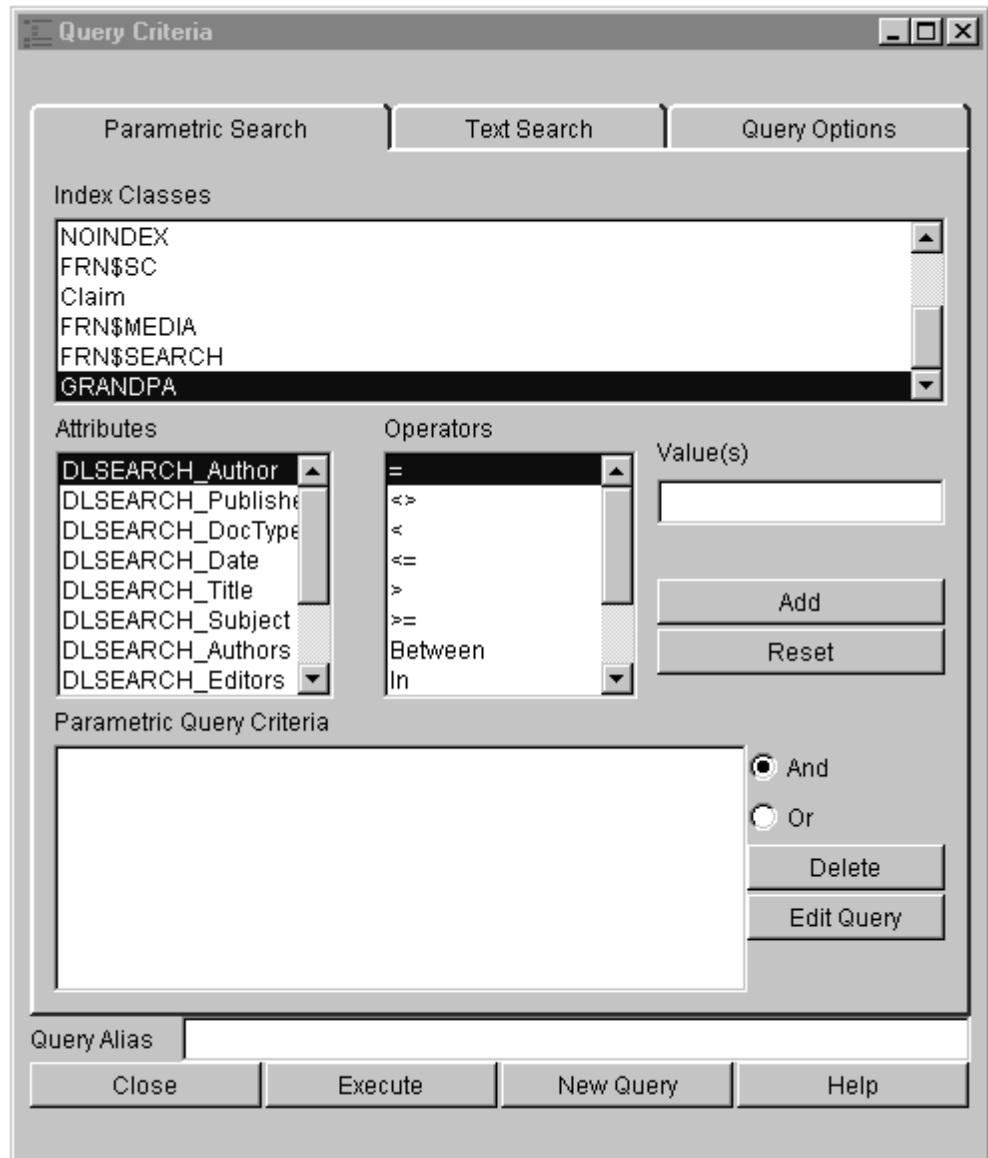


Figure 31. Parametric Search page of the Query Criteria notebook

The Parametric Search page displays the list of available index classes and the attributes associated with each index class. Clicking on the attributes and operators and entering the values constructs a parametric search string which is shown in the lower section of the page.

Users can change the query by either selecting the current criteria and resetting them, or by clicking **Edit Query** and modifying the parametric search string directly. Modifications made using the **Edit Query** push button disassociate the search string from other modification methods until a new query is done. Figure 32 on page 172 shows an example page from the Query Criteria notebook.

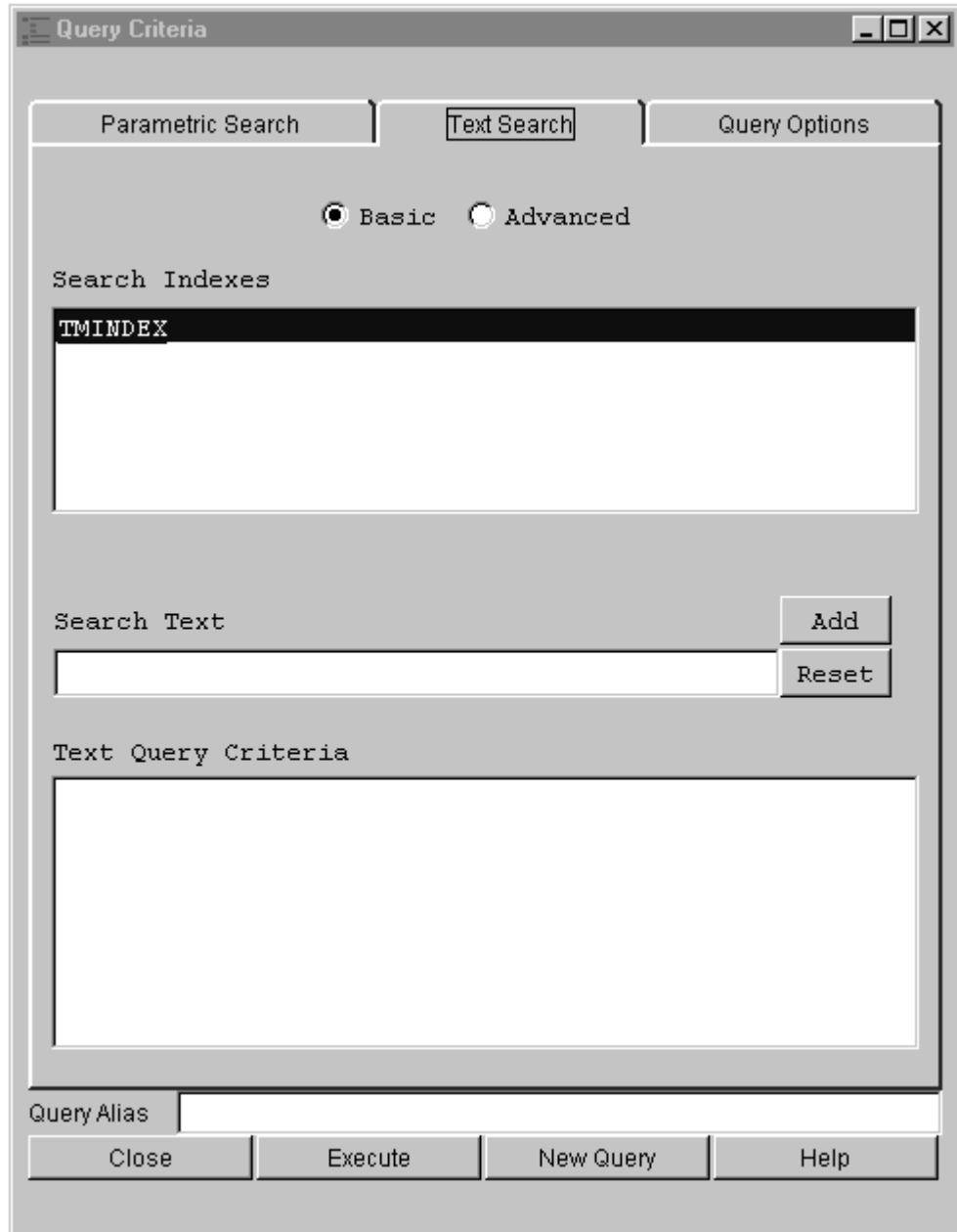


Figure 32. Text search page of the Query Criteria notebook

The Text search page displays the list of text search index classes and provides a field for entering a search string. Users can either enter a simple boolean search, or click **Advanced** to create a more complicated query.

Combining the Parametric Search page and the Text search page allows you to do a combined search. If a search string is entered in both the Parametric Search page and the Text search page, the application performs a combined search and displays the intersection of the two search results.

The last page, Query Options, allows users to set generic options for all queries. The Query applet does not have any parameters.

Understanding the view applet

The view applet displays the search results in a tree structure. Users can click to expand or collapse an item in the result list. Users can also create, retrieve, update, and delete objects if they have the privilege to do so. Select an item from the result list and click the right mouse button to display the pop-up menu that shows the actions that you can perform on that data object. Figure 33 shows an example of a view applet display.

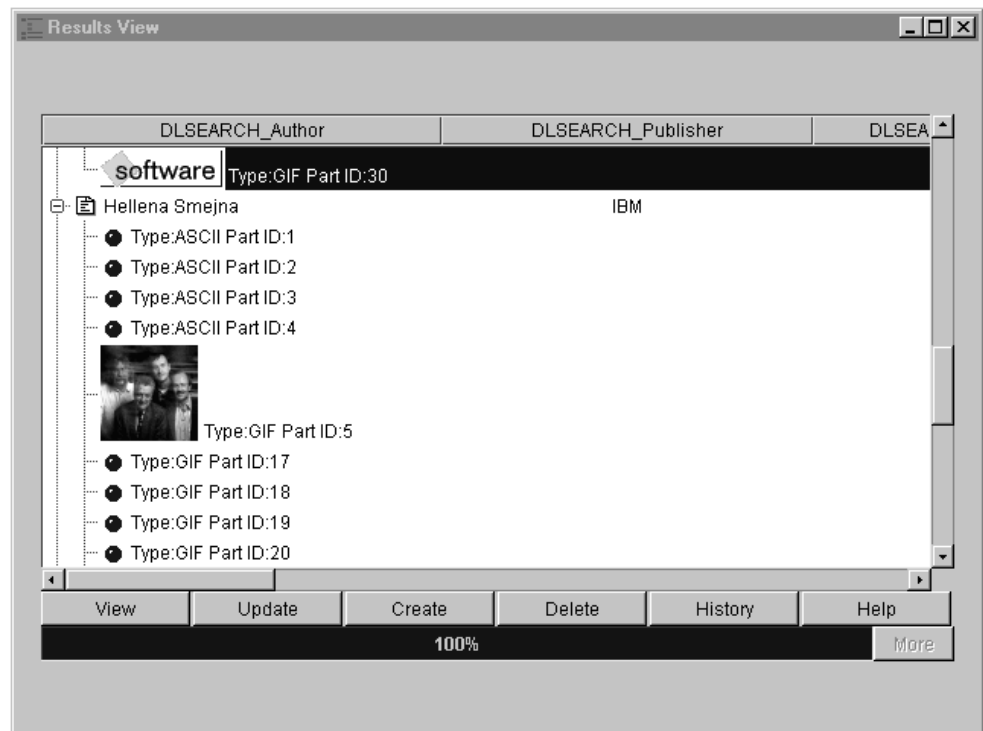


Figure 33. Results View window

Individual item parts can be viewed separately with the view applet, which displays GIF and JPEG files, or in conjunction with the retrieve servlet, which displays the part through a Web browser. To view the item in its entirety, users right-click an item and select the menu item for the Dynamic Page Builder website that has been set up with the corresponding connections and data. All of the item parts are brought together and displayed as one part through the Web.

The view applet has the following parameters:

cgisource

The Web address for the retrieve servlet. For example, <http://voodoo.stl.ibm.com:8080/servlet/DTRRetrieve?>

indexClassList

The list of index classes, their attributes, and the thumbnail part. For example, (GRANDPA)(1 2 3)(1) means show the first three attributes of index class GRANDPA, and the first image part is the thumbnail (specifying 0 does not show the thumbnail).

advance

The flag for showing the advanced features—create, update, and delete. The default value is true.

Understanding the retrieve servlet

The retrieve servlet is used by the view applet to retrieve and display various types of parts from Enterprise Information Portal. The servlet is located in the `cmbroot\lib\servlet` directory, which contains the following four files:

DTRetrieve.java

The source code for the servlet

You can modify the source as needed and use it to better understand how the servlet works. As new MIME types are required, you can modify DTRetrieve to handle them.

DTRetrieve.class

The executable file for the servlet

Servlets.properties

The file where you put initialization parameter values

DTRetrieve has three initialization parameters:

datastore

The name of the library server to connect to

Restriction: Currently, the Content Manager client for AIX is single-threaded and the servlet servers operate by spawning new threads for each request. Two or more requests arriving at the retrieve servlet at the same time cause exceptions and the servlet server might stop.

user ID

The user ID to connect with

password

The password to connect with

Classes.zip

The servlet class library files

For the servlet to run, you need either a Web server capable of running servlets or `srun.exe`, which is part of the Java Servlet Development Package available from JavaSoft at <http://www.javasoft.com>.

The Solaris package works for AIX also. To run `srun`, enter `srun -s servlets.properties` from the directory `cmbroot\lib\servlet`. This starts a servlet server with the initial arguments set in `servlets.properties`. If your Web server supports servlets, refer to your Web server documentation to set up the servlet (for example, where to put the servlet).

If you are using `srun.exe`, enter

```
srun.exe -s servlets.properties
```

in the servlet directory. For more information, enter `srun -?`.

For more information about servlets, go to <http://www.javasoft.com> and search for the Java Servlet Development Kit (JSDK).

Running applets in a Java application

DTApp.bat is the file that runs the applets as a Java application. Enter dtapp at a command prompt to start the applets, retrieve the list of available content servers and open the window shown in Figure 34.

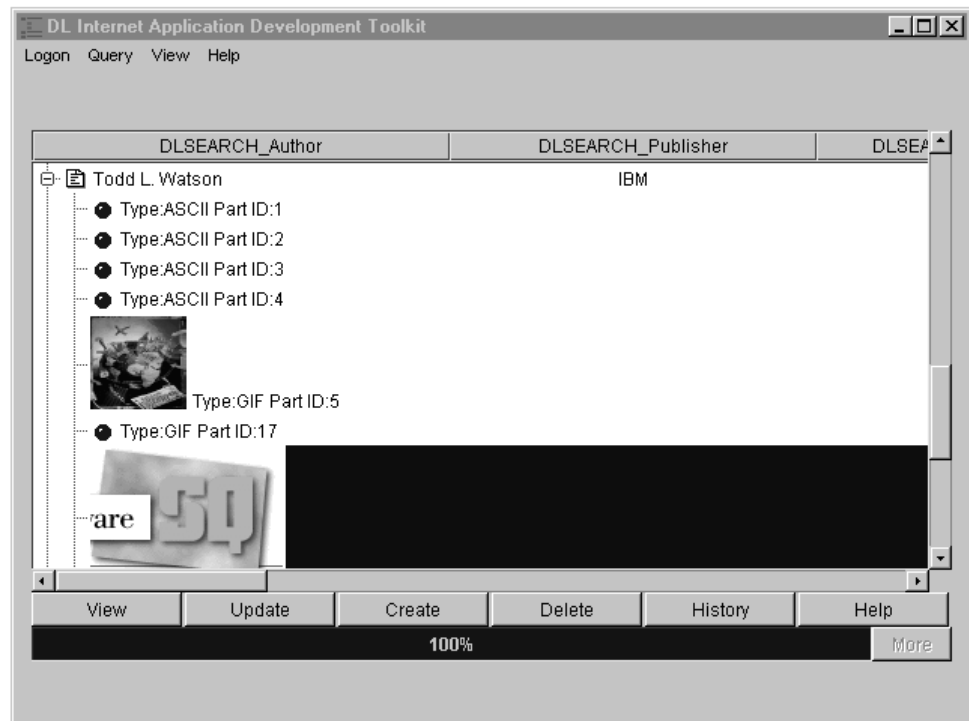


Figure 34. Main application window

You can set two view settings:

View→Set Web Browser

Allows you to specify the Web browser to use when viewing parts that are not GIF or JPEG files. For example, you might type:
c:\netscape\program\netscape.exe

View→Set Web Server

Allows you to specify the Web server address to use. For example:
http://webserver/servlet/DTRetrieve?

When retrieving files that are not GIF or JPEG types, ensure that both of the above options are set correctly and that srun is running correctly for the servlet.

Accessing local applets

A sample HTML page, dtappsvr.html, incorporates the DTConnectApplet, DTQueryApplet, and DTTreeViewApplet and allows users to log onto, query, and view data from Enterprise Information Portal. To access this HTML page:

1. Go to the cmbroot\lib\servlet directory and modify the servlets.properties file to have the correct content server name, user ID, and password for connecting to the correct server.

If you do not modify `servlets.properties`, the current `DTRetrieve.class` is compiled with `server = LIBSRVR2`, `user ID = FRNADMIN`, and `password = password`.

2. In a Web browser page `file:/cmbroot/dtappsrvr.html` where `cmbroot` is the directory path for `dtappsrvr.html`. For example, `c:\cmbroot`. This starts the three applets.

The connection window opens. Connect to the content server with the correct user ID and password, run a query, and view the results.

Restriction: You must use a Web browser that is enabled for JDK 1.1. Currently, the HotJava Browser 1.0 FCS is the only Web browser fully enabled for JDK 1.1. Within the HotJava Browser, make sure that the security level for unsigned applets is medium.

Accessing remote applets

A sample HTML page, `dtappclt.html`, incorporates three applets—`DTConnectApplet`, `DTQueryApplet` and `DTTreeViewApplet`—and allows users to remotely log onto, query, and view data from Enterprise Information Portal. To access this HTML page:

1. Make sure that `cmbroot` is set to an HTML-accessible directory for the Web server residing on your machine. If not, consult either your Web administrator or your Web server documentation to have this set up.
2. Go to the `cmbroot\lib\servlet` directory and modify the `servlets.properties` file to set the correct content server name, user ID, and password for connecting to the correct server.

If you do not modify `servlets.properties`, the current `DTRetrieve.class` is compiled with `server = LIBSRVR2`, `user ID = FRNADMIN`, and `password = password`.

3. Edit `dtappclt.html` to remove the comment tags from `RemoteHost` and `RemotePort`, after the definition of `DTConnectApplet.class`. Set `RemoteHost` to the host name of the Web server. Set `RemotePort` to the port number that the RMI server is using.

Tip: Alternatively, the `cmbclient.ini` file, which is located in the `cmbroot` directory, can be modified with the Web server's host name at the `HOSTNAME` keyword and a port number at the `PORTNUMBER` keyword. The port number should be a unique port number, which will be used for RMI communication. The `cmbclient.ini` file should be stored in the same directory as the client Web browser's invocation directory.

4. On Windows 95 and Windows NT, modify the `cmbregist.cmd` file. On AIX, modify the placeholder file, which is located in the `cmbroot\lib` directory. Modify the file to use the same port number as the `RemotePort` parameter in `dtappclt.html`, or the same `PORTNUMBER` in `cmbclient.ini`, if `cmbclient.ini` is used.
5. Go to the `cmbroot\lib` directory and enter `cmbregist` to initialize the RMI server.
6. Go to a browser and invoke the Web page `dtappclt.html` from your Web server's alias, `http://webserver/cmbroot/dtappclt.html`. This brings up the Web page and starts the three applets.

The connection window opens when the applets have been properly loaded and have started up. Connect with the correct user ID/password/server combination, make a query against the index class, and view the items sent back.

Restrictions:

- You must use a Web browser that is enabled for JDK 1.1. Currently, the HotJava Browser 1.0 FCS is the only fully JDK 1.1 enabled Web browser. Within the HotJava Browser, make sure that the security level for unsigned applets is medium.
- The Content Manager client for AIX is currently single-threaded, so any one process on AIX can have only one connection at a time. This means that there can be only one remote connection at a time through one RMI process.

Displaying part information

The info function displays Content Type, Size and other information for the selected part. A part must have a part ID and can also have a pre-defined type to describe its content. The following screen captures show parts indexed, respectively, by a text search server, an image search server, and nothing. Figure 35 shows an Info window example for an ASCII file indexed by Text search. Figure 36 on page 178 shows an Info window example of a GIF file indexed by Image Search. Figure 37 on page 178 shows an example of a non-indexed GIF file.

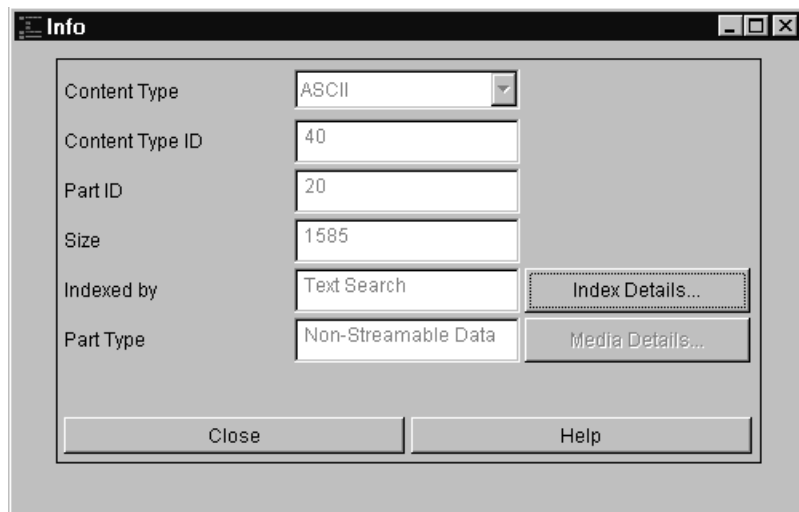


Figure 35. Info window for an ASCII file

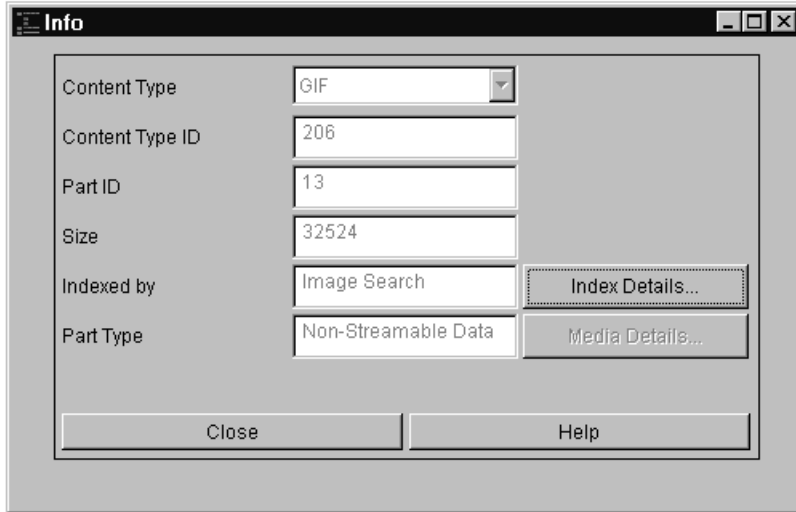


Figure 36. Info window for a GIF file indexed by Image Search

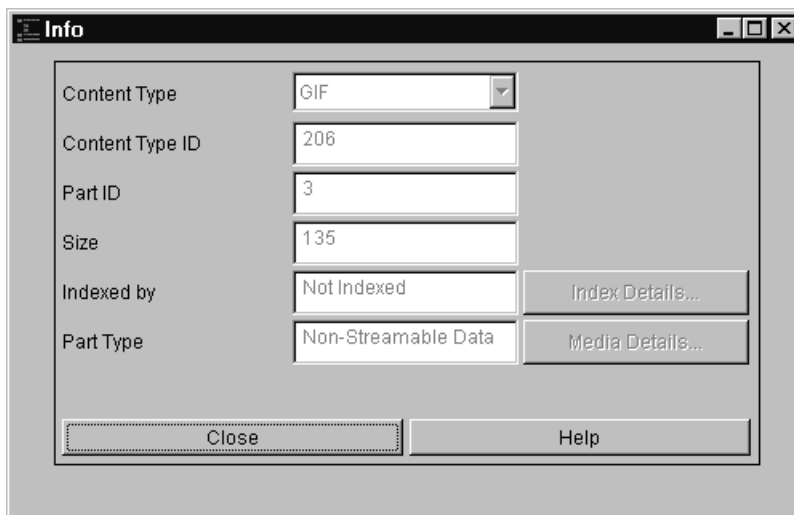


Figure 37. Info window for a non-indexed GIF file

The Index Details window shows the search engine type, server, and index information for the selected part. The screen captures in Figure 38 on page 179 and Figure 39 on page 179 show examples of parts indexed by Text search or an image search server.

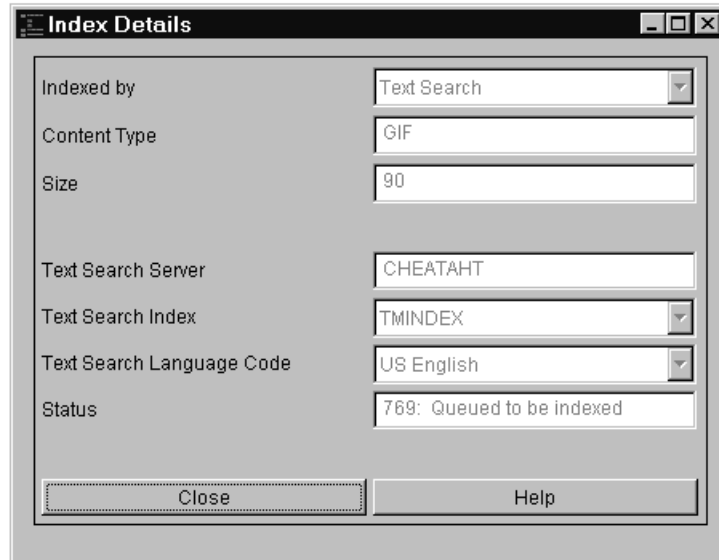


Figure 38. Index Details for GIF file indexed by Text Search

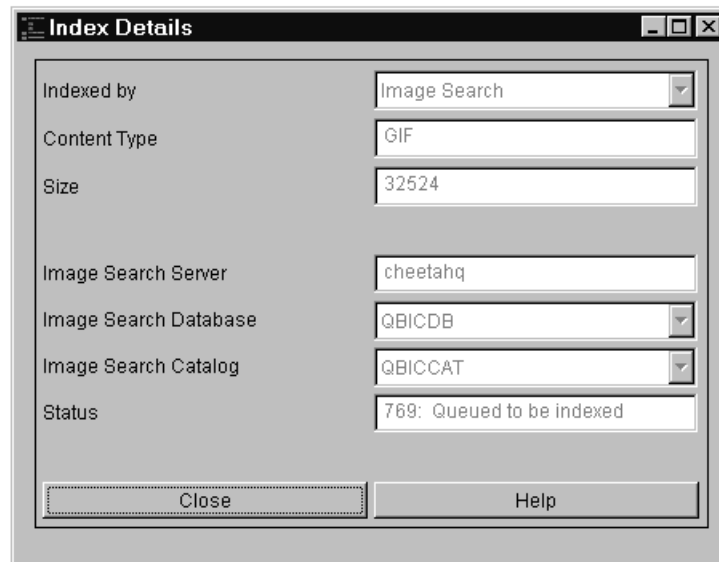


Figure 39. Index Details for GIF file indexed by QBIC

Indexing a part

Use the index part function to index a part while loading, updating, or indexing an existing part. If you set the indexing information when you create a part, the part is created with its indexing information. When you update a part, if the part has not been previously indexed, you can set the indexing information and it will be indexed when it is updated. If the part was previously indexed, it is indexed using existing indexing information when it is updated. To index a part while the part is being loaded or updated, click **Set Indexing** in the Create or Update window. Figure 40 on page 180 shows an example of a Text part in the Create window.

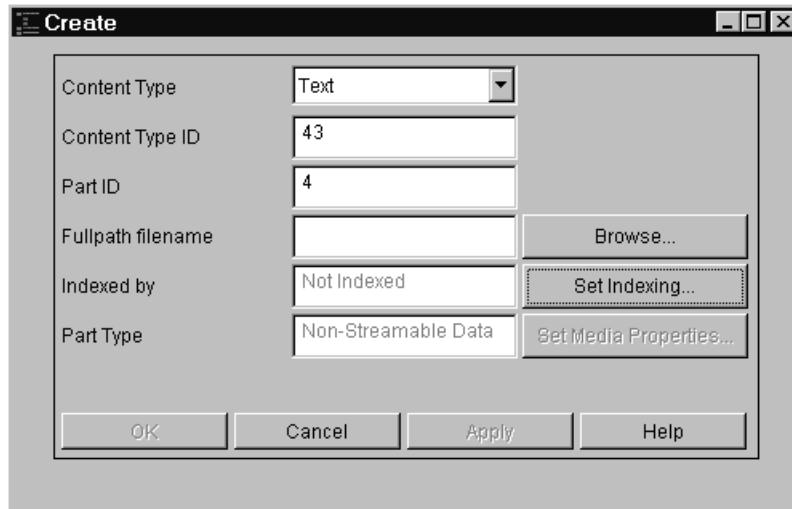


Figure 40. the Create window

To index an existing part, select a part from the query result tree and then select **Index** from the pop-up menu. The window shown in Figure 41 opens.

If the search server is not connected when you select a search type, the Logon window will automatically pop up. At this point, you must select a server to connect to. You need to know which search server is associated with the currently connected server. If the parts have already been indexed by any search engine, those parts cannot be indexed again. In this situation, selecting the **Index** menu choice will open the Index Details window. This rule applies to the update method as well.

Selecting the search engine type from the **Indexed by** list updates the window with the necessary values. The windows are shown in Figure 41, and Figure 42 on page 181.

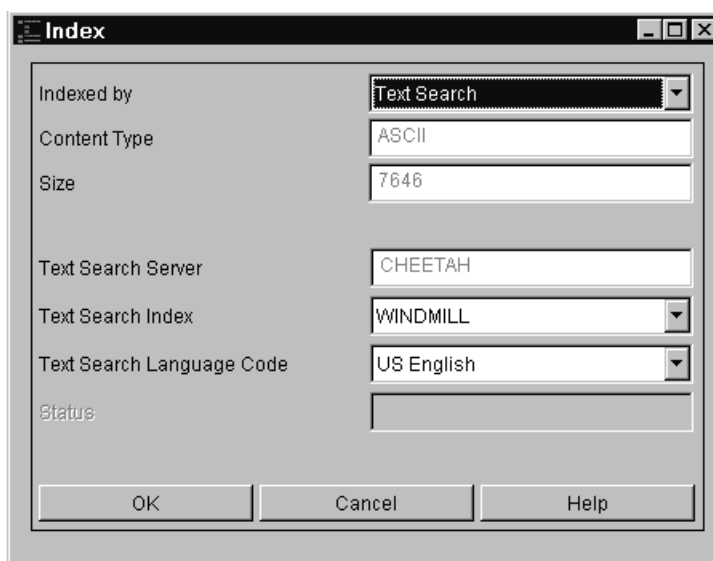
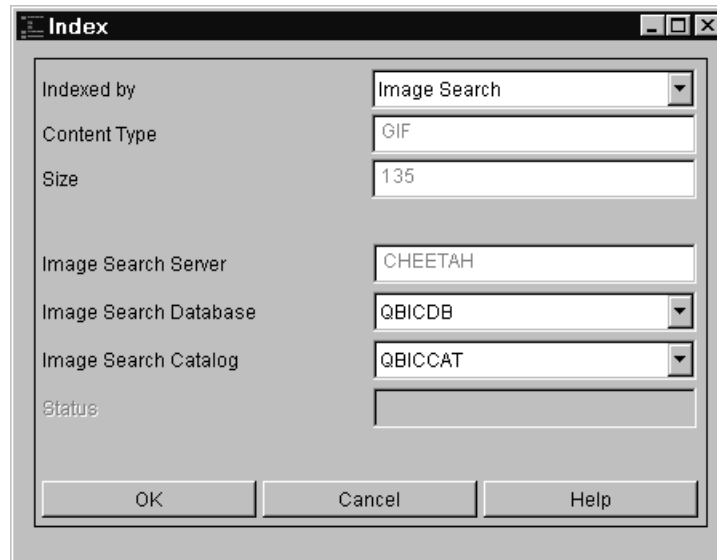


Figure 41. Sample Index window from Text Search

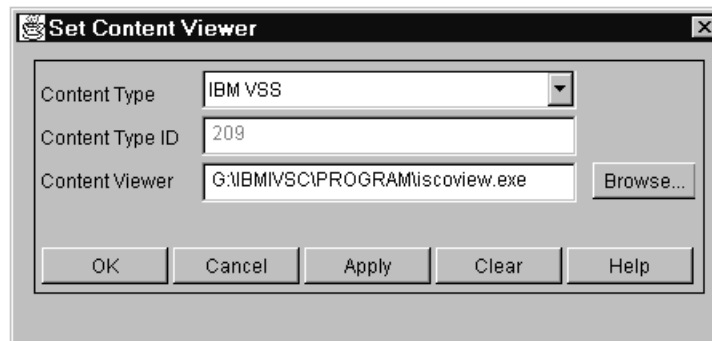
Figure 42. Sample Index window from Image Search



When you update an indexed part, the update function will automatically invoke the indexing process using existing indexing information.

Setting the content viewer

The set content viewer window allows you to specify a viewer for displaying the content of a particular type of object. Each type can have only one viewer. The settings will be saved into a file; therefore, the settings will always be available throughout different sessions. After you have set a viewer, you can invoke the viewer from the query result tree by selecting a part and clicking the **View** push button.



Content Type

The choices for the content type.

After a type is selected, its ID will be displayed in the **Content Type ID** field.

Content Viewer

The viewer application name with full path.

Click **Browse** push button to navigate the local file system. A file selection box will pop up if you click on this button.

OK Save the displayed settings and close the window.

Apply Save the displayed settings.

Clear Clear the displayed settings.

Cancel

Close this window without committing any changes that have not been changed using the **Apply** or **Clear** push buttons.

The settings are saved into the file named `dtcviewer.set`. For AIX, the default directory of this file is under `$HOME`; for Windows, the default directory is under `%CMBROOT%`.

Loading video streams

This function loads video streams into Content Manager. Each video stream is a digitized file and must reside on a machine that is running an FTP server. The FTP server is necessary to transfer the video file to the VideoCharger server.

To load a video stream, first select **Create Part** from the query result tree menu. A Create window will pop up as shown in the screen capture below. From that window, you must select the **IBM VSS** content type for loading video stream parts. This selection will enable the **Set Media Properties** push button and will require you to enter video asset properties.

Content Type	IBM VSS
Content Type ID	209
Part ID	
Fullpath filename	<input type="text"/> Browse...
Indexed by	Not Indexed Set Indexing...
Part Type	Streamable Data Set Media Properties...

OK Cancel Apply Help

The following screen capture shows the window that pops up when you press the **Set Media Properties** push button.

The screenshot shows a dialog box titled "Media Properties". It contains the following fields and controls:

- FTP Host:** Text input field containing "annchen".
- Local/Remote:** Radio buttons for "Local" (selected) and "Remote".
- FTP User ID:** Empty text input field.
- FTP Password:** Empty text input field.
- Media File Fullpath:** Empty text input field with a "Browse..." button to its right.
- Single Media Object/List of Media Segments:** Radio buttons for "Single Media Object" (selected) and "List of Media Segments".
- Maximum Concurrent Users:** Text input field containing "1".
- Asset Group Name:** Empty text input field.
- Buttons:** "OK", "Cancel", and "Help" buttons at the bottom.

The required inputs are:

FTP Host

The machine host name that has the video assets and an FTP server running

Local/Remote radio group

Specifies whether the FTP host is local to the machine that this application runs from

Selecting **Remote** will disable the **Browse** push button.

FTP User ID/Password

The user ID and password to log on to the host machine

Media File Fullpath

The video file full path and name

If this file is local, the user can click the **Browse** push button to navigate the local file system to find the video file.

Single Media Object/List of Media Segments radio group

Indicates whether the file displayed in the **Media File Fullpath** field is the actual video stream file, or a control file which contains a list of fullpaths containing a set of video segments

Maximum Concurrent Users

Specifies maximum concurrent users for this video asset

This value is not enforced by the current version of VideoCharger (Version 2). Any positive, non-zero number will be accepted.

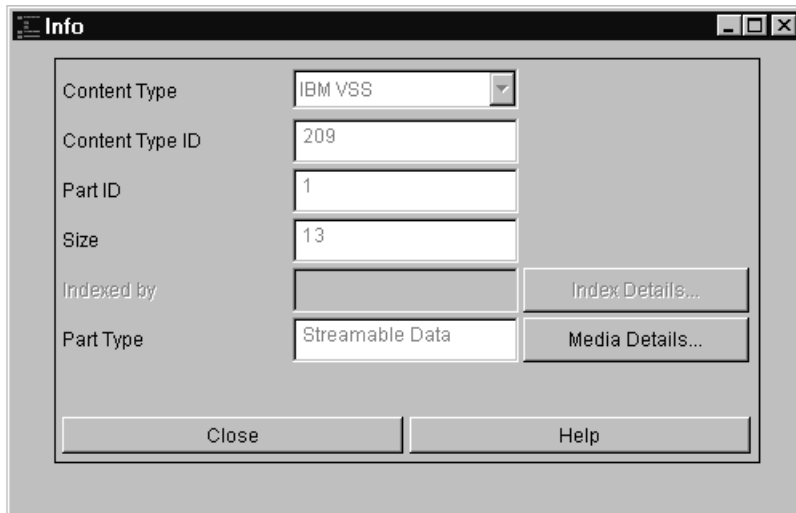
Asset Group Name

The storage entity name as defined in VideoCharger

Currently, VideoCharger has only one asset group. During VideoCharger installation, this default asset group name is AG. However, the installer may change this name; you must know the actual asset group name in order to load video parts.

Displaying video stream parts information

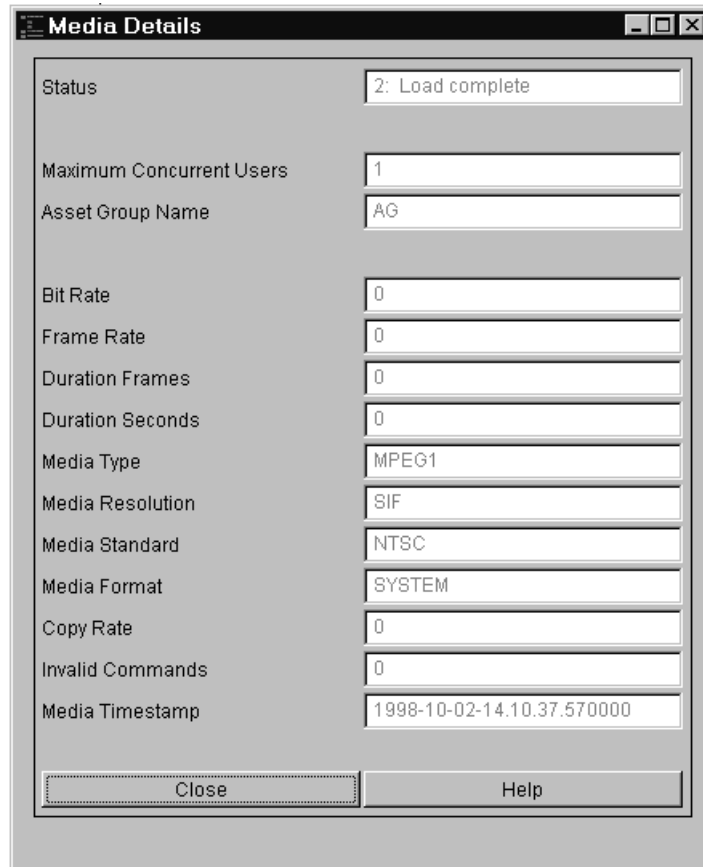
This function allows you to retrieve video parts information and loading status. From the query result tree, select a video part and **Info** pop-up menu item. The generic part information window displays the following information:



The screenshot shows a dialog box titled "Info" with a standard Windows window control bar (minimize, maximize, close). The dialog contains the following fields and buttons:

Content Type	IBM VSS	
Content Type ID	209	
Part ID	1	
Size	13	
Indexed by		Index Details...
Part Type	Streamable Data	Media Details...
Close		Help

From the above window, click **Media Details** to display video stream part information:



The screenshot shows a dialog box titled "Media Details" with a list of parameters and their values:

Status	2: Load complete
Maximum Concurrent Users	1
Asset Group Name	AG
Bit Rate	0
Frame Rate	0
Duration Frames	0
Duration Seconds	0
Media Type	MPEG1
Media Resolution	SIF
Media Standard	NTSC
Media Format	SYSTEM
Copy Rate	0
Invalid Commands	0
Media Timestamp	1998-10-02-14.10.37.570000

At the bottom of the dialog box, there are two buttons: "Close" and "Help".

The **Status** field displays the video loading states, including:

1. Load in process
2. Load complete
3. Load failed

This value will be updated automatically during the loading process.

Playing video streams

To play video streams, you must have the VideoCharger Player installed. The VideoCharger Player is currently available for Windows only. To play a video stream, set the content viewer for video stream parts, then double-click on a video part from the query result tree.

Working in conjunction with Dynamic Page Builder

If Dynamic Page Builder is set up and configured, the applets also have a feature to view data that has been set up to be displayed by Dynamic Page Builder and Net.Data®. In the view applet, when the list of items returned are also viewable from a Dynamic Page Builder setup site, click the right mouse button after selecting one of the document items. A pop-up menu appears with Dynamic Page as one of the menu items. Select that menu item to display a window that prompts you for the part in this item that describes how the document should be brought together and for the URL of the Dynamic Page Builder setup. Make sure that

frnpage.d2w for Dynamic Page Builder is set up correctly. See “Configuring the Dynamic Page Builder with Net.Data” on page 393 for more information on how to configure Dynamic Page Builder.

Chapter 6. Using the sample thin client

The thin client uses JavaServer Pages™ (JSP) on a Web server. The thin client sample application uses the CMBDocumentServices and related classes in the .jsp files. A client can communicate with the Web server using HTTP with parameters that are passed in HTML forms.

Unlike the Enterprise Information Portal client the Thin client does not:

- Require Java on the client
- Require native code, such as the server connector or viewer, on the client
- Introduce firewall problems because HTTP is used

Note: The thick client also does not introduce firewall problems.

For more information about the thin client, see the `thinClientInstall.html` file located in your `CMBROOT/Samples/jsp/clientapp` directory.

Sample JavaServer Pages

This section explains how to use the sample Thin client to demonstrate how to use the non-visual beans in a JavaServer Pages-based application and to cover the function provided in the sample Thin client.

A set of sample JSPs are provided to demonstrate building a JSP application using EIP non-visual beans. These files, listed and explained below can be found in the `CMBROOT/Samples/jsp/clientapp` directory:

addtoworkpacket.jsp

Displays a form for adding an item to an existing workpacket.

banner.jsp

Displays the EIP banner, provides a search, workflow, and logout button. The search and workflow buttons provide a way to switch between workflow client tasks and federated search tasks.

changeworkflow.jsp

Changes the workpacket to a different workflow.

contineworkpacket.jsp

Performs the continue action on a workpacket.

deletenotification.jsp

Deletes a notification from the list of notifications.

deleteworkpacket.jsp

Deletes a workpacket.

editattributes.jsp

Displays the attributes of an item. It can also be invoked with parameters to cause item update.

editworkpacketattributes.jsp

Displays a form for editing the attributes of a work packet.

expertsearch.jsp

Displays the entities, or the attributes of a particular entity if the name of

the entity is passed as a parameter. Filling in the form and pressing search launches a search, invoking search.jsp.

error.jsp

Formats error messages and displays them.

errorreport.jsp

Consists of jsp code to be included that generates a report when given an exception. Included by login.jsp, useridmapping.jsp, and searchresults.jsp.

foldercontents.jsp

Displays details on a folder. The folder is passed as a parameter. The details are displayed as a table.

functions.js

Consists of common javascript functions used by the JSPs.

index.html

Displays login.jsp.

inttostrings.jsp

Consists of jsp code to be included that provides common functions used by workflow related JSPs. Included by addtoworkpacket.jsp and worklist.jsp.

itemlistdisplay.jsp

Consists of jsp code to be included that generates the html page of a list of items. Included by foldercontents.jsp, searchresults.jsp, and worklist.jsp.

login.jsp

Displays a form for logging into EIP. After the UserID, password, and server name are specified, it performs the login and displays mainframes.jsp. Also provides a way to change the password and manage your UserID mapping. login.jsp uses errorreport.jsp to generate the error reports.

logout.jsp

Terminates the EIP session and provides a link to the login.jsp.

mainframes.jsp

provides the main display of the application, which is divided into two sections: document search on the bottom and the banner on the top.

Master.css

A style sheet used by all of the JSPs. By modifying this style sheet, the look of all the pages can be changed.

newworkpacket.jsp

Displays a form to create a new workpacket.

print.jsp

Displays a form to display the document in a print-friendly fashion.

prioritizeworkpacket.jsp

Displays a dialog to change the priority of a workpacket.

reindex.jsp

Displays the current index. It can be invoked with parameters to cause the item to be reindexed.

removefromworkpacket.jsp

Removes an item from a workpacket.

resumeworkpacket.jsp

Resumes a suspended workpacket.

searchframes.jsp

A set of frames for the left hand side of the main frame. It will display the template search and search results in frames on top and bottom respectively. Alternatively, if passed a parameter, it will display entity search or worklist.

searchresults.jsp

Performs a search and displays the search results. The search string is provided as a parameter and the results are returned as a table. It uses `itemlistdisplay.jsp` to generate the html output. It also displays a form to update the userid mapping for a federated userid if the search fails due to an unsuccessful connection with the content server. It uses `errorreport.jsp` to generate the error reports.

searchtemplate.jsp

Displays the search templates, or a particular template if provided the template name as a parameter. Filling out the template and pressing Search launches a search, invoking `searchresults.jsp`. Choosing a template invokes `templatesearchtask.jsp` to display the particular template and the search results.

suspendworkpacket.jsp

Suspends a workpacket.

templatesearchtask.jsp

Displays a set of frames for the bottom section of the main frame. It will display the template search form in the top frame and search results in the bottom frame.

thinclient.css

Consists of a style sheet used by the JSPs. By modifying this style sheet, the look of those pages can be changed.

UserIDmapping.jsp

Displays the current userid mapping for a federated userid. It also performs mapping updates when invoked with parameters. It uses `errorreport.jsp` to generate the error reports.

viewframes.jsp

Displays a set of frames to view a document. It has a toolbar in the top frame and a page view in the bottom frame.

viewpage.jsp

Displays a single page of a document.

viewtoolbar.jsp

Displays the toolbar for controlling the viewed page of a document and moving from page to page.

workflowstyle.css

Consists of a style sheet used by all of the JSPs dealing with workflow tasks. By modifying this style sheet, the look of those pages can be changed.

worklist.jsp

Displays the list of worklists, or the details of a particular worklist if passed the name or a worklist as a parameter, or the details of a particular work packet if passed the worklist and workpacket id's as parameters.

Chapter 7. Working with information mining

Building an application using the Information Mining beans

Before you can use the Information Mining beans, you must install and configure the EIP information mining feature on the machine that will run the beans.

These are the Information Mining beans:

- CMBSummarizationService
- CMBCategorizationService
- CMBAdvancedSearchService
- CMBAdvancedSearchQueryBuilder
- CMBTextAnalysisAdapter
- CMBWebCrawlerService

The following examples demonstrate how you can use the Information Mining beans to build applications that:

- **Categorization sample:** Gather information in preparation for information mining - a typical step for a librarian. See Figure 43 on page 192. In this sample, you make a standard EIP search to find information in the EIP content server. (Gathering information from the Web is in the Web Crawler sample.) You categorize the information and make it available for search within the categories (the advanced search sample). The category information is placed with the document IDs in storage known as the *catalog*.
- **The summarization sample:** Another typical step for a librarian. As in the categorization sample, you make a standard EIP search to find information in the EIP content server. You then summarize the information and store the summaries in the catalog. See Figure 45 on page 197.
- **The advanced search sample:** This is an information mining step. You make an advanced EIP search, which allows you to search for information using a flexible query, and to search in specific categories. The search results contain the found information, but not the categories from which the information originated. To restore the category information from the catalog to the search results, you again run the categorization step.

Then follows a step that restores also the summary information from the catalog to the results.

This is shown in Figure 46 on page 200.

- **The Web Crawler sample:** Get information from the Web by crawling and make the information available for search within the categories (the advanced search sample). This is also a librarian step, similar to the categorization sample, except that you gather the information from the Internet or an intranet rather than from the EIP content server.

See Figure 48 on page 206.

Location of the sample files

The samples described in this chapter are provided in
CMBROOT\samples\jsp\clientapp\infomining:

Sample	Location
--------	----------

Categorization application

`\samples\java\beans\infomining\categorization`

Summarization application

`\samples\java\beans\infomining\summarization`

Advanced search application

`\samples\java\beans\infomining\advancedsearch`

Web Crawler application

`\samples\java\beans\infomining\webcrawler`

Content provider sample

`\samples\java\beans\infomining\contentprovider`

Taxonomy files

`\samples\java\beans\infomining\taxonomy`

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.

The JSP applications are in the following directory:

JSP applications

`\SAMPLES\jsp\clientapp\infomining`

The categorization sample: Categorizing information found by a standard EIP search

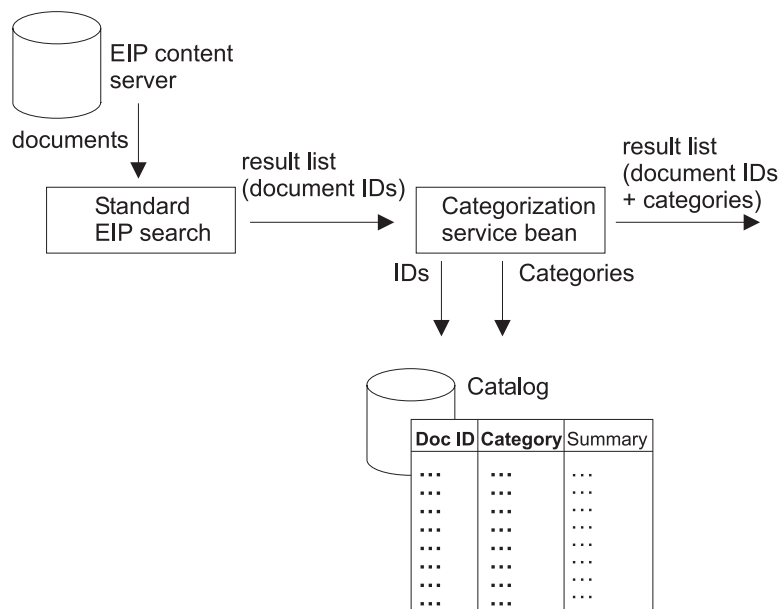


Figure 43. The categorization sample

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 (the advanced search sample). See “Location of the sample files” on page 191.

In Figure 43 on page 192, a standard EIP search is made on documents held in the content server. The categorization service bean takes the IDs of the found documents and stores them in the catalog. The categorization service bean then uses the IDs to locate the found documents (not shown) and to determine to which categories they belong. The bean stores the category information with the document IDs in the catalog. 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)
- CMBTextAnalysisAdapter
- CMBCategorizationService

For this sample, the application:

1. Creates the beans
2. Customizes the beans so that the categorization result is stored and the appropriate documents are available for advanced search
3. Connects the beans
4. Runs the query
5. Displays the text analysis results for verification

An explanation of each of the preceding five steps follows the source for `Categorization.java`.

Complete source for `Categorization.java`

```
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.infomining.*;
import java.util.Vector;

public class Categorization implements CMBResultListener, CMBExceptionListener
{
    public Categorization() throws Exception
    {
        // creating beans
        CMBConnection connection = new CMBConnection();
        CMBQueryService queryService = new CMBQueryService();
        CMBSearchResults searchResults = new CMBSearchResults();
        CMBCategorizationService categorizationService = new CMBCategorizationService();
        CMBTextAnalysisAdapter adapter1 = new CMBTextAnalysisAdapter();
        CMBTextAnalysisAdapter adapter2 = new CMBTextAnalysisAdapter();

        // customizing beans
        CMBInfoMiningUtilities.setContentProvider(new CMBDefaultContentProvider());
        connection.setConnectionType(CMBConnection.CMB_CONNTYPE_REMOTE,
            "file:///C:/cmbroot/cmbclient.ini");
        connection.setServerName("content server name");
        connection.setUserid("userid");
        connection.setPassword("password");
        categorizationService.setTaxonomyFileName("simple.taxonomy");
        categorizationService.setCategorySchemaFileName("simple.tcd");
        categorizationService.setReadContentFromCatalogEnabled(false);
        categorizationService.setReadResultFromCatalogEnabled(false);
        categorizationService.setWriteContentToCatalogEnabled(true);
        categorizationService.setWriteResultToCatalogEnabled(true);

        // connecting beans
        connection.addCMBConnectionReplyListener(queryService);
        connection.addCMBConnectionReplyListener(searchResults);
    }
}
```

```

connection.addCMBConnectionReplyListener(categorizationService);
queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(categorizationService);
categorizationService.addCMBTextAnalysisReplyListener(adapter2);
adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// running query
connection.connect();
CMBSchemaManagement schema = connection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate("template name");
String[] searchValues = {"search value"};
searchTemplate.setSearchCriterion("criterion name",
    CMBBaseConstant.CMB_OP_EQUAL, searchValues);
queryService.setQueryObject(searchTemplate);
queryService.runQuery();
connection.disconnect();
}

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
    Vector cmbItemVector = (Vector)e.getData();

    if(cmbItemVector == null)
        return;

    for(int i = 0; i < cmbItemVector.size(); i++)
    {
        CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

        System.out.println("PID      : " + currentItem.getPidString());
        System.out.print("Categories: ");

        Vector categoryVector = currentItem.getCategories();
        CMBCategory category = null; //we display only the first one
        if((categoryVector != null) && (categoryVector.size() > 0))
            category = (CMBCategory)categoryVector.elementAt(0);

        while(category != null)
        {
            System.out.print(" < " + category.getName());
            category = category.parent();
        }
        System.out.println("\n");
    }
}

//implementing CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
    ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
    try
    {
        new Categorization();
    }
    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 `CMBCConnection` bean.

The beans `CMBQueryService` and `CMBSearchResults` are required to perform a federated search. The documents are made available for advanced search using the `CMBCategorizationService` bean, which will assign the documents to one or more categories. The two adapters are required to convert search result events to text analysis request events and then text analysis reply events back to search result events.

Customizing the beans

The code shown in the customizing section must be adapted according to your installation. On the connection bean you need to specify the connection type, the name of the content server to connect, a user ID, and the appropriate password.

The line

```
connection.setConnectionType(CMBCConnection.CMB_CONNTYPE_REMOTE,
                             "file:///C:/cmbroot/cmbclient.ini");
```

specifies that the sample is to be run on the client. If this line is not included this would be a server application.

Before you can run the categorization service bean, you have to associate it with a taxonomy using a taxonomy file and a category schema file. Both files are created by the categorizer training tool as described in *Planning and Installing Enterprise Information Portal*.

You have to set the `readContentFromCacheEnabled` and `readResultFromCacheEnabled` properties to false to specify that the categorization service beans must not use the catalog to read document content or the text analysis results. With these two properties set to false, the bean retrieves the latest version of the document content from the content server and analyzes the text.

The content and the text analysis results are written to the catalog, which prepares the document for advanced search. The advanced search provides a search in certain categories, which is possible only if the appropriate category information is stored in the catalog.

There are other catalog-related properties that are documented in the online API reference for the information mining beans.

Connecting the beans

Figure 44 on page 196 illustrates the flow of events among the beans:

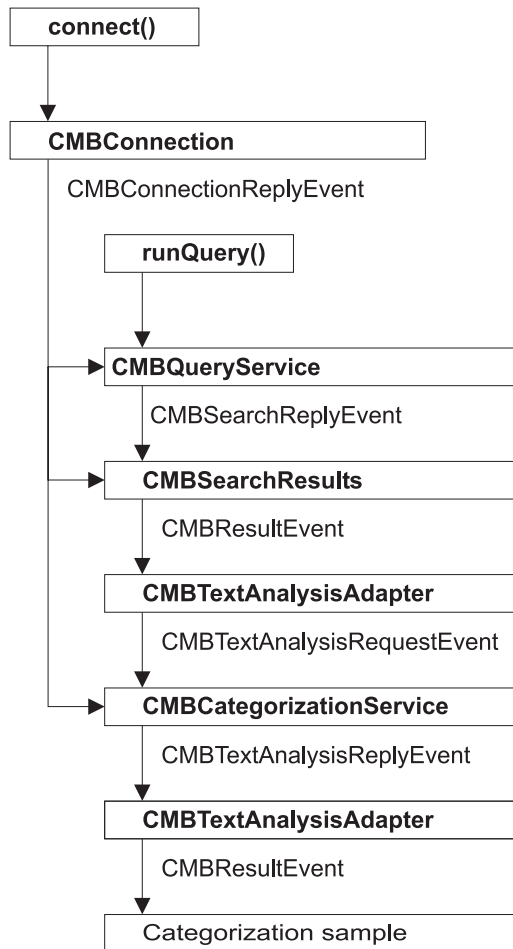


Figure 44. The categorization sample: event flow

Three 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 that then starts the event flow through the other beans.

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 receive exceptions.

You can also add the `CMBSummarization` service in this sample to create summaries for the documents. See “The advanced search sample: Make an advanced search and analyze the results” on page 200 for more information.

The `CMBTextAnalysisAdapter` allows the text analysis beans to respond not only to a `CMBResultEvent` but to other events as well. It does this by converting the `CMBResultEvent` to a generic event called `CMBTextAnalysisRequestEvent`. The text analysis beans themselves produce a `CMBTextAnalysisReplyEvent` which the `CMBTextAnalysisAdapter` converts back to a `CMBResultEvent`. In this way, the adapter allows text analysis beans to be inserted anywhere where a `CMBResultEvent`, or a similar event, occurs.

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 federated search, you need to specify a search template, a criterion of the template, the compare operator, and a search value. Refer to the Javadoc to get information about the query syntax.

To close the current connection, call the disconnect() method:

```
connection.disconnect();
```

Displaying text analysis results

The Categorization class implements the CMBResultListener interface 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 an 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);  
System.out.println("PID      : " + currentItem.getPidString());
```

as well as the results of text analysis beans if there were some in the event flow:

```
Vector categoryVector = currentItem.getCategories();
```

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.

Scenario 2: Import documents and metadata using federated search

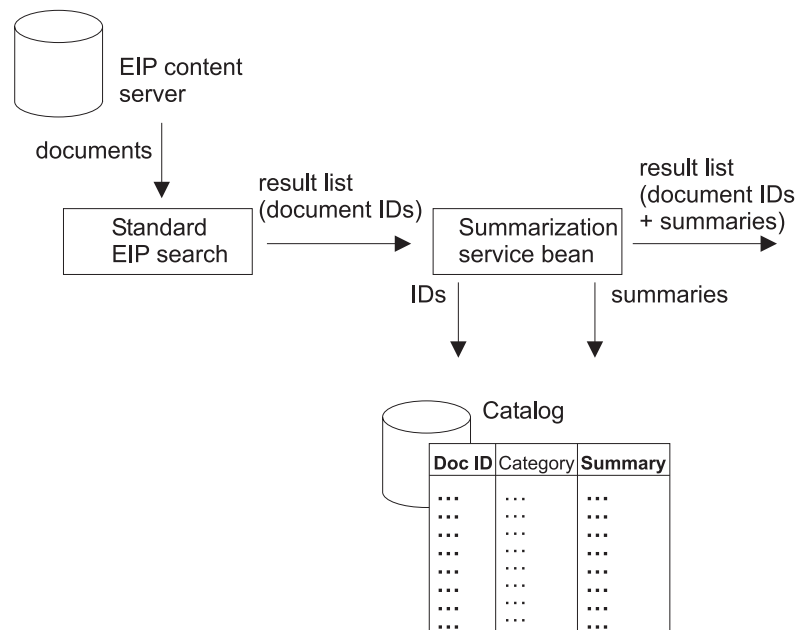


Figure 45. The summarization sample

This sample demonstrates how you can summarize documents that have been retrieved by a standard EIP search. See “Location of the sample files” on page 191.

In Figure 45 on page 197, a standard EIP search is made on documents held in the EIP content server. The summarization service bean takes the IDs of the found documents and stores them in the catalog. The summarization service bean then uses the IDs to locate the found documents (not shown) and to create summaries of them. The bean stores the summarization information with the document IDs in the catalog. This information, the document IDs and the summaries, is then made available for further processing.

Tip: You can combine the use of the categorization service bean and the summarization service bean, one after the other, in any sequence, to add analysis information to the catalog.

Complete source for Summarization.java

Here is the complete source of the summarization sample. For further details, see “The categorization sample: Categorizing information found by a standard EIP search” on page 192.

```
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.infomining.*;
import java.util.Vector;

public class Summarization implements CMBResultListener, CMBExceptionListener
{
    public Summarization() throws Exception
    {
        // creating beans
        CMBConnection connection = new CMBConnection();
        CMBQueryService queryService = new CMBQueryService();
        CMBSearchResults searchResults = new CMBSearchResults();
        CMBSummarizationService summarizationService = new CMBSummarizationService();
        CMBTextAnalysisAdapter adapter1 = new CMBTextAnalysisAdapter();
        CMBTextAnalysisAdapter adapter2 = new CMBTextAnalysisAdapter();

        // customizing beans
        CMBInfoMiningUtilities.setContentProvider(new CMBDefaultContentProvider());
        connection.setConnectionType(CMBConnection.CMB_CONNTYPE_REMOTE,
            "file:///C:/cmbroot/cmbclient.ini");
        connection.setServerName("content server name");
        connection.setUserid("userid");
        connection.setPassword("password");
        summarizationService.setReadContentFromCatalogEnabled(false);
        summarizationService.setReadResultFromCatalogEnabled(false);
        summarizationService.setWriteContentToCatalogEnabled(true);
        summarizationService.setWriteResultToCatalogEnabled(true);

        // connecting beans
        connection.addCMBConnectionReplyListener(queryService);
        connection.addCMBConnectionReplyListener(searchResults);
        connection.addCMBConnectionReplyListener(summarizationService);
        queryService.addCMBSearchReplyListener(searchResults);
        searchResults.addCMBResultListener(adapter1);
        adapter1.addCMBTextAnalysisRequestListener(summarizationService);
        summarizationService.addCMBTextAnalysisReplyListener(adapter2);
        adapter2.addCMBResultListener(this);

        connection.addCMBExceptionListener(this);
        queryService.addCMBExceptionListener(this);
        searchResults.addCMBExceptionListener(this);
        summarizationService.addCMBExceptionListener(this);
        adapter1.addCMBExceptionListener(this);
        adapter2.addCMBExceptionListener(this);
    }
}
```

```

// running query
connection.connect();
CMBSchemaManagement schema = connection.getSchemaManagement();
CMBSearchTemplate searchTemplate = schema.getSearchTemplate("template name");
String[] searchValues = {"search value"};
searchTemplate.setSearchCriterion("criterion name",
    CMBBaseConstant.CMB_OP_EQUAL, searchValues);
queryService.setQueryObject(searchTemplate);
queryService.runQuery();
connection.disconnect();
}

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
    Vector cmbItemVector = (Vector)e.getData();

    if(cmbItemVector == null)
        return;

    for(int i = 0; i < cmbItemVector.size(); i++)
    {
        CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

        System.out.println("PID      : " + currentItem.getPidString());
        System.out.println("Summary  : " + currentItem.getSummary());
    }
}

//implementing CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
    ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
    try
    {
        new Summarization();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
}

```

The advanced search sample: Make an advanced search and analyze the results

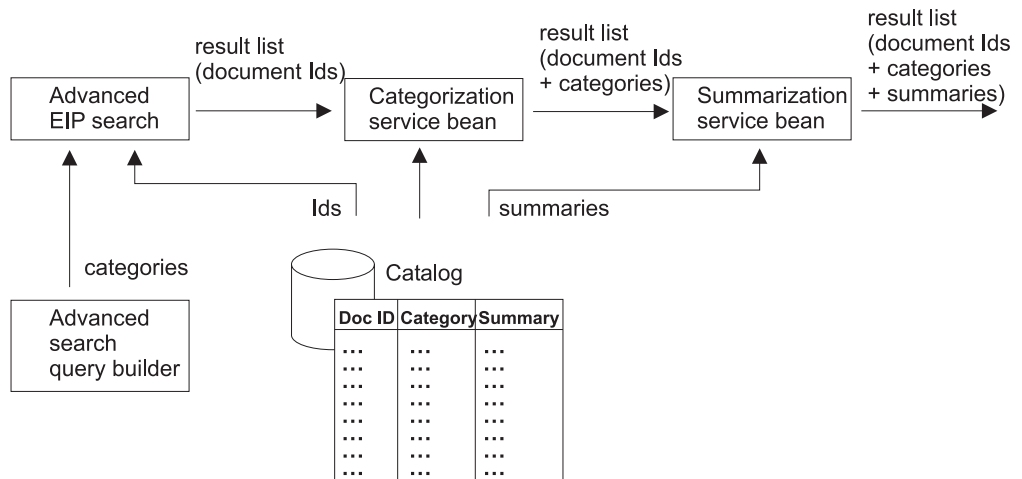


Figure 46. The advanced search sample

This sample demonstrates how to run an advanced search and conduct a text analysis (summarization and categorization) on 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 191.

In contrast to the previous samples where a standard EIP search is carried out on the entire EIP content server, this sample begins by making a so-called *advanced* search only on the documents whose IDs are stored in the catalog. To narrow the search even further, you tell the advanced search query builder not only the text you want to search for, but also the categories of documents you want to search in.

Tips: 1. An advanced search can be made only on documents whose IDs and categories are stored in the catalog; that is, on documents resulting from the categorization sample.

2. The result of an advanced search contains only the IDs of the found documents – not the name of the categories in which the documents were found. To obtain the category information, you make use again of the categorization service bean.

When the advanced EIP search has produced a result list, the IDs of the found documents are used by the categorization service bean to retrieve the category information previously stored there by the same bean in the categorization sample.

Note that in this sample, the bean reads the category information from the catalog, whereas in the categorization sample the bean writes the category information to the catalog. You can, however, force the categorization service bean not to read the category information previously stored in the catalog, and to recategorize the documents (write it again to the catalog). You do this using the `ReadContentFromCacheEnabled` and `ReadResultFromCacheEnabled` flags. Note, however, that if these flags are set to true (the default value), and the catalog does not contain the required category or summary information, then an analysis is automatically started and the catalog or summary information is written to the catalog.

The advanced search sample continues by using the summarization bean to retrieve from the catalog the summaries of the documents found by the advanced search. Also here you can force the summarization service bean not to read the summary information previously stored in the catalog, and to recreate it (write it again to the catalog).

In each stage of the advanced search sample, the result is enriched: the initial result contains only the IDs of the found documents, then the category information is added, and then the summary information.

The following beans are used in this sample:

- CMBConnection
- CMBAdvancedSearchQueryBuilder
- CMBAdvancedSearchService
- CMBTextAnalysisAdapter
- CMBCategorizationService
- CMBSummarizationService

For this sample the application:

1. Creates the beans
2. Connects the beans
3. Runs the query
4. Displays the text analysis results

An explanation of each of the preceding four steps follows the source for `AdvancedSearch.java`.

Complete source for `AdvancedSearch.java`

```
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.infomining.*;
import java.util.Vector;

public class AdvancedSearch implements CMBResultListener, CMBExceptionListener
{
    public AdvancedSearch() throws Exception
    {
        // creating beans
        CMBConnection connection = new CMBConnection();
        CMBAdvancedSearchQueryBuilder queryBuilder = new CMBAdvancedSearchQueryBuilder();
        CMBAdvancedSearchService searchService = new CMBAdvancedSearchService();
        CMBCategorizationService categorizationService = new CMBCategorizationService();
        CMBSummarizationService summarizationService = new CMBSummarizationService();
        CMBTextAnalysisAdapter adapter1 = new CMBTextAnalysisAdapter();
        CMBTextAnalysisAdapter adapter2 = new CMBTextAnalysisAdapter();

        // customizing beans
        connection.setConnectionType(CMBConnection.CMB_CONNTYPE_REMOTE,
            "file:///C:/cmbroot/cmbclient.ini");
        connection.setServerName("content server name");
        connection.setUserid("userid");
        connection.setPassword("password");
        queryBuilder.setTaxonomyFileName("simple.taxonomy");
        categorizationService.setTaxonomyFileName("simple.taxonomy");
        categorizationService.setCategorySchemaFileName("simple.tcd");
        categorizationService.setReadContentFromCatalogEnabled(true);
        categorizationService.setReadResultFromCatalogEnabled(true);
        categorizationService.setWriteContentToCatalogEnabled(true);
        categorizationService.setWriteResultToCatalogEnabled(true);
        summarizationService.setReadContentFromCatalogEnabled(true);
    }
}
```

```

summarizationService.setReadResultFromCatalogEnabled(true);
summarizationService.setWriteContentToCatalogEnabled(true);
summarizationService.setWriteResultToCatalogEnabled(true);
// connecting beans
connection.addCMBConnectionReplyListener(searchService);
connection.addCMBConnectionReplyListener(categorizationService);
connection.addCMBConnectionReplyListener(summarizationService);
queryBuilder.addCMBAdvancedSearchRequestListener(searchService);
searchService.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(categorizationService);
categorizationService.addCMBTextAnalysisReplyListener(summarizationService);
summarizationService.addCMBTextAnalysisReplyListener(adapter2);
adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
queryBuilder.addCMBExceptionListener(this);
searchService.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);

// running query
connection.connect();
queryBuilder.setTextSearchQuery("query string");
queryBuilder.setCategoryNames("categories");
queryBuilder.runQuery();
connection.disconnect();
}

// implementing com.ibm.mm.beans.CMBResultListener
public void onCMBResult(CMBResultEvent e)
{
    Vector cmbItemVector = (Vector)e.getData();

    if(cmbItemVector == null)
        return;

    for(int i = 0; i < cmbItemVector.size(); i++)
    {
        CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

        System.out.println("PID      : " + currentItem.getPidString());
        System.out.print("Summary  : " + currentItem.getSummary());
        System.out.print("Categories: ");

        Vector categoryVector = currentItem.getCategories();
        CMBCategory category = null; //we display only the first one
        if((categoryVector != null) && (categoryVector.size() > 0))
            category = (CMBCategory)categoryVector.elementAt(0);

        while(category != null)
        {
            {
                System.out.print(" < " + category.getName());
                category = category.parent();
            }
            System.out.println("\n");
        }
    }
}

//implementing CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
    ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
    try

```

```

        {
            new AdvancedSearch();
        }
    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 beans `CMBAdvancedQueryBuilder` and `CMBAdvancedSearchService` are required to perform an advanced EIP search. A summary and category information can be created using the beans `CMBSummarizationService` and `CMBCategorizationService`. The two adapters are required to convert search result events to text analysis request events and then text analysis reply events back to search result events.

Here is the code that creates the beans:

```

CMBConnection connection = new CMBConnection();
CMBAdvancedSearchQueryBuilder queryBuilder = new CMBAdvancedSearchQueryBuilder();
CMBAdvancedSearchService searchService = new CMBAdvancedSearchService();
CMBCategorizationService categorizationService = new CMBCategorizationService();
CMBSummarizationService summarizationService = new CMBSummarizationService();
CMBTextAnalysisAdapter adapter1 = new CMBTextAnalysisAdapter();
CMBTextAnalysisAdapter adapter2 = new CMBTextAnalysisAdapter();

```

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 connection type, the name of the content server to connect, a user ID and the appropriate password. By specifying the client INI file, you indicate that the sample has been written to run on an EIP client.

Before you can run the advanced query builder and the categorization service bean you need to associate them with a taxonomy using a taxonomy file. In addition, the category schema file has to be set on the categorization service bean. Both files are created by the categorizer training tool as described in *Enterprise Information Portal Planning and Installing Enterprise Information Portal*.

Here is the code that does the customization for the sample:

```

connection.setConnectionType(CMBConnection.CMB_CONNTYPE_REMOTE,
    "file:///C:/cmbroot/cmbclient.ini");
connection.setServerName("content server name");
connection.setUserid("userid");
connection.setPassword("password");
queryBuilder.setTaxonomyFileName("simple.taxonomy");
categorizationService.setTaxonomyFileName("simple.taxonomy");
categorizationService.setCategorySchemaFileName("simple.tcd");
categorizationService.setReadContentFromCatalogEnabled(true);
categorizationService.setReadResultFromCatalogEnabled(true);
categorizationService.setWriteContentToCatalogEnabled(true);
categorizationService.setWriteResultToCatalogEnabled(true);
summarizationService.setReadContentFromCatalogEnabled(true);
summarizationService.setReadResultFromCatalogEnabled(true);
summarizationService.setWriteContentToCatalogEnabled(true);
summarizationService.setWriteResultToCatalogEnabled(true);

```

The code assumes that the categorization input comes from the catalog.

```

categorizationService.setReadContentFromCatalogEnabled(true);
categorizationService.setReadResultFromCatalogEnabled(true);

```

These are the default values, so you can omit these statements. However, if you want the content or the result to be read from the content server and not from the catalog, make these statements false, like this:

```

categorizationService.setReadContentFromCatalogEnabled(false);
categorizationService.setReadResultFromCatalogEnabled(false);

```

Connecting the beans

Figure 47 illustrates the flow of events among the beans in this sample:

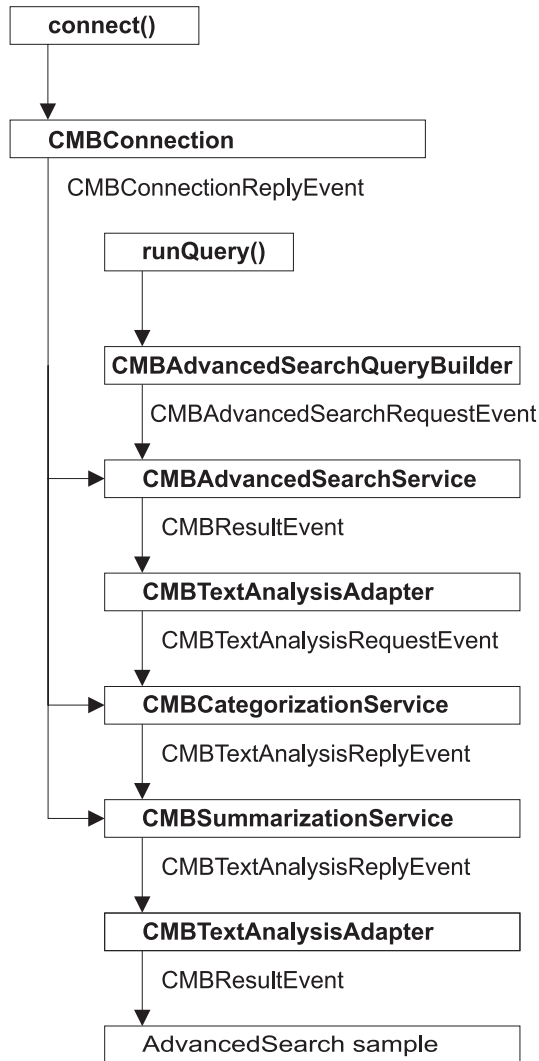


Figure 47. The advanced search sample: Event flow

Three of the beans used in this sample listen to the `CMBConnectionReplyEvent` to get the connection handle. The `CMBAdvancedSearchQueryBuilder` bean initiates a search that results in an event which then starts the event flow through the other beans.

Here is the code that connects the beans:


```

connection.addCMBConnectionReplyListener(searchService);
connection.addCMBConnectionReplyListener(categorizationService);
connection.addCMBConnectionReplyListener(summarizationService);
queryBuilder.addCMBAdvancedSearchRequestListener(searchService);
searchService.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(categorizationService);
categorizationService.addCMBTextAnalysisReplyListener(summarizationService);
summarizationService.addCMBTextAnalysisReplyListener(adapter2);
adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
queryBuilder.addCMBExceptionListener(this);
searchService.addCMBExceptionListener(this);
categorizationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.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 `CMBCConnection` bean:

```
connection.connect();
```

To start the advanced EIP search, you need to specify a search query string and the categories to search in.

You need to adapt the following code according to your configuration:

```

queryBuilder.setTextSearchQuery("query string");
queryBuilder.setCategoryNames("categories");
queryBuilder.runQuery();

```

To close the current connection, just 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 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);
System.out.println("PID      : " + currentItem.getPidString());

```

as well as the results of text analysis beans if there were some in the event flow.

Then you get the summary information:

```
System.out.print("Summary  : " + currentItem.getSummary());
```

and the category information:

```
Vector categoryVector = currentItem.getCategories();
```

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.

The Web Crawler sample: Getting information from crawling the Web

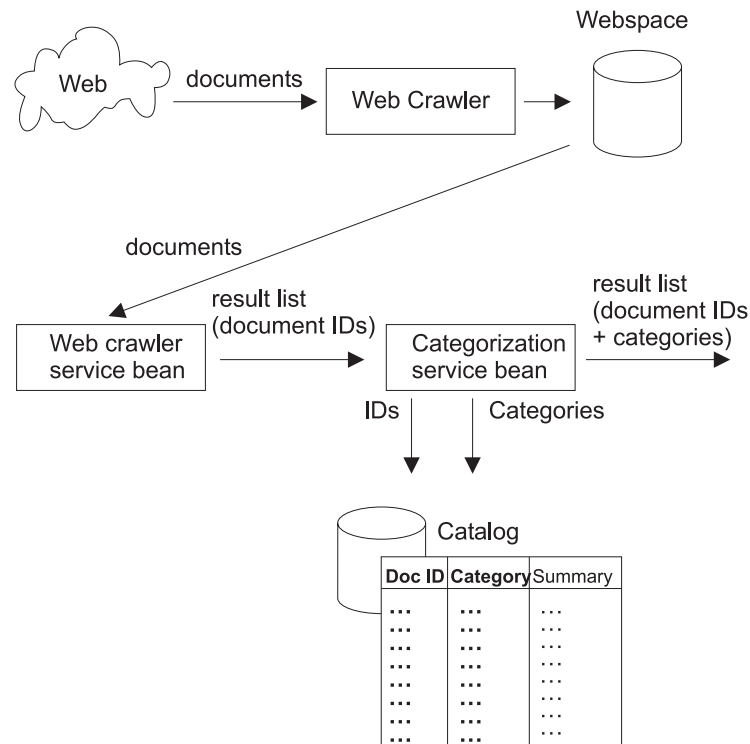


Figure 48. The Web Crawler sample

This sample demonstrates how to import crawled documents into the information mining feature and conduct a text analysis (categorization) on the documents that have been imported. This can only be done if IBM Web Crawler has run previously and new or changed objects have been saved within the defined Web space. The saved pages must contain URL metadata. Set the web crawler to include the following metadata: edit your webspaces `imy.ini` file and set the `[STORE]` section `SAVE_HEADLINESk` option equal to 1 before you start the crawl. See “Location of the sample files” on page 191.

As in samples 1 and 2, the results of the Web Crawler search bean can be made available for a subsequent advanced search (see the advanced search sample).

The following beans are used in this sample:

- `CMBCConnection`
- `CMBWebCrawlerService`
- `CMBTextAnalysisAdapter`
- `CMBCategorizationService`

For this sample the application:

1. Creates the beans
2. Customizes the beans so that the text analysis results are read from the catalog

3. Connects the beans
4. Sets or changes the default properties of CMBWebCrawlerService
5. Starts the Web Crawler service
6. Displays the text analysis results

An explanation of each of the preceding steps follows the source for WebCrawler.java.

Complete source for WebCrawler.java

```
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.infomining.*;
import java.util.Vector;

public class WebCrawler implements CMBResultListener, CMBExceptionListener
{
    public WebCrawler() throws Exception
    {
        // creating beans
        CMBConnection connection = new CMBConnection();
        CMBWebCrawlerService crawlerService = new CMBWebCrawlerService();
        CMBCategorizationService categorizationService = new CMBCategorizationService();
        CMBTextAnalysisAdapter adapter1 = new CMBTextAnalysisAdapter();
        CMBTextAnalysisAdapter adapter2 = new CMBTextAnalysisAdapter();

        // customizing beans
        connection.setConnectionType(CMBConnection.CMB_CONNTYPE_REMOTE,
            "file:///C:/cmbroot/cmbclient.ini");
        connection.setServerName("content server name");
        connection.setUserid("userid");
        connection.setPassword("password");
        crawlerService.setRootDir("C:\\webspaces");
        crawlerService.setWebSpace("webpace name");
        categorizationService.setTaxonomyFileName("simple.taxonomy");
        categorizationService.setCategorySchemaFileName("simple.tcd");
        categorizationService.setReadContentFromCatalogEnabled(true);
        categorizationService.setReadResultFromCatalogEnabled(false);
        categorizationService.setWriteContentToCatalogEnabled(true);
        categorizationService.setWriteResultToCatalogEnabled(true);

        // connecting beans
        connection.addCMBConnectionReplyListener(categorizationService);
        connection.addCMBConnectionReplyListener(crawlerService);
        crawlerService.addCMBResultListener(adapter1);
        adapter1.addCMBTextAnalysisRequestListener(categorizationService);
        categorizationService.addCMBTextAnalysisReplyListener(adapter2);
        adapter2.addCMBResultListener(this);

        connection.addCMBExceptionListener(this);
        crawlerService.addCMBExceptionListener(this);
        categorizationService.addCMBExceptionListener(this);
        adapter1.addCMBExceptionListener(this);
        adapter2.addCMBExceptionListener(this);

        // running web crawler service
        connection.connect();
        crawlerService.start();
        connection.disconnect();
    }

    // implementing com.ibm.mm.beans.CMBResultListener
    public void onCMBResult(CMBResultEvent e)
    {
        Vector cmbItemVector = (Vector)e.getData();

        if(cmbItemVector == null)
            return;
    }
}
```

```

for(int i = 0; i < cmbItemVector.size(); i++)
{
    CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

    System.out.println("PID      : " + currentItem.getPidString());
    System.out.print("Categories: ");

    Vector categoryVector = currentItem.getCategories();
    CMBCategory category = null; //we display only the first one
    if((categoryVector != null) && (categoryVector.size() > 0))
        category = (CMBCategory)categoryVector.elementAt(0);

    while(category != null)
    {
        System.out.print(" < " + category.getName());
        category = category.parent();
    }
    System.out.println("\n");
}
}

//implementing CMBExceptionListener
public void onCMBException(CMBExceptionEvent e)
{
    ((Exception)e.getData()).printStackTrace();
}

public static void main(String[] args)
{
    try
    {
        new WebCrawler();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
}

```

Creating the beans

You build a connection to a content server for security reasons because 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 CMBCConnection bean.

The bean CMBWebCrawlerService is used to import the previously crawled documents into the information mining component and to throw a CMBResultEvent notification. Summary and category information can be created using the beans CMBSummarizationService and CMBCategorizationService. In this sample, the CMBCategorizationService service bean is used. The two adapters convert crawler service result events to text analysis request events, and then text analysis reply events back to the sample result event.

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 connection type, the name of the content server to connect, a user ID and the appropriate password. Because the sample has been written to run on an EIP client, the client INI file has to be specified.

Before you can run the Web Crawler service and the categorization service bean, you need to associate them with a taxonomy using a taxonomy file. Also the

rootDir on 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.

In addition, the category schema file has to be set on the categorization service bean. Both files are created by the categorizer training tool as described in *Planning and Installing Enterprise Information Portal*.

It is essential that ReadContentFromCatalogEnabled and ReadResultFromCatalogEnabled is set to true (the default value), because otherwise it would try to read from the EIP content server. This also implies that, for this sample to work, the content or result must already exist in the catalog.

Connecting the beans

Figure 49 illustrates the flow of events among the beans in this sample:

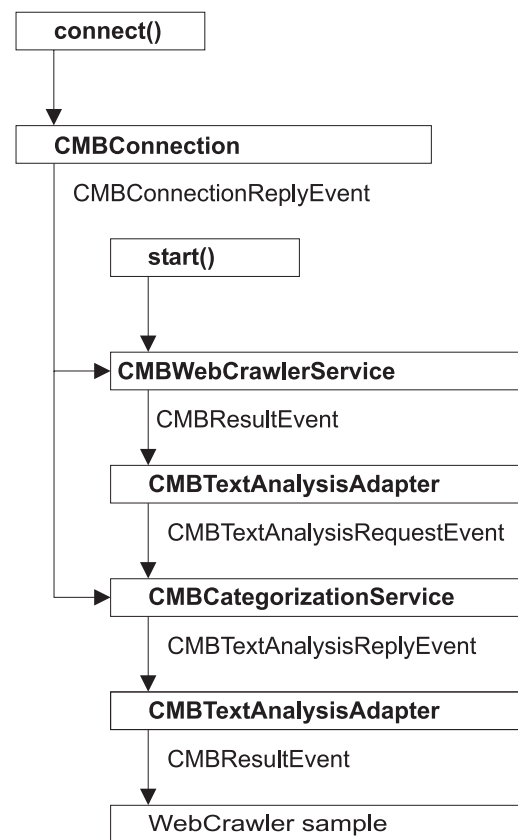


Figure 49. The Web Crawler sample: Event flow

Two 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 that then starts the event flow through the other beans.

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 receive 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();
```

Here is the code to start the Web crawler service:

```
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 list the documents that have been found on the Web space and imported to the information mining component and to display the category information created for each document. The `CMBResultEvent`, received as an 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);  
System.out.println("PID : " + currentItem.getPidString());
```

as well as the results of text analysis beans if there were some in the event flow.

In this sample, you get `Categories` which is the only text analysis bean in the event flow:

```
Vector categoryVector = currentItem.getCategories();
```

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 need to build your own content provider when you need to apply custom 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 custom 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.

- 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.

```
public String getContent(CMBConnection connection, CMBItem item)  
    throws CMBContentProviderException;
```

To tell the system which ContentProvider to use, use the method `setContentProvider(CMBCContentProvider)` 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. See "Location of the sample files" on page 191 for the location of the sample content provider.

A default content provider is provided with Information Mining. It employs INSO Corporation's OutsideIn technology to process a large number of document formats. It extends the `CMBCContentProvider` 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());
```

These are the methods provided:

```
/**
 * Default constructor.
 * All parts in CMBItem are processed (provided they do not exceed the
 * default size cutoff value of 16 MB) and the texts bits are concatenated.
 */
public CMBDefaultContentProviderINSO() ( )

/**
 * Constructor defining the part to be extracted.
 * This constructor allows specifying one part in the CMBItems to be
 * accessed which contains the text parts to be processed.
 * @param partIndex index of the part in CMBItem to be processed for
 * text content
 */
public CMBDefaultContentProviderINSO(int partIndex) ( )

/**
 * Constructor defining the parts to be extracted.
 * This constructor allows specifying the parts in the CMBItems
 * which contain the text parts to be processed. If multiple parts
 * are specified the resulting text bits will be concatenated.
 * @param partIndices indices of parts to be processed
 */
public CMBDefaultContentProviderINSO(int[] partIndices) ( )

/**
 * Set the value of maxPartSize in bytes.
 * This value specifies the largest part size that is processed by
 * the retrieval subsystem. The cutoff is introduced to avoid system
 * crashes due to trying to load video streams into main memory for
 * processing.
 * @param v number of bytes to assign to maxPartSize.
 */
public void setMaxPartSize(int v) ( )

/**
 * Set the part indices of the parts to be processed.
 * Use this method if there is a need to change the part mask
 * during the ongoing operation.
 * @param array of part indices
 */
```

```

public void setPartsMask(int[] partIndices) ( )

/**
 * Get textual contents of a CMBItem.
 * This is the default method for text retrieval on CMBItems. The standard
 * behaviour
 * is to look at all available parts for text content, collect this content
 * and concatenate it.
 *
 * @return text content of all parts in the specified CMBItem
 * @param connection EIP connection
 * @param item CMBItem to be processed
 */
public String getContent( CMBCConnection connection, CMBItem item )
    throws CMBCContentProviderException ( )

```

Understanding the Information Mining JSP applications

There are two Information Mining Java Server Page (JSP) applications:

- **fedSearch.jsp** searches within Enterprise Information Portal. It categorizes, summarizes, and indexes the documents into the Information Mining component. Then it displays the found documents in a category structure.
- **advSearch.jsp** searches for text within the Information Mining component where you can search for documents within categories. It, too, displays the found documents in a category structure.

The location of the JSP applications is given in “Location of the sample files” on page 191. The directory contains these files:

fedSearch.jsp The source code for the fedSearch JSP.

This JSP provides the Enterprise Information Portal search-specific form-handling code and form-formatting instructions (HTML). It also acts as the controller for selecting views of the data.

advSearch.jsp The source code for the advSearch JSP.

This JSP 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 also contains initialisation instructions specific to the advanced search, in particular the availability of categories that can be searched in.

Settings.jsp The source code for the Settings JSP.

This JSP allows you to customize settings to adapt them to your environment.

catView.jsp This JSP 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 JSP provides the logic and bean-connection code. It contains only java code. There is an implementation of a simple data structure class used for 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 either Enterprise Information Portal search or the advanced search.

The source code is a sample of using the Information Mining beans, and it contains a description of how the code works: 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 not only Enterprise Information Portal but also the Information Mining component installed, with Enterprise Information Portal connected to a back-end data source. 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>.

Chapter 8. Using the C++ application programming interfaces

The C++ APIs are a set of classes that access and manipulate locally or remotely stored data.

This chapter describes the C++ API, the C++ implementation of multiple search facilities, and Internet connectivity.

The C++ API supports:

- Multiple search and update across a heterogeneous combination of content servers
- A common object model for data access
- A flexible mechanism for using a combination of search engines; for example, Text Search Engine and query by image content (QBIC).

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).

Setting up the Windows and AIX environment

When you set up your Windows or AIX environment, you must establish the settings described in this section. Table 19 lists Library, AIX shared and DLL requirements.

Requirement: To use C++, you must install DB2 Client Application Enabler (CAE) on all remote servers running the Enterprise Information Portal database. The CAE user ID and password must be the same user ID and password you use with the Enterprise Information Portal database. For details, see the *Managing Enterprise Information Portal*.

Table 19. Library, shared objects and DLL environment information

Library	Shared objects for AIX	Windows DLLs
cmbcm716.lib	libcmbcm71366.a	cmbcm716.dll
cmbcm716d.lib,	libcmbdb271366.a	cmbcm716d.dll
cmbdl716.lib, cmbdl716d.lib	libcmbdl71366.a	cmbdb2716.dll
	libcmbdlfac71366.so	cmbdb2fac716.dll
cmbip716.lib		cmbdb2716d.dll
		cmbdb2fac716d.dll
cmbip716d.lib	libcmbdb2fac71366.so	cmbdl716.dll
cmbv4716.lib	libcmbdj71366.a	cmbdl716d.dll
cmbv4716d.lib	libcmbdjfac71366.so	cmbdlfac716.dll
cmbdd716.lib		cmbdlfac716d.dll
cmbdd716d.lib		cmbodbc716.dll
		cmbodbcfac716.dll
cmbdes716.lib		cmbodbc716d.dll
		cmbodbcfac716d.dll
cmbdes716d.lib		cmbfed716.dll
		cmbfedfac716.dll
cmbdb2716.lib		cmbfed716d.dll
		cmbfedfac716d.dll

Table 19. Library, shared objects and DLL environment information (continued)

Library	Shared objects for AIX	Windows DLLs
cmbdb2716d.lib		cmbdd716.dll cmbddf716.dll
cmbdj716d.lib		cmbdd716d.dll cmbddf716d.dll
cmbdj716.lib		cmbip716.dll cmbipfac716.dll cmbip716d.dll cmbipfac716d.dll cmbv4716.dll cmbv4fac716.dll cmbv4716d.dll cmbv4fac716d.dll cmbdes716.dll cmbdesfac716.dll cmbdes716d.dll cmbdesfac716d.dll cmbdj716.dll cmbdjfac716.dll cmbdj716d.dll cmbdjfac716d.dll de_db2.dll de_db2_d.dll de_ora.dll de_ora_d.dll

Setting AIX environment variables

Use the `-qalign=packed` compiler option to properly align objects. Refer to the sample makefiles 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 `cmbenv71.sh`
2. For a C shell, use `cmenv71.csh`
3. For a Korn shell, use `cmenv71.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
```

Setting Windows environment variables

You can open a DOS command prompt with the environment configured for developing EIP applications by selecting **Start** → **Programs** → **IBM Enterprise**

Information Portal for Multiplatforms 7.1 → **Development Window**. As an alternative, you can run `CMBenv71.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 your drive

INCLUDE

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

where *x* is your drive

Building C++ programs on Windows

To create a project and makefile in Microsoft® Developers Studio version 5.2 or higher:

1. Select the **Win32** console application from the left frame.
2. In the window that opens, select **empty project**.
3. If you used a different program to write your application, click **Project** → **Add to Project**.
4. Select **Files** and then the .cpp file you want to import.
5. Select **Project/Settings**. Click the **General** tab in Microsoft Foundation Classes.
6. Select **Use MFC in a Shared DLL**.
7. Click the **C/C++** tab. Change the category to Preprocessor. Type the path for the Development Kit - OO API include directories are located in the "Additional include directories" text field.
8. Click the **Link** tab, type in under "Object/library modules" where the Development Kit - OO API lib is located (for example, *x:\cmbroot\lib* where *x* is the drive).
9. You can now build and run your project.

To use an existing makefile using Microsoft Developers Studio version 5.2 or higher, you must complete the following steps:

1. Open the workspace and select the sample makefile.
2. Click **Project Settings**.
3. Select the **C/C++** tab. Change the category to **Preprocessor**, type in the pathname where the Development Kit - OO API include directories are located in the "Additional include directories" text field.
4. Click the **Link** tab, type in under the Object/library modules where the Development Kit - OO API lib is located.
5. You can now build and run your project.

Setting console subsystem for code page conversion on Windows

```
#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);
...
}

```

Multiple search facilities

Use the multiple search facilities to search within a given content server, using one or a combination of supported query types (defined below) or to search the results of previous search. Each search type is supported by one or more search engines.

Parametric query

Queries requiring an exact match between the condition specified in the query and stored data.

Text query

Queries requiring an approximate match between the given query and stored text.

Not all content servers support multiple search facilities. For more information about specific content servers and multiple search, see “Using specific content servers” on page 258.

Tracing information

You can use any of the following environment variable settings to get tracing information.

For text queries using Text Search Engine

The Text Search Engine and all of its functions can only be used with the Content Manager server.

The following environment variable settings put a Text Search Engine query, in binary format, into a specified file.

- CMBTMDSTREAMTRACE=<fileName> (for example, `.\tm.out` for Windows or `<./tm.out>` for AIX)

The following environment variable settings put the Text Search Engine API calls used during a text query into a specified file.

- CMBTMTRACE=<fileName> (for example, `.\tm.api` for Windows or `./tm.api` for AIX)

The following environment setting puts the text search terms into a specified file.

- CMBTMTERM=<fileName> (for example, `.\tmterm.out`)

For parametric queries

The following environment variable setting puts the parametric query passed to the folder manager into the specified file.

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

Catching a DKException

A DKException, once caught, allows you to see any error messages, error codes, and error states that occurred while running. If an error is caught below the DKException name, an error is issued along with the location of where the exception was thrown. The error ID and exception ID are also given.

```
try {
    DKDatastoreDL dsDL;
    dsDL.connect("TM","","","");
    dsDL.disconnect();
}
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;
}
}
```

Connecting to content servers

A DKDatastore xx (where xx is the suffix representing the specific content server, for example, Content Manager (DL), ImagePlus for OS/390 (IP), and so forth) represents and manages a connection to a content server, provides transaction support, and runs server commands.

Establishing a connection

A content server provides methods for connecting to it and disconnecting from it. The following example shows how to connect to a Content Manager library server named LIBSRVRN, using the user ID USER1 and password PASSWORD. In a typical application, you create a datastore, connect to it, work with it, then disconnect from it, as shown in the following example.

```
...
try {
    DKDatastoreDL dsDL;
    cout << "Datastore DL created" << endl;
    cout << "connecting to datastore" << endl;
    dsDL.connect(libsrv,userid,pw);
    cout << "datastore connected " << libsrv << " userid - " << userid << endl;
    dsDL.disconnect();
    cout << "datastore disconnected" << endl;
}
...
```

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

Setting and getting content server options

Each content server provides some processing or informational options that you can set or get using its methods. The following example shows how to set and get the option for establishing an administrative session. See the online API reference for the list of options for each content server and their descriptions.

```

DKAny input_option = DK_DL_SS_CONFIG;
DKAny output_option;
dsDL.setOption(DK_DL_OPT_ACCESS,input_option);
dsDL.getOption(DK_DL_OPT_ACCESS,output_option);

```

Listing content servers

Each content server provides a method for listing the content servers it can connect to.

Restriction: The Domino.Doc datastore does not provide such a method.

The list of servers is returned in a sequential collection of DKAny objects containing DKServerInfoxx (where xx is the datastore suffix in which you want to work, for example, Content Manager (DL), ImagePlus for OS/390 (IP), and so forth) objects. After you obtain a DKServerDefxx object you can retrieve the server name and server type, and use the server name to establish a connection. The following example shows how to retrieve the list of servers:

```

DKDatastoreDL dsDL;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKServerDefDL *pSV = 0;
DKString strServerName;
DKString strServerType;
DKAny a;
long i = 0;
cout << "list servers" << endl;
a = dsDL.listDataSources();
pCol = (DKSequentialCollection*)((dkCollection*)a);
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
{
    i++;
    pSV = (DKServerDefDL*)((void*)(*pIter->next()));
    strServerName = pSV->getName();
    strServerType = pSV->getServerType();
    ... // Process the list of servers as appropriate
    delete pSV;
}
delete pIter;
delete pCol;

```

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

Listing a content server's schema

Each content server provides methods for listing its schema. In a content server, these methods list entities and their attributes.

The list of entities is returned in a DKSequentialCollection object of dkEntityDef objects. The list of attributes for an index class 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 specific information about these two methods, see the online API reference.

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


```

...
cout << "list index classes" << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsDL.listDataSources());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
{
    i++;
    pEnt = (DKIndexClassDefDL*)((void*)(*pIter->next()));
    strIndexClass = pEnt->getName();
    cout << "index class name [" << i << "] - " << strIndexClass << endl;
    cout << " list attributes for " << strIndexClass << " index class" << endl;
    pCol2 = (DKSequentialCollection*)((dkCollection*)dsDL.listEntities(strIndexClass));
    pIter2 = pCol2->createIterator();
    j = 0;
    while (pIter2->more() == TRUE)
    {
        j++;
        pA = pIter2->next();
        pAttr = (DKAttrDefDL*) pA->value();
        cout << " Attribute 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;
        delete pAttr;
    }
    cout << " " << j << " attributes listed for " << strIndexClass
        << " index class" << endl;
    delete pIter2;
    delete pCol2;
    delete pEnt;
    ...
}

```

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

Tip: Instead of deleting pEnt immediately, you can defer it and use the apply method to delete the attribute definition inside of the collection. See “Managing memory in collections” on page 244 for more information.

```

...
pCol2->apply(deleteDKAttrDefDL);
delete pCol2;
...

```

Using 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

DKDDO can be regarded as a container of attributes. An attribute, has a name, value, and properties. Each attribute is identified by a sequential, unique data ID number. For example, if a container had 10 attributes, the first attribute would have the data ID of one, and the last attribute would have a data ID of 10.

Because the data ID number, name, value, and property of an attribute can vary, DKDDO provides flexible mechanisms to represent data originating from different content servers and in different formats. DKDDO can represent data items from different index classes in Content Manager, or rows from different tables in a relational database. The DKDDO itself has properties that apply to the whole DKDDO, instead of to only one attribute.

You must associate a DKDDO with a content server before you can call the add, retrieve, update, and delete methods to send its attributes into the content server and retrieve them. You associate a DKDDO with a content server by calling the proper DKDDO constructor or by calling setDatastore method.

Every DKDDO has a persistent identifier (PID). The PID contains information for locating the attributes in the datastore. For example, in Content Manager, a DKDDO represents an item, which could be a document or a folder.

Creating a DKDDO

Example 1 shows the simplest way to create a DKDDO, by calling its constructor, which takes no parameter.

```
DKDDO addo;
```

If you know the amount of DDO attributes, you can pass this information to the constructor as shown in Example 2.

```
DKDDO bddo(10);
```

In Example 2 , the DKDDO is constructed and can hold up to ten attributes. Example 2 is more efficient than the Example 1, which had no parameters, because addo grows dynamically to accommodate more attributes.

You can create a DKDDO by supplying content server and object type as input.

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

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 a 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 methods. For the add method, the content server creates and returns the item ID.

To create a DDO for retrieving a known item, do the following:

```

DKDatastoreDL dsDL;           // create a Content Manager content server
DKPid pid;
pid.setObjectType("GRANDPA"); // set the index class name it belongs to
pid.setId("LN#U5K6ARLGM3DB4"); // set the item id
DKDDO* ddo = new DKDDO(&dsDL, pid); // create a DDO with PID and associate it to dsDL

```

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

Adding data items and properties

Suppose the index class GRANDPA has the attributes shown in Table 20.

Table 20. Attribute and data_id information

Attribute	data_id=1	2
Name	Title	Subject
Type	String	String
Nullable	No	Yes

You can represent the information shown in Table 3 on page 27 as follows:

```

// create a Content Manager content server
DKDatastoreDL dsDL;
// create a DDO to hold an object type GRANDPA in dsDL
DKDDO* cddo = new DKDDO(&dsDL, "GRANDPA");
DKAny any;
DKBoolean yes = TRUE;
DKBoolean no = FALSE;
// add the first attribute named "Title"
unsigned short data_id = cddo->addData("Title");

// add a property named: "type", set to value : variable length string
any = DK_CM_DATAITEM_TYPE_STRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);

// add a property named: "nullable", set to value : boolean false
any = no;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// add the second attribute named "Subject"
data_id = cddo->addData("Subject");

// add a property named: "type", set to value : variable length string
any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);

// add a property named: "nullable", set to value : boolean true
any = yes;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

```

The above example illustrates the standard attribute properties of type and nullable. You can have as many additional properties as required by your application.

Adding properties to a DDO

The DKDDO has all the required attribute information. However, there is no information to indicate if the DKDDO is either a document or a folder. The following example sets the DKDDO property to indicate that the DKDDO is a document:

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

Setting and getting data item values

In this section, you set the values for the Title and Subject data items you created in “Adding data items and properties” on page 223.

```
// set Title value to the given string
// assume we know the data_id for the data_item "Title" is 1
any = DKString("One dark and stormy night");
cddo->setData(1, any);

// set Subject value to the given string
// assume we do not know the data_id for the data_item "Subject"
// find data_id for data_item named "Subject"
data_id = cddo->dataId("Subject");
any = DKString("Mystery");
cddo->setData(data_id, any);
```

Use the `getData` method to get the values back for Title and Subject:

```
any = cddo->getData(1);
cout << "Title = " << any << endl; // displays "One dark and stormy night"
cout << "Subject = " << cddo->getData(data_id) << endl; // displays "Mystery"
```

Getting the DDKDO and attribute properties

The code example below shows how to retrieve the DKDDO properties:

```
unsigned short prop_id =
    cddo->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
if (prop_id > 0) {
    unsigned short type = (unsigned short) cddo->getProperty(prop_id);
    switch(type) {
        case DK_CM_DOCUMENT:
            // process document
            ...
            break;
        case DK_CM_FOLDER:
            // process folder
            ...
            break;
    }
}
```

The code example below shows how to retrieve attribute properties of the “Title” attribute. You must have the attribute `data_id` to retrieve the properties.

```
// get data_id of Title
data_id = cddo->dataId("Title");
// how many props does it have?
unsigned short number_of_data_prop = cddo->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 = " << cddo->
        getDataPropertyName(data_id, i) << " value = " << cddo->
        getDataProperty(data_id, i) << endl;
}
```

Important: Both `data_id` and `property_id` start from 1. If you specify 0 you receive an exception.

Displaying the DDO

Follow the example below to display the DKDDO content. During application development, you may need to display the DKDDO content for debugging purposes.

```
unsigned short number_of_attribute = cddo->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 = " << cddo->getPropertyName(k) <<
        ",\t value = " << cddo->getProperty(k) << endl;
}
// list data-items and their properties
for (unsigned short i = 1; i <= number_of_attribute; i++) {
    cout << i << " Attr. Name = " << cddo->getDataName(i) <<
        << ",\t value = " << cddo->getData(i) << endl;
    number_of_data_prop = cddo->dataPropertyCount(i);
    for (unsigned short j = 1; j <= number_of_data_prop; j++) {
        cout << "\t" << j << " Data Prop. Name = "
            << cddo->getDataPropertyName(i, j)
            << ",\t value = " << cddo->getDataProperty(i, j)
            << endl;
    }
}
```

Deleting a DDO

There are two ways to delete a DDO.

1. You delete a DKDDO by calling its destructor. The DDO is deleted in memory, but the persistent copy in the content server is unchanged.
2. You use the `del` method in DKDDO, which deletes the persistent copy in the content server. The DKDDO representation in memory does not change. 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`.

Using XDOs

An XDO represents a single part in Enterprise Information Portal. One type of XDO is called `DKBlobxx`, (where *xx* is the suffix representing the specific server. For example, *xx* could represent Content Manager (DL), OnDemand (OD), ImagePlus for OS/390 (IP), or other servers. `DKBlobxx` requires the datastore `DKDatastorexx` as an input to create the object instance.

Using an XDO PID

An XDO needs a PID to store its data persistently. For Content Manager, VI400 and IP390, the item ID and part ID of `DKPidXDOxx` are required for XDO to locate the persistent data in a datastore. Relational Databases require the table, column and datapredicate string to locate the persistent data in a datastore.

Understanding XDO data members

For object content of Content Manager to be indexed by a search engine correctly, you must set the values for the following XDO properties. You use the methods of the `DKBlobxx` to set these properties where they apply. The required values to set are `SearchEngine`, `SearchIndex` and `SearchInfo`. All the properties are not available for all content server types. If not set, the default values are used.

Note: The following values are for Content Manager only.

RepType (representation type)

FRN\$NULL

Attention: The only representation type (or RepType) supported by Content Manager for AS/400 is " ", eight blank spaces surrounded by leading and trailing quotation marks.

ContentClass

DK_DL_CC_UNKNOWN

AffiliatedType

DK_DL_BASE

AffiliatedData

NULL

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

DB2, ODBC and DataJoiner configuration strings for C++

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

CC2MIMEFILE=(filename)

Specify the cmbcc2mime.ini file (optional).

DSNAME=(datastore name)

Specify the datastore name (optional). **Note:** When this datastore is used by Federated, this option is set automatically.

AUTOCOMMIT=ON | OFF

Specify autocommit is on or off. Default is off (optional). **Note:** When this datastore 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 methods (optional).

Programming tips

For Content ManagerContent 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 method, if you set part ID to 0, the system assigns an available part ID for it. You can retrieve the part ID value after add if you want to do some other operation with that object later.

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

```
unsigned long partID = ((DKPidXDODL*)(axdo->getPidObject()))
                    ->getPartId();
```

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

Using XDO as a part of DDO instead of a stand-alone XDO

An XDO represents a single part object when a DDO is a document that is a collection of part objects. You can manipulate the XDO as a component of the DDO or as a stand-alone object. To handle as a part of the DDO, you must get the item ID for the XDO from the DDO. To handle it as a stand-alone object, you must know the existing item ID for the XDO.

XDO as a part of DDO

The major statements used to relate the XDO with the DDO are listed in the following code sample:

```
//create DDO
DKPid pid;
pid.setObjectType(indexClassName);
DKDDO* ddo = new DKDDO(&dsDL, pid);
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, DK_CM_DOCUMENT);
...
DKParts* parts = new DKParts;
DKAny any;
//create XDO
DKBlobDL* axdo = new DKBlobDL(&dsDL);
DKPidXDODL apid;
apid.setPartId(partId);
axdo->setPidObject(apid);
axdo->setContentClass(DK_DL_CC_GIF);
axdo->setAffiliatedType(DK_DL_BASE);
axdo->setContentFromClientFile(imageNames[i]);

//add XDO to the DKParts collection
any = (dkDataObjectBase*)axdo;
parts->addElement(any);
...
//add DDO
short partsDataId = -1;
partsDataId = ddo->addData(DKPARTS);
ddo->addDataProperty(partsDataId, DK_CM_PROPERTY_TYPE, DK_CM_COLLECTION_XDO);
any = (dkCollection*)(parts);
ddo->setData(partsDataId, any);
ddo->add();
```

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

Stand-alone XDO

The following code examples specific for Content Manager are for a stand-alone XDO. For RDB and other connectors, please refer to the sample programs in the CMBROOT\Samples directory.

Adding an XDO from buffer: This example shows how to add an XDO from the buffer. To use this sample, you must know the existing XDO item ID.

```
void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, fileName;
    unsigned long partId;
```



```

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;
}
} //end of main

```

Adding an annotation object to an XDO: To add an annotation object, you must insert the following statements in your program before the add method.

```

//----- set DKAnnotationDL ----- (using extension object)
axdo->setAffiliatedType(DK_DL_ANNOTATION);
DKAnnotationDL ann;
ann.setPart(14);
ann.setPageNumber(1);
ann.setX(5);
ann.setY(5);
axdo->setExtension("DKAnnotationDL", (dkExtension*)&ann);

```

Retrieving, updating, and deleting an XDO: To retrieve, update, or delete an XDO in a content server, provide the correct item ID, part ID, and RepType to identify the object.

```

void main(int argc, char *argv[])
{
    DKDatastoreDL dsDL;
    DKString itemId, fileName;
    unsigned long partId;
    partId = 17; //part ID of object
    itemId = "CPPIORH4JBIXWIY0"; //existing item ID
    fileName = "g:\\test\\choice.gif"; //file content to update
    try
    {
        //connection to datastore
        dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
        DKBlobDL* axdo = new DKBlobDL(&dsDL); //create XDO
        DKPidXDODL* apid = new DKPidXDODL; //create Pid
        apid->setPartId(partId); //set part ID
        apid->setPrimaryId(itemId); //set item ID
        axdo->setPidObject(apid); //set Pid to XDO
        axdo->retrieve(); //retrieve the object
        axdo->setContentFromClientFile(fileName); //set file content to buffer area
        axdo->update(); //update the object with buffer data
        axdo->retrieve("new.gif"); //retrieve content to a file
        axdo->del(); //delete object from datastore
        delete axdo; //call destructor
        delete apid; //call destructor
        dsDL.disconnect(); //disconnect from datastore
    }
    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;
}
} //end of main

```

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

```

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;

        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();

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: For every media object added, an entry is created in the FRN\$MEDIA table. That 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.

```

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<<"you 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, userid, 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;

// *** 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: This example shows how to delete an XDO media object. For this example you must know the item ID, partID, and RepType of the XDO.

```

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, 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;

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

```

```

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

    }
    cout<<"about to do del()"<<endl;
    axdo->del();
    cout<<"del successfully..."<<endl;
    flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
    cout<<"after delete isMediaObject? = "<<flag2<<endl;
    delete axdo;
    delete apid;
    dsDL.disconnect();
    cout<<"datastore disconnected"<<endl;
}
catch(DKException &exc)
{
    cout << "Error id" << exc.errorId() << endl;
    cout << "Exception id " << exc.exceptionId() << endl;
    for(unsigned long i=0;i< exc.textCount();i++)
    {
        cout << "Error text:" << exc.text(i) << endl;
    }
    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: This example shows how to retrieve an XDO media object. The retrieved object contains only the media metadata, not the media object itself. For the example below, you must know the item ID and part ID of the XDO.

```

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;

    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;

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, you must use the extension object `DKStorageManageInfoDL`.

```

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 datastore
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 files TxdoAddBsmsDL.cpp, TxdoAddFsmsDL.cpp, TxdosAddBsmsDL.cpp, TxdosAddFsmsDL.cpp, and TxdomAddsmsDL.cpp in the samples directory as examples of adding search indexed objects and media objects to Content Manager.

Changing the storage collection of an XDO: You can change the storage collection of an existing XDO. After setting up the extension object DKStorageManageInfoDL you call the changeStorage method.

```

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

```

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

Creating 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 XDOs. You set the DKPARTS attribute when you retrieve or create a DDO, as shown in the following code example:

```

DKDatastoreDL dsDL;
// create a new DKParts, collection of parts
DKParts* parts = new DKParts;
// create a new XDO blob
DKBlobDL* blob = new DKBlobDL(&dsDL);
// create Pid for this XDO object
DKPidXDODL pid;
// set part number to 5
pid.setPartId(5);
// the item-id this part belongs to
pid.setId("LN#U5K6ARLGM3DB4");
// set the Pid for the XDO blob
blob->setPid(&pid);
// set content class type GIF
blob->setContentClass(DK_CC_GIF);
// set rep type for the part
blob->setRepType(DK_REP_NULL);
// set the blob's content
blob->setContentFromClientFile("choice.gif");
// the viewer program on AIX
blob->setInstanceOpenHandler("xv");

DKAny any = (dkDataObjectBase*) blob;
// add the blob to the parts collection
parts->addElement(any);

...           // create and add some more blobs to the
...           // collection as necessary

```

```

// create a ddo
DKDDO* ddo = new DKDDO;
...           // sets some of its attributes
// set the type to document DDO
DKAny any = DK_CM_DOCUMENT;
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);

// create DKPARTS attribute and sets it to refer to the DKParts object
// add attribute "DKParts"
unsigned short data_id = ddo->addData(DKPARTS);
// add type property
any = DK_COLLECTION_XDO;
ddo->addDataProperty(data_id,DK_CM_PROPERTY_TYPE,any);
// add nullable property
any = (DKBoolean) TRUE;
ddo->addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE ,any);
any = (dkCollection*) parts;
// sets the attribute value
ddo->setData(data_id, any);

```

After you set DKPARTS as an attribute value of a DDO, the DDO owns it.

To get the parts from a DDO, use the following example:

```

// get DKPARTS data-id
data_id = ddo->dataId(DKPARTS);
// parts not found
if (data_id == 0) {
    DKException exc(" parts data-item not found");
    DKTHROW exc;
}
// get the parts collection
any = ddo->getData(data_id);
DKParts* pCol = (DKParts*) any.value();
// create iterator and process the part collection member one by one
if (pCol != NULL) {
    DKAny* element;
    DKBlobDL* blob;
    dkIterator* iter = pCol->createIterator();
    while (iter->more()) {
        element = iter->next();
        blob = (DKBlobDL*) element->value();
        if (blob != NULL) {
            // display the blob using the viewer
            blob->open();
            // other processing
            ...
        }
    }
    delete iter;
}

```

Creating and using the DKFOLDER attribute

In a folder DDO, the DKFOLDER attribute represents a collection of folders and documents that belong to the folder. The value of this attribute is a DKFolder object, which is a collection of DDOs. Similar to DKPARTS, DKFOLDER is set when you retrieve or create a DDO, as shown in the following code sample:

```

DKDatastoreDL dsDL;
DKFolder* folder = new DKFolder;
// create a new DKFolder, collection of DDO
DKDDO* member = new DKDDO;
// create the first member of this folder
// sets the member DDO attributes and properties
...

```

```

// add member to the folder collection
folder->addElement(member);
...
// create and add some more member DDO to the
// DDO collection as necessary
...
// create a folder ddo
DKDDO* ddo = new DKDDO;
// sets some of its attributes
...
// set the type to folder DDO
DKAny any = DK_CM_FOLDER;
ddo->addProperty(DK_CM_PROPERTY_ITEM_TYPE, any);

// create DKFOLDER attribute and sets it to refer to the DKFolder object
unsigned short data_id = ddo->addData(DKFOLDER);
// add attribute "DKFolder"

any = DK_COLLECTION_DDO;
// add type property
ddo->addDataProperty(data_id,DK_CM_PROPERTY_TYPE,any);
// add nullable property
any = (DKBoolean) TRUE;
ddo->addDataProperty(data_id,DK_CM_PROPERTY_NULLABLE ,any);
any = (dkCollection*) folder;
// sets the attribute value
ddo->setData(data_id, any);

```

After you set DKFOLDER as an attribute of a DDO, the DDO owns it.

To get the folder from a DDO, use the following example:

```

// get DKFOLDER data-id
data_id = ddo->dataId(DKFOLDER);
if (data_id == 0) { // folder not found
    DKException exc(" folder data-item not found");
    DKTHROW exc;
}
// get the parts 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) {
            // process the member DDO
            item->retrieve();
            // other processing
            ...
        }
    }
    delete iter;
}

```

Using DKAny

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

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` method, 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.

```
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` method to find the type code of the object inside DKAny.

```
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.

```

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:

```

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:

```

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

```

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

```

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:

```

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

```

Displaying DKAny

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

```

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:

```

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

```

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

```

Using collections and iterators

dkCollection is an abstract class which provides the interface to collection methods. DKSequentialCollection provides the concrete implementation of those methods. Other collections are derived as a subclass of DKSequentialCollection. These collections contain DKAny objects as members.

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.

Collection members are usually objects of the same type. However, you can have members of different types in one collection.

Using sequential collection methods

DKSequentialCollection provides methods for adding, retrieving, removing, and replacing its members. In addition, it has the apply and sort methods. The following example illustrates how to add a new member to a collection:

```

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

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 methods; and its subclass DKSequentialIterator, which contains more methods. An iterator is created by calling the createIterator method on the collection. This method 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 above code could be rewritten as follows:

```

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.

```

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 follows:

```

...
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

```

The above check is necessary in order to avoid skipping the next member after removing the current one.

Managing memory in collections

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:

```

// 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` methods.

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

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

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

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

You could write your own delete method 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` methods.

Sorting the collection

Use the `sort` method 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.hpp`; so you can write your own sort method for sorting your specific collection. The following example illustrates how to sort a collection of DDOs based on each DDO's item ID:

```
DKResults* rs;
... // execute a query to fill DKResults with DDOs
...

DKSortDDOid sortId; // the sort function; sort on item-id
rs->sort(&sortId); // by default, sort in ascending order
...
```

The definition of a function object `DKSortString` is provided in the `samples` directory as an example of creating a sort function object.

Programming tips

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

Understanding federated collection and iterator

An application uses a federated collection to process as a group data objects resulting from a query. At the same time the federated collection preserves the subgrouping relationships that exist between them.

A federated collection can contain an infinite number of nested collections.

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.

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 50 shows the structure and behavior of DKFederatedCollection.

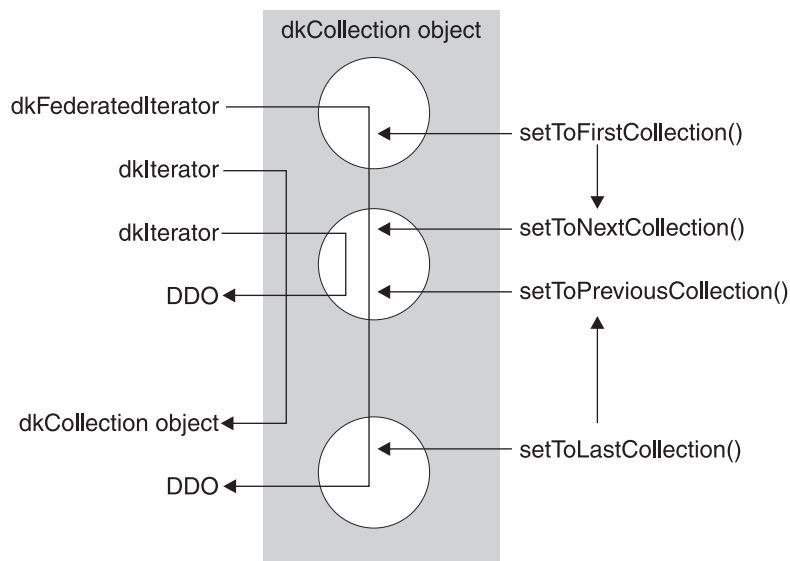


Figure 50. Structure and behavior of DKFederatedCollection

In Figure 50, 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 contained in the federated collection, therefore it returns a DDO member from the DKResults collection.

The setToFirstCollection method 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 method is invoked, it sets the iterator position to the first DDO of the second DKResults collection.

The `setToLastCollection` method 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` method 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. You can create a query object to represent your query, then invoke the `execute` method or `evaluate` method 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.

There are three query object types: parametric, text, and combined. The combined query is composed of both text and parametric queries. Not all content servers can perform combined queries.

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

You can also use `dkResultSetCursor` to run a query again by calling the `close` and `open` methods. This is described in “Using the result set cursor” on page 255.

`DKResults` contains all of the results from the query. You can move an iterator either forwards or backwards over the items in the collection. The `DKResults` collection can be queried and used as a scope for another query. See “Querying collections” on page 257 for more information.

Restriction: Although Domino.Doc content servers return a `DKResults` object, this object cannot be queried nor used 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.
Attention: The Domino.Doc content server cannot rerun a query.
- The `DKResults` object contains the entire result set and supports a bi-directional iterator.

Using parametric query

This section explains how to formulate, and execute different kinds of parametric queries.

Formulating a parametric query

The following example is a query string representing a query on the index class `DLSAMPLE`. The query is searching for all documents or folders with an attribute of

DLSEARCH_DocType <> null. The maximum number of results returned is limited to five. The content is set to YES, so that contents of the document or folder are returned.

The query also specifies that a Content Manager server use dynamic SQL for this query and that all folders and documents be searched. 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 quotation marks (").

```
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)";
```

Formulating a parametric query on multiple criteria

You can specify multiple search criteria using a parametric query. The following example shows how to specify a query on two index classes.

```
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

Each content server provides a method for creating a query object. You can use that query object to execute the query and obtain the results. The following example shows how to create a parametric query object in a Content Manager server and then execute that query. After the query executes, the results are returned in a DKResults collection.

```
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.

Attention: When you delete a DKResults object, all of its members are also deleted. Make sure that you do not delete the element twice. See "Using collections and iterators" on page 243 for more information.

Executing a parametric query from the content server

Each content server provides a method for executing a query. The following example shows how to execute a parametric query in a Content Manager server. After the query executes, the results are returned in a dkResultSetCursor object.

```
...
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 (TExecutedDL.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Evaluating a parametric query from the content server

Each content server provides a method for evaluating a query. The following example shows how to evaluate a parametric query in a content server. After the query executes, the results are returned in a DKResults collection.

```
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

This section explains text queries.

Formulating a text query

The following example shows a query for a text index called TMINDEX. The query searches for all text documents with the word UNIX or member. The maximum number of results returned is five.

```

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.

```

DKString cmd = "SEARCH=(COND=(UNIX OR member));";
cmd += "OPTION=(SEARCH_INDEX=(TMINDEX,TMINDEX2); MAX_RESULTS=5)";

```

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.

Executing a text query

Each content server provides a method for creating a query object. You can use that query object to execute the query and obtain the results. The following example shows how to create a text query object and execute that query. After a query executes, the results are returned in a DKResults collection.

```

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/d1 directory.

Executing a text query from the datastore

Each content server provides a method for executing a query. The following example shows how to execute a text query in a Content Manager server. After the query executes, the results are returned in a dkResultSetCursor object.

```
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/d1 directory.

Evaluating a text query from the datastore

Each content server provides a method for executing a query. The following example shows how to execute a text query in a Content Manager server. After the query executes, the results are returned in a DKResults collection.

```
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 for each text query result item

This example retrieves match highlighting information for each text query result item during a text query, by setting the MATCH_INFO option to YES. The MATCH_DICT option specifies whether the highlighting information should be obtained using a dictionary. The match information is returned in the DKMATCHESINFO attribute of the DKDDO returned from a text query. The value of the DKMATCHESINFO attribute will be a DKMatchesInfoTS object.

Attention: This process is time consuming because the document is retrieved from the content server and linguistically analyzed to determine potential matches.

```
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 :
                {
                    // 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

This example retrieves match highlighting information for a specific item returned from a text query. The match information contains the text of the document and the highlighting information for every match of the corresponding query. The `dkResultSetCursor` passed into this routine must be in an open state.

Attention: This process is time consuming because the document is retrieved from the content server and linguistically analyzed to determine potential matches.

```

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 *pDObj = 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())
            {
                ...
            }
        }
        // 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 DDOs and does not appear until you fetch an element from it. The collection set resulting from a query submitted to the content server.

Important: When you stop using the cursor, call the `destroy` method to close it and prevent memory leaks.

Opening and closing the result set cursor to re-execute the query

When you create a result set cursor, it is open. To run a query again, you close and reopen the cursor, as shown in the following example:

```

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 both set and get the current cursor position. The following example creates and runs a query. Inside the while loop, the cursor position is set to the first (or next) valid position. Then a DDO is fetched from that position. Finally, the cursor position is retrieved, and assigned to the variable `i`. A null is returned from the `fetchObject` method if the cursor is past the last result item.

```

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:

```

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. The following example skips every other item in the result set:

```

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

You can use the result set cursor to populate a collection with a specified number of items from the result set. In the following example, all items from the result set are fetched into a sequential collection. The first parameter specifies how many items to put into the collection. A zero in the first parameter of the `fetchNextN` method indicates that all result set items will be put into the collection. If `fItems` is `TRUE`, at least one item was returned.

```

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 evaluation set or more refined results. A concrete implementation of a queryable collection is a DKResults object. DKResults is a collection of DDOs, which are the result of a query.

Getting query results

The following example illustrates how to submit a parametric query and get results:

```
// 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();
```

The results are in rs, which is a DKResults object. You can use previous code examples to process the collection and get the DDO.

Evaluating a new query

As shown in the code example below, you can query the result from the query to further refine it. For example:

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

A DKResults object called any contains the refined results. The combined results of both queries are:

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

You can repeat this step until you get satisfactory results. After you start with one type of query, the subsequent queries must be of the same type, because you might get a null result.

The following example is for text queries:

```
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);
```

A DKResults object called any contains the DKResults object containing the combined results of both queries:

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

Using queryable collection instead of combined query

Evaluating a queryable collection is similar to other Java classes. One such class is combined query. You can use a combined query 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.

The result of a combined query is a DKResults object, so you can theoretically evaluate another parametric query against it; although it might not always work. You cannot perform 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. This is quite useful for dynamically browsing a content server 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.

Using specific content servers

Each content server uses the dkDatastore classes, or data definition classes, as the primary interface to the Enterprise Information Portal database. Each content server has a separate content server class that implements the dkDatastore class to provide information specific to the content server.

Each content server type is represented by a class called DKDatastore xx , where xx is an abbreviation that identifies the specific content server as shown in table Table 21.

Table 21. Server type and class name terminology

Server type	Class name
Content Manager	DKDatastoreDL
VisualInfo for AS/400	DKDatastoreV4
ImagePlus for OS/390	DKDatastoreIP
Domino.Doc	DKDatastoreDD
Domino Extended Search	DKDatastoreDES
relational database	DKDatastoreDB2, DKDatastoreJDBC, DKDatastoreODBC (for C++ on WinNT), DKDatastoreDJ
Information Catalog	DKDatastoreIC
relational database	DKDatastoreDB2, DKDatastoreODBC (for C++ on WinNT), DKDatastoreDJ
Information Catalog	DKDatastoreIC

When creating a content server you must implement one concrete class for each of the following classes or interfaces:

dkDatastore

Represents and manages a connection to the content server, transactions, and the execution of content server commands. dkDatastore is an abstract version of the query manager class. It supports the evaluate method, so it can be considered a subclass of query manager.

dkDatastoreDef

Defines methods to access items stored in the content server; it can also create, list, and delete its entities. It maintains a collection of dkEntityDefs. Examples of concrete classes for this interface are:

- DKDatastoreDefDL
- DKDatastoreDefV4

dkEntityDef

Defines methods to:

- Create and delete an entity.
- Access entity information.
- Create and delete attributes.

In this class, all methods related to subentities generate DKUsageError objects indicating that the datastore does not then support subentities. However, if the content server supports multiple-level entities the subclass for the content servers must implement methods to overwrite the exceptions, for example. Examples of concrete classes for the dkEntityDef interface are:

- DKIndexClassDefDL
- DKEntityDefIP

dkAttrDef

Defines methods to access attribute information and to create and delete attributes. Examples of concrete classes for dkAttrDef are:

- DKAttributeDefDL
- DKAttrDefV4

dkServerDef

Defines methods to access server information. Examples of concrete classes for dkServerDef are:

- DKServerDefDL
- DKServerDefDD

dkResultSetCursor

A content server cursor that manages a virtual collection of DDO objects. The collection is a query result set. Elements of the collection do not materialize until a datastore fetch operation is run. To use the addObject, deleteObject, and updateObject methods, you must set the datastore option DK_CM_OPT_ACCESS_MODE to DK_CM_READWRITE.

dkBlob

An abstract class that 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 polymorphic processing of BLOB collections originating from heterogeneous content servers. Examples of concrete classes for dkBlob are:

- DKBlobDL
- DKBlobDD

For more information on dkDatastore and other common classes, see “Developing custom content servers” on page 316.

Working with Content Manager

This section describes how to:

- Handle large objects
- Use DDOs in the server
- Use XDOs in a search engine
- Use combined query
- Use Text Search Engine
- Use image search (QBIC)
- Use workflows and workbaskets

Handling large objects

This section describes how Content Manager handles large objects.

Setting Java heap size: 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 uses objects larger than the default maximum heap size, your program will fail during execution. To increase the maximum heap size for your application, use the `-mx` option when you run your Java application program, for example:

```
java -mx40000000 yourApplication
```

Using DDOs to represent Content Manager's data

A DDO associated with DKDatastoreDL has some specific information to represent the Enterprise Information Portal document metaphor: document, folder, parts, item, item ID, rank, and so forth. The following sections describe how you record this information.

DDO properties: The type of an item, either 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:

```
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 223 and “Getting the DDKDO and attribute properties” on page 224 for more information.

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 datastore. See “Creating a persistent identifier (PID)” on page 222.

Representing documents: A DDO representing a document has the property `DX_DL_PROPERTY_ITEM_TYPE` equal to `DX_DL_DOCUMENT`. Its PID contains the index class name as the object type, and the item ID in the PID's ID.

The parts inside a document are represented as DXPartsDL objects, which are collections of binary large objects (BLOBs), each of which is represented as a DXBlobDL object. A document DDO has a specific attribute with the reserved name `DX_DL_DKPARTS`, whose value is also DKParts object. To get to each part in a document, you must retrieve DKParts first, then create an iterator to retrieve each part. If the document does not have a part at all, DKPARTS is null. Documents

associated with a combined query (a combination of a parametric and text query) can have a transient attribute called 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 265, and “Creating and using the DKPARTS attribute” on page 238.

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.

The table of contents inside a folder is represented as a DKFolder object, which is a collection of DDOs. Each collection represents an item—either a document or another folder—belonging to this folder. A folder DDO has a specific attribute with a reserved name DKFOLDER, whose value is also DKFolder object.

To get to each DDO member of the folder, you must retrieve DKFOLDER first, then create an iterator to retrieve each item member. If the folder does not have a member, DKFOLDER is null.

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 265, and “Creating and using the DKFOLDER attribute” on page 239.

Creating, updating, and deleting documents or folders

This section describes creating, updating, and deleting documents and folders.

Creating a document: To create a document and save it in a content server, you must create a DDO, setting all of its attributes and other information, except its item ID. The item ID is assigned and returned by the content server. Some of the previous examples are combined in the following example:

```
// 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 to 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

// 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();

```

After the last step, the document is created 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 listed in DKPARTS. This also applies to adding a folder DDO, the DKFOLDER collection members are added to the datastore as a component of the folder. A folder contains a table of contents of its members, which are existing documents and folders. Therefore, all folder members must be created in the datastore before you can add a folder DDO. 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 you must set the item ID and object type. Then, update the desired attributes, or add parts to the collection. Finally, call the update method to store the change, for example:

```

// 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 datastore 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. Updating a folder DDO requires similar steps: you update the attribute values, or add or remove elements from `DKFolder`, then call the update method.

Updating parts: The collection of parts in a document is represented in a `DKParts` object called `DKPARTS`. `DKParts` is a subclass of `DKSequentialCollection`. In addition to inheriting the sequential collection functions, `DKParts` has two additional members for adding a part to, and removing a part from, the collection and immediately saves those changes. The document must already exist in the content server.

Adding and removing a part: The following example adds a part to a document:

```

// 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);

```

Similarly, to remove `newPart` from the collection and the content server, enter:

```

parts->removeMember(ddo, newPart);

```

The `removeMember` method in `DKParts` actually deletes the part from the content server.

Differences between update, add, and remove on a document DDO: `DKParts` `addMember` and `removeMember` methods provide conveniences for adding and removing a part in the collection and the content server. Compared to the update method in a document DDO, the `addMember` and `removeMember` are faster.

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 shown in the following example:

```

...
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: The collection of documents and folders within a folder is represented as a DKFolder object. In the content server, a folder holds a table of contents referring to its objects instead of all of the 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 to, or removing a member from, the collection and immediately stores those changes.

The added document or folder to be added must already exist in the content server, as must the folder to be added to.

Adding and removing a member: The following example adds another document or folder to a folder:

```

// 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);

```

Similarly, to remove newMember from the collection and the content server, enter:

```
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 method it does not delete the member from memory or from the content server.

Differences between update, add, and remove on a folder DDO: DKFolder addMember and removeMember methods provide conveniences for adding and removing a document or folder the collection and the content server. Compared with the update method in a folder DDO, add and remove methods are faster.

The update method on a DDO updates all of the attributes in the DDO, including DKFolder and all of its members, whereas add and remove member methods involve only adding and removing one particular member to or from 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.

```
ddo->del();
```

The DDO must have its item ID and object type set, and your program must have a valid connection to the content server.

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. See “Creating a document” on page 261 for more information.

Retrieving a document or folder

To retrieve a document from DKDatastoreDL, you must know the document’s index class name and item ID. You must first associate the DDO to a content server and establish a connection.

```
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();
```

To retrieve the DDO, call:

```
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 datastore. 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, it is not desirable to 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 was run 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 identified in DKPARTS, as follows:

```
DKAny any = ddo->getDataByName(DKPARTS);
DKParts* parts = (DKParts*) any.value();
```

This example assumes that the DKPARTS attribute exists. An exception is generated if the attribute does not exist. See “Retrieving a folder” on page 266 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 and using the DKPARTS attribute” on page 238.

```
// 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 information retrieval. To bring its content and related information into memory, call:

```
blob->retrieve();
```

Retrieving a folder: Retrieve a folder DDO the same way as you would retrieve a document DDO. After being retrieved, the folder DDO has all of its attributes set, including a special attribute, DKFOLDER. This attribute value is set to a DKFolder object, a collection of DDO members in this folder. Like the parts in DKParts, these member DDOs contain only enough information to retrieve them. You can retrieve a folder DDO as follows:

```

// 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 and using the DKFOLDER attribute” on page 239.

Understanding text searching (Text Search Engine)

The Text Search Engine product can specify boolean, proximity, global text retrieval (GTR), hybrid, and free text queries. You can use the text search item ID, part number, and ranking information returned by the query to create an XDO that retrieves the text document contents in a Content Manager server.

The DKDatastoreTS object does not support add, update, retrieve, and delete methods. You can query this content server. Refer to “Loading data to be indexed by Text Search Engine” on page 272 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. A phrase is a sequence of words enclosed in apostrophes ('), that is searched for as a literal string.

The following example is searching for all text documents with the word WWW or the phrase Web site in the TMINDEX text search index:

```
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 is searching for all text documents with the free text web site in the TMINDEX text search index:

```
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 example is searching for all text documents with the words IBM and UNIX, as well as the free text web site in the TMINDEX text search index.

```
DKString cmd = "SEARCH=(COND=(IBM AND UNIX {Web site}))";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Proximity query: A proximity query relates to a sequence of search arguments found in the same document, paragraph, or sentence. The following example is searching for all text documents with the phrase rational numbers and the word math in the same paragraph.

```
DKString cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}))";  
cmd += "OPTION=(SEARCH_INDEX=TMINDEX)";
```

Note: 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, but also supports single-byte character set (SBCS) languages. The following example shows a search for all text documents with the phrase IBM marketing. All double-byte characters should be enclosed in apostrophes ('). The phrase to be searched for should be in the specified character code set and language. The MATCH keyword is set to indicate the degree of similarity for the phrase.

```
DKString cmd = "SEARCH=(COND=($CCSID=850, LANG=6011,MATCH=1$ "  
cmd += "' IBM marketing'))";  
cmd += "OPTION=(SEARCH_INDEX=TMGTRX)";
```

Make sure that the text search datastore 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 example. You must specify both CCSID and LANG; one value cannot be specified with the other.

Representing Text Search Engine information using DDOs: A DDO associated with a DKDatastoreTS object has specific information for representing results from text searches. DKDatastoreTS 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 has a set of standard attributes, described below.

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:

datastore type

TS

datastore 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 methods for connecting and a method for disconnecting. Normally, you create a DKDatastoreTS object, connect to it, run a query, then disconnect when done. The following example shows the first connection method using the text search server TM.

```
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/d1 directory.

The following example shows the second connection method 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 method 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 connection method using the text search server name TM, library server LIBSRVRN, user ID FRNADMIN, and password PASSWORD:

```
dsTS.connect("TM","", "", "LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)");
```

Note that 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 methods. See the online API referenceonline API referencefor 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.

```
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);
```

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 referenceonline API referencefor 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 in “Global text retrieval (GTR) query” on page 69.

Listing servers: The DKDatastoreTS object provides a method to list the text search servers that it can connect to. The following example shows how to retrieve the list of servers.

```
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 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.

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

Listing schema:: A `DKDatastoreTS` object provides methods 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.

```

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.

See “Managing memory in collections” on page 244 for information about deleting the collection.

Indexing XDOs by search engines: If you want to index object content using Text Search Engine, the values of `SearchEngine`, `SearchIndex`, and `SearchInfo` are required.

The SearchIndex value is a combination of two names: the search service name and search index name. For example, if the system administration program refers to a text search server named TM for which you defined a search index named TMINDEX, then the correct value for the SearchIndex is TM-TMINDEX.

The value of SearchEngine must be SM for text search. The value of SearchEngine must be QBIC for image search (query by image content).

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 the files FLoadSampleTSQBICDL.cpp and LoadFolderTSQBICDL.cpp inside the samples directory, for examples of how to load data, or create a folder and then 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:

```
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, userid, 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;

//--- 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;
}

```

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 by running the samples TListCatalogDL.cpp and TListCatalogTS.cpp. Before running the samples, update them with your server, user ID, and so forth.

To create parts in Content Manager that are indexed by the Text Search Engine, refer to "Using XDOs" on page 225.

After the data is loaded into Content Manager, use the `wakeUpService` method in the `DKDatastoreDL` class to place the documents on the document queue.

From the Content Manager text search Administration window:

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 about text search administration, refer to the *System Administration Guide*.

Using text structured document support: Text structured documents are composed using text, for example an HTML file. A document model defines how the text document is laid out. For example, an HTML file contains tags for beginning and ending the file. Text Search Engine can perform searches on words or phrases between the HTML tags.

You can perform text queries on structured documents as follows:

1. Create a document model. The document model contains sections which include the section name and document tag in the text document, for example:

```
<HTML>
<HEAD>
<TITLE>My Corp<br></TITLE>
</HEAD>

<BODY>

<H1>My Corp<BR></H1>
<P><B>My Corp<BR></B><BR>
<P>Robert Summers<BR>
<P><ADDRESS>My Corporation<BR></ADDRESS>

<HR>
<H2>My 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 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.

```
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/d1 directory.

This example shows how to create a document model.

```

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("SAMP CORPMOD");
docSection->setName("SAMP CORPTITLE");
docSection->setTag("TITLE");
docModel->addDocSection(docSection);
docSection2->setName("SAMP CORP BODY");
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/d1 directory.

This example shows how to create and set the indexing rules for a text search index and that uses a document model.

```
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);

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/d1 directory. The example shows how to start the indexing process by using the system administration program.

```

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 << "getDocsInSecondaryIndex " << pIndexFuncStatus->getDocsInSecondaryIndex()
    << 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/d1 directory.

The indexing process is asynchronous so this program just begins the process. To check the status, you must use the system administration program.

The sample named TCheckStatusTS.cpp can be used to see if a queued request has been moved from the scheduled document queue to the primary or secondary queues. If an indexing error occurs, you can check imldiag.log file. For more information about the imldiag.log file see the *Text Search Engine Application Programming Reference*.

This example shows how to execute a structure document text query based on the document model and the text search index defined above.

```

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/d1 directory.

Searching images by content

In the past, image queries were limited to file names and descriptions associated with images. You can use the IBM Image Search server to search for stored images by specifying the image type or by providing an example of the image.

Figure 51 on page 278 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.

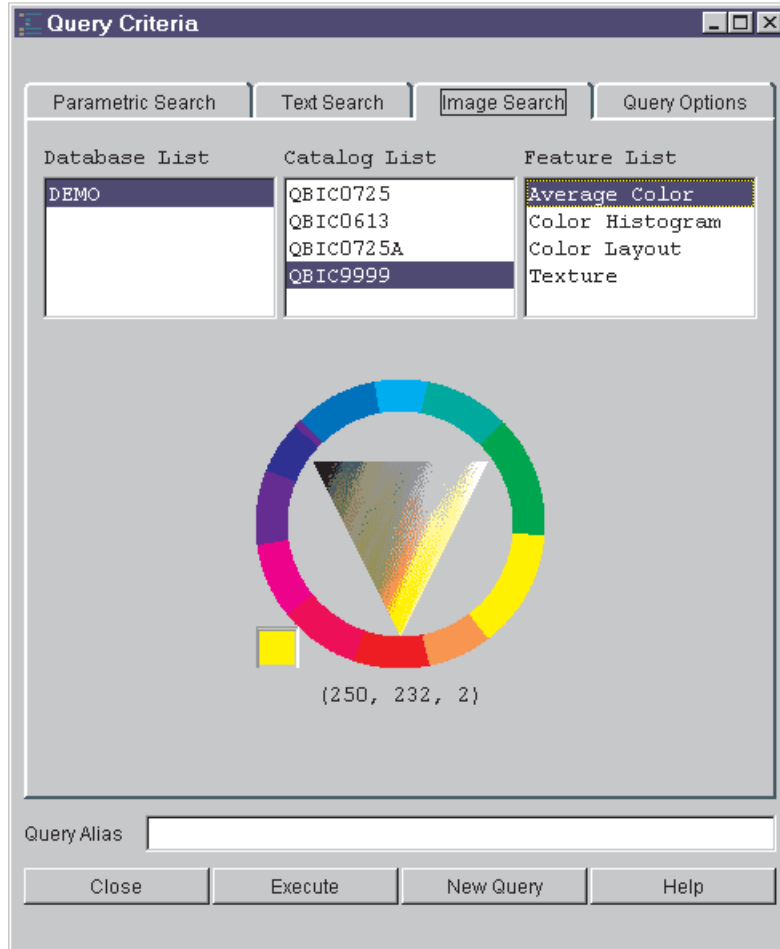


Figure 51. Image search sample client application

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 the entire Content Manager system. It also describes *features* that are the searchable visual characteristics of images.

Understanding image search server, databases, and catalogs: A Content Manager system uses an image search server to search for images. Content Manager applications store image objects in the object server; the image search server analyzes images and stores the image information. The image search server does not store images themselves. Figure 52 on page 279 shows an example of an image search server.

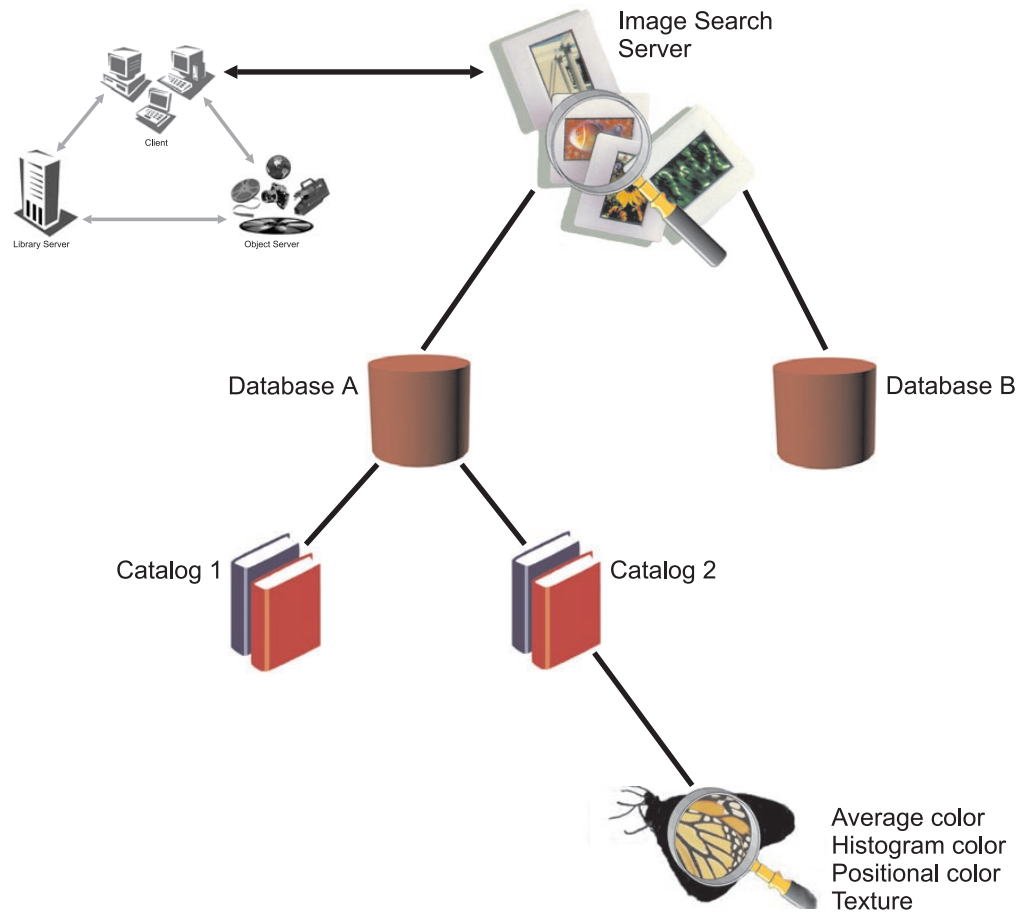


Figure 52. An image search server in a Content Manager system. The image search server communicates with the other Content Manager components through the clients.

The image search server contains one or more catalogs that hold information about one or more of the four image search features:

- Average Color
- Color Histogram
- Color Layout
- Texture

Understanding image search features: This section explains the four image search features.

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 will have an average color of orange.

`QbColorFeatureClass` is 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

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.

Requirement: 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 53 on page 281 shows an example of the image search client search and retrieve process.

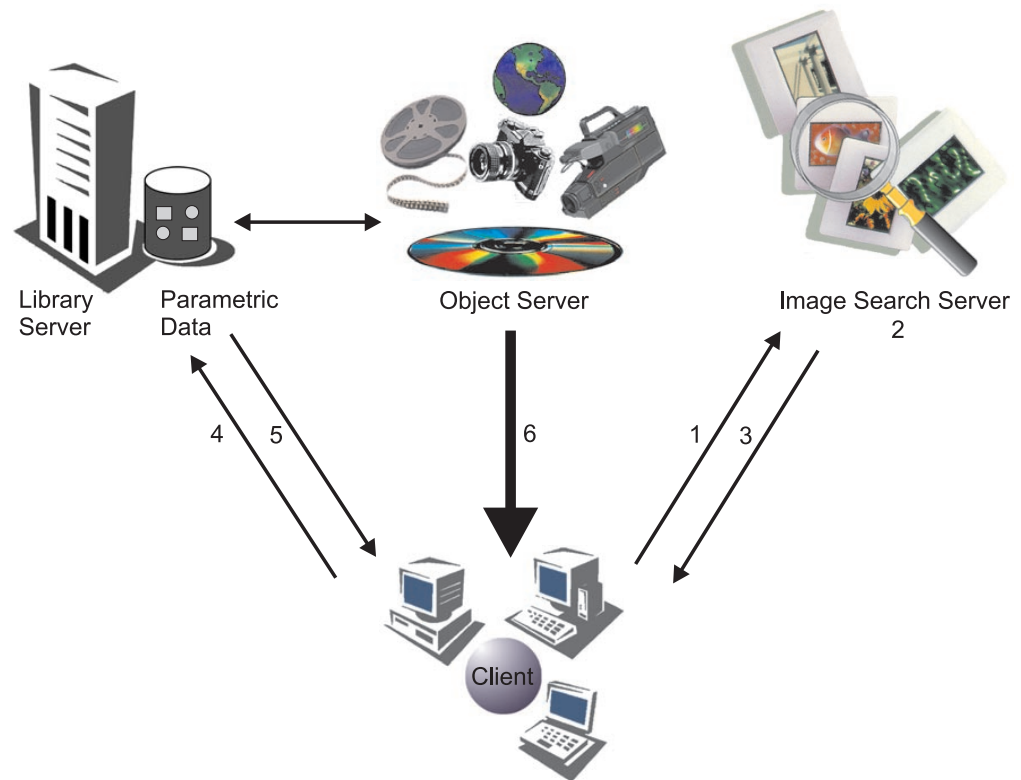


Figure 53. How image search clients search and retrieve images

1. A client application 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.

The query string: An image query is a character string that specifies the search criteria. The search criteria consist of:

Feature name The features used in the search.

Feature value The values of those features. Table 22 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

The maximum number of matches for the query to 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. 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. When you specify multiple features in a query, you can assign a weight to one or more of the features. `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 may also specify the maximum number of matches that a query returns. To specify the maximum results, append `max_results` to your query. `max_results` consists of the keyword `max`, an equal sign (=), and an integer greater than 0. Table 22 describes Image search query feature names and values.

Table 22. Image search query valid feature names and values

Feature name	Values
QbColorFeatureClass or QbColor	<p>color = < <code>rgbValue</code> , <code>rgbValue</code> , <code>rgbValue</code> > where <code>rgbValue</code> is an integer from 0 to 255.</p> <p>file = < <code>fileLocation</code> , " <code>fileName</code> " > where <code>fileLocation</code> is either <code>server</code> or <code>client</code>, <code>fileName</code> is the complete file path specified in the format appropriate for the system on which the file resides.</p>
QbColorHistogramFeatureClass or QbHistogram	<p>histogram = < histList > where <code>histList</code> consists of one or more <code>histClause</code> separated by a comma (,).</p> <p>A <code>histClause</code> is specified as (<code>histValue</code> , <code>rgbValue</code> , <code>rgbValue</code> , <code>rgbValue</code>), where <code>histValue</code> is an integer from 1 to 100 (a percentage value), and <code>rgbValue</code> is an integer from 0 to 255.</p> <p>file = < <code>fileLocation</code> , " <code>fileName</code> " > where <code>fileLocation</code> is either <code>server</code> or <code>client</code>, <code>fileName</code> is the complete file path specified in the format appropriate for the system on which the file resides.</p>

Table 22. Image search query valid feature names and values (continued)

Feature name	Values
QbDrawFeatureClass or QbDraw	<p>description = < " descString " > where descString is a special encoded string describing a picker file. Format of the description string:</p> <ol style="list-style-type: none"> 1. D,w,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. The colon character (:) is used as a separator. <p>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.</p>
QbTextureFeatureClass or QbTexture	<p>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.</p>

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

Establishing a connection in QBIC

Image search provides methods 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.

```

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/d1 directory.

The image search connection allows an application to connect to an image search server.

After connecting, your program can use methods that access the image search server, except for the methods that are not related to image search catalogs, such as listDatabases. An openCatalog method is required to open a catalog for processing. A closeCatalog method is called after processing is done. The following example shows how to connect, open a catalog, close the catalog, and disconnect.

```

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 method 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.

```

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/d1 directory.

Listing image search databases, catalogs, and features

DKDatastoreQBIC provides a method 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.

```

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++;
            pAttr = (DKFeatureDefQBIC*) pA->value();
            cout << "    Attribute name [" << k << "] - "
                << pAttr->getName() << endl;
            cout << "        datastoreName " << pAttr->datastoreName()
                << endl;
            cout << "        datastoreType " << pAttr->datastoreType()
                << endl;
            cout << "        attributeOf " << pAttr->getEntityName()
                << endl;
            delete pAttr;
        }
        delete pIter3;
        delete pCol3;
        delete pEntCat;
    }
}

```

```

    cout << " " << j << " features listed for catalog: "
        << strCatName << endl;
    delete pIter2;
    delete pCol2;
    delete pEntDB;
}
delete pIter;
delete pCol;
cout << i << " databases listed" << endl;
dsQBIC.disconnect();

```

The complete sample application from which this example was taken (TListCatalogQBIC.cpp) is available in the Cmbroot/Samples/cpp/d1 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:

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.

DKDLITEMID

The item ID specifying where this image part is stored. Use this item ID to retrieve the whole item from the content server.

DKPARTNO

A long integer part number identifying the location of this image part. In Java, DKPARTNO is an integer part number. Use with the item ID to retrieve this part from the content server.

DKREPTYPE

A representation type string with a default value of FRN\$NULL. This attribute is reserved.

DKRANK

A long integer rank signifying the relevance of this part to the result set from the image query. The rank is within the range 0 to 100. A higher rank means a better match.

Working with image queries

This section describes how to create, run, and evaluate image queries.

Creating an image query: The following example shows a query string that searches for all images with average color red. "QbColorFeatureClass" represents the feature for average color.

```
String cmd = "QbColorFeatureClass color=<255, 0, 0>";
```

Running an image query: The content server lets you 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.

```

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/d1 directory.

Running an image query from the content server: DKDatastoreQBIC provides a method for running a query. The following example shows how to run an image query on the content server. Results are returned in a dkResultSetCursor object.

```

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/d1 directory.

Evaluating an image query from the datastore: DKDatastoreQBIC provides a method 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.

```

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.

Query based on average color: A query based on average color consists of a feature name and its value. The following example shows how to search for all images based on average color red:

```
String cmd = "QbColorFeatureClass color=<255, 0, 0>";
```

Query based on color percentages: A query based on color percentages consists of a feature name and its value. The following example shows how to search for all images based on a histogram of three colors: 10% red, 50% blue, and 40% green.

```
String cmd = "QbColorHistogramFeatureClass ";
cmd += "histogram=<(10, 255, 0, 0),(50, 0, 255, 0),(40, 0, 0, 255)>";
```

Query based on color layout: A query based on color layout consists of a feature name and its value. The following example shows how to search for all images based on color layout described by an image in a file on the client:

```
String cmd = "QbDrawFeatureClass file=<client, \\patterns\pattern1.gif>";
```

Query based on texture: A query based on texture consists of a feature name and its value. The following example shows how to search for all images based on the texture value provided by an image in a file on the client:

```
String cmd = "QbTextureFeatureClass file=<client, \\patterns\pattern2.gif>";
```

Query with multiple features: You can specify a query with multiple features. The following example shows how to search for all images based on 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 average color is twice that of the texture:

```
String cmd = "QbColorFeatureClass color=<255, 0, 0> weight=2.0 and ";
cmd += "QbTextureFeatureClass file=<client, \\patterns\pattern2.gif>";
```

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. This example tests the `setToBeIndexed` method of the `DKBlobDL` class.

```
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;

//--- 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:

```

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_PARAM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
// to signal the end of parameter list
par[2].setName(DK_PARAM_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, you can slightly modify the previous example to insert the scope as an additional parameter:

```

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_PARAM_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_PARAM_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 workflow and workbasket functions

This section describes the workflow and workbasket functions.

Understanding the 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 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 methods.

The following example shows how to connect to a Content Manager server named LIBSRVRN, using the user ID FRNADMIN and the password PASSWORD.

```
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/d1 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 method add ()

The example follows the six steps to create a workflow.

```
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/d1 directory

Important: If you connect to the datastore 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 method for listing the workflows in the system as shown in the following example. The list is returned in a sequential collection of DKWorkFlowDL objects.

```
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/d1 directory.

Creating a 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 method

The following example follows these steps to create a workbasket. A complete code sample named TCreateDelWorkBasket.cpp is available in the samples directory. If you connect to the datastore as a normal user (DK_SS_NORMAL), you do not get the workbasket defined after you connect. Therefore, this sample uses DK_SS_CONFIG.

```
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();
```

Listing workbaskets: DKWorkflowServiceDL provides a method for listing the workbaskets in the system as shown in the following example. The list is returned in a sequential collection of DKWorkBasketDL objects.

```
DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1=(DKSequentialCollection *)wfDL.listWorkBaskets();
if (wfList1 != NULL)
{
    dkIterator * pIter1 = wfList1->createIterator();
    DKWorkBasketDL * pwbl;
    while (pIter->more())
    {
        pwbl = (DKWorkBasketDL *)((void*)(*pIter1->next()));
        pwbl->retrieve();
        ... // do some work
        delete pwbl;
    }
}
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 a workflow: DKWorkflowServiceDL provides a method 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.

```
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());
    }
}
```

```

        ...
    }
}
dsDL.disconnect();

```

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

Executing a workflow: DKWorkflowServiceDL provides methods 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 EP8L80R9MHH##QES
- Use a valid workflow ID instead of HI7MOPALUPFQ1U47
- Use a valid workbasket ID instead of E3PP1UZ0ZUFQ1U3M

```

DKDatastoreDL dsDL;
DKWorkflowServiceDL wfDL(&dsDL);
DKString itemID = DKString("EP8L80R9MHH##QES");
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
DKString itemIDWB = DKString("E3PP1UZ0ZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkflowItem(itemID,           // itemID
                      itemIDWF,        // itemIDWB
                      NULL,             // default(the first workbasket)
                      TRUE,             // overload
                      DK_WIP_DEFAULT_PRIORITY // initial_priority
                      );
...
wfDL.routeWipItem(itemID,           // itemID
                  itemIDWF,        // itemIDWB
                  TRUE,             // overload
                  DK_NO_PRIORITY_CHANGE // initial_priority
                  );
...
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 ImagePlus for OS/390

ImagePlus for OS/390 includes the following features:

- Connecting and disconnecting from the datastore
- Retrieving categories
- Retrieving attribute fields
- Retrieving folders
- Retrieving documents

Restriction: ImagePlus for OS/390 does not support:

- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

Listing entities and attributes

The following example lists all of the entities found in an ImagePlus for OS/390 content server.

```

// 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 uses the `getAttr` and `listAttrNames` methods to list all of the attributes associated with each entity.

```

// Method 1:
cout << "---List attributes using listAttrNames and getAttr methods---" << 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<tmpcount; 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 uses the `listEntityAttrs` method to list all the attributes associated with each entity.

```

// Method 2:
cout << "---List attributes using listEntityAttrs method---" << 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:

```

SEARCH=(COND=(ip_search_expression),ENTITY={entity_name | mapped_entity_name}
        [,MAX_RESULTS=maximum_results]);
        [OPTION=( [CONTENT={YES | ATTRONLY | NO}; ][PENDING={YES | NO}; ])]

```

The valid variables are:

ip_search_expression

ip_search_criteria [[*binary_operator ip_search_criteria*] ...]

Restriction: Only the boolean operator AND is supported

ip_search_criteria

{attr_name | mapped_attr_name} operator literal

attr_name

Name of the entity attribute to search for.

mapped_attr_name

Attribute name mapped with the attribute to search for.

operator

For DATE attributes , the following operators are supported:

- == equality
- > greater than
- < less than
- >= greater than or equal to
- <= less than or equal to

For all other attributes, only equality (==) is supported

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 for

mapped_entity_name

the Entity name mapped with the entity to search for

maximum_results

The desired maximum number of results to be returned

The valid keywords are:

CONTENT option

Controls the amount of information returned in the results

YES The PIDs, attributes, and their values for a document or folder are set. If there are parts in a document, the XDO PIDs are set. If there are documents in a folder, the document PIDs are set.

YES is the default.

NO Only the document or folder PIDs are set.

ATTRONLY

Only the PIDs, attributes, and their values for a document or folder are set.

PENDING option

Controls whether or not documents that do not have parts are included in the results. This option only applies when ENTITY is set to DOCUMENT or to an entity mapped to DOCUMENT.

YES YES is the default value. Includes pending documents in the results

NO Does not include pending documents in the results

Working with VisualInfo for AS/400

The API classes provided for VisualInfo for AS/400 are very similar to those provided for Content Manager.

Restriction: VisualInfo for AS/400 does not support:

- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

Listing index classes and attributes

The following example lists index classes and attributes in VisualInfo for AS/400.

```
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.

Executing a query

Figure 54 on page 301 shows an example of how to run a query in VisualInfo for AS/400 and process the results.


```

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;
                }

                case DKAny::tc_long :
                {
                    lVal = a;

```

Figure 54. Query in VisualInfo for AS/400 (Part 1 of 3)

```

    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;
        }
    }
}

```

Figure 54. Query in VisualInfo for AS/400 (Part 2 of 3)

```

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;

```

Figure 54. Query in VisualInfo for AS/400 (Part 3 of 3)

The complete sample application from which this application was taken (TExecuteV4.cpp) is available in the Cmbroot/Samples/cpp/v4 directory.

Executing a parametric query

The following example runs a parametric query.

```

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:

- Ability to search binders
- Ability to retrieve documents
- ODMA compliance so that users can work in familiar applications

Restriction: Domino.Doc does not support:

- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

With the Domino.Doc API, you build an expression to return an object. This section describes the design of the Domino.Doc API and how the objects fit into the hierarchy, and thus how to build the expression. Figure 55 on page 305 shows how the components of the object model are related.

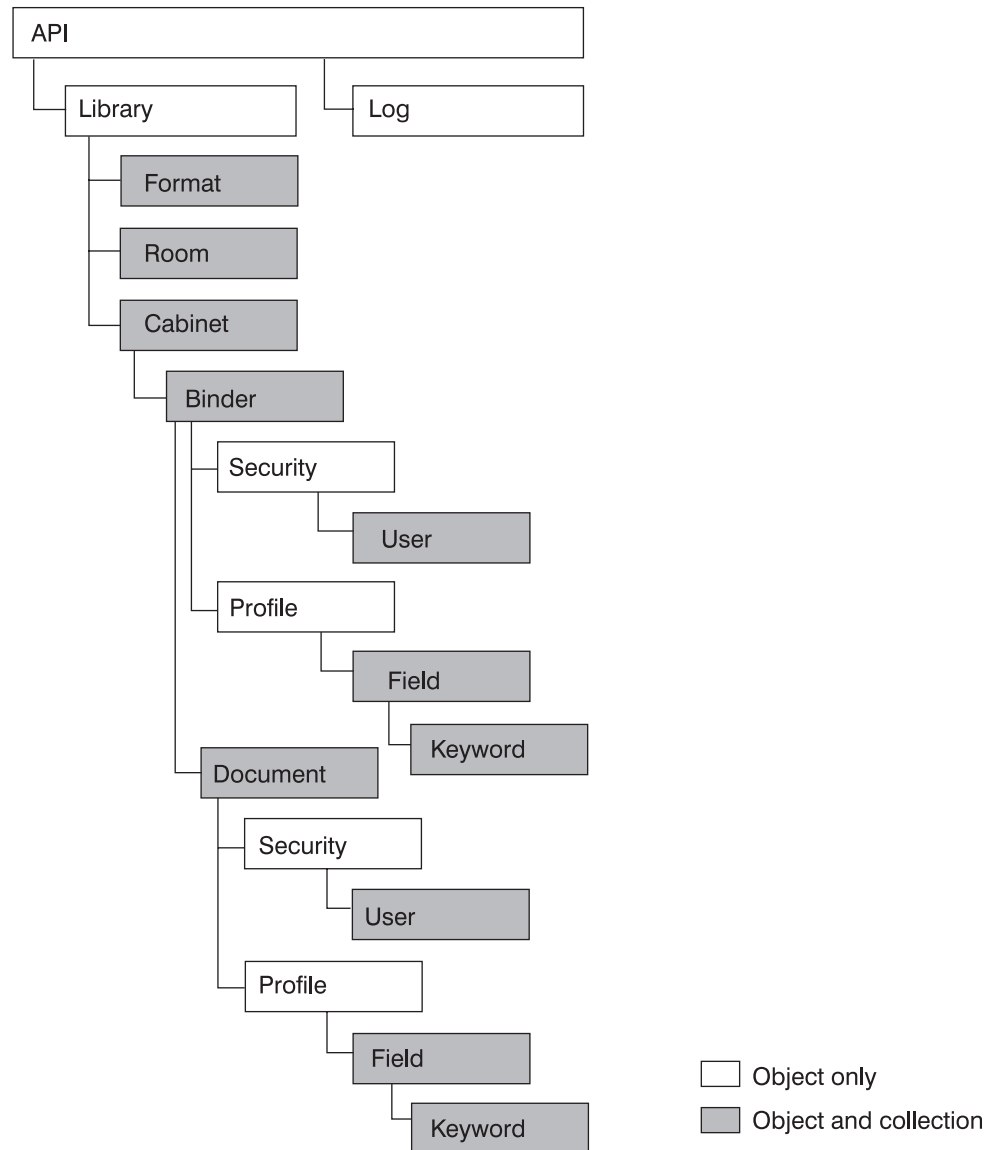


Figure 55. Domino.Doc object model

The elements contained in Domino.Doc are arranged so 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 subtentities

The following example lists the entities in Domino.Doc. In the example, rooms are the entities, and cabinets, binders and documents are subtentities.

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

```
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 following example lists the subentities (cabinets, binders, and documents) associated with an entity (in this case a room).

```
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. So, when DKDatastoreDD lists searchable entities it lists cabinets.

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".

```
"ENTITY=<"cabinetTitle"> QUERY=<"lotusQueryString">"
```

QUERY= is a required parameter.

In Domino.Doc a query string looks like this:

Use the FTSearch method to query the Domino.Doc content server. The Domino.Doc content server must be fully text indexed for this method to work efficiently. To test for an index, use the IsFTIndexed property. To create an index, use the UpdateFTIndex method.

The FTSearch method searches all of the documents in a content server—to search documents within a particular view, use the FTSearch method in NotesView; to search documents within a particular document collection, use the FTSearch method in NotesDocumentCollection.

If you don't 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 must be enclosed in apostrophes ('). Remember to use quotation marks (") if, 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:

1. near
2. sentence
3. 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:

1. =,
2. >,
3. >=, <>,
4. <=

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 Domino Extended Search (DES)

Domino Extended Search (DES) allows you to query and retrieve documents from:

- Lotus Notes databases
- NotesPump databases
- File systems
- Web search engines

This section provides instructions on:

- Listing DES servers
- Listing databases and fields
- Ability to use Generalized Query Language (GQL) to perform searches
- Ability to retrieve documents

Restriction: DES does not support:

- Adding, updating, and deleting documents
- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

All DES features are accessed and controlled by the DES 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 DES servers

To provide access to multiple DES servers, you can create a file named `cmbdes.ini` that contains the server information. Store this file in

`x:\CMBROOT`

(where *x* 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`

TCP/IP is the TCP/IP address of the DES server and the *port number* is the port number defined for accessing the server (for example: `PORT=80`).

Listing databases and fields

When you build a query to search a DES server, you must know the available database and field names. The `DKDatastoreDES` object provides the `listEntities`

method to list the databases and the `listEntityAttrs` method to list the fields for each database. The following example shows how to retrieve a list of databases and their fields.

```

...
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 << "    Attribute 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/des directory.

Using Generalized Query Language (GQL)

DES uses the Generalized Query Language (GQL) to perform searches. Table 23 contains examples of valid GQL expressions.

Table 23. 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

DES 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])]
```

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 results expected are 5.

```
String cmd = "SEARCH=(DATABASE=(Notes Help);" +
             "COND=(EQ \"View\" \"How Do I?\");" +
             "OPTION=(MAX_RESULTS=5)"
```

This example runs a GQL query for DES.

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore DES 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/des directory.

DDO properties in DES

A DDO in DES always has the type DK_CM_DOCUMENT. To get the item type of the DDO, you call:

```
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 DES

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 11 and “Creating a persistent identifier (PID)” on page 27.

Contents of a DES 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 DK_CM_COLLECTION or 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. DKPARTS object can store the uniform resource locator (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:

```
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);
        switch (a.typeCode())
        {
            case DKAny::tc_string :
                {
                    strData = a;
                    cout << "attribute name : " << strDataName << " value : " << strData << endl;
                }
                break;
            case DKAny::tc_long :
                {
                    long l = a;
                    cout << "attribute name : " << strDataName << " value : " << l << endl;
                }
                break;
            case DKAny::tc_double :
                {

```

```

        double db = a;
        cout << "attribute name : " << strDataName << " value : " << db << endl;
    }

    break;
    case DKAny::tc_timestamp :
    {
        DKTimestamp tt = a;
        cout << "attribute name : " << strDataName << " value : "
        << tt.getMonth() << "/" << tt.getDay() << "/" << tt.getYear() << " "
        << tt.getHours() << ":" << tt.getMinutes() << ":" << tt.getSeconds() << endl;
    }

    break;
    case DKAny::tc_dobase :
    {
        pDOBase = a;
        pXDO = (dkXDO*)pDOBase;
        cout << "attribute name : " << strDataName << " value : " << endl;
        pidXDO = (DKPidXDOES*)pXDO->getPid();
        cout << "XDO pid database name " << pidXDO->getDatabaseName() << endl;
        cout << "XDO pid docId " << pidXDO->getDocId() << endl;
        cout << "XDO mimetype " << pXDO->getMimeType() << endl;
        ((DKBlobDES*)pXDO)->getContentToClientFile("c:\\temp\\temp.html", 1);
    }

    break;

    case DKAny::tc_collection :
    {
        pdCol = a;
        cout << strDataName << " collection name : " << strDataName << endl;
        cout << "-----" << endl;
        pdIter = pdCol->createIterator();
        ushort b = 0;
        while (pdIter->more() == TRUE)
        {
            b++;
            cout << "-----" << endl;
            a = *(pdIter->next());
            switch (a.typeCode())
            {
                case DKAny::tc_string :
                {
                    strData = a;
                    cout << "attribute name : " << strDataName
                    << " value : " << strData << endl;
                }

                break;
                case DKAny::tc_long :
                {
                    long l = a;
                    cout << "attribute name : " << strDataName << " value : " << l << endl;
                }

                break;
                case DKAny::tc_double :
                {
                    double db = a;
                    cout << "attribute name : " << strDataName << " value : " << db << endl;
                }

                break;
                case DKAny::tc_timestamp :
                {
                    DKTimestamp tt = a;
                    cout << "attribute name : " << strDataName << " value : "
                    << tt.getMonth() << "/" << tt.getDay() << "/" << tt.getYear() << " "
                    << tt.getHours() << ":" << tt.getMinutes() << ":" << tt.getSeconds() << endl;
                }

                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/des 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:

```

DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore DES 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/des directory.

Retrieving a BLOB

To retrieve a BLOB from a DKDatastoreDES object, you must know the name of the database, the ID of the document that contains the BLOB, and the name of the field that contains the BLOB. You must also associate the DDO to a content server and establish a connection.

In the following example, the database named DES files contains an HTML file named D:\desdoc\README.html. The field that contains the HTML file is named Doc\$Content. The sample code retrieves the HTML file and saves it as D:\DESReadme.html.

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore DES 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("DES files");
pid.setFieldName("Doc$Content");
pid.setPrimaryId("DES 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/des directory.

Associating MIME types with documents

DES does not directly support 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 DES 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 DES

When you create federated queries, the syntax used in DES is similar to SQL syntax. The federated query expressions are converted to GQL syntax before they are submitted to DES. Because SQL and GQL grammar have differences however, only a subset of the SQL grammar is supported by Enterprise Information Portal.

Table 24 summarizes the SQL to GQL conversion of the supported comparison and logical operators.

Table 24. SQL and GQL operators

SQL operator	GQL operator
--------------	--------------

Table 24. SQL and GQL operators (continued)

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 custom content servers

In Enterprise Information Portal you have the option of creating your own server definitions for custom content servers. If you integrate a custom server into Enterprise Information Portal you must provide your own C++ classes to support the definition.

Developing custom content servers

For information about 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
- link to the `cmbcm716.dll`, non-debugged version, and `cmbcm716d.dll`, debugged version, files in order to access the common framework

Enterprise Information Portal database infrastructure: The primary interface between content servers and an Enterprise Information Portal is the `dkDatastore` classes. Each content server has a separate class that implements the `dkDatastore` class to provide its specific implementation information. Each content server type is represented by a class called `DKDatastorexx`, where `xx` identifies the name or type of the specific content server. Table 25 lists the content servers provided in Enterprise Information Portal.

Table 25. Enterprise Information Portal content servers

Server type	Class names
Content Manager	DKDatastoreDL
OnDemand	DKDatastoreOD
VisualInfo for AS/400	DKDatastoreV4
ImagePlus for OS/390	DKDatastoreIP
Domino.Doc	DKDatastoreDD
Domino Extended Search	DKDatastoreDES
DB2 Universal Database	DKDatastoreDB2
DB2 DataJoiner	DKDatastoreDJ
ODBC	DKDatastoreODBC

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 methods 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 datastore, or a document in OnDemand.

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 methods.

The dkXDO class extends the public interface of dkXDOBase by defining independent content server access, create, retrieve, update, and delete methods. These methods 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 PD 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 datastores, and mapped to notes for the OnDemand datastores.

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.

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 method, 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 datastore

Restriction : Some content servers do not support these methods.

registerServices(), unregisterServices()

Registers search engines

changePassword(userid, oldPasswd, newPasswd)

Changes the login password for the current user id on 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 method.

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 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 dkSchemMapping object

dkDatastoreDef

The dkDatastoreDef interface defines methods to access content server information and to create, list, and delete its entities. It maintains a collection of dkEntityDef objects.

Table 26 contains examples of concrete classes for the dkDatastoreDef interface.

Table 26. Concrete classes for dkDatastoreDef

Server type	Class name
Content Manager	DKDatastoreDefDL
OnDemand	DKDatastoreDefOD
VisualInfo for AS/400	DKDatastoreDefV4
ImagePlus for OS/390	DKDatastoreDefIP
Domino.Doc	DKDatastoreDefDD
Domino Extended Search	DKDatastoreDefDES
DB2 Universal Database	DKDatastoreDefDB2,
DB2 DataJoiner	DKDatastoreDefDJ
ODBC	DKDatastoreDefODBC

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 methods with names that are familiar to that content server. For example, the DKDatastoreDefDL class contains these specific methods:

- listIndexClassNames()
- listIndexClasses()

The DKDatastoreDefOD class contains these specific methods:

- listAppGrps()
- listAppGrpNames()

dkEntityDef

The dkEntityDef class defines methods to:

- Create and delete the entity
- Access entity information
- Create and delete attributes

In the dkEntityDef class, all methods 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 methods to overwrite the exceptions.

Table 27 contains examples of concrete classes for the dkEntityDef class.

Table 27. Concrete classes for dkEntityDef

Server type	Class name
Content Manager	DKIndexClassDefDL
OnDemand	DKAppGrpDefOD
VisualInfo for AS/400	DKIndexClassDefV4
ImagePlus for OS/390	DKEntityDefIP
Domino.Doc	DKCabinetDefDD
Domino Extended Search	DKDatabaseDefDES
DB2 Universal Database	DKTableDefDB2, DKColumnDefDB2
DB2 DataJoiner	DKTableDefDJ, DKColumnDefDJ
ODBC	DKTableDefODBC, DKColumnDefODBC

The main methods 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 methods for accessing attribute information and creating and deleting attributes.

Table 28 contains examples of concrete classes for the dkAttrDef class.

Table 28. Concrete classes for dkAttrDef

Server type	Class name
Content Manager	DKAttributeDefDL
OnDemand	DKFieldDefOD
VisualInfo for AS/400	DKAttrDefV4
ImagePlus for OS/390	DKAttrDefIP
Domino.Doc	DKAttrDefDD
Domino Extended Search	DKFieldDefDES
DB2 Universal Database	
DB2 DataJoiner	
ODBC	

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 29 contains examples of concrete classes for the dkServerDef class.

Table 29. Concrete classes for dkServerDef

Server type	Class name
Content Manager	DKServerDefDL
OnDemand	DKServerDefOD
VisualInfo for AS/400	DKServerDefV4
Domino.Doc	DKServerDefDD
Domino Extended Search	DKServerDefDES
DB2 Universal Database	DKServerDefDB2
DB2 DataJoiner	DKServerDefDJ
ODBC	DKServerDefODBC

The main methods 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()

Get 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 methods 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

fetchNextN(int how_many, dkCollection collection)
Retrieves as the next *n* elements of the result set and inserts them into the specified collection

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 datastore

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 datastore type to which the cursor belongs

handle(int type)
Gets the resultset handle that is associated with the resultset cursor, by type

Requirement: To use the `addObject`, `deleteObject`, and `updateObject` methods, you must set the datastore option `DK_OPT_ACCESS_MODE` to `DK_READWRITE`.

dkBlob

`dkBlob` is an abstract class that declares a common public interface for binary large object (BLOB) data types.

Table 30 contains examples of concrete classes for the `dkBlob` class.

Table 30. Concrete classes for dkBlob

Server type	Class name
Content Manager	DKBlobDL
OnDemand	DKBlobOD
VisualInfo for AS/400	DKBlobV4
ImagePlus for OS/390	DKBlobIP
Domino.Doc	DKBlobDD
Domino Extended Search	DKBlobDES
DB2 Universal Database	DKBlobDB2
DB2 DataJoiner	DKBlobDJ
ODBC	DKBlobODBC

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 *startPos*, 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 method 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 31 contains examples of concrete classes for the dkClob class.

Table 31. Concrete classes for dkClob

Server type	Class name
DB2 Universal Database	DKClobDB2
DB2 DataJoiner	DKClobDJ
ODBC	DKClobODBC

The main methods 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:

Data access

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 *startPos*, 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 method 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 `DKBlobxx` class, where the `dkBlob` object is the representation of the binary annotation data and the `DKParts` collection.

dkDatastoreExt

The `dkDatastoreExt` class defines the standard datastore extension classes.

Table 32 contains examples of concrete classes for the `dkDatastoreExt` class.

Table 32. Concrete classes for dkDatastoreExt

Server type	Class name
Content Manager	DKDatastoreExtDL
OnDemand	DKDatastoreExtOD
VisualInfo for AS/400	DKDatastoreExtV4
ImagePlus for OS/390	DKDatastoreExtIP
Domino.Doc	DKDatastoreExtDD
Domino Extended Search	DKDatastoreExtDES
DB2 Universal Database	
DB2 DataJoiner	
ODBC	

The main methods 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 method.

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 33 contains examples of concrete classes for the DKPidXDO class.

Table 33. Concrete classes for DKPidXDO

Server type	Class name
Content Manager	DKPidXDODL
OnDemand	DKPidXDOOD
VisualInfo for AS/400	DKPidXDOV4
ImagePlus for OS/390	DKPidXDOIP
Domino.Doc	DKPidXDODD
Domino Extended Search	DKPidXDODES
DB2 Universal Database	DKPidXDODB2
DB2 DataJoiner	DKPidXDODJ
ODBC	DKPidXDOODBC

dkUserManagement

The dkUserManagement class represents and processes all of the content server's user management methods.

Table 34 contains examples of concrete classes for the dkUserManagement class.

Table 34. Concrete classes for dkUserManagement

Server type	Class name
Content Manager	DKUserMgmtDL
VisualInfo for AS/400	DKUserMgmtV4
ImagePlus for OS/390	DKUserMgmtIP
DB2 Universal Database	
DB2 DataJoiner	
ODBC	

DKConstant

All common constants are defined in the DKConstant class. Each content server has its own DKConstantxx 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 DKMessageIdxx class for defining its own message IDs.

Recommendation: All content servers use the common messages whenever possible.

The following property files contain two common warning and error messages:

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.

Figure 56 is an example of the datastore structure used in Enterprise Information Portal, using the OnDemand content server as a model.

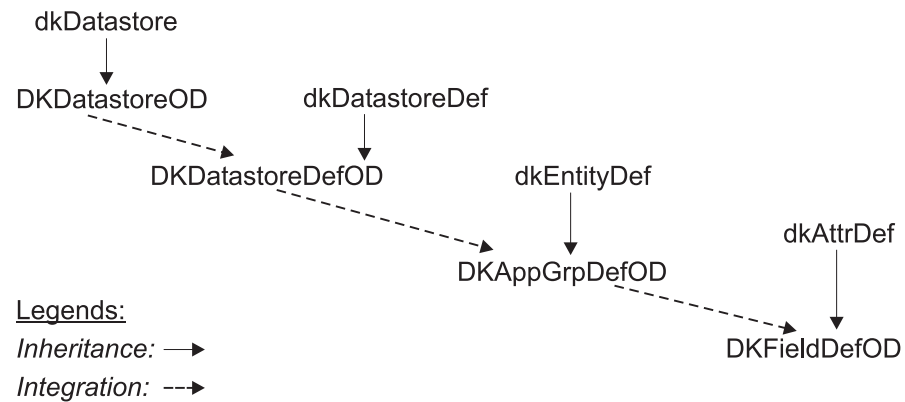


Figure 56. Datastore structure example

Figure 57 on page 330 shows an example of the data structure used in Enterprise Information Portal, using the OnDemand content server as a model.

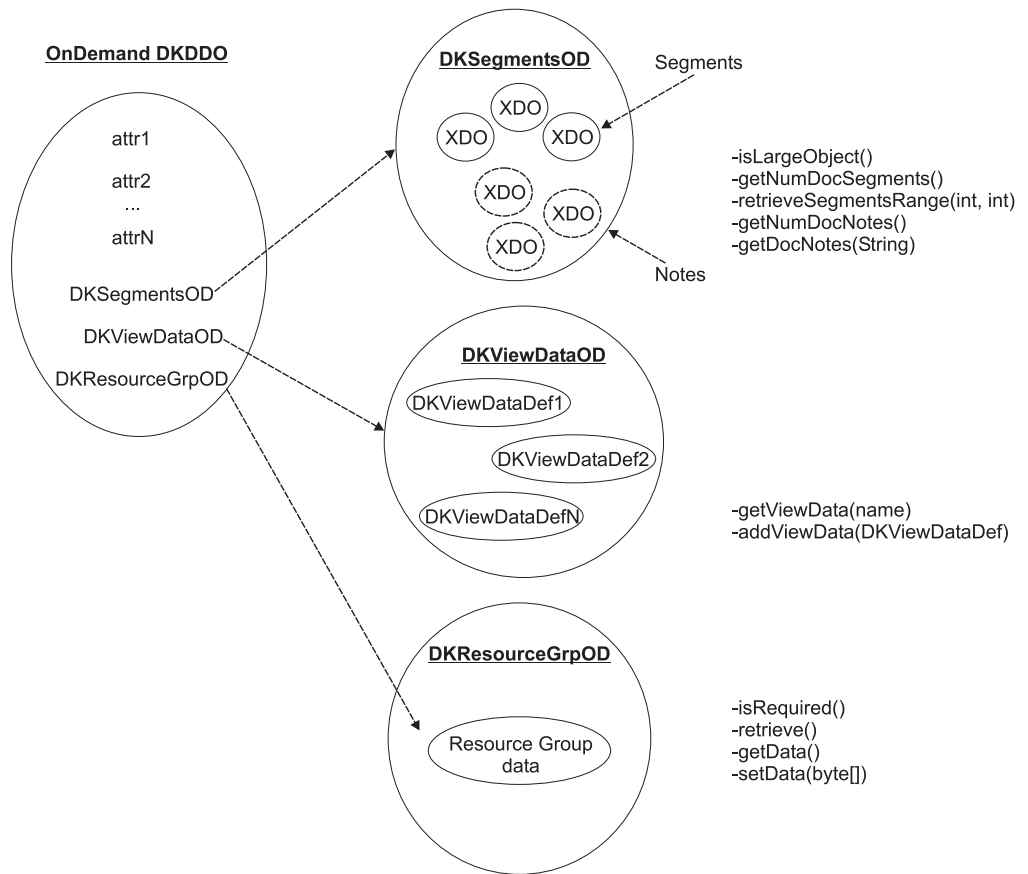


Figure 57. Data structure example

Chapter 9. Using the ActiveX (OLE) application programming interface

The ActiveX application programming interfaces (API) are a set of classes that access and manipulate locally or remotely stored data.

This section describes the design of the ActiveX API and the ActiveX implementation of multiple search facilities. Using any macro language or programming tool that supports automation (for example, VBA, Lotus script, and so forth) you create custom applications for searching and updating across multiple content servers.

The ActiveX API supports:

- Multiple search and update across a heterogeneous combination of content servers
- A common object model for data access
- A flexible mechanism for using a combination of search engines; for example, Text Search Engine and query by image content (QBIC).

In addition to the code segments described in this chapter, three sets of samples are provided on the Enterprise Information Portal CD-ROM, one for each type of query. Visual Basic forms are created with Visual Basic Version 6. The forms incorporate relatively simple user interfaces and are intended as general guidelines only. You are encouraged to explore and use some of the more sophisticated features of Visual Basic, such as the tree and image list controls, or Lotus Approach's Forms and Picture Plus fields for displaying data.

See the examples in the `samples` directory, for information on how to use each sample.

The ActiveX classes are automation servers that support the COM IDispatch interface. A type library is shipped with the .DLL so that a programming tool can determine which interfaces an object supports, as well as the names of its members. Because automation servers are not meant to be subclassed, the ActiveX classes represent the leaf classes in the C++ class hierarchy and include all of the methods of their C++ counterparts, plus those of their parents.

For methods that return non-leaf class objects, the return type is either replaced by the concrete equivalent or by the type object. Visual Basic users can use the built-in `TypeOf` method to determine the real type. Two APIs will be described in subsequent sections, which use the same samples as those in the sections but translated into Visual Basic. In general, the differences between the C++ and ActiveX API are:

- Operators are either replaced by functions, or in cases where the functionality can be replaced by one or more existing functions, eliminated entirely.
- Because the data types must allow automation, methods that take a `DKAny` object as argument take a `VARIANT` object instead. This allows the methods to perform run-time type checking to ensure that the data passed is the correct type.

Running in client/server mode

The ActiveX APIs can be configured to run in a client/server mode. ActiveX APIs rely on DCOM as the underlying mechanism for accessing remote objects on the network. DCOM requires Windows NT 4.0 Service Pack 2 (or higher) which provides support for launching DLL-based objects remotely using a surrogate.

All of the server components are installed by default. To enable the ActiveX classes to operate in the client/server mode, you must update the registry using regedit, DCOMCnfg or OLEView. DCOMCnfg and regedit come with the operating system. OLEView is supplied with Visual C++ and the Win32 Software Developer's Kit (SDK). You can also download OLEView from the Microsoft Web site

Updating the registry using regedit or DCOMCnfg

1. Ensure that HKEY_LOCAL_MACHINE\Software\Microsoft\OLE has the value Y. This value globally affects whether any remote clients can launch or connect to already running objects. The registry entries relevant to remote objects are found under AppID in HKEY_CLASSES_ROOT.
2. For servers, the AppID entry for a content server should have a value named DllSurrogate added with an empty string value.
3. For servers: Add a value named RunAs which can be either a local or domain account, or the string Interactive User, to specify that the account of the logged-on user should be used. If RunAs is missing, the COM service control manager uses Launching User as the string value, that is, the user that has requested the object. However, this only works when the client allows it with the requested impersonation level.
4. For clients, add a value named RemoteServerName with a string value that is the name of the remote machine on which the server resides. Also,
5. For clients to function as a client, remove the following registry entry:
HKEY_CLASSES_ROOT\CLSID\{clsid}\InProcServer32

Updating the registry using OLEView

If you are using OLEView to change your registry, complete the following steps:

1. Click **Expert mode** and expand the node on **All Objects**.
All the automation classes appear as content server classes.
2. Select a class and click the Implementation tab. On the **Implementation** panel, select **use surrogate process**.
3. The Activation page can be used to specify the remote machine name (this is equivalent to the RemoteServerName registry entry) and the activation options Interactive user and Launching user.

To access an object on the server, the launch and activate permissions on the server must be set to allow access by a client account. This can be done using DCOMCnfg or OLEView. Set the authentication level to something other than none. The impersonation level can be set to either identify or impersonate (the default values are connect and identify.)

Setting up the Windows environment

When you set up your Windows or AIX environment, you must establish the settings described in this section.

DLLs for Windows

- cmbxfed716.dll
- cmbxdl716.dll
- cmbxdes716.dll
- cmbxdd716.dll
- cmbxip716.dll
- cmbxv4716.dll

Miscellaneous files

- cmbxfed716.tlb
- cmbxdl716.tlb
- cmbxdes716.tlb
- cmbxdd716.tlb
- cmbxip716.tlb
- cmbxv4716.tlb

Setting Windows environment variables

Set the following environment variables:

PATH

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

where *x* is your drive

INCLUDE

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

where *x* is your drive

Using DXInstallDL, DXInstallDES, and DXInstallFed, DXInstallDD, DXInstallIP, and DXInstallV4

The ActiveX API is installed as part of the object-oriented toolkit. You must run the following files to register the classes before you can use the automation classes:

```
DXInstallDL.exe  
DXInstallDES.exe  
DXInstallFed.exe  
DXInstallDD.exe  
DXInstallIP.exe  
DXInstallV4.exe
```

Registering classes

To register classes with the Windows registry:

1. Open a command prompt and go to the directory that contains Enterprise Information Portal (for example, C:\CMBROOT).
2. To run the appropriate executable file enter one of the following three commands:

DXInstallDL
DXInstallDES
DXInstallFed

3. You get a message indicating the success or failure of the process.
4. If an error occurs, go to the log directory and locate the DXInstallDL.log, DXInstallDES.log, or DXInstallFed.log file; in it you will find information regarding which classes have failed to register. You must manually edit those entries in the registry.

Removing registration

To remove the classes registration during the uninstall process, you must run an additional file.

1. Locate the files named DXUninstallDL.exe, DXUninstallDES.exe, and DXUninstallFed.exe
2. Follow the same procedure as before, and check the appropriate log file if you encounter any errors.

Requirement: Make sure that the FRNADDRON environment variable in Visual Basic is set to yes before you use the APIs.

Multiple search facilities

Use the multiple search facilities to:

- Search within a given content server, using one or a combination of supported query types:

Parametric query

Query requiring an exact match between the condition specified in the query and stored data.

Text query

Query requiring an approximate match between the given query and stored text.

Each search type is supported by one or more search engines.

- Search the results of previous search.

Not all content servers support multiple search facilities. For more information about specific content servers and multiple search see "Using specific content servers" on page 258.

Connecting to content servers

A DXDatastore xx (where xx is the suffix representing the specific content server, for example, Content Manager (DL), Domino Extended Search (DES), and so forth) represents and manages a connection to a content server, provides transaction support, and runs server commands.

Establishing a connection

In a typical example, you create and connect to a content server, work with it and then disconnect. The following example shows how to connect to a Content Manager library server named LIBSRVRN, using the user ID FRNADMIN and password PASSWORD.

```

Dim dsDL As New DXDatastoreDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
dsDL.disconnect

```

Setting and getting datastore options

The content server provides some options that you can set or get using its methods. The following example shows how to set and get the option to establish an administrative session. The online API reference lists and describes the options for each content server.

```

Dim session_type As Long
Dim outType As Variant
Dim dsDL As New DXDatastoreDL
dsDL.setOption DX_DL_OPT_ACCESS, DX_DL_SS_CONFIG
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
dsDL.getOption DX_DL_OPT_ACCESS, outType
session_type = outType

If session_type = DX_DL_SS_CONFIG Then
    MsgBox "Datastore access is an administrative session"
End If

dsDL.disconnect

```

Listing servers

The content server can connect to different library servers. The content server provides a method to list the library servers it can connect to. **Restriction:** The Domino.DOC content server does not provide such a method.

The following section lists the three steps to retrieve the list of library servers and shows a code sample. The steps are:

1. Create a DXDatastoreDL
2. List the Content Manager servers
3. Create an iterator for the collection and loop through the DXServerDefDL objects to get the server name and type

```

Dim dsDL As New DXDatastoreDL
Dim col As DXSequentialCollectionDL
Set col = dsDL.listDataSources
Dim serverDef As DXServerDefDL
Dim iter As DXSequentialIteratorDL
Set iter = col.createIterator
Dim i As Long
i = 0
Do While iter.more
    i = i + 1
    Set serverDef = iter.Next
    MsgBox "Server Name [" & i & "] - " & serverDef.getName
        & " Server Type - " & serverDef.getServerType
Loop

```

Listing schema and schema attributes

A content server provides methods for listing its schema. In a content server, these methods list index classes and their attributes. The following example shows how to retrieve the list of index classes, as well as the list of attributes. The steps are:

1. Create a DXDatastoreDL
2. Connect to the Content Manager server LIBSRVRN using the user ID FRNADMIN and the password PASSWORD
3. List the attributes for each index class

4. Disconnect from the Content Manager server

```
Dim dsDL As New DXDatastoreDL
Dim col As DXSequentialCollectionDL
Dim col2 As DXSequentialCollectionDL
Dim serverDef As DXServerDefDL
Dim iter As DXSequentialIteratorDL
Dim iter2 As DXSequentialIteratorDL
Dim entDef As DXIndexClassDefDL
Dim strEntity As String
Dim attrDef As DXAttrDefDL
Dim i As Long
Dim j As Long
dsDL.Connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Set col = dsDL.listEntities
Set iter = col.createIterator
i = 0
Do While iter.more
    i = i + 1
    Set entDef = iter.Next
    MsgBox "Entity Name [" & i & "] - " & entDef.getName
    strEntity = entDef.getName
    Set col2 = dsDL.listEntityAttrs(strEntity)
    Set iter2 = col2.createIterator
    j = 0
    Do While iter2.more
        j = j + 1
        Set attrDef = iter2.Next
        MsgBox " Attribute Name [" & j & "] - " & attrDef.getName
    Loop
Loop
dsDL.disconnect
```

Using DDOs

DXDDO can be regarded as an attribute container. An attribute has a name, value, and properties. Each attribute is identified by a data ID.

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

You must associate a DXDDO with a content server before you can call the add, retrieve, update and delete methods to send its attributes into the content server and retrieve them. You do this by calling the proper DXDDO constructor or by calling setDatastore method.

Every DXDDO has a persistent identifier (PID) that contains information for locating the attributes in the datastore. For example, in Content Manager, a DDO represents an item, which could be a document or a folder.

Creating a persistent identifier (PID)

Each DDO requires a persistent identifier (PID). The PID contains information about the content server name, content server type, ID, and object type. The PID provides the location in the content server of the DDO's persistent data. For example, in a Content Manager datastore, this PID is the item ID. The item ID is

one of the most important parameters for the retrieve, update, and delete methods. For the add method, the item ID is created and returned by the content server.

To create a DDO to retrieve a known item:

```
Dim dsDL as new DXDatastoreDL      'create a Content Manager datastore
Dim ddo as new DXDDODL
ddo.setObjectType "GRANDPA"        'set the index class name it belongs to
ddo.setPid "LN#U5K6ARLGM3DB4"     'set the item ID
ddo.setDatastore dsDL              'associate ddo with dsDL
```

Then connect to the content sever and call the retrieve method to retrieve this DDO.

Adding data items and properties

Suppose the index class GRANDPA has the attributes shown in Table 35.

Table 35. Grandpa attributes

Attribute	data_id=1	2
Name	Title	Subject
Type	String	String
Nullable	No	Yes

You can represent the information above in a DXDDO as follows:

```
Dim dsDL as new DXDatastoreDL      'create a Content Manager datastore
Dim cddo as new DXDDODL            'create a DDO to hold an object type
cddo.setObjectType "GRANDPA"       'set the index class name it belongs to
cddo.setDatastore dsDL             'associate ddo with dsDL
Dim data_id as Integer

'Add the first attribute
data_id = cddo.addData("Title")    'add a new attribute named "Title"

'Add a property named: DX_DL_PROPERTY_TYPE, set to value : variable length string
cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_TYPE, DX_VString

'Add a property named: DX_DL_PROPERTY_NULLABLE, set to value : boolean false
cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_NULLABLE, False

'Add the second attribute
data_id = cddo.addData("Subject")  'add a new attribute named "Subject"

'Add a property named: DX_DL_PROPERTY_TYPE, set to value : variable length string
cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_TYPE, DX_VString

'Add a property named: DX_DL_PROPERTY_NULLABLE, set to value : boolean true
cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_NULLABLE, True
```

The above example illustrates the standard properties that an attribute should have, namely DX_DL_PROPERTY_TYPE and DX_DL_PROPERTY_NULLABLE. You can have as many additional properties as required by your application.

Adding properties to a DDO

The DXDDO has all the required attribute information. However, there is no information to indicate if the DXDDO is either a document or a folder. The following example sets the DXDDO property to indicate that the DXDDO is a document:

```
cddo.addProperty DX_DL_PROPERTY_ITEM_TYPE, DX_DL_DOCUMENT 'it is a document
```

Setting and getting data item values

After you create a data item and its properties, you can set its value:

```
'Set Title value to the given string assume we know the data_id for  
'the data_item "Title" is 1  
cddo.setData 1, "One dark and stormy night"
```

'Set Subject value to the given string assume we do not know the data_id

```
'for the data_item "Subject"  
data_id = cddo.dataId("Subject") 'find data_id for data_item named "Subject"  
cddo.setData data_id, "Mystery"
```

Use the `getData` method to retrieve the value:

```
Dim data as String  
data = cddo.getData(1)  
MsgBox "Title = " & data 'displays "One dark and stormy night"  
MsgBox "Subject = " & cddo.getData(data_id) 'displays "Mystery"
```

Getting the properties

When processing a DXDDO, the first thing you want to know is its type: document or folder. The following code determines the DXDDO type:

```
Dim prop_id as Integer  
Dim myType as Integer  
prop_id = cddo.propertyId(DX_DL_PROPERTY_ITEM_TYPE)  
If prop_id > 0 Then  
    myType = cddo.getProperty(prop_id)  
    Select Case myType  
        case DX_DL_DOCUMENT  
            'process document  
        ...  
        case DX_DL_FOLDER  
            'process folder  
        ...  
    End Select  
End If
```

To retrieve attribute properties, you must have the attribute `data_id`:

```
data_id = cddo.dataId("Title") 'get data_id of Title  
'How many prop does it have  
Dim number_of_data_prop as Integer  
number_of_data_prop = cddo.dataPropertyCount(data_id)  
'Displays all data properties belonging to this attribute  
'Notice that the loop index starts from 1, 1 <= i <= number_of_data_prop  
Dim i as Integer  
For i = 1 To number_of_data_prop  
    MsgBox Str(i) & " Property Name = "  
        & cddo.getDataPropertyName(data_id, i) & "  
        " value = " & cddo.getDataProperty(data_id, i)  
Next
```

Important: Both `data_id` and `property_id` start from 1; if you specify 0, you receive an exception.

Displaying the DDO

During application development, you might need to display the content of a DXDDO for debugging purpose. For example:

```

Dim number_of_attribute as Integer
number_of_attribute = cddo.dataCount
Dim number_of_prop as Integer, k as Integer, j as Integer
Dim number_of_data_prop as Integer, i as Integer
number_of_prop = cddo.propertyCount
'List DDO properties
For k = 1 To number_of_prop
    MsgBox Str(k) & " Property Name =
        " & cddo.getPropertyName(k) & ", value =
        " & cddo.getProperty(k)
Next
'List data items and their properties
For i = 1 To number_of_attribute
    MsgBox Str(i) & " Attr. Name =
        " & cddo.getDataName(i) & ", value =
        " & _cddo.getData(i)
    number_of_data_prop = cddo.dataPropertyCount(i)
    For j = 1 To number_of_data_prop
        MsgBox Str(j) & " Data Prop. Name =
            " & cddo.getDataPropertyName(i, j) & ", value =
            " & cddo.getDataProperty(i, j)
    Next
Next

```

Deleting a DDO

If you delete a DXDDO by calling its destructor, the DDO is deleted in memory, but the persistent copy in the content server is unchanged. Conversely, the `del` method in DXDDO deletes the persistent copy in the content server, with its representation in memory remaining unchanged.

Using XDOs

An XDO represents a single part in Enterprise Information Portal. There is one type of XDO for binary objects called DXBlobxx (where xx is the suffix representing the specific server, for example, Content Manager (DL), Domino Extended Search (DES), and so forth). DXBlobxx requires the datastore DXDatastorexx as an input to create the object instance.

Using an XDO PID

An XDO needs to have a PID in order to store its data persistently. The item ID and part ID of DXPidXDOxx are required for XDO to locate the persistent data in a datastore.

Understanding XDO data members

You must set the values for XDO `RepType`, `ContentClass`, `AffiliatedType`, `AffiliatedData`, `SearchEngine`, `SearchIndex`, and `SearchInfo`. If you do not set these values, the XDO content is indexed by a search engine and the following default values are used:

RepType

DX_DL_DK_REP_NULL

Attention: The only representation type (or `RepType`) supported by `VisualInfo` for AS/400 is " ", eight blank spaces surrounded by leading and trailing quotation marks.

ContentClass

DX_DL_CC_UNKNOWN

AffiliatedType
DX_DL_BASE

AffiliatedData
NULL

Tip: For the valid values of ContentClass, see the enumerated constant CONTENT_CLASS provided with Content Manager.

Using XDO in a datastore

To retrieve, update or delete a content server object, provide the correct item ID, part ID, and repType to identify the object.

```
Dim partId as Integer
partId = 17                                'partId of object
Dim itemId as String, fileName as String
itemId = "CPPIORH4JBIXWIY0"              'existing itemId
fileName = "g:\\test\\choice.gif"         'file to be updated
Dim dsDL as new DXDatastoreDL              'required datastore
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD" 'connection to datastore
Dim axdo as new DXBlobDL
axdo.init dsDL                             'create XDO
axdo.setPartId partId                       'set partId
axdo.setPid itemId                          'set itemId
axdo.retrieve                               'retrieve the object
axdo.setContentFromClientFile fileName      'set file content to buffer area
axdo.update                                 'update the object with buffer data
axdo.retrieve "new.gif"                    'retrieve content to a file
axdo.del                                    'delete object from datastore
dsDL.disconnect                             'disconnect from datastore
```

Programming tips

You identify an XDO by the combination of item ID, part ID, and RepType. 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 method, if you set part ID to 0, the system assigns an available part ID for it. 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.

Using XDO as a part of DDO instead of stand-alone XDO

An XDO represents a single part object when a DDO is a document that is a collection of part objects. You can manipulate the XDO as a component of the DDO or as a stand-alone object. To handle as a part of the DDO, you must get the item ID for the XDO from the DDO. To handle it as a stand-alone object, you must know the existing item ID for the XDO.

XDO as part of DDO

The major statements used to relate the XDO with the DDO are:

```
'Create DDO
Dim ddo as new DXDDODL
ddo.setObjectType indexClassName
ddo.setDatastore dsDL
ddo.addPropertyAndValue DX_DL_PROPERTY_ITEM_TYPE, DX_DL_DOCUMENT
...
...
```



```

Dim parts as new DXPartsDL
'Create XDO
Dim axdo as new DXBlobDL
axdo.init dsDL
axdo.setPartId partId 'set partId
axdo.setContentClass DX_DL_CC_GIF
axdo.setAffiliatedType DX_DL_BASE
axdo.setContentFromClientFile imageNames(i)
'Add XDO to the DXPartsDL collection
parts.addElement axdo
...
...
'Add DDO
dataId = ddo.addData(DX_DL_DKPARTS)
ddo.addDataPropertyAndValue dataId, DX_DL_PROPERTY_TYPE, DX_Collection_XDO
ddo.setData dataId, parts
ddo.add

```

The complete sample application from which this example was taken (wfs.frm, written in Visual Basic) is available in the Cmbroot/Samples/activex/dl directory.

Stand-alone XDO

The following code examples are specific to Content Manager for a stand-alone XDO. For relational databases, please 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 the buffer. To use this sample, you must know the existing item ID of the XDO.

```

Dim dsDL as new DXDatastoreDL
Dim itemId as String, fileName as String
Dim partId as Integer
partId = 37 'partId 37 not being used yet
itemId = "CPPIORH4JBIXWIY0" 'existing itemId
fileName = "g:\test\cheetah.gif" 'file to be added
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD" 'connection to datastore
Dim axdo as new DXBlobDL
axdo.init dsDL 'create XDO
axdo.setPartId partId 'set partId
axdo.setPid itemId 'set itemId
axdo.setContentClass DX_DL_CC_GIF 'set ContentClass
axdo.setContentFromClientFile fileName 'set file content to buffer area
axdo.add 'add from the buffer
MsgBox "after add partId=" & axdo.getPartId
dsDL.disconnect 'disconnect from datastore

```

Adding an XDO from a file: This example adds an XDO from a file. To add an XDO from a file, add the following to the code from the previous example.

```

axdo.setContentFromClientFile fileName
axdo.add
axdo.add fileName

```

Adding an annotation object to an XDO: To add an annotation object, you must insert the following statements in your program before the add method.

```

axdo.setAffiliatedType DX_Annotation
axdo.setAffiliatedData 14, 1, 5, 5

```

Retrieving, updating and deleting an XDO: To retrieve, update, or delete an XDO in a content server, provide the correct item ID, part ID, and RepType to identify the object.

```

Private Sub cout(str As String)
Results.Text = Results.Text & Chr$(13) & Chr$(10) & str
End Sub

Private Sub Add_Click()

'following itemId should be already exist
'modify it with your own known itemId
'itemId = "POL025$BQVH$TZBS"
itemId = "FPN3ZP$MPZI@U0C0"
partId = 95
Set dsDL = New DXDatastoreDL
Set xdo = New DXBlobDL
xdo.init dsDL
xdo.setPartId partId
xdo.setPid itemId

'media content class is 209
xdo.setContentClass DX_CC_IBMVSS
cout "Do new DXMediaStreamInfoDL.."
Dim aVS As New DXMediaStreamInfoDL
aVS.setMediaFullFileName "/icing.mpg1"
aVS.setMediaObjectOption DX_VS_SingleObject
aVS.setMediaHostName "<insert hostname here>"
aVS.setMediaUserId "<insert user ID here>"
aVS.setMediaPassword "<insert password here>"
aVS.setMediaNumberOfUsers 1
aVS.setMediaAssetGroup "AG"
aVS.setMediaType "MPEG1"
aVS.setMediaResolution "SIF"
aVS.setMediaStandard "NTSC"
aVS.setMediaFormat "SYSTEM"
cout "set mediaStream extension obj..."
xdo.setExtension aVS
On Error GoTo Errors
cout "about to add media object..."
xdo.Add
On Error GoTo Errors
cout "add successfully..."
Add.Enabled = False
Exit Sub

Errors:
cout "Errors:" & str(Err.Number) & Err.Description

End Sub

Private Sub Retrieve_Click()

cout "itemId=" & xdo.getPid
cout "partId=" & xdo.getPartId
cout " "
Dim state As Integer
Dim mflag As Boolean
mflag = xdo.isCategoryOf(DX_Media_Object)
On Error GoTo Errors
cout "isCategoryOf media obj = " & mflag
If (mflag) Then
cout "Get mediaStream extension obj..."
Dim aVS2 As Variant
Set aVS2 = xdo.getExtension("DXMediaStreamInfoDL")
On Error GoTo Errors
cout " aVS2.getMediaCopyRate=" & aVS2.getMediaCopyRate
cout " aVS2.getMediaInvalidCommands=" & aVS2.getMediaInvalidCommands
cout " aVS2.getMediaDurSeconds=" & aVS2.getMediaDurSeconds
cout " aVS2.getMediaDurFrames=" & aVS2.getMediaDurFrames
cout " aVS2.getMediaFrameRate=" & aVS2.getMediaFrameRate

```

```

        cout " aVS2.getMediaBitRate=" & aVS2.getMediaBitRate
        cout " aVS2.getMediaNumberOfUsers=" & aVS2.getMediaNumberOfUsers
        cout " aVS2.getMediaAssetGroup=" & aVS2.getMediaAssetGroup
        cout " aVS2.getMediaType=" & aVS2.getMediaType
        cout " aVS2.getMediaResolution=" & aVS2.getMediaResolution
        cout " aVS2.getMediaStandard=" & aVS2.getMediaStandard
        cout " aVS2.getMediaFormat=" & aVS2.getMediaFormat
        cout " aVS2.getMediaState=" & aVS2.getMediaState
        state = xdo.retrieveObjectState(DX_Media_Object)
        cout "xdo.retrieveObjectState(media) =" & state
    On Error GoTo Errors
End If

    cout "===before retrieve==="
    cout " getLength=" & xdo.getLength
    cout " getSize=" & xdo.getSize
    cout " updatedTimestamp=" & xdo.getUpdatedTimestamp
    cout " createdTimestamp=" & xdo.getCreatedTimestamp
    cout " mimeType=" & xdo.mimeType
    cout "===about to retrieve as filedx.ivs ==="
    xdo.Retrieve ("filedx.ivs")
    On Error GoTo Errors
    cout " retrieve successfully"
    cout "===after retrieve==="
    cout " getLength=" & xdo.getLength
    cout " getSize=" & xdo.getSize
    cout " getContentClass=" & xdo.getContentClass
Exit Sub

Errors:
    cout "Errors:" & str(Err.Number) & Err.Description

End Sub

Private Sub Delete_Click()

    cout "itemId=" & xdo.getPid
    cout "partId=" & xdo.getPartId
    Dim mflag As Boolean
    mflag = xdo.isCategoryOf(DX_Media_Object)
    cout "isCategoryOf media obj = " & mflag
    If (mflag) Then
        cout "setting delete Option..."
        xdo.setOption DX_Opt_DL_Delete_Option, DX_Delete_NoDropitemMediaAvail
        Dim ln As Long
        xdo.getOption DX_Opt_DL_Delete_Option, ln
        cout "getOption deleteOpt=" & ln
    End If
    cout "===about to call delete==="
    xdo.del
    cout " delete successfully"
    cout "===after delete==="
    mflag = xdo.isCategoryOf(DX_Media_Object)
    cout " isCategoryOf media obj = " & mflag
    Add.Enabled = True
    Retrieve.Enabled = False
    Delete.Enabled = False
Exit Sub

Errors:
    cout "Errors:" & str(Err.Number) & Err.Description

End Sub

```

Creating and using the DX_DL_DKPARTS attribute

The DX_DL_DKPARTS attribute in a DDO represents the collection of parts in a document. The value of this attribute is a DXParts object, which is a collection of XDOs. You set the DKPARTS attribute when you retrieve a DDO, as shown below:

```
Dim dsDL as new DXDatastoreDL
Dim parts as new DXPartsDL      'create a new DXPartsDL , collection of parts
Dim blob as new DXBlobDL      'create a new XDO blob
blob.init dsDL
blob.setPartId 5                'set part number to 5
blob.setPid "LN#U5K6ARLGM3DB4" 'the item ID this part belongs to
blob.setContentClass DX_DL_CC_GIF 'set content class type GIF
blob.setRepType DX_DL_DK_REP_NULL 'set rep type for the part
blob.setContentFromClientFile "choice.gif" 'set the blob's content
blob.setInstanceOpenHandler "Netscape", True 'the viewer program
parts.addElement blob          'add the blob to the parts collection
... 'create and add some more blobs to the collection as necessary
Dim ddo as new DXDDO           'create a ddo
... 'sets some of its attributes
ddo.addProperty DX_DL_PROPERTY_ITEM_TYPE, DX_DL_DOCUMENT
'Create DX_DL_DKPARTS attribute and sets it to refer to the DXPartsDL object
Dim data_id as Integer
data_id = ddo.addData DX_DL_DKPARTS 'add attribute DX_DL_DKPARTS
ddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_TYPE, DX_Collection_XDO
ddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_NULLABLE, True
ddo.setData data_id, parts      'sets the attribute value
```

After you set a DXParts as an attribute value of a DDO, the DDO owns it and will take care of deleting it:

```
data_id = ddo.dataId(DX_DL_DKPARTS) 'get DX_DL_DKPARTS data-id
If data_id = 0 Then
    MsgBox " parts data item not found"
End If
Dim col as DXPartsDL
Set col = ddo.getData(data_id) 'get the parts collection
'Create iterator and process the part collection member one by one
If col Is Nothing Then
Else
    Dim iter as DXSequentialIterator
    Dim blob as DXBlobDL
    Set iter = col.createIterator
    Do While iter.more
        Set blob = iter.next
        If blob Is Nothing Then
        Else
            blob.open                'display the blob using the viewer
            ... 'other processing
        End If
    Loop
End If
```

Creating and using the DX_DL_DKFOLDER attribute

In a folder DDO, the DX_DL_DKFOLDER attribute represents a collection of folders and documents belonging to this folder. For example, a DX_DL_DKFOLDER attribute represents a collection of folders and documents in Content Manager. The value of this attribute is a DXFolder object, a collection of DDOs. Similar to DX_DL_DKPARTS, DX_DL_DKFOLDER is set during DDO retrieve, or it can be created and set by you, as shown below:

```
Dim dsDL as new DXDatastoreDL
Dim folder as new DXFolderDL    'create a new DXFolderDL, collection of DDO
Dim member as new DXDDODL      'create the first member of this folder
... 'sets the member DDO attributes and properties
```

```

folder.addElement member          'add member to the folder collection
... 'create and add some more member DDO to the DDO collection as necessary
Dim ddo as new DXDDODL           'create a folder ddo
... 'sets some of its attributes
ddo.addPropertyAndValue DX_DL_PROPERTY_ITEM_TYPE, DX_DL_FOLDER
'Create DX_DL_FOLDER attribute and sets it to refer to the DXFolderDL object
Dim data_id as Integer
data_id = ddo.addData(DX_DL_DKFOLDER) 'add attribute DX_DL_DKFOLDER
ddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_TYPE, DX_Collection_DDO
ddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_NULLABLE, True
ddo.setData data_id, folder      'sets the attribute value

```

After a DXParts is set to be an attribute value of a DDO, the DDO owns it and will take care of deleting it:

```

data_id = ddo.dataId(DX_DL_DKFOLDER) 'get DX_DL_DKFOLDER data-id
If data_id = 0 Then 'folder not found
    MsgBox "folder data item not found"
End If
Dim col as DXFolderDL
Set col = ddo.getData(data_id) 'get the parts collection
'Create iterator and process the DDO collection member one by one
If col Is Nothing Then
Else
    Dim item as DXDDODL
    Dim iter as DXSequentialIteratorDL
    Set iter = col.createIterator
    Do While iter.more
        Set item = iter.next
        If item Is Nothing Then
        Else
            item.retrieve 'process the member DDO
            ... 'other processing
        End If
    Loop
End If

```

Using collections and iterators

The ActiveX collection classes are made up of DXSequentialCollectionxx, DXResultsxx, DXPartsxx, and DXFolderxx where xx is the suffix representing the specific server, for example, Content Manager (DL), or Domino Extended Search (DES). DXSequentialCollectionxx contains VARIANTS as members. DXResults and DXFolder contain DXDDOs, and DXParts objects contain DXBlobs.

DXSequentialCollectionDL is used for retrieving datastore information such as schemas and servers, so it provides a minimum set of methods. The other collection classes provide methods for adding, retrieving, removing, and replacing members.

Create an iterator by calling the createIterator method on a collection. Use iterators to step through collection members. The following code steps through a collection:

```

Dim iter as DXSequentialIteratorDL
Set iter = coll.createIterator 'create an iterator for coll
Do While iter.more 'while there are more members
    Set member = iter.next 'get the current member and advance iter to the next member
    'Do something with the member
    ...
Loop

```

DXSequentialCollectionDL provides methods for adding, retrieving, removing, and replacing its members. The code could be rewritten as follows:

```

Dim iter as DXSequentialIteratorDL
Set iter = coll.createIterator 'create an iterator for coll
Do While iter.more           'while there are more members
  Set member = iter.at 'get the current member
  'Do something with the member
  ...
  iter.setToNext           'advance to the next position
Loop

```

All of the collection classes except DXSequentialCollectionDL allow you to perform operations at the current member before moving to the next member. When you remove the current member, the iterator advances to the next member. Therefore, when removing a member inside a loop, construct a check to avoid skipping the next member after removing the current one. For example:

```

...
If removeCondition = True Then
  coll.removeElementAt '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
End If

```

Querying a content server

You can search a content server and receive results in a DXResultSetCursor or DXResults object. You can create a query object to represent your query, then invoke the execute method or evaluate method 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.

There are three query object types: parametric, text, and combined. The combined query is composed of both text and parametric queries. Not all content servers can perform combined queries.

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

You can also use dkResultSetCursor to run a query again by calling the close and open methods. This is described in “Using the result set cursor” on page 255.

DKResults contains all of the results from the query. You can move an iterator either forwards or backwards over the items in the collection. The DKResults collection can be queried and used as a scope for another query. See “Querying collections” on page 257 for more information.

Restriction: Although Domino.Doc content servers return a DKResults object, this object cannot be queried nor used as a scope for another query.

Using parametric query

This section explains the parametric query function.

Formulating a parametric query

The following example is a query string representing a query on the index class DLSAMPLE. The query is searching for all documents or folders with an attribute of DLSEARCH_DocType <> null. The maximum number of results returned is limited to five. The content is set to YES, so that contents of the document or folder are returned.

The query also specifies that a Content Manager server use dynamic SQL for this query and that all folders and documents be searched. 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 quotation marks (").

```
Dim cmd as string
cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,"
cmd = cmd & "MAX_RESULTS=5,"
cmd = cmd & "COND=(DLSEARCH_DocType <> NULL));"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;"
cmd = cmd & "TYPE_FILTER=FOLDERDOC)"
```

Formulating a parametric query on multiple criteria

You can specify multiple search criteria using a parametric query. The following example shows how to specify a query on two index classes.

```
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE,MAX_RESULTS=3,";
cmd += cmd & "COND=(DLSEARCH_DocType <> NULL));";
cmd += cmd & "INDEX_CLASS=DLSAMPLE,MAX_RESULTS=8,";
cmd += cmd & "COND=('First name' == \"Robert\");";
cmd += cmd & "OPTION=(CONTENT=YES;";
cmd += cmd & "TYPE_QUERY=DYNAMIC;";
cmd += cmd & "TYPE_FILTER=FOLDERDOC)";
```

Executing a parametric query

Using Content Manager as a model, the steps are:

1. Create a DXDatastoreDL object.
2. Connect to the Content Manager server LIBSRVRN using the user ID FRNADMIN and the password PASSWORD.
3. Create a parametric query. The following query specifies to find all documents for the GP2DLS3 index class with the attribute last name equal to SUMMERS.
4. Run the query and examine the results using DXResultsDL.
5. Disconnect from the Content Manager server.

```
Dim dsDL as new DXDatastoreDL
Dim qry As DXParametricQueryDL
Dim results as DXResultsDL
Dim item as DXDDODL
Dim iter as DXSequentialIteratorDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=GP2DLS3,"
cmd = cmd & "COND=('Last name' == \"SUMMERS\");";
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;TYPE_FILTER=DOC)"
Set qry = dsDL.createQuery(cmd, DX_DL_PARAMETRIC_QL_TYPE)
qry.execute
Set results = qry.result
Set iter = results.createIterator
Do While iter.more
```

```

    Set item = iter.next
    'Do something with the DDO...
Loop
dsDL.disconnect

```

Executing a parametric query from the content server

Using Content Manager as a model, the steps are:

1. Create a DXDatastoreDL object.
2. Connect to the Content Manager server LIBSRVRN using the user ID FRNADMIN and the password PASSWORD.
3. Search all folders and documents for the DLSAMPLE index class.
4. Execute the query and examine the results using DXResultSetCursor. A null is returned from `fetchNext` if the cursor is past the last item in the result.
5. Disconnect from the Content Manager server.

```

Dim dsDL as new DXDatastoreDL
Dim cur As DXResultSetCursorDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)"
Set cur = dsDL.execute(cmd, DX_DL_PARAMETRIC_QL_TYPE)
Dim item as DXDDODL
Do While cur.isValid
    Set item = cur.fetchNext
    'Do something with the ddo item
Loop
cur.destroy 'in case the variable is being used by another query
dsDL.disconnect

```

Evaluating a parametric query from the datastore

Using Content Manager as a model, the steps are:

1. Create a DXDatastoreDL object.
2. Connect to the Content Manager server LIBSRVRN using the user ID FRNADMIN and the password PASSWORD.
3. Search all folders and documents for the GP2DLS5 index class with a condition of `DLSEARCH_Date >= 1995` and `DLSEARCH_Date <= 1996`. Because content is equal to NO, only the document and folder PIDs appear in the results.
4. Run the query and examine the results using DXResultsDL.
5. Disconnect from the Content Manager server.

```

Dim dsDL as new DXDatastoreDL
Dim results As DXResultsDL
Dim iter as DXSequentialIteratorDL
Dim item as DXDDODL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=GP2DLS5,;"
cmd = cmd & "COND=((DLSEARCH_Date >= ""1995"") AND "
cmd = cmd & "(DLSEARCH_Date <= ""1996"")));"
cmd = cmd & "OPTION=(CONTENT=NO;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;TYPE_FILTER=FOLDERDOC)"
Set results = dsDL.evaluate(myQuery, DX_DL_PARAMETRIC_QL_TYPE)
Set iter = results.createIterator
Do While iter.more
    Set item = iter.next
    'Do something with the DDO item
Loop

```


Using text query

This section explains text queries.

Formulating a text query

The following example shows a query for a text index called TMINDEX. The query searches for all text documents with the word UNIX or member. The maximum number of results returned is five.

```
Dim cmd as String
cmd = "SEARCH=(COND=(UNIX OR member));"
cmd = 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.

```
Dim cmd as String
cmd = "SEARCH=(COND=(UNIX OR member));";
cmd += cmd & "OPTION=(SEARCH_INDEX=(TMINDEX,TMINDEX2); MAX_RESULTS=5)";
```

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.

Executing a text query

Using Content Manager as a model, the steps are:

1. Create a DXDatastoreTS object.
2. Connect to the text search server TM.
3. Create a text query. The following query searches documents with the phrase UNIX operating and the word system.
4. Run the query and examine the results using DXResults.
5. Disconnect from the text search server.

```
Dim dsTS as new DXDatastoreTS
Dim qry As DXTextQueryTS
Dim results as DXResultsDL
Dim item as DXDDODL
Dim iter as DXSequentialIteratorDL dsTS.connect "TM", "", "", ""
Dim cmd as String
cmd = "SEARCH="
cmd = cmd & "(COND=('UNIX operating' AND system));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
Set qry = dsTS.createQuery(cmd, DX_TS_TEXT_QL_TYPE)
qry.execute
Set results = qry.result
Set iter = results.createIterator
Do While iter.more
    Set item = iter.next
    'Do something with the DDO item
Loop
dsTS.disconnect
```

Executing a text query from the content server

Using Content Manager as a model, the steps are:

1. Create a DXDatastoreTS object.

2. Connect to the text search server by host name and port number.
3. Search for all text documents with the free text Web site.
4. Run the query and examine the results using DXResultSetCursor. A null is returned from fetchNext if the cursor is past the last item in the result.
5. Disconnect from the text search server.

```
Dim dsTS as new DXDatastoreTS
Dim cur As DXResultSetCursorDL
Dim item as DXDDODL
dsTS.connect "zebra", "7502", DX_CTYP_TCPIP
dim cmd as String
cmd = "SEARCH="
cmd = cmd & "(COND=({web site}));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
Set cur = dsTS.execute(cmd, DX_TS_TEXT_QL_TYPE)
Do While cur.isValid
    Set item = cur.fetchNext
    'Do something with the DDO item
Loop
cur.destroy 'in case the variable is being reused by another query
dsTS.disconnect
```

Evaluating a text query from the content server

Using Content Manager as a model, the steps are:

1. Create a DXDatastoreTS object.
2. Connect to the text search server TM.
3. Search for all text documents with words that start with the letters UN.
4. Run the query and examine the results using DXResultsDL.
5. Disconnect from the text search server.

```
Dim dsTs as new DXDatastoreTS
Dim item as DXDDODL
Dim results As DXResultsDL
Dim iter as DXSequentialIteratorDL
dsTS.connect "TM", "", "", ""
Dim cmd as String
cmd = "SEARCH="
cmd = cmd & "(COND=($MC=*$ UN*));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
Set results = dsTS.evaluate(myQuery, DX_TS_TEXT_QL_TYPE)
Set iter = results.createIterator
Do While iter.more
    Set item = iter.next
    'Do something with the DDO item
Loop
dsTS.disconnect
```

Getting match highlighting information for each text query result item

This example retrieves match highlighting information for each text query result item during a text query, by setting the MATCH_INFO option to YES. The MATCH_DICT option specifies whether the highlighting information should be obtained using a dictionary. The match information is returned in the DKMATCHESINFO attribute of the DKDDO returned from a text query. The value of the DKMATCHESINFO attribute will be a DKMatchesInfoTS object.

Attention: This process is time consuming because the document is retrieved from the content server and linguistically analyzed to determine potential matches.

```
Dim dsTS As new DXDatastoreTS
Dim cur as DXResultSetCursorDL
Dim item as DXDDODL
```

```

Dim count As Integer, i As Integer
Dim pidId As String, dataName As String
Dim data As Variant
Dim mInfo As DXMatchesInfoTS
Dim mSect As DXMatchesDocSectionTS
Dim mPara As DXMatchesParagraphTS
Dim mText As DXMatchesTextItemTS
Dim li As Integer, lj As Integer, lk As Integer
Dim numSects As Integer, numParas As Integer, numTextItems As Integer
Dim numNewLines As Integer
Dim strDataName As String
Dim iCCSID As Integer, iLang As Integer
Dim iOffset As Integer, iLen As Integer
Dim cmd As String
dsTS.Connect "TM", "", "", "LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)"
cmd = "SEARCH=(COND=(UNIX));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX;MATCH_INFO=YES;MATCH_DICT=NO"
cmd = cmd & "MAX_RESULTS=2)"
Set cur = dsTS.execute(cmd, DX_TS_TEXT_QL_TYPE)
Do While cur.isValid
    Set item = cur.fetchNext
    pidId = item.getPid
    count = item.dataCount
    For i = 1 To count
        dataName = item.getDataName(i)
        If dataName = DX_DL_DKMATCHESINFO Then
            Set mInfo = item.GetData(i)
            strDataName = mInfo.getDocumentName
            numSects = mInfo.numberOfSections
            For li = 1 To numSects
                Set mSect = mInfo.getSection(li)
                strDataName = mSect.getSectionName
                numParas = mSect.numberOfParagraphs
                For lj = 1 To numParas
                    Set mPara = mSect.getParagraph(lj)
                    iCCSID = mPara.getCCSID
                    iLang = mPara.getLanguageId
                    numTextItems = mPara.numberOfTextItems
                    For lk = 1 To numTextItems
                        Set mText = mPara.getTextItem(lk)
                        strDataName = mText.GetText
                        If mText.isMatch = True Then
                            iOffset = mText.getOffset
                            iLen = mText.getLength
                        End If
                    Next
                    numNewLines = mText.numberOfNewLines
                Next
            Next
        Else
            data = item.GetData(i)
        End If
    Next
Loop
cur.destroy
dsTS.disconnect

```

Getting match highlighting information for a particular text query result item

This example retrieves match highlighting information for a specific item returned from a text query. The match information contains the text of the document and the highlighting information for every match of the corresponding query. The `dkResultSetCursor` passed into this routine must be in an open state.

Attention: This process is time consuming because the document is retrieved from the content server and linguistically analyzed to determine potential matches.

```
Dim dsTS As new DXDatastoreTS
Dim cur as DXResultSetCursor
Dim item as DXDDO
Dim count As Integer, i As Integer
Dim pidId As String, dataName As String
Dim data As Variant
Dim mInfo As DXMatchesInfoTS
Dim mSect As DXMatchesDocSectionTS
Dim mPara As DXMatchesParagraphTS
Dim mText As DXMatchesTextItemTS
Dim li As Integer, lj As Integer, lk As Integer
Dim numSects As Integer, numParas As Integer, numTextItems As Integer
Dim numNewLines As Integer
Dim strDataName As String
Dim iCCSID As Integer, iLang As Integer
Dim iOffset As Integer, iLen As Integer
Dim docId As String, indexName As String
Dim cmd As String
dsTS.Connect "TM", "", "", "LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)"
cmd = "SEARCH=(COND=(UNIX));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX"
cmd = cmd & "MAX_RESULTS=2)"
Set cur = dsTS.execute(cmd, DX_TS_TEXT_QL_TYPE)
Do While cur.isValid
    Set item = cur.fetchNext
    pidId = item.getPid
    docId = pidId
    count = item.dataCount
    indexName = item.getObjectType
    For i = 1 To count
        dataName = item.getDataName(i)
        data = item.GetData(i)
    Next
    Set mInfo = ds.getMatches(cur, docId, indexName, False)
    If (mInfo Is Nothing) = False Then
        Set mInfo = item.GetData(i)
        strDataName = mInfo.getDocumentName
        numSects = mInfo.numberOfSections
        For li = 1 To numSects
            Set mSect = mInfo.getSection(li)
            strDataName = mSect.getSectionName
            numParas = mSect.numberOfParagraphs
            For lj = 1 To numParas
                Set mPara = mSect.getParagraph(lj)
                iCCSID = mPara.getCCSID
                iLang = mPara.getLanguageId
                numTextItems = mPara.numberOfTextItems
                For lk = 1 To numTextItems
                    Set mText = mPara.getTextItem(lk)
                    strDataName = mText.GetText
                    If mText.isMatch = True Then
                        iOffset = mText.getOffset
                        iLen = mText.getLength
                    End If
                    numNewLines = mText.numberOfNewLines
                Next
            Next
        Next
    End If
Loop
cur.destroy
dsTS.disconnect
```

Using result set cursor

The `DXResultSetCursor` is a content server cursor that manages a virtual collection of DDOs and does not appear until you fetch an element from it. The collection set resulting from a query submitted to the content server.

Important: When you stop using the cursor, call the `destroy` method to close it and prevent memory leaks.

Opening and closing the result set cursor to re-execute the query

When you create a result set cursor, it is open. To run a query again, you close and reopen the cursor, as shown in the following example:

```
Dim cmd as String
cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;"
cmd = cmd & "TYPE_FILTER=FOLDERDOC"
Dim cur as DXResultSetCursorDL
...
Set cur = dsDL.execute(cmd, DX_DL_PARAMETRIC_QL_TYPE)
cur.close
cur.open
```

Setting and getting positions in a result set cursor

The result set cursor allows you to set and get the current cursor position. The following example completes the following steps:

1. Run a query.
2. Set the position to next, and fetch the DDO.
3. Get the current position of the result set cursor. A null value is returned from `fetchObject` if the cursor is past the last item in the result.

```
Dim cmd as string
cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE);"
cmd = cmd & "OPTION=(CONTENT=YES;"
cmd = cmd & "TYPE_QUERY=DYNAMIC;"
cmd = cmd & "TYPE_FILTER=FOLDERDOC"
Dim cur as DXResultSetCursorDL
Dim item as DXDDODL
Dim i as Long
i = 0
...
Set cur = dsDL.execute(cmd, DX_DL_PARAMETRIC_QL_TYPE)
Do while cur.isValid
    cur.setToNext
    Set item = cur.fetchObject
    If item is Nothing Then
    Else
        i = cur.getPosition
        'Do something with the DDO
    End If
Loop
cur.destroy 'in case the variable is being reused by another query
```

Another way to do this is:

```
Set cur = dsDL.execute(cmd, DX_DL_PARAMETRIC_QL_TYPE)
Do While cur.isValid
    cur.setPosition DX_DL_NEXT, 1 'the 1 is ignored for DX_NEXT
    Set item = cur.fetchObject
    If item is Nothing Then
```

```

Else
    i = cur.getPosition
    'Do something with the DDO item
End If
Loop
cur.destroy 'in case the variable is being reused by another query

```

You can use relative positioning. In the following example, every other item in the result set cursor is skipped.

```

Dim increment as Long
increment = 2
Set cur = dsDL.execute(cmd, DX_DL_PARAMETRIC_QL_TYPE)
Do While cur.isValid
    cur.setPosition DX_DL_RELATIVE, increment
    Set item = cur.fetchObject
    If item is Nothing Then
    Else
        i = cur.getPosition
        'Do something with the DDO
    End If
Loop
cur.destroy 'in case the variable is being reused by another query

```

Creating a collection from a result set

You can use the result set cursor to populate a collection with a specified number of items from the result set. In the following example, all items from the result set are fetched into a sequential collection. The first parameter specifies how many items to put into the collection. A zero in the first parameter of the `fetchNextN` method indicates that all result set items will be put into the collection. If `fItems` is `TRUE`, at least one item was returned.

```

Dim seqColl as DXSequentialCollection
Dim fItems as Boolean
fItems = False
Dim how_much as Long
how_much = 0
fItems = cur.fetchNextN(how_much, seqColl)

```

Querying collections

A *queryable collection* is a collection that can be queried further, thus providing a smaller evaluation set or more refined results. A concrete implementation of a queryable collection is a `DKResults` object. `DKResults` is a collection of DDOs, which are the result of a query.

Getting the result of a query

The following example illustrates how to submit a parametric query and get results:

```

'Establish a connection
Dim dsDL as new DXDatastoreDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
'Create a query object
Dim query1 as String
query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> NULL));"
Dim pq as DXParametricQuery
Set pq = dsDL.createQuery(query1, DX_DL_PARAMETRIC_QL_TYPE)
pq.execute
Dim rs as DXResultsDL
Set rs = pq.result

```

The results are in `rs`, which is a `DXResults` object. You can use previous code examples to process the collection and get the DDO.

Evaluating a new query

You can query the result from the example shown in the previous section to further refine it. For example:

```
Dim query2 as String
query2 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Subject == 'Mystery'));"
Dim rs2 as DXResults
Set rs2 = rs.evaluate(query2, DX_Parametric_QL_type
```

The totals of both queries would be equivalent to:

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

You can repeat this step until you get satisfactory results. After you start with one type of query, the subsequent queries must be of the same type, because you might get a null result.

The following example is for text queries:

```
Dim dsTS as new DXDatastoreTS
dsTS.connectPort "TM","","",""
Dim tquery1 as String
tquery1 = "SEARCH=(COND=(IBM));OPTION=(SEARCH_INDEX=TMINDEX)"
Dim tq as DXTextQuery
Set tq = dsTS.createQuery(tquery1)
tq.execute
Dim trs as DXResults
Set trs = tq.result
Dim tquery2 as String
tquery2 = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)"
Dim trs2 as DXResults
Set trs2 = trs.evaluate(tquery2, DX_TS_TEXT_QL_TYPE)
```

The sum of both queries is equivalent to:

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

Using queryable collection instead of combined query

Evaluating a queryable collection is similar to other Java classes. One such class is combined query. You can use a combined query 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.

The result of a combined query is a `DKResults` object, so you can theoretically evaluate another parametric query against it; although it might not always work. You cannot perform 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. This is quite useful for dynamically browsing a content server 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.

Using specific content servers

Each content server uses the `dkDatastore` classes, or data definition classes, as the primary interface to the Enterprise Information Portal database. Each content server has a separate content server class that implements the `dkDatastore` class to provide information specific to the content server.

Each content server type is represented by a class called `DKDatastorexx`, where `xx` is an abbreviation that identifies the specific content server as shown in Table 36.

Table 36. Server type and class name terminology

Server type	Class name
Content Manager	<code>DXDatastoreDL</code>
Domino Extended Search	<code>DXDatastoreDES</code>

When creating a content server you must implement one concrete class for each of the following classes or interfaces:

DXDatastorexx

Represents and manages a connection to the content server, transactions, and the execution of content server commands. `DXDatastore` is an abstract version of the query manager class. It supports the `evaluate` method, so it can be considered a subclass of query manager.

DXDatastoreDefxx

Defines methods to access items stored in the content server; it can also create, list, and delete its entities. It maintains a collection of `DXEntityDefs`. Examples of concrete classes for this interface are:

- `DXDatastoreDefDL`
- `DXDatastoreDefDES`

DXEntityDefxx

Defines methods to:

- Create and delete an entity
- Access entity information
- Create and delete attributes

In this class, all methods related to subentities generate `DKUsageError` objects indicating that the datastore does not then support subentities. However, if the content server supports multiple-level entities the subclass for the content servers must implement methods to overwrite the exceptions, for example. Examples of concrete classes for the `DXEntityDef` interface are:

- `DXIndexClassDefDL`
- `DXEntityDefTS`

DXAttrDefxx

Defines methods to access attribute information and to create and delete attributes. Examples of concrete classes for `DXAttrDefxx` are:

- `DXAttrDefDL`
- `DXFieldDefDES`

DXServerDefxx

Defines methods to access server information. Examples of concrete classes for `DXServerDefxx` are:

- DXServerDefDL
- DXServerDefDES

DXResultSetCursorxx

A content server cursor that manages a virtual collection of DDO objects. The collection is a query result set. Elements of the collection do not materialize until a datastore fetch operation is run. To use the `addObject`, `deleteObject`, and `updateObject` methods, you must set the datastore option `DK_DL_OPT_ACCESS_MODE` to `DK_READWRITE`. Examples of concrete classes for `dkResultSetCursor` are:

- DXResultSetCursorDL
- DXResultSetCursorV4

DXBlobxx

An abstract class that declares a common public interface for binary large object (BLOB) data types in each content server. The concrete classes derived from `DXBlob` share this common interface, allowing polymorphic processing of BLOB collections originating from heterogeneous content servers. Examples of concrete classes for `DXBlobxx` are:

- DXBlobDL
- DXBlobDD

Working with Content Manager

This section describes how to:

- Handle large objects
- Use DDOs in the server
- Use XDOs in a search engine
- Use combined query
- Use Text Search Engine
- Use image search (QBIC).
- Use workflows and workbaskets

Using DDOs to represent datastore information

A DDO associated with `DKDatastoreDL` has some specific information to represent the Enterprise Information Portal document metaphor: document, folder, parts, item, item ID, rank, and so forth. The following sections describe how you record this information.

DDO properties: The type of an item, either a document or folder, is a property under the name `DK_Property_Item_Type`. To get the item type of the DDO, you call:

```
Dim a as VARIANT
Dim item_type as Integer
a = cddo.getPropertyByName(DX_DL_PROPERTY_ITEM_TYPE)
If IsEmpty(a) Then
Else
    item_type = a
End If
```

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 223 and “Getting the DDKDO and attribute properties” on page 224 for more information.

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 datastore. See “Creating a persistent identifier (PID)” on page 222.

Representing documents: A DDO representing a document has the property `DX_DL_PROPERTY_ITEM_TYPE` equal to `DX_DL_DOCUMENT`. Its PID contains the index class name as the object type, and the item ID is in the PID’s ID.

The parts inside a document are represented as `DXPartsDL` objects, which are collections of binary large objects (BLOBs), each of which is represented as a `DXBlobDL` object.

A document DDO has the specific attribute with the reserved name `DX_DL_DKPARTS`, whose value is also `DXPartsDL` object.

To get to each part in a document, you must retrieve `DXPartsDL` first, then create an iterator to retrieve each part. If the document does not have a part at all, `DKParts` is null.

For more information on creating and processing a `DKParts` object, see “Retrieving a document or folder” on page 361 and “Creating and using the `DX_DL_DKPARTS` attribute” on page 344.

Documents associated with a combined query (a combination of a parametric and text query) can have a transient attribute called `DX_DKRANK`, whose value is an object containing an integer rank computed by the Text Search Engine.

Representing folders: A DDO representing a folder has a property `DX_DL_PROPERTY_ITEM_TYPE` equal to `DX_DL_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.

The table of contents inside a folder is represented as a `DXFolderDL` object, which is a collection of DDOs. Each collection represents an item—either a document or another folder—belonging to this folder. A folder DDO has a specific attribute with a reserved name `DX_DL_DKFOLDER`, whose value is also `DXFolderDL` object.

To get to each DDO member of the folder, you must retrieve `DXFolderDL` first, then create an iterator to retrieve each item member. If the folder does not have a member, `DXFolderDL` is null.

For more information on creating and processing a `DKFolder` object, see “Retrieving a document or folder” on page 361 and “Creating and using the `DX_DL_DKFOLDER` attribute” on page 344.

Creating, updating, and deleting documents or folders

This section describes how to create, update, and delete documents and folders.

Creating a document: To create a document and save it in a content server, you must create a DDO, setting all of its attributes and other information, except its item ID. The item ID is assigned and returned by the content server. The following example combines parts of the previous examples:

```
'step 1: create a datastore and connect to it
Dim dsDL as new DXDatastoreDL
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"
Dim cddo as new DXDDO
```

```

cddo.setObjectType "GRANDPA"      'set the index class name it belongs to
cddo.setDatastore dsDL            'associate to dsDL

'step 2.a: add attributes according to index class GRANDPA
Dim data_id as Integer
data_id = cddo.addData("Title") 'add a new attribute named "Title"
cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_TYPE,
    DX_VString 'add type property VSTRING
    cddo.addDataPropertyAndValue data_id,
        DX_DL_PROPERTY_NULLABLE, False 'add nullable property
data_id = cddo.addData("Subject") 'add a new attribute named "Subject"
cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_TYPE, DX_VString
cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_NULLABLE, True
'Add some more attributes as necessary
' ...

'step 2.b: add DX_DL_DKPARTS attribute
Dim blob as new DXBlobDL
blob.setDatastore dsDL            'create a new XDO blob
blob.setPartId 5                  'set part number to 5
blob.setContentClass DX_DL_CC_GIF 'set content class type GIF
blob.setRepType DX_DL_DK_REP_NULL 'set rep type for the part
blob.setContentFromClientFile "choice.gif" 'set the blob's content
blob.setInstanceOpenHandler "netscape", true 'set the blob's viewer
parts.addElement blob            'add the blob to the parts collection
... 'create and add some more blobs to the collection as necessary
'Create DX_DL_DKPARTS attribute and sets it to refer to the DXPartsDL object
data_id = ddo.addData(DX_DL_DKPARTS) 'add attribute DX_DL_DKPARTS
cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_TYPE,
    DX_Collection_XDO 'add type property
    cddo.addDataPropertyAndValue data_id, DX_DL_PROPERTY_NULLABLE,
        True 'add nullable property
    cddo.setData data_id, parts 'sets the attribute value

'step 2.c: sets the item type : document
cddo.addPropertyAndValue DX_DL_PROPERTY_ITEM_TYPE, DX_DL_DOCUMENT

'step 3: make it persistent
cddo.add 'a document is created in the datastore

```

After the last step, the document is created 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 listed in DXPARTSDL.

This also applies to adding a folder DDO, the DXFolderDL collection members are added to the datastore as a component of the folder. A folder contains a table of contents of its members, which are existing documents and folders. Therefore, all folder members must be created in the datastore before you can add a folder DDO.

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
Dim dsN as new DXDatastoreDL
dsN.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
'Update the Pid
ddo.setObjectType "GRANDPA2"      'set the new index class
ddo.setPid ""                      'blank the item ID
ddo.setDatastoreName "LIBSRVRN"   'set the new datastore name'
ddo.setDatastore dsN              'set the new datastore
ddo.add                            'add it

```

Updating a document or a folder: To update a document, you must set the item ID and object type. Then, update the desired attributes or add to the parts collection. Finally, call the update method to store the change. For example:

```
'Update the value of attribute Title
Dim data_id as Integer
data_id = ddo.GetDataByName("Title")
ddo.SetData data_id, "Guess who is behind all this"
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.

Updating a folder DDO requires similar steps: you update the attribute values, or add or remove elements from DXFolderDL, then call the update method.

Updating parts: The collection of parts in a document is represented in a DXPartsDL object called DKPARTS.

DKParts has two additional members for adding a part to, and removing a part from, the collection and immediately saves those changes.

The document must already exist in the content server.

Adding and removing a part: The following example adds a part to a document:

```
Dim ddo as DXDDO                'a document DDO
Dim newPart as DXBlobDL         'a new part to be added
... 'ddo and newPart are initialized somewhere along the line
Dim parts as DXPartsDL
Set parts = ddo.GetDataByName(DX_DL_DKPARTS) 'get DXPartsDL
parts.addMember ddo, newPart
```

Similarly, to remove newPart from the collection and the content server, enter:

```
parts.removeMember ddo, newPart
```

The removeMember method in DXPartsDL deletes the part from the content server.

Differences between update, add, and remove on a document DDO: DKParts addMember and removeMember methods provide conveniences for adding and removing a part in the collection and the content server. Compared to the update method in a document DDO, the addMember and removeMember are faster.

The update method on a DDO updates all of the attributes in the DDO, including DXPartsDL and all of its members that changed. The steps are:

```
...
Dim parts as DXPartsDL
Set parts = ddo.GetDataByName(DX_DL_DKPARTS) 'get DXPartsDL , assume it exists
parts.addElement newPart
ddo.update                                'updates the whole ddo
```

Updating folders: The collection of documents and folders within a folder is represented as a DKFolder object. In the content server, a folder holds a table of contents referring to its objects instead of all of the actual objects.

DXFolderDL has two additional members for adding a member to, or removing a member from, the collection and immediately stores those changes.

The added document or folder to be added must already exist in the content server, as must the folder to be added to.

Adding and removing a member: The following example adds another document or folder to a folder:

```
Dim folderDDO as DXDDO      'a folder DDO
Dim newMember as DXDDO      'a new DDO to be added as a member of this folder
... 'folderDDO and newMember are initialized somewhere along the line
Dim folder as DXFolderDL
'get DXFolderDL , assume it exists
Set folder = folderDDO.getDataByName(DX_DL_DKFOLDER)
folder.addMember folderDDO, newMember
```

To remove newMember from the collection and the content server, enter:

```
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 method it does not delete the member from memory or from the content server.

Differences between update, add, and remove on a folder DDO: DXFolderDL addMember and removeMember methods provide conveniences for adding and removing a document or folder in the collection and the content server. Compared with the update method in a folder DDO, add and remove methods are faster.

The update method on a DDO updates all of the attributes in the DDO, including DXFolderDL and all of its members, whereas add and remove member methods involve only adding and removing one particular member to or from 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.

```
ddo.del
```

The DDO must have its item ID and object type set, and your program must have a valid connection to the content server.

Use the statement above to delete a folder as well. Only the 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 datastore later. See “Creating a document” on page 358 for more information.

Retrieving a document or folder

To retrieve a document from DXDatastoreDL, you must know the document’s index class name and item ID. You must first associate the DDO to a content server and establish a connection.

```
Dim dsDL as new DXDatastoreDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim ddo as new DXDDO
ddo.setObjectType "GRANDPA"      'set the index class name it belongs to
ddo.setPid "LN#U5K6ARLGM3DB4"    ' set the item ID
ddo.setDatastore dsDL            'associate to dsDL
ddo.retrieve                      'retrieve it
```

To retrieve the DDO, call:

```
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 datastore. If the document has parts, the DX_DL_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, it is not desirable to 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 was run 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 identified in DX_DL_DKPARTS attribute, as follows:

```
Dim parts as DXPartsDL
Set parts = ddo.getDataByName(DX_DL_DKPARTS)
```

The example assumes that the DX_DL_DKPARTS attribute exists. An exception is generated if the attribute does not exist. See "Retrieving a folder" for an example of extracting 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 and using the DX_DL_DKPARTS attribute" on page 344.

```
'Create iterator and process the part collection member one by one
If parts Is Nothing Then
Else
  Dim blob as DXBlobDL
  Dim iter as DXSequentialIterator
  Set iter = parts.createIterator
  Do While iter.more
    Set blob = iter.next
    If blob Is Nothing Then
    Else
      blob.retrieve          'retrieve the blob's content
      blob.open              'other processing, as needed
    End If
  Loop
End If
```

Similar to the DDO results of a parametric query, each part XDO inside the DXPartsDL collection is empty (does not have its content set). However, it has all the information needed for information retrieval. To bring its content and related information into memory, call:

```
blob.retrieve
```

Retrieving a folder: Retrieve a folder DDO the same way as you would retrieve a document DDO. After being retrieved, the folder DDO has all of its attributes set, including a special attribute, DX_DL_DKFOLDER. This attribute value is set to a DKFolder object, a collection of DDO members in this folder. Like the parts in DXPartsDL, these member DDOs contain only enough information to retrieve them. You can retrieve a folder DDO as follows:

```
data_id = ddo.dataId(DX_DL_DKFOLDER)      'get DX_DL_DKFOLDER data-id
If data_id = 0 Then                        'folder not found
  MsgBox " folder data item not found"
  Exit sub
End If
Dim col as DXFolderDL
col = ddo.getData(data_id)                'get the folder collection
'Create iterator and process the DDO collection member one by one
```

```

If col Is Nothing Then
Else
  Dim item as DXDDOD1
  Dim iter as DXSequentialIterator
  Set iter = col.createIterator
  Do While iter.more
    Set item = iter.next
    If item Is Nothing Then
    Else
      item.retrieve          'retrieve the member DDO
      .... 'other processing
    End If
  Loop
End If

```

See also “Creating and using the DX_DL_DKFOLDER attribute” on page 344.

Understanding text searching (Text Search Engine)

A DXDatastoreTS object represents the Text Search Engine. Text Search Engine does not actually store the data, it merely indexes the data stored in 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 DXDatastoreTS object does not support add, update, retrieve, and delete methods. You can query this datastore. Refer to “Adding an XDO to be indexed by Text Search Engine” on page 367 for information on adding data to Content Manager that is indexed by Text Search Engine.

The Text Search Engine product can specify boolean, proximity, global text retrieval (GTR), hybrid, and free text queries. You can use the text search item ID, part number, and ranking information returned by the query to create an XDO that retrieves the text document contents in a Content Manager server.

Boolean query: A boolean query is made up of words and phrases, separated by boolean operators. A phrase is a sequence of words enclosed in apostrophes ('), to be searched as a literal string.

The following example is searching for all text documents with the word WWW or the phrase Web site in the TMINDEX text search index:

```

Dim cmd as String
cmd = "SEARCH=(COND=(www OR 'web site'));"
cmd = 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. In the following example, all text documents with the free text web site in the TMINDEX text search index are being searched for:

```

Dim cmd as String
cmd = "SEARCH=(COND={{web site}});"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"

```

Hybrid query: A hybrid query is made up of a boolean query followed by a free text query. The example is searching for all text documents with the words IBM and UNIX, as well as the free text web site in the TMINDEX text search index.

```

Dim cmd as String
cmd = "SEARCH=(COND=(IBM AND UNIX {web site}));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"

```


Proximity query: A proximity query relates to a sequence of search arguments found in the same document, paragraph, or sentence. The following example is searching for all text documents with the phrase rational numbers and the word math in the same paragraph.

```
Dim cmd as String
cmd = "SEARCH=(COND=($PARA$ {'rational numbers' math}));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMINDEX)"
```

Note: 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, but also supports single-byte character set (SBCS) languages. The following example shows a search for all text documents with the phrase IBM marketing. All double-byte characters should be enclosed in apostrophes ('). The phrase to be searched for should be in the specified character code set and language. The MATCH keyword is set to indicate the degree of similarity for the phrase.

```
Dim cmd as String
cmd = "SEARCH=(COND=($CCSID=850,LANG=6011,MATCH=1$ "
cmd = cmd & "'IBM marketing'));"
cmd = cmd & "OPTION=(SEARCH_INDEX=TMGTRX)"
```

Make sure that the text search datastore options DX_TS_OPT_CCSID (coded character set identifiers) and DX_TS_OPT_LANG (language identifiers) are set properly. The default for DX_TS_OPT_CCSID is DX_TS_CCSID_00850. The default for DX_TS_OPT_LANG is DX_TS_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 with the other.

Representing Text Search Engine information using DDOs: A DDO associated with a DXDatastoreTS object has specific information for representing results from text searches. DKDatastoreTS does not have a property item type as does a DXDatastoreDL object. The format of its ID is also different. A DDO resulting from a text query corresponds to a text part inside an item. It has a set of standard attributes, described below.

DX_DL_DKDLITEMID

The item ID that this text is part of. Use this item ID to retrieve the whole item from the content server.

DX_DL_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.

DX_DL_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.

DX_DL_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 *Text Search Engine Application Programming Reference* for further information.

DX_DL_DKDSIZE

An integer number representing word occurrences in the results of boolean queries. See *Text Search Engine Application Programming Reference* for further information.

DX_DL_DKRCNT

An integer number representing boolean search matches. See *Text Search Engine Application Programming Reference* for further information.

The PID for a text search DDO has the following information:

datastore type

TS

datastore 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 methods for connecting and a method for disconnecting. In a typical example, you create a DKDatastoreTS object, connect to it, run a query, then disconnect when done. The following example shows the first connection method using the text search server TM.

```
Dim dsTS as new DXDatastoreTS
dsTS.connect "TM", "", "", ""
dsTS.disconnect
```

The following example shows the second connection method using the text search server with the hostname apollo, port number 7502, and TCP/IP communication type DX_CTYP_TCPIP:

```
dsTS.connectPort "apollo", "7502", DX_TS_CTYP_TCPIP
```

The following example shows the first connection method 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 connection method using the text search server name TM, library server LIBSRVRN, user ID FRNADMIN, and password PASSWORD:

```
dsTS.connect "TM", "", "", "LIBACCESS=(LIBSRVRN,FRNADMIN,PASSWORD)"
```

Note that you can use the last parameter, also called the connect string, to pass a sequence of parameters in one string.

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 methods. 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.

```
Dim session_type As Long
Dim ccsid As Variant
Dim dsTS As New DXDatastoreTS
dsTS.setOption DX_TS_OPT_CC SID, DX_TS_CC SID_00850
dsTS.setOption DX_TS_OPT_LANG, DX_TS_LANG_ENU
dsTS.connect "TM", "", "", ""
dsTS.getOption DX_TS_OPT_CC SID, ccsid
```

```

If ccsid = DX_CCSID_00850 Then
    MsgBox "Datastore character code set is 850"
End If
dsTS.disconnect

```

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 DXDatastoreTS options, DX_TS_OPT_CCSID and DX_TS_OPT_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 364.

Listing servers: The DXDatastoreTS object provides a method to list the text search servers that it can connect to. The following example shows how to retrieve the list of servers.

```

Dim dsTS As New DXDatastoreTS
Dim col As DXSequentialCollectionDL
Set col = dsTS.listDataSources
Dim serverDef As DXServerDefTS
Dim iter As DXSequentialIteratorDL
Set iter = col.createIterator
Dim i As Long
i = 0
Do While iter.more
    i = i + 1
    Set serverDef = iter.Next
    MsgBox "Server Name [" & i & "] - " & serverDef.getName
        & " Server Loc - " & serverDef.getServerLocation
Loop

```

Listing schema: A DXDatastoreTS object provides methods for listing the schema. For text search, these are text search indexes. The following example shows how to retrieve the index list.

```

Dim dsTS As New DXDatastoreTS
Dim col As DXSequentialCollectionDL
Dim iter As DXSequentialIteratorDL
Dim entDef As DXSearchIndexDefTS
Dim i As Long
dsTS.Connect "TM", "", "", ""
Set col = dsTS.listEntities
Set iter = col.createIterator
i = 0
Do While iter.more
    i = i + 1
    Set entDef = iter.Next
    MsgBox "Entity Name [" & i & "] - " & entDef.getName
Loop
dsTS.disconnect

```

Indexing XDOs by search engine: If you want to index object content using Text Search Engine, the values of SearchEngine, SearchIndex and SearchInfo are required.

The SearchIndex value is a combination of two names: the search service name and search index name. For example, if the system administration program, refers to a text search server named TM for which you defined a search index named TMINDEX for it, then the correct value for the SearchIndex is TM-TMINDEX.

The value of SearchEngine must be SM for text search. The value of SearchEngine must be Query By Image Content (QBIC).

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.

Important: Do not set the RepType when you add a part object for a search engine to index. The Text Search Engine works only with the default RepType FRN\$NULL.

Adding an XDO to be indexed by Text Search Engine:

```
Dim partId as Integer
partId = 0 'let system decide the partId
Dim itemId as String, fileName as String
itemId = "CPPIORH4JBIXWIYO" 'existing itemId
fileName = "g:\\test\\cheetah.gif" 'file content to be indexed
Dim dsDL as new DXDatastoreDL 'required datastore
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD" 'connection to datastore
Dim axdo as new DXBlobDL
axdo.init dsDL 'create XDO
axdo.setPartId partId 'set partId
axdo.setPid itemId 'set itemId
axdo.setContentClass DX_DL_CC_ASCII 'set ContentClass to text

'---set searchEngine ----- (deprecated)
axdo.setSearchEngine "SM"
axdo.setSearchIndex "TM-TMINDEX"
axdo.setSearchInfo "ENU"

'---set searchEngine ----- (using an extension object)
Dim srcheng As New DXSearchEngineInfoDL
srcheng.setSearchEngine "SM"
srcheng.setSearchInfo "ENU"
srcheng.setSearchIndex "TMMUF-TMINDEX"
xdo.setExtension srcheng

axdo.setContentFromClientFile fileName 'set file content to buffer area
axdo.add 'add from the buffer
MsgBox "after add partId = " & axdo.getPartId 'display the partId after add
axdo.retrieve 'retrieve object
axdo.setInstanceOpenHandler "notepad", true 'set open viewer
axdo.open 'open content with viewer
dsDL.disconnect 'disconnect from datastore
```

Loading data to be indexed by Text Search Engine: You must create an index and a text search index when you load data for Content Manager to index.

Make sure the text search server is running before you create a text search index. To verify your environment setup, run the samples TListCatalogDL.frm and TListCatalogTS.frm. Before you run the samples, update them with your server, user ID, and related information.

To create parts indexed by TM in Content Manager, refer to “Using XDOs” on page 339.

After you load the data into Content Manager, the wakeUpService method in the DXDatastoreDL class places the documents on the document queue. From the Content Manager text search administration window, double-click the text search server. Double-click the text search index. Click INDEX.

When you click **INDEX**, you index the documents on the document queue. After Content Manager completes the indexing, 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 have a text-based structure. An HTML file is an example of a text-based document. A document model defines the document layout. For example, an HTML file contains tags such as <HEAD>, </H1> and <BODY>. You can use Text Search Engine to search for words or phrases between the HTML tags.

You can perform text queries on structured documents as follows:

1. Create a document model that contains sections made up of the section name and document tag in the text document. For example:

```
<HTML>
<HEAD>
<TITLE>My Corp<br></TITLE>
</HEAD>

<BODY>

<H1>My Corp<BR></H1>
<P><B>My Corp<BR></B><BR>
<P>Robert Summers<BR>
<P><ADDRESS>My Corporation<BR></ADDRESS>

<HR>
<H2>My 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 document model.
3. Set the indexing rules for the text search index and specify the default document format. For example, use DK_TS_DOCFMT_HTML for HTML files.
4. Add parts objects to the Content Manager server.
5. Start the text search index indexing process.

You must use the system administration program to check the indexing process status.

Check the `imldiag.log` file in the text search instance directory if an indexing error occurs. For more information on the `imldiag.log` file see the *Text Search Engine Application Programming Reference*.

Searching images by content

You can use the IBM Image Search server to search for stored images. You specify the image type or provide an example of the image.

Figure 58 on page 369 shows an example of the Image Search window in the application that connects to the image search server. The image search server uses

Query by Image Content (QBIC) technology to support searches based on Average Color, Color Layout and other features.

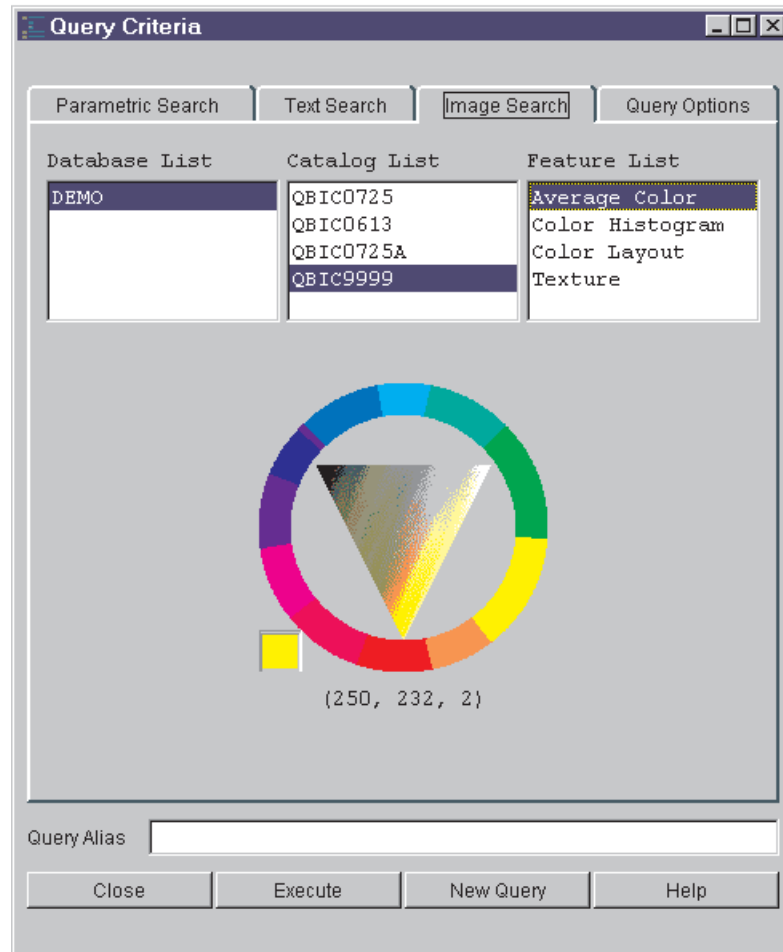


Figure 58. Image search window example. Users select search criteria, such as Average Color or Texture

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 the entire Content Manager system. It also describes *features* that are the searchable visual image characteristics.

Understanding image search server, databases, and catalogs: A Content Manager system searches for images using an image search server. Content Manager applications store image objects in the object server. The image search server analyzes and stores the image information. 59 shows an example of the image search server. Image servers do not store images.

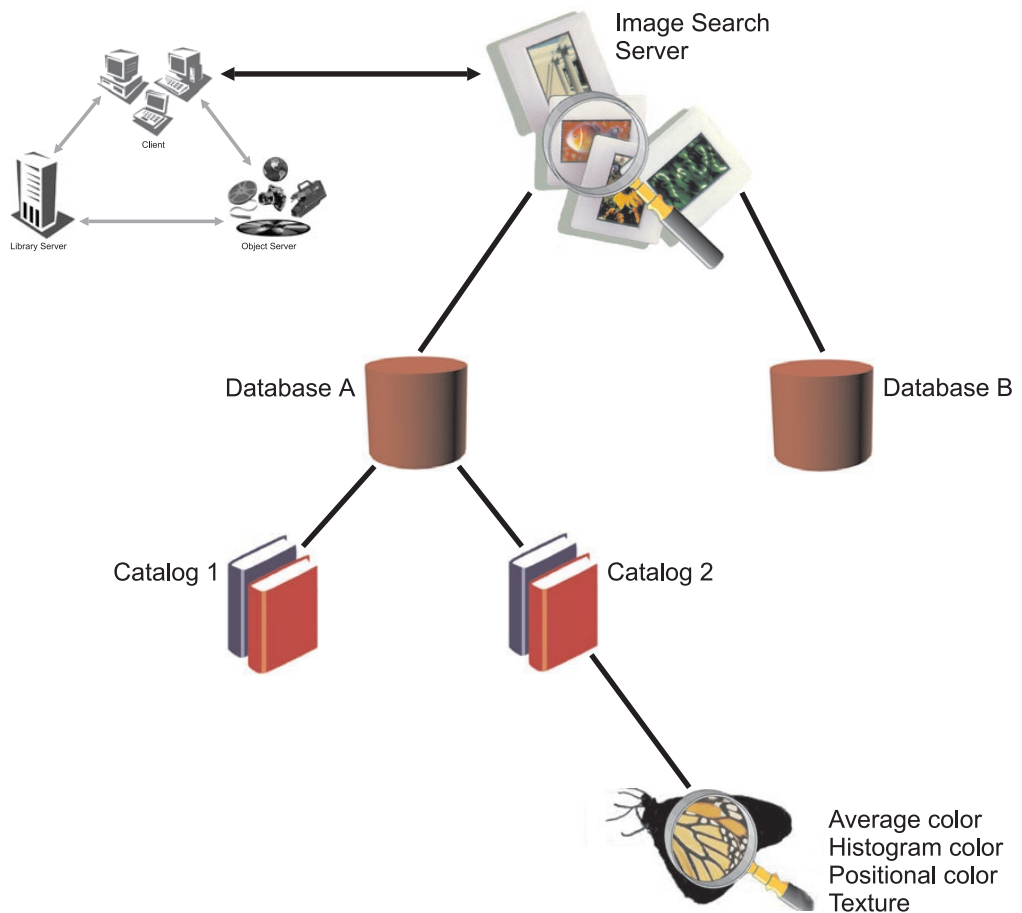


Figure 59. An image search server in a Content Manager system. The image search server communicates with the other Content Manager components through the clients.

The image search server contains one or more catalogs that hold information about one or more of the four image search features:

- Average Color
- Color Histogram
- Color Layout
- Texture

Understanding the Image Search Features List terms: This section explains the four image search features and provides the feature names used in the search.

Average color Use average color to search for images that have a predominant color. For example, images that contain equal portions of red and yellow will have an average color of orange. If you search for an image with an average color of orange, the search engine could find an image of fruit with an orange-colored peel, such as an orange and a tangerine.

The feature name you enter to search by Average Color is `QbColorFeatureClass`.

Histogram color

Use Histogram Color to search for images containing a similar variety of colors. 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 a meadow and blue sky has a histogram color with a high frequency of blue and green. The feature name you enter to search by Histogram Color is `QbColorHistogramFeatureClass`.

Color Layout

Use Color Layout to search for images with a similar layout. Color Layout measures the average color value for the pixels in a specified area of an image. For example, a image search for Color Layout that specified with picture with bright red objects in the middle could return an image of a sunset. The feature name you enter to search by Color Layout is `QbDrawFeatureClass`.

Texture

Use Texture to search for images that have a particular pattern. The Texture feature measures image coarseness, contrast, and direction. Coarseness indicates the size of repeating items in an image. Contrast identifies the brightness variations in an image. Direction indicates whether a direction predominates in an image. For example, an image that contains a wood grain has a similar texture to other images that contain a wood grain. The feature name you enter to search by Color Layout is `QbTextureFeatureClass`.

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, you must index the images and store the content information 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 client and server (see Figure 60 on page 372).

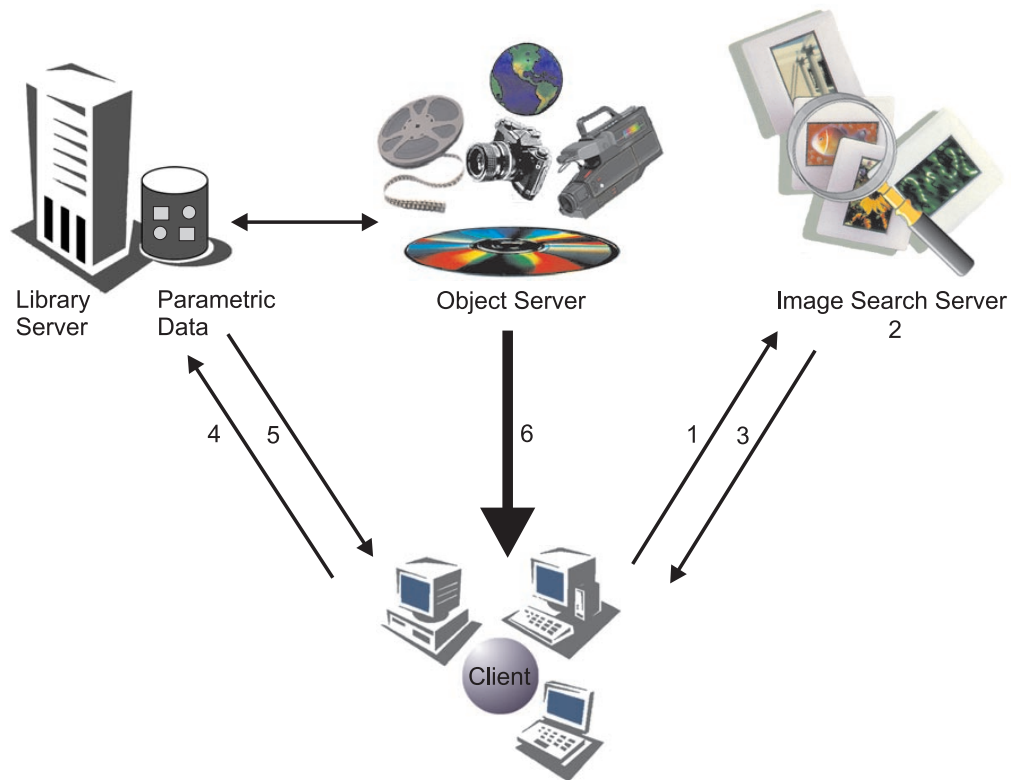


Figure 60. How image search clients search for and retrieve images

The following list describes how image search clients search for and retrieve images.

1. A client application builds a QBIC query string and sends it to an image search server.
2. An 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.

The query string: An image query is a character string that specifies the search criteria. The search criteria consist of:

Feature name The features used in the search.

Feature value The values of those features. Table 37 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. When you specify multiple features in a query, you can assign a weight to one or more of the features. `weight` is the combination of the keyword `weight=>0.0`.

You can also specify the maximum number of matches that a query returns. To specify the maximum matches, append `max_results` to your query. `max_results` consists of the keyword `max=>0`. Table 37 lists valid image search query feature names and values.

Table 37. Image search query valid feature names and values

Feature name	Values
Average Color	color = < <i>rgbValue</i> , <i>rgbValue</i> , <i>rgbValue</i> > where <i>rgbValue</i> is an integer from 0 to 255.
QbColorFeatureClass or	
QbColor	
	file = < <i>fileLocation</i> , " <i>fileName</i> " > where <i>fileLocation</i> is either <code>server</code> or <code>client</code> , <i>fileName</i> is the complete file path specified in the format appropriate for the system on which the file resides.

Table 37. Image search query valid feature names and values (continued)

Feature name	Values
Histogram Color QbColorHistogramFeatureClass or QbHistogram	<p>histogram = < <i>histList</i> > where <i>histList</i> consists of one or more <i>histClause</i> separated by a comma (,).</p> <p>A <i>histClause</i> is specified as (<i>histValue</i>, <i>rgbValue</i> , <i>rgbValue</i> , <i>rgbValue</i>), where <i>histValue</i> is an integer from 1 to 100 (a percentage value), and <i>rgbValue</i> is an integer from 0 to 255.</p> <p>file = < <i>fileLocation</i> , " <i>fileName</i> " > where <i>fileLocation</i> is either <i>server</i> or <i>client</i>, <i>fileName</i> is the complete file path specified in the format appropriate for the system on which the file resides.</p>
Color Layout QbDrawFeatureClass or QbDraw	<p>description = < " <i>descString</i> " > where <i>descString</i> is a special encoded string describing a picker file. Description string format:</p> <ol style="list-style-type: none"> 1. <i>Dw,h</i> specifies the outer dimensions of the image with width <i>w</i> and height <i>h</i>. 2. <i>Rx,y,w,h,r,g,b</i> specifies that a rectangle of width <i>w</i> and height <i>h</i> is to be positioned with its upper left corner at the coordinates (<i>x,y</i>). 3. Use the colon (:) as a separator. <p>file = < <i>fileLocation</i> , " <i>fileName</i> " > where <i>fileLocation</i> is either <i>server</i> or <i>client</i>, <i>fileName</i> is the complete file path specified in the format appropriate for the system on which the file resides.</p>
Color Texture QbTextureFeatureClass or QbTexture	<p>file = < <i>fileLocation</i> , " <i>fileName</i> " > where <i>fileLocation</i> is either <i>server</i> or <i>client</i>, <i>fileName</i> is the complete file path specified in the format appropriate for the system on which the file resides.</p>

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 texture weight 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 limit the returned matches to five:

```
QbColorFeatureClass color=<255,0,0> and max=5
```

Establishing a connection in QBIC

Image search provides methods 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.

```
Dim dsQBIC as new DXDatastoreQBIC
dsQBIC.connect "QBICSRV", "QBICUSER", "PASSWORD", ""
...
// do some work
dsQBIC.disconnect
```

The image search connection allows an application to connect to an image search server. After connecting, your program can use methods that access the image search server except for the methods that are not related to image search catalogs, such as `listDatabases`. An `openCatalog` method is required to open a catalog for processing. A `closeCatalog` method is called after processing is done. The following example shows how to connect, open a catalog, close the catalog, and disconnect.

```
Dim dsQBIC as new DXDatastoreQBIC
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 method for listing the image search servers that it can connect to. The following example shows how to retrieve in a `DXSequentialCollection` object the list of servers that contain `DXServerInfoQBIC` objects. After you get a `DXServerInfoQBIC` object, you can retrieve the server name, the host name, and the port number.

```
Dim dsQBIC as new DXDatastoreQBIC
Dim serverInfo as DXServerInfoQBIC
Dim col as DXSequentialCollection
Dim iter as DXSequentialIterator
Dim serverName as String, hostName as String, portNumber as String
Dim i as Long
i = 0
Set col = dsQBIC.listServers
Set iter = col.createIterator
Do While iter.more
    i = i + 1
    Set serverInfo = iter.next
    serverName = serverInfo.serverName
    hostName = serverInfo.hostName
    portNumber = serverInfo.portNumber
    MsgBox "Server Name [" & i & "] - " & serverName &
        "Host Name " & hostName & "Port Number " & portNumber
Loop
```

Listing image search databases, catalogs and features

`DXDatastoreQBIC` provides a method for listing all of the image search databases, catalogs, and features on an image search server. The list is returned in a `DXSequentialCollection` object that contains `DXIndexQBIC` objects. After you get a `DXIndexQBIC` object, you can retrieve the database, catalog, and feature name. The following example shows how to retrieve the list of databases, catalogs, and features.

```

Dim index as DXIndexQBIC
Dim databaseName as String
Dim col as DXSequentialCollection
Dim iter as DXSequentialIterator
Dim i as Long
i = 0

Set col = dsQBIC.listDatabases
Set iter = col.createIterator
Do While iter.more
    i = i + 1
    Set index = iter.next
    databaseName = index.name
    MsgBox "Database Name [" & i & "] - " & databaseName
Loop

```

The following example shows how to retrieve the list of QBIC catalogs. The list of catalogs is returned in a sequential collection of DXIndexQBIC objects. Once a DXIndexQBIC object is obtained, you can pull out the QBIC catalog name.

```

Dim index as DXIndexQBIC
Dim catalogName as String
Dim col as DXSequentialCollection
Dim iter as DXSequentialIterator
Dim i as Long
i = 0

Set col = dsQBIC.listCatalogs("DEMO")
Set iter = col.createIterator
Do While iter.more
    i = i + 1
    Set index = iter.next
    catalogName = index.name
    MsgBox "Catalog Name [" & i & "] - " & catalogName
Loop

```

Representing image search information with a DDO

A DDO associated with DXDatastoreQBIC contains specific information for representing image search results. A DDO from an image query corresponds to an image part inside an item; it has the following set of standard attributes:

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 result set.

As a convention, the attribute value is always an object.

DKDLITEMID

The item ID specifying where this image part is stored. Use this item ID to retrieve the whole item from the content server.

DKPARTNO

A long integer part number identifying the location of this image part. In Java, DKPARTNO is an integer part number. Use with the item ID to retrieve this part from the content server.

DKREPTYPE

A representation type string with a default value of FRN\$NULL. This attribute is reserved.

DKRANK

A long integer rank signifying the relevance of this part to the result set from the image query. The rank is within the range 0 to 100. A higher rank means a better match.

Working with image queries

This section describes how to create, run, and evaluate image queries.

Creating an image query: The following example shows a query string that searches for all images with average color red. "QbColorFeatureClass" represents the feature for average color.

```
String cmd = "QbColorFeatureClass color=<255, 0, 0>";
```

Running an image query: The datastore provides a method to create a query object. You use the 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 DXResultsDL collection.

```
Dim dsQBIC as new DXDatastoreQBIC
dsQBIC.connect "QBICSRV", "QBICUSER", "PASSWORD"
dsQBIC.openCatalog "DEMO", "QBIC0725"
```

```
Dim cmd as String
cmd = "QbColor color=<255,0,0>"
Dim query as DXImageQuery
Set query = dsQBIC.createQuery(cmd)
query.execute
```

```
Dim results as DXResults
Dim iter as DXSequentialIterator
Dim ddo as DXDDO
Set results = query.results
Set iter = results.createIterator
Do While iter.more
    Set ddo = iter.next
    'Do something with the DDO
```

Loop

```
dsQBIC.closeCatalog
dsQBIC.disconnect
```

Running an image query from the content server: DKDatastoreQBIC provides a method for running a query. The following example shows how to run an image query on the content server. Results are returned in a dkResultSetCursor object.

```
Dim dsQBIC as new DXDatastoreQBIC
dsQBIC.connect "QBICSRV", "QBICUSER", "PASSWORD"
dsQBIC.openCatalog "DEMO", "QBIC0725"
```

```
Dim cmd as String
cmd = "QbColor color=<255,0,0>"
Dim cursor as DXResultSetCursor
Set cursor = dsQBIC.execute(cmd)
```

```
Dim ddo as DXDDO
Do While cursor.isValid
    Set ddo = cursor.fetchNext
    'Do something with the DDO
```

Loop

```
cursor.destroy
dsQBIC.closeCatalog
dsQBIC.disconnect
```

Evaluating an image query from the datastore: DKDatastoreQBIC provides a method to evaluate a query. The following example shows how to evaluate an image query from the content server. Results are returned in a DXResultsDL collection.

```
Dim dsQBIC as new DXDatastoreQBIC
dsQBIC.connect "QBICSRV", "QBICUSER", "PASSWORD"
dsQBIC.openCatalog "DEMO", "QBIC0725"

Dim cmd as String
cmd = "QbColor color=<255,0,0>"
Dim results as DXResults
Dim iter as DXSequentialIterator
Dim ddo as DXDDO
Set results = dsQBIC.evaluate(cmd)
Set iter = results.createIterator
Do While iter.more
    Set ddo = iter.next
    'Do something with the DDO

Loop

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.

Query based on average color: A query based on average color consists of a feature name and its value. The following example shows how to search for all images based on average color red:

```
String cmd = "QbColorFeatureClass color=<255, 0, 0>";
```

Query based on color percentages: A query based on color percentages consists of a feature name and its value. The following example shows how to search for all images based on a histogram of three colors: 10% red, 50% blue, and 40% green.

```
String cmd = "QbColorHistogramFeatureClass ";
cmd += "histogram=<(10, 255, 0, 0),(50, 0, 255, 0),(40, 0, 0, 255)>";
```

Query based on color layout: A query based on color layout consists of a feature name and its value. The following example shows how to search for all images based on color layout described by an image in a file on the client:

```
String cmd = "QbDrawFeatureClass file=<client, \\patterns\pattern1.gif\>";
```

Query based on texture: A query based on texture consists of a feature name and its value. The following example shows how to search for all images based on the texture value provided by an image in a file on the client:

```
String cmd = "QbTextureFeatureClass file=<client, \\patterns\pattern2.gif\>";
```

Query with multiple features: You can specify a query with multiple features. The following example shows how to search for all images based on 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 average color is twice that of the texture:

```
String cmd = "QbColorFeatureClass color=<255, 0, 0> weight=2.0 and ";
cmd += "QbTextureFeatureClass file=<client, \\patterns\pattern2.gif\>";
```

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. Verify your environment settings before you run a query.

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.

```
Private Sub cout(str As String)
Results.Text = Results.Text & Chr$(13) & Chr$(10) & str
End Sub

Private Sub Run_Click()

itemId = "N2JJBERBQFK@WTVL"
partId = 13
Set dsDL = New DXDatastoreDL
Set xdo = New DXBlobDL
xdo.init dsDL
xdo.setPartId partId
xdo.setPid itemId

cout "Check if was indexed, if not then do indexing:"
Dim catflag As Boolean
catflag = xdo.isCategoryOf(DX_Indexed_Object)
cout "isCategoryOf indexed obj = " & catflag
If (catflag) Then
    cout "==alreay indexed, do getExtension to show informations:"
    Dim srcheng2 As Variant
    Set srcheng2 = xdo.getExtension("DXSearchEngineInfoDL")
    On Error GoTo Errors
    cout " srcheng2.getSearchEngine=" & srcheng2.getSearchEngine
    cout " srcheng2.getSearchIndex=" & srcheng2.getSearchIndex
    cout " srcheng2.getSearchInfo=" & srcheng2.getSearchInfo
    cout " srcheng2.getTextIndex=" & srcheng2.getTextIndex
    cout " srcheng2.getCatalog=" & srcheng2.getCatalog
    cout " srcheng2.getDataBase=" & srcheng2.getDataBase
    cout " srcheng2.getServerName=" & srcheng2.getServerName
    cout " srcheng2.getSearchClassId=" & srcheng2.getSearchClassId
    cout " srcheng2.getSearchTimestamp=" & srcheng2.getSearchTimestamp
    cout " xdo.retrieveObjectState=" & xdo.retrieveObjectState(DX_Indexed_Object)
Else
    cout "==not indexed, do new DXSearchEngingInfoDL to prepare:"
    Dim srcheng As New DXSearchEngineInfoDL
    srcheng.setSearchEngine "SM"
    srcheng.setSearchInfo "ENU"
    srcheng.setSearchIndex "TMMUF-TMINDEX"
    cout " set srcheng extension obj..."
    xdo.setExtension srcheng
    On Error GoTo Errors
    cout " about to setToBeIndexed..."
    xdo.setToBeIndexed
    On Error GoTo Errors
    cout " setToBeIndexed success..."
End If
End Sub
```

```

End If
    cout "isCategoryOf media obj=" & xdo.isCategoryOf(DX_Media_Object)
    cout "isCategoryOf indexed obj=" & xdo.isCategoryOf(DX_Indexed_Object)

Exit Sub
Errors:
    cout "Errors:" & str(Err.Number) & Err.Description

End Sub

```

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 that is 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 combined query result 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:

```

Dim dsDL as new DXDatastoreDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim dsTS as new DXDatastoreTS
dsTS.connectPort "TM", "", DX_TS_CTYP_TCPIP 'TM is a local alias for the TM server
'Create a parametric query
Dim pquery as String
pquery = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(DLSEARCH_Date > 1994));"
Dim pq as DXParametricQuery
Set pq = dsDL.createQuery(pquery)
'Create a text query
Dim tquery as String
tquery = "SEARCH=(COND=(Tivoli));OPTION=(SEARCH_INDEX=TMINDEX)"
Dim tq as DXTextQuery
Set tq = dsTS.createQuery(tquery)
'Create a combined query
Dim cq as new DXCombinedQuery
'Package the queries in DXNVPair as input parameters
Dim par(1 To 3) As New DXNVPair
par(1).Set DX_PARAM_QUERY, pq
par(2).Set DX_TEXT_QUERY, tq
par(3).setName DX_PARAM_END 'to signal the end of the parameter list
'Execute the combined query
cq.execute par
'Get the results
Dim res as DXResults
Set res = cq.result
If res Is Nothing Then
Else
    'Process the results
    ...
End If

```


The last if-statement is necessary to ensure that the DKResults object is not null.

Using a scope: If you have a DXResultsDL object that you want to use as the scope, you can slightly modify the previous example to insert the scope as an additional parameter:

```
Dim scope as DXResults
'assume that this is the scope
'initialized somewhere as a result
'of some parametric query
...
'Package the query in DXNVPair as input parameters
Dim par(1 to 4) as new DXNVPair
par(1).set DX_PARM_QUERY, pq
par(2).set DX_TEXT_QUERY, tq
par(3).set DX_SCOPE_DL, scope
par(4).setName DX_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 DX_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 workflow and workbasket functions

This section describes the workflow and workbasket functions.

Understanding the workflow service: The ActiveX APIs provide the workflow service for the Content Manager datastore, encompassing the workflow and workbasket functions that are available in the Folder Manager. 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 to route work through the process until completion.

The workflow model used 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 located in the same workflow more than once
- A document or folder can only be stored in one workflow at a time; however, workbaskets can be located in multiple workflows
- A document or folder can be stored in a workbasket that is not located in a workflow.

The `DXWorkflowServiceDL` class represents the workflow service of the Content Manager datastore. This class provides the capability to start, change, remove, route, and complete a document or folder in a workflow. Additionally, the `DXWorkflowServiceDL` class allows you to retrieve information about workbaskets and workflows.

The `DXWorkflowDL` and `DXWorkBasket` 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 methods. The connect method allows an application to connect to Content Manager. After you create a workflow service, you can run subsequent methods of the workflow service.

The following example shows how to connect to a Content Manager server named LIBSRVRN, using the user ID FRNADMIN and the password PASSWORD.

```
Dim dsDL As New DXDatastoreDL
Dim wfDL As New DXWorkflowServiceDL
wfDL.init (dsDL)

dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
...           ' do some work
dsDL.disconnect
```

The complete sample application from which this example was taken (`wfs.frm`, written in Visual Basic) is available in the `Cmbroot/Samples/activex/dl` directory.

Creating a workflow: The workflow service allows you to create a new workflow. The typical workflow creation process follows these steps:

1. Create an instance of `DXWorkflowDL`
2. Set the workflow name to GOLF
3. Set the workbasket sequence to NULL to indicate that this workflow contains no workbaskets
4. Set the privilege to **All Privileges**
5. Set the disposition to `DX_WF_SAVE_HISTORY`
6. Call the add method

The following example uses these steps to create a workflow. If you connect to the datastore as a normal user (`DX_SS_NORMAL`), you will not get the workflow that is defined after you connect. Therefore, this sample uses `DX_SS_CONFIG`.

```
Dim dsDL As New DXDatastoreDL
Dim wfDL As New DXWorkflowServiceDL
wfDL.init dsDL
dsDL.setOption DX_OPT_DL_ACCESS, DX_SS_CONFIG
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim newwf As New DXWorkflowDL
newwf.init wfDL
newwf.setName "GOLF"
```

```

newwf.setAccessList "All Privileges"
newwf.setHistoryDisposition DX_WF_SAVE_HISTORY
newwf.add
... ' do some work
dsDL.disconnect

```

The complete sample application from which this example was taken (wfs.frm, written in Visual Basic) is available in the Cmbroot/Samples/activex/dl directory.

Listing workflows: DKWorkflowServiceDL provides a method for listing the workflows in the system as shown in the following example. The list is returned in a sequential collection of DKWorkFlowDL objects.

```

Dim dsDL As New DXDatastoreDL
Dim wfDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"

Dim wfList1 As New DXSequentialCollection
Set wfList1 = wfDL.listWorkFlows
Dim pIter1 As New DXSequentialIterator
Set pIter1 = wfList1.createIterator

Dim pwf1 As DXWorkFlowDL
Do While pIter1.more
    Set pwf1 = pIter1.Next
    pwf1->retrieve();
    ... ' do some work
Loop
dsDL.disconnect

```

The complete sample application from which this example was taken (wfs.frm, written in Visual Basic) is available in the Cmbroot/Samples/activex/dl directory.

Creating a workbasket: The workflow service allows you to create a new workbasket. The typical workbasket creation process follows these steps:

1. Create an instance of DXWorkBasketDL
2. Set the workbasket name to Hot Items
3. Set the privilege to **All Privileges**
4. Call the add method

The following example uses these steps to create a workflow. If you connect to the datastore as a normal user (DX_SS_NORMAL), you will not get the workflow that is defined after you connect. Therefore, this sample uses DX_SS_CONFIG.

```

Dim dsDL As New DXDatastoreDL
Dim wfDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.setOption DX_OPT_DL_ACCESS, DX_SS_CONFIG
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim newwb As New DXWorkBasketDL
newwb.init 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 (wfs.frm, written in Visual Basic) is available in the Cmbroot/Samples/activex/dl directory.

Listing workbaskets: DKWorkflowServiceDL provides a method for listing the workbaskets in the system as shown in the following example. The list is returned in a sequential collection of DKWorkBasketDL objects.

```
Dim dsDL As New DXDatastoreDL
Dim dsDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"

Dim wbList1 As New DXSequentialCollection
Set wbList1 = wfDL.listWorkFlows
Dim pIter1 As New DXSequentialIterator
Set pIter1 = wbList1.createIterator

Dim pwb1 As DXWorkBasketDL
Do While pIter1.more
    Set pwb1 = pIter1.Next
    pwb1->retrieve();
    ... ' do some work
Loop
dsDL.disconnect
```

The complete sample application from which this example was taken (wfs.frm, written in Visual Basic) is available in the Cmbroot/Samples/activex/dl directory.

Listing items in a workflow: The workflow service provides a method 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 DXString objects.

```
Dim dsDL As New DXDatastoreDL
Dim dsDL As New DXWorkFlowServiceDL
wfDL.init dsDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim itemIDWF As String
itemIDWF = "H17MOPALUPFQ1U47"
Dim wf As New DXWorkFlowDL
wf.init wfDL
wf.setID itemIDWF
wf.retrieve

Dim pColDoc1 As New DXSequentialCollection
Set pColDoc1 = wf.listItemIDs
Dim pIter1 As New DXSequentialIterator
Set pIter1 = pColDoc1.createIterator

Dim DocID As String
Do While pIter1.more
    DocID = pIter1.Next
    ... ' do some work
Loop
dsDL.disconnect
```

The complete sample function from which this example was taken (wfs.frm, written in Visual Basic) is available in the Cmbroot/Samples/activex/dl directory.

Executing a workflow: DKWorkflowServiceDL provides methods 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 EP8L80R9MHH##QES
- Use a valid workflow ID instead of H17MOPALUPFQ1U47
- Use a valid workbasket ID instead of E3PP1UZ0ZUFQ1U3M

```
Dim dsDL As New DXDatastoreDL
Dim dsDL As New DXWorkFlowServiceDL
wfdL.init dsDL
dsDL.connect "LIBSRVRN", "FRNADMIN", "PASSWORD"
Dim itemID As String
Dim itemIDWF As String
Dim itemIDWB As String
itemID = "EP8L80R9MHH##QES"
itemIDWF = "H17MOPALUPFQ1U47"
itemIDWB = "E3PP1UZ0ZUFQ1U3M"
wfdL.startWorkFlowItemInFirstWorkBasket itemID, itemIDWF, TRUE,
                                         DX_WIP_DEFAULT_PRIORITY
...                                     ' do some work
wfdL.routeWipItem itemID, itemIDWB, TRUE, DX_NO_PRIORITY_CHANGE
...                                     ' do some work
wfdL.completeWorkFlowItem itemID
dsDL.disconnect
```

The complete sample function from which this example was taken (`wfs.frm`, written in Visual Basic) is available in the samples directory.

Working with Domino Extended Search (DES)

Domino Extended Search (DES) allows you to query and retrieve documents from:

- Lotus Notes databases
- NotesPump databases
- File systems
- Web search engines

With DES you can:

- Connect and disconnect from one or more DES servers
- List DES servers
- List databases and fields
- Perform searches using Generalized Query Language (GQL)
- Retrieve documents

Restriction: DES does not support:

- Adding, updating, and deleting documents
- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

All DES features are accessed and controlled by the DES 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 DES servers

To provide access to multiple DES servers, you can create a file named `cmbdes.ini` that contains the server information. Store this file in `x:\CMBROOT` (where `x` is the drive letter). The `cmbdes.ini` file must contain one line, in the following format, for each server, :

```
DATASOURCE=TCP/IP address;PORT=port number
```

TCP/IP is the TCP/IP address of the DES server and the *port number* is the port number defined for server access (for example: `PORT=80`).

Listing databases and fields

When you build a query to search a DES server, you must know the available database and field names that are available. The `DXDatastoreDES` object provides the `listEntities` method to list the databases and the `listEntityAttrs` method to list the fields for each database. The following example shows an example of a list of databases and their fields.

```
cout "list entities"

Set pCol = dsDES.listEntities
Set pIter = pCol.createIterator

Dim pCol2 As DXSequentialCollectionDES
Dim pIter2 As DXSequentialIteratorDES

Dim pEnt As DXDatabaseDefDES
Dim pAttr As DXFieldDefDES

i = 0
Do While pIter.more
    i = i + 1

    Set pEnt = pIter.Next
    cout "database name [" & i & "] - " & pEnt.getName
    cout "dispname: " & pEnt.getDisplayName
    cout "helptext: " & pEnt.getHelpText
    cout "lang: " & pEnt.getLanguage
    cout "NumVals: " & pEnt.getNumVals
    cout "datatype: " & pEnt.getDataType
    cout "searchable:" & pEnt.isSearchable
    cout "retrievable" & pEnt.isRetrievable

    cout " list attributes for " & strDBName & " database name"

    Set pCol2 = dsDES.listEntityAttrs(pEnt.getName)
    Set pIter2 = pCol2.createIterator

    j = 0
    Do While pIter2.more
        j = j + 1
        Set pAttr = pIter2.Next
        cout " Attribute name [" & j & "] - " & pAttr.getName
        cout " datastoreName " & pAttr.datastoreName
        cout " datastoreType " & pAttr.datastoreType
        cout " type " & pAttr.getType
        cout " size " & pAttr.getSize
        cout " nullable " & pAttr.isNullable
        cout " precision " & pAttr.getPrecision
        cout " scale " & pAttr.getScale
        cout " string type " & pAttr.getStringType
        cout " display name " & pAttr.getDisplayName
        cout " help text " & pAttr.getHelpText
        cout " language " & pAttr.getLanguage
        cout " isQueryable " & pAttr.isQueryable
        cout " isRetrievable " & pAttr.isRetrievable
```

```

Loop
    cout " " & j & " attributes listed for " & pEnt.getName & " database name"
Loop
cout i & " entities listed\n"

dsDES.disconnect
cout "datastore disconnected"

cout "done ..."

```

The complete sample application from which this example was taken (TListCatalogDES.frm) is available in the Cmbroot/Samples/activex/des directory.

Using Generalized Query Language (GQL)

DES uses the Generalized Query Language (GQL) to perform searches. Table 38 contains GQL expression examples.

Table 38. GQL expressions

GQL expression	Description
"software"	Search for documents containing the word software
(TOKEN:WILD "exec*")	Search for documents containing any word beginning with exec
(AND "software" "IBM")	Search for documents containing 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

DES uses the query type DX_DES_GQL_QL_TYPE which 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])]

```

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 results expected are 5.

```

myQuery = "SEARCH=(DATABASE=(Notes Help);"
myQuery = myQuery & "COND=(EQ \"View\" \"How Do I?\");"
myQuery = myQuery & "OPTION=(MAX_RESULTS=5)"

```

This example runs a GQL query for DES. After the query runs, the dkResultSetCursor object returns the results:

```

...
Private Sub Execute_Click()
Dim myQuery As String
Dim paramList(0) As DXNVPairDES
Dim vParamList As Variant

```

```

vParamList = paramList()
Dim i As Long
i = VarType(vParamList)
'Replace the following query string with your own -
'or get text from Query text box - remember to
'strip out the line feed and carriage control characters
'myQuery = QueryString.Text
myQuery = "SEARCH=(DATABASE=(DES files));"
myQuery = myQuery & "COND=((IN ""Doc$Content"" ""a""));";
myQuery = myQuery & "OPTION=(MAX_RESULTS=2;TIME_LIMIT=15)"
QueryString.Text = myQuery
Dim rsCursor As DXResultSetCursorDES
Set rsCursor = dsdes.Execute(myQuery, DX_DES_GQL_QL_TYPE)
On Error GoTo HandleError
Dim res As New ProcessRes 'create text box
res.ProcessResults rsCursor
res.Show
Exit Sub
...

```

A complete example of the application from which this sample was taken (TExecuteDES.frm) is available in the Cmbroot/Samples/activex/dl directory.

DDO properties in DES

A DDO in DES always has the type DX_DES_DOCUMENT. To get the DDO object type:

```

Object obj = ddo.getPropertyByName(DX_DL_PROPERTY_TYPE);
short type = ((Short) obj).shortValue();

```

Creating PIDs in DES

The PID contains specific document information. The object type identifies the database where the document was found. To create a PID, use the database name, followed by the logical (l) or character | character and the document ID. For example:

```
database name|documentId ()
```

For more information on PIDs, see “Understanding persistent identifiers (PID)” on page 11 and “Creating a persistent identifier (PID)” on page 27.

DES document contents

Each item in the DDO represents 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:

- DX_CM_VSTRING
- DX_CM_FLOAT
- DX_CM_XDOOBJECT
- DX_CM_DATE
- DX_CM_SHORT

Collection

When a field has multiple values, the field name is in the item name. The item value is a DXSequentialCollection object. The property can be DX_CM_COLLECTION, or DX_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. You can use DKPARTS to store the document Uniform Resource Locator (URL) information. DKPARTS can also contain an XDO with its contents as a string representing the document URL.

This example processes the contents of a DDO:

```
...
cout "query string " & cmd
cout "create query"
Dim pQry As DXDESQueryDES
Set pQry = dsDES.createQuery(cmd, DX_DES_GQL_QL_TYPE)
cout "executing query"
pQry.execute
cout "query executed"
cout "get query results"

Dim presults As DXResultsDES
Set presults = pQry.result

processResults presults

dsDES.disconnect
cout "datastore disconnected"
End Sub

Private Sub processResults(results As DXResultsDES)
    Dim iter As DXSequentialIteratorDES
    Set iter = results.createIterator

    cout " cardinality of results " & results.cardinality

    Dim item As DXDDODES

    Do While iter.more
        Set item = iter.Next

        If item.getPropertyByName("item-type") = DX_DES_DOCUMENT Then
            cout " Item is a document, number of data item "
                & item.dataCount
            For i = 1 To item.dataCount
                dataName = item.getDataName(i)
                cout "    data item[" & i & "] = " & dataName & ",
                    \t\t value " & item.GetData(i)
                usCount = item.dataPropertyCount(i)
                For k = 1 To usCount
                    cout " property " & k & " " & item.getDataProperty(i, k)
                Next
            Next
        Else
            cout " Item is not recognized "
        End If
    Loop
...

```

A complete example of the application from which this sample was taken (TQueryDES.frm) is available in the samples directory.

Retrieving a document

To retrieve a document from a DXDatastoreDES 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.

Retrieving a BLOB

To retrieve a BLOB from a DXDatastoreDES object, you must know the name of the database, the ID of the document that contains the BLOB, and the name of the field that contains the BLOB. You must also associate the DDO to a content server and establish a connection.

In the following example, the database named DES files contains an HTML file named D:\desdoc\README.html. The field that contains the HTML file is named Doc\$Content. The sample code retrieves the HTML file and saves it as D:\DESReadme.html.

```
Private Sub Retrieve_Click()
    Set xdo = New DXBlobDES
    xdo.init dsdes
    Set pid = New DXPidXDODES
    pid.setDocId ("d:\desdoc\README.html")
    pid.setDatabaseName ("DES files")
    pid.setPrimaryId ("DES files\D:\desdoc\README.html")
    xdo.setPidObject pid
    xdo.setFieldName ("Doc$Content")
    cout "===before retrieve==="
    xdo.Retrieve ("c:\Temp\DESReadme.html")

    On Error GoTo Errors
    cout " retrieve successfully"

Exit Sub
```

A complete example of the application from which this sample was taken (TxdoRetrieveDES.frm) is available in the samples directory.

Associating MIME types with documents

DES does not directly support 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 DES 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 DES

When you create federated queries, the syntax used in DES is similar to SQL syntax. The federated query expressions are converted to GQL syntax before they are submitted to DES. Because SQL and GQL grammar have differences however, only a subset of the SQL grammar is supported by Enterprise Information Portal.

Table 39 summarizes the SQL to GQL conversion of the supported comparison and logical operators.

Table 39. SQL and GQL operators

SQL operator	GQL operator
AND	AND
OR	OR
NOT	not supported
IN	not supported
BETWEEN	BETWEENE
EQ	EQ
NEQ	not supported
GT	GT
LT	LT
LIKE	not supported
GEQ	GTE
LEQ	LTE
NOTLIKE	not supported

Table 39. SQL and GQL operators (continued)

NOTIN	not supported
NOTBETWEEN	not supported

Chapter 10. Using the Dynamic Page Builder

This chapter describes how to use the Dynamic Page Builder (DPB) to create a Web application to make Content Manager data available over the Internet or an intranet. DPB is a set of application programming interfaces (APIs) that lets you build applications that retrieve Content Manager text and image content. You can also dynamically create documents, including video streams, for display in a Web browser.

The DPB requires IBM's Net.Data as a front-end parsing and formatting tool. Net.Data combines the functions of Common Gateway Interface (CGI), Connection Manager, and Web page generation. Net.Data has a wizard to help you access Net.Data function from your application.

See the Net.Data documentation for more information about using Net.Data. To view the documentation from the Windows NT desktop, go to **Start → Programs → Net.Data → Net.Data Documentation**.

Configuring the Dynamic Page Builder with Net.Data

Net.Data uses an initialization file (db2www.ini) to customize settings and search paths. The file location, contents and syntax depends on your operating system and Web server. The following list shows an example of a Net.Data path:

ICS for AIX	/usr/lpp/internet/server_root/pub
NCSA for AIX	/usr/httpd/htdocs
IIS for Windows NT	\inetpub\wwwroot
ICS for Windows NT	\www\html

The Dynamic Page Builder uses the following three special path statements and one environment statement:

ENVIRONMENT (DTW_DLDPB)

MACRO_PATH

INCLUDE_PATH

HTML_PATH

The ENVIRONMENT statement (DTW_DLDPB)

The following statement is for the Dynamic Page Builder on Net.Data; it provides the server setting for the cliette program and macro files:

```
ENVIRONMENT(DTW_DLDPB) dtwdl dpb (IN DATABASE, LOGIN, PASSWORD,  
OUT RETURN_CODE) CLIETTE "DTW_DLDPB:$(DATABASE)"
```

dtwdl dpb

is a required, shared DLL providing the non-cliette access to Content Manager. Not currently supported by Dynamic Page Builder.

(IN DATABASE, LOGIN, PASSWORD, OUT RETURN_CODE)

is the variable setting for macro files, and it provides the capability to

allow different users to log on to the cliette. If you want to run in the single user mode, remove this line from db2www.ini and the macro files to improve performance.

The MACRO_PATH statement

The MACRO_PATH identifies one or more directories to search for macro files. The syntax of the MACRO_PATH is:

```
MACRO_PATH [=] path[;path;..]
```

The INCLUDE_PATH statement

The INCLUDE_PATH identifies one or more directories in which to search for a file specified on the %INCLUDE statement in a macro file:

```
Net.Data db2www.ini file
INCLUDE_PATH=c:\inetpub\wwwroot
Macro file
%INCLUDE "dlheader.html"
```

The HTML_PATH statement

This is the special path variable. When Content Manager returns a large object, it saves all of the objects to the tmplobs directory. tmplobs is the subdirectory under the first directory name specified in the HTML_PATH statement.

The syntax of the HTML_PATH statement is similar to the MACRO_PATH statement:

```
HTML_PATH [=] path[;path;..]
```

Dynamic Page Builder functions

This section describes the Dynamic Page Builder APIs, the parameters passed to them, the parameters passed through in-line data, and the variables to be defined.

API functions

Table 40 shows the terms that Dynamic Page Builder reserves for API functions. The online API reference defines API function terms.

Table 40. API function terms

Term	Definition
DP_DLConnect	Connect to Content Manager
DP_DLListServer	List all of the Content Manager servers
DP_TMConnect	Connect to the text search server
DP_QBConnect	Connect to the image search server
DP_DisConnect	Disconnect from a search engine
DP_IndexClass	Get the index classes
DP_IndexAttribute	Get the index attributes
DP_QBOpenCatalog	Access the QBIC catalog
DP_QBCloseCatalog	Exit the QBIC catalog
DP_QBListCatalog	List the QBIC catalogs
DP_PMQuery	Perform a parametric search
DP_PMQuerySQL	Perform a parametric search using DB2 access; the parameters are the same as DP_PMQuery()

Table 40. API function terms (continued)

Term	Definition
DP_PMQuerySQL2	Perform a parametric search using DB2 access; the parameters are the same as DP_PMQuery() with one exception: DP_ATTRIBUTE_NAME
DP_TMQuery	Perform a text search
DP_TMQuerySQL	Perform a text search using DB2 access; the parameters are the same as DP_TMQuery()
DP_QBQuery	Perform an image search
DP_QBQuerySQL	Perform an image search using DB2 access; the parameters are the same as DP_QBQuery()
DP_TMQuery2	Perform a text search while connected with Text Engine
DP_QBQuery2	Perform an image search while connected
DP_CBQuery	Perform a combined search
DP_CBQuerySQL	Perform a combined search using DB2 access; the parameters are the same as DP_CBQuery()
DP_Folders	Get a folder collection
DP_Parts	Get a parts collection
DP_MetaData	Get the metadata
DP_Retrieve	Get a data object
DP_WorkflowService	Access the workflow services class
DP_ListWorkFlows	List the workflows
DP_ListWorkBaskets	List the workbaskets in the system or within the specified workflow
DP_ListItems	List the items in the specified workflow or workbasket
DP_WorkFlowDL	Access the DKWorkFlowDL class
DP_WorkBasketDL	Access the DKWorkBasketDL class
DP_Object	Execute an object
DP_SQLTrigger	Run SQL through the input parameter DP_SQLSTATEMENT

Input parameters

Table 41 describes the API function input parameters and application. Each function allows only one input parameter.

Table 41. API function input parameter application and description

Parameter	Applied at	Description
CONNECT_STRING	DP_DLConnect	Connection string for Content Manager
CONNECT_STRING	DP_TMConnect	Connection string for the TM
DP_DATASTORE	DP_DisConnect	Content server identifier (DL for Content Manager, TM for Text Search, QBIC for image search)
PMQUERY_STRING	DP_PMQuery	Parametric search query string (DL for Content Manager, TM for Text Search)

Table 41. API function input parameter application and description (continued)

Parameter	Applied at	Description
PMQUERY_STRING	DP_CBQuery	Parametric search query string (DL for Content Manager, TM for Text Search)
TMQUERY_STRING	DP_TMQuery	Text search query string
TMQUERY_STRING	DP_CBQuery	Text search query string
INDEX_CLASS	DP_IndexAttribute	Content Manager index class name
DP_THUMBID	DP_Retrieve	Part ID of item retrieved
DP_DESCRIBEID	DP_Parts	Part ID of TOC Part
PID	DP_Folders	Part Item ID
PID	DP_Parts	Part Item ID
PID	DP_MetaData	Part Item ID
PID	DP_Retrieve	Part Item ID
DP_DESCRIBEID	DP_Retrieve	Part ID of TOC Part; file name for thumbnail image

An example of a function call with input parameters is:

```
%FUNCTION(DTW_DLDPB) DP_DisConnect(DP_DATASTORE)
{ DP_DisConnect; %}
```

Inline data

You can also pass parameters to Dynamic Page Builder using inline data as shown in Table 42.

Table 42. Dynamic Page Builder inline data

Parameter	Term	Definition
For DP_DLConnect(), DP_TMConnect() and DP_QBConnect()	CONNECT_STRING	Connection string for Content Manager
For DP_DisConnect()	DP_DATASTORE	Content server identifierDL for Content Manager, TM for text search, QBIC for image search
For DP_PMQuery() and DP_CBQuery()	DP_THUMBID	Part ID of retrieved item
	DP_DESCRIBEID	Part ID of TOC part; file name for thumbnail image
PMQUERY_STRING	Parametric search query string	Ignores remaining parameters if PMQUERY_STRING is supplied
INDEX_CLASS	Content Manager index class names	
PM_MAX_RESULTS	Maximum parametric search results	
PM_CONDITION	Query expression	
TYPE_FILTER	Filter (DOC, FOLDER, FOLDERDOC)	

Table 42. Dynamic Page Builder inline data (continued)

Parameter	Term	Definition
DP_ORDERBY	Order by attribute name; default is ascending.	Add an exclamation point (!) in front of the attribute name to perform descending sorting. For example, !DLSEARCH_Author
DP_MAX_DISPLAY	Maximum display results	
For DP_TMQuery() and DP_CBQuery()	DP_MAP2DL=yes/no	Displays Content Manager attribute information
	TMQUERY	Text search string
	TM_CONDITION	Query expression
	TM_MAX_RESULTS	Maximum text search results
	SEARCH_INDEX	TM index class name
	DP_TIMELIMIT	Text search time limit
	DP_ORDERBY	Perform a parametric search
	DP_MAX_DISPLAY	Maximum display results
	DP_THUMBID	Part ID display for the part retrieved from Content Manager. Returns Thumbnail file name in the column of the header name DP_THUMBID, if inline parameters provided correct DP_THUMBID
	DP_DESCRIBEID	TOC Part ID. Provides original file name for thumbnail. Generates random file name if it finds incorrect DP_DESCRIBEID for thumbnail
	DP_MAP2DL=yes/no	Return DP_PMQuery output format instead. (yes/no)
For "DP_QBQuery" and "DP_CBQuery"	QBIC_FEATURE_NAME	QBIC feature class
	QBIC_FEATURE_VALUE	QBIC search value
	DP_THUMBID	Text search query string
	DP_DESCRIBEID	Text search query string
	DP_MAP2DL	Text search query string
	DP_MAX_DISPLAY	Text search query string
	DP_TMQuery2()	Text search query string plus one additional parameter, CONNECT_STRING, to connect to Text Search Engine;

Table 42. Dynamic Page Builder inline data (continued)

Parameter	Term	Definition
	DP_QBQuery2()	Takes the same parameters as DP_QBQuery() plus CONNECT_STRING to connect to image search server; also, QBIC_DATABASE and QBIC_CATALOG to open the QBIC catalog.
For DP_MetaData()	PID	Parts item ID
For DP_Retrieve()	PID	Parts item ID
	DP_THUMBID	Part ID of item retrieved
	DP_IMAGENAME	Returns as file name (yes/no). Default is yes. The file content will be returned if set to no
	DP_DESCRIBEID	Part ID of TOC part; file name for thumbnail image
For DP_Parts()	PID	Parts item ID
	DP_DESCRIBEID	Part ID of TOC part; file name for Thumbnail image
For DP_Folders()	PID	Parts item ID
	DP_THUMBID	Part ID of item retrieved
	DP_DESCRIBEID	Part ID of TOC Part; file name for Thumbnail image
For "DP_QBListCatalog()"	QBIC_DATABASE	QBIC database name
For "DP_QBOpenlog()"	QBIC_DATABASE	QBIC database name
	QBIC_CATALOG	QBIC catalog name
	DP_TMQuery2	Combines DP_TMConnect(), DP_TMQuery(), and DP_DisConnect() as one API that reduces the HTTP traffic. It takes the same parameters (input/inline) as DP_TMQuery() with additional inline parameter (CONNECT_STRING) for TM connection.

Table 42. Dynamic Page Builder inline data (continued)

Parameter	Term	Definition
	DP_QBQuery2	Similar to DP_TMQuery2. This is the combined API that represents DP_QBConnect, DP_QBOpenCatalog, DP_QBQuery, DP_QBCloseCatalog, and DP_DisConnect in one API call. It takes the same parameters as DP_QBQuery() with additional inline parameters CONNECT_STRING for QBIC connection, QBIC_DATABASE, and QBIC_CATALOG for QBIC open catalog.

Variable definition

Define the variables listed in the db2www.ini file so that you can use them in the macro files:

```
ENVIRONMENT(DTW_DLDAPB) dtwdldpb (IN DATABASE, LOGIN, PASSWORD,
START_ROW_NUM, OUT RETURN_CODE) CLIETTE "DTW_DLDAPB:$(DATABASE)"
```

DATABASE

Content Manager server name

LOGIN

Content Manager user ID

PASSWORD

Content Manager password

START_ROW_NUM

Starting row number for display in report section

RETURN_CODE

Return code from DP API

Special output variable

DP API assigns the Number of Rows Affected (NRA) to the second title column of the first returned row to display the total NRA by DP_PMQuery() and DP_CBQuery.

This is an example of a report section of DP_PMQuery():

```
%FUNCTION(DTW_DLDAPB) DP_PMQuery(in...)
...
%REPORT {
  Total Number of Hits $(N2)
  %ROW{
    ...
  }
%}
%}
%}
```

Developing a Net.Data macro for the Dynamic Page Builder

This section shows two sample macro files that help illustrate the Dynamic Page Builder macro file elements.

Sample macro 1

Sample macro 1 retrieves all of the index class information from the Content Manager server. The sample macro displays the index class information as a list. From this list, you select an index class and retrieve its index attributes. A second HTML page displays the index attributes.

This sample macro file has four sections:

- The definition section
- The function definition section
- The HTML input section
- The HTML report section

There is no limit on the HTML sections in a macro file.

To build a macro, add a macro statement to be processed dynamically at the server. Two examples are: @DP_IndexClass in the HTML_INPUT section and @DP_IndexAttribute in the HTML_REPORT section.

Examine the sample macro, section by section, to understand the macro execution:

Definition section

```
%(----- Define Section -----%}  
%DEFINE{  
    DLTable = %TABLE(100)  
    PROGRAM = "/cgi-bin/db2www.exe/frndp2.d2w"  
%}
```

After you define one or more variables in a single definition section, the variables can be referenced anywhere in the macro file using the syntax: \$(Variable)

Function definition section

```
%(----- FUNCTION Definition Section -----%}  
%FUNCTION(DTW_DLDPB) DP_IndexClass (OUT table) {  
    <← This function has no need for an →  
    <← input parameter, but will return →  
    <← the result in a table.           →  
  
    DP_IndexClass;  
  
    <← This function invokes the Dynamic Page Builder API (DP_IndexClass).→  
  
    %REPORT{ %row{ <OPTION value=$(V1)>$(V1) %} %}  
    <← The formatted return result is →  
    <← set to be the option value of the →  
    <← selection box.                 →  
  
    %MESSAGE {  
    <← error detection.           →  
        -160: "Error: 160: Catalog lookup failure" : exit  
    %}  
%}  
%FUNCTION(DTW_DLDPB) DP_IndexAttribute (in idx, OUT table) {  
    DP_IndexAttribute;  
    <← This function invokes the Dynamic →
```

```

<-- Page Builder API (DP_IndexAttribute) -->
    INDEX_CLASS = $(Index_Class);
<-- pass index class from inline data -->

    %REPORT {
<-- format return result -->
        %row {
<-- loop through each row -->
            <TR>
<-- Table row -->
                <TD> $(ROW_NUM) </TD>
<-- display row number -->
                <TD> $(V1) </TD>
<-- display 1st column -->
            </TR>
        %}
    %}

%MESSAGE {
    -162:    "Error: 162: Catalog lookup failure" : exit
    %}
%}

```

This function definition section contains two function declarations. The first, `DP_IndexClass`, is the Dynamic Page Builder API "get" index class. It takes no input parameters and stores the return result in a single column table.

```

    %REPORT{ %row{ <OPTION value=$(V1)>$(V1)
    %}
%}

```

The return result is assigned to `<OPTION value=>` format, and builds the items of the selection box for you to pick the index class and display the index attributes at the second HTML page. The second function definition `DP_IndexAttribute` is the Dynamic Page Builder API "get" index attribute. It takes the single parameter `INDEX_CLASS` and returns the results in a single column table. The passing of input parameters as inline data is available in most of Dynamic Page Builder APIs. In the sample macro, `INDEX_CLASS` is passed through inline data, not input parameters. *Inline data will be ignored if both input parameters and inline data are provided.*

HTML input section

```

%{----- HTML Input Section -----%}

%HTML_INPUT{
<-- Identifies the name of this section -->
<HTML>
    <FORM METHOD=POST ACTION="$(PROGRAM)/report">
<-- Form tag with variable substitution; -->
<-- this will invoke the $(PROGRAM)/report -->
<-- section when this form is submitted. -->
        <center>
            <h2> Result of Index Class in Content Manager server </h2>
            <p> Select the categories for the attributes associated. <br><p>

            <TABLE border=1 width=50%>
            <TR>
            <TD align=center> <SELECT NAME="Index_Class" SIZE=8 WIDTH=80>
                @DP_IndexClass(DLTable)
<-- This line contains a call to DPP -->
            </SELECT> </TD>
            </TR>
            </TABLE>
            <BR><BR>
        </center>
    </FORM>
%}

```

```

        <TD> <INPUT TYPE="submit" NAME="get_IndexAttr"
            VALUE="Get Attribute">
        </TD> </TR>
    </CENTER>
</FORM>
</HTML>
%}

```

The entire HTML section is surrounded by the HTML section identifier:
 %HTML_INPUT { ... %} INPUT identifies the name of this section, which has been used in the web address also: <http://www.ibm.com/cgi-bin/db2www/sample.d2w/INPUT>

This section contains an example of a function call. The expression of @DP_IndexClass(DLTable) is a call to the Dynamic Page Builder API. This function is defined in the function definition section described in "Dynamic Page Builder functions" on page 394. The result of the DP_IndexClass function is inserted into the HTML text in the same location as the @DP_IndexClass() expression.

Figure 61 shows an example of what the input section looks like in the sample application.

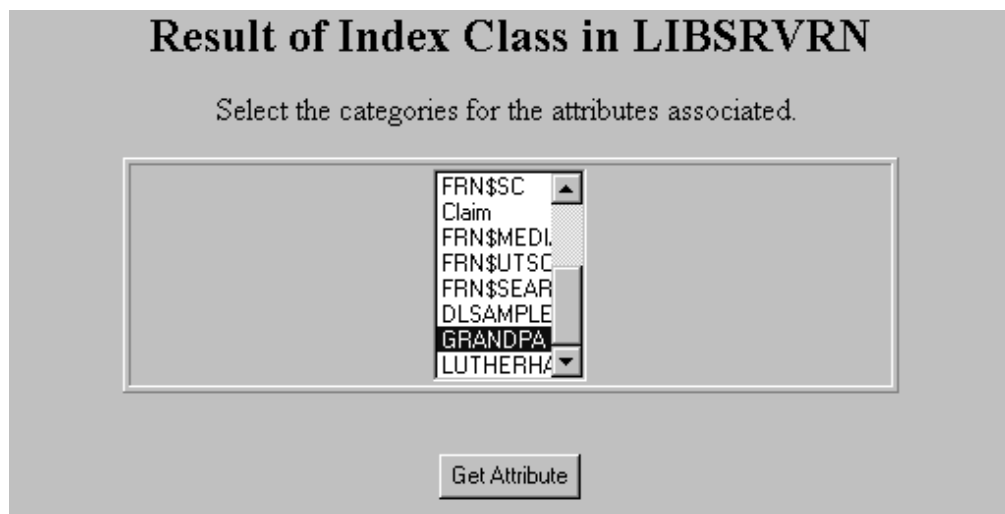


Figure 61. Index Class result

HTML report section

```

%{----- HTML Report Section -----}%
%HTML_REPORT{
<HTML>
    <CENTER>
        <h2>Index Attribute of $(Index_Class)</h2>
        ← Substitution from INPUT section →
    <TABLE cellpadding=1 cellspacing=5 border=1 width=80%>
        <TR>
            <TH>Row Number</TH>
            <TH>Index Attribute</TH>
        </TR>
        @DP_IndexAttribute(Index_Class,DLTable)
        ← call to a function →
    </TABLE>
    </CENTER>
</HTML>
%}

```

Like the INPUT section, this section is standard HTML, enhanced with a macro statement that substitutes a variable and a function call.

The Index_Class variable is substituted into the title statement. DP_IndexAttribute is the call to the previously defined function. It passes the content of the variable Index_Class, which was selected from the INPUT form (see Figure 61 on page 402, shows the category Grandpa selected. Figure 62 shows the HTML report listing the 12 index attributes of Grandpa.

Index Attribute of GRANDPA	
Row Number	Index Attribute
1	DLSEARCH_Author
2	DLSEARCH_Publisher
3	DLSEARCH_DocType
4	DLSEARCH_Date
5	DLSEARCH_Title
6	DLSEARCH_Subject
7	DLSEARCH_Authors
8	DLSEARCH_Editors
9	DLSEARCH_Source
10	DLSEARCH_Volume
11	DLSEARCH_Issue
12	DLSEARCH_Relation

Figure 62. Attributes of index class Grandpa

DP_IndexAttribute also accepts inline parameters, as shown in the following statement:

```
@DP_IndexAttribute(""  
,DLTable)
```

DP_IndexAttribute requires a variable defined using inline data in the function definition section:

```
%FUNCTION(DTW_DLDPB) DP_IndexAttribute (in idx, OUT table) {  
    DP_IndexAttribute;  
    INDEX_CLASS = $(Index_Class);  
←      Index_Class referenced here      →  
    ...  
%}  
%(------ HTML Report Section -----%)  
%HTML_REPORT{  
<HTML>  
    <CENTER>  
    <h2>Index Attribute of $(Index_Class)</h2>  
    <TABLE cellpadding=1 cellspacing=5 border=1 width=80%>  
    <TR>  
    <TH>Row Number</TH>  
    <TH>Index Attribute</TH>  
    </TR>  
    @DP_IndexAttribute(Index_Class,DLTable)
```

```

        </TABLE>
      </CENTER>
    </HTML>
  %}

```

Sample macro 2

Sample macro 2 performs a parametric search and returns a report that includes hyperlinks to the returned files.

This macro file consists of three major sections:

- Definition section
- The function definition section
- The HTML Input Section

Sample Macro 2 calls the HTML input section in the program with the PPID parameter set to Not NULL.

Definition section

You need the DATABASE, LOGIN, and PASSWORD variables in the Define section if they are defined in the db2www.ini file as: ENVIRONMENT(DTW_DLDPB) dtwdldpb (IN DATABASE, LOGIN, PASSWORD, OUT RETURN_CODE) CLIETTE "DTW_DLDPB:\$(DATABASE)".

```

%{----- Define Section -----%}
%DEFINE{
  DATABASE = "LIBSRVRN"
  <---      Library server name      --->
  LOGIN    = "FRNADMIN"
  <--- User logon ID --->
  PASSWORD = "PASSWORD"
  <---      User logon password      --->
  d1Result = %TABLE(ALL)
  <---      Result set table        --->
  TOC_PART = "2"
  <---      Part number of description part --->
  THUMBID  = "5"
  <--- Part number to display --->
  PROGRAM  = "/cgi-bin/db2www.exe/frndp59.d2w"
  qst      = "SEARCH=(INDEX_CLASS=GRANDPA,MAX_RESULTS=10);"
  <---      Parametric Query string  --->
%}

```

Function definition section

This function definition section contains two function declarations. The first, DP_Parts, is the Dynamic Page Builder API declaration for retrieving collections. It retrieves all of the individual parts in the document and stores them in the tmplobs subdirectory of the HTML root directory. It has one input parameter, part item ID and stores the returned result in a five-column table.

\$(V1) Part item ID

\$(V2) File size

\$(V3) File type

\$(V4) File name

\$(V5) Part ID

DP_PMQuery is the Dynamic Page builder API for processing parametric queries. It has one input parameter, PMQUERY_STRING, and returns the results in the variable columns in the five-column table.

\$(V1) Part item ID

\$(V2) Filter (Document, Folder, or Unknown)

\$(V3) DKFolder. In the results, you might receive the folder collection after the Parametric query. In this case, \$(V3) is set to YES to indicate that there is a folder collection, and that you need to call DP_Folder() to process it. \$(V3) can be blank to indicate that no folder collection is associated with it. \$(V3) can also be the file name of a thumbnail image if DP_THUMBID is provided and no folder collection is found.

\$(V4) DKParts. Setting \$(V4) to yes indicates that there is an associated parts collection. You can call DP_Parts() to retrieve all of the parts within the current parts collection.

\$(V5)...\$(Vn)

Values of index attributes.

```
%(----- FUNCTION Definition Section -----%)
```

```
%FUNCTION(DTW_DLDAPB) DP_Parts (in PID, OUT table) {
    DP_Parts;
    <--          Invoke DP_API          -->
    DP_DESCRIBEID=$(TOC_PART);
    <--          Assign TOC part number  -->
    %REPORT{
        %ROW {
            <--          Loop through each row          -->
                %IF (ROW_NUM == "3")
                    <FRAME SRC="$(V4)">
                %ENDIF
            <--          If the 3rd row, display the 4th column  -->
        %}
    %}

%FUNCTION(DTW_DLDAPB) DP_PMQuery (in PMQUERY_STRING,OUT table) {
    DP_PMQuery;
    <-- Invoke DP_API          -->
    DP_THUMBID=$(THUMBID);
    <-- Assign part number to be shown          -->
    DP_DESCRIBEID=$(TOC_PART);
    <-- Provide file name in TOC part for thumbnail;-->
    <-- random file name will be          -->
    <-- generated if DP_DESCRIBEID does          -->
    <-- not provide, or can't find the TOC part. -->
    %REPORT {
        %ROW {
            @DTW_DIVREM(ROW_NUM,"3","5",REM)
            <-- Get remainder of dividing "3";          -->
            <-- For showing 3 records per row          -->
            %IF (REM == "1")
                <-- Remainder = "1"          -->
                <TR>
                <-- New table row          -->
                %ENDIF
                <TD WIDTH=33.33% ALIGN=CENTER> <IMG SRC="$(V3)">
            <-- Display the 3rd column          -->
                <BR>
                %IF ($(V4) == "YES")
                    <A HREF=$(PROGRAM)/input?PPID=$(V1)>$(V9)
                <-- Hyperlink with PPID value          -->
                %ENDIF
            </TD>
            %IF (REM == "0")
                </TR>
            <-- Close table row          -->
        %ENDIF
    %}
}
```

```
%}  
  
%MESSAGE {  
  -170:   "Error: 170: Query failure"   : exit  
%}
```

HTML input section

To run this program, send a URL request to your Web server. For example, `http://WWW/cgi-bin/db2www.exe/frndp59.d2w/input`, where WWW is the main Web server directory.

To process the input section (because PPID begins as NULL), the %ELSE portion is executed to invoke `DP_PMQuery()` for data processing. This displays a thumbnail image and builds a link to assign a PPID value. You can display the document by clicking the Title link to call the program and pass PPID as a parameter. The If statement is satisfied with the PPID assigned and `DP_Parts()` is invoked to retrieve the document. Figure 63 shows an example of a query string for the `INDEX_CLASS=GRANDPA`. Figure 64 on page 407 shows an example of a search results report that includes links.

```
Query String  
  
SEARCH=(INDEX_CLASS=GRANDPA,MAX_RESULTS=10);  
  
Result Sets
```

Figure 63. Query string example



Figure 64. Search results report

If you receive a broken image, make sure that:

- The image object is available in the object server
- The thumbnail ID is valid
- The object that resides in the object server is not corrupt

See the Net.Data documentation for more information about macro programming.

```

%(------ HTML Section -----%)
%HTML(INPUT){
<HTML>
  <head>
  <BASE HREF="http://jasonwu.stl.ibm.com/" >
  </head>
  <center>
  %IF (PPID != "")
<----- detect PPID variable, the first time ----->
<----- through will be NULL ----->
  </center>
  <FRAMESET FRAMEBORDER="no" ROWS="100%,*,*">

```

```

<NOFRAMES>
<BODY BGCOLOR="#ffffff" LINK="#000080" ALINK="#00ff00"
VLINK="#557f55">
</BODY>
</NOFRAMES>
@DP_Parts(PPID,d1Result)
←      Invoke DP_API      →
</FRAMESET>
%ELSE
<br> <font+1><strong> Query String </strong></font> <br><br>
$(qst) <br> <br>
<font+1><strong> Result Sets </strong></font> <br>
<HR size=3> <br>
<TABLE cellpadding=7 cellspacing=4 border=1 width=80%>
    @DP_PMQuery(qst,d1Result)
←      Invoke DP_API      →
</TABLE>
%ENDIF
</center>
</HTML>
%}

```

Improving performance

Live Connection Manager

This section explains how to use Content Manager Dynamic Page Builder with the Net.Data Live Connection Manager to improve performance by eliminating start-up overhead. A live connection consists of a connection manager and cliettes.

Cliettes are single-threaded processes that the connection manager starts and keeps resident while the server runs, processes data, and communicates with the Net.Data environment.

Live Connection has two advantages:

Enhanced performance

Avoids the library server connection to save time.

Sequential display

Offers the ability to use next and previous options, which help navigate large result sets in an application. By setting the variables `START_ROW_NUM` and `RPT_MAX_ROW`, you can break down large result sets into smaller segments.

Reuse dynamic pages

Dynamic Page Builder retrieves image content from Content Manager at run time and stores it on the server with the file name provided in the TOC part.

A TOC part is one of the parts within the document that identifies the relationship between an individual part and the file name it is associated with. An example TOC part is shown below (anynet.htm is an HTML file):

```

anynet.htm:3
abstract.txt:4
anynet1t.gif:5
sqlcloud0.gif:17
sqlclouda.gif:18
bigsq.gif:19
dot_clea.gif:20
dot_clea.gif:21
anynet1.gif:22
anynet2.gif:23

```

sqhome.gif:24
tellsq.gif:25
getsq.gif:26
software.gif:27

With the TOC part, you can reduce the overhead associated with file input and output by setting the environment variable `DLPD_FILEMODE=2` when you invoke the Live Connection Manager (see “The ENVIRONMENT statement (DTW_DLDPB)” on page 393.) With `DLPD_FILEMODE=2`, the Dynamic Page Builder does not overwrite existing files, so you can reuse a file.

Invoking the wizard

Net.Data offers a wizard to help you access Net.Data function from your application.

How you start the Content Manager Dynamic Page Builder wizard depends on the platform and shell script you use (see Table 43).

Table 43. Platforms and script files

Platform	Script file
Windows NT	DPWizard.bat
AIX Korn Shell	DPWizard.ksh
AIX C Shell	DPWizard.csh

After you complete all of the data fields in the Dynamic Page Builder wizard, a macro file creates a file in the current directory with the file name you specified. Copy the created macro file to the `MACRO_PATH` directory as defined in the `db2www.ini` file, so that you can execute the file from your Web application.

Starting the Dynamic Page Builder sample

This section describes how to use some of the different kinds of available Web servers.

The following examples assume that `C` is your drive, `DB2WWW` is the root directory for ICS on Windows NT, and `InetPub` is IIS root directory on Windows NT.

Web server configuration

Windows NT with the IIS V2 Web server

1. Start Microsoft Internet Service Manager. Double-click on the computer name of WWW service to display the WWW service properties for your machine.
2. Click on **Directories** to create `cgi-bin`, `tmplobs`, and `cmbroot` aliases. Make sure you specify the corresponding directories and make sure the directories exist.
3. Choose READ and EXECUTE access rights for `cgi-bin`. Choose READ only for `tmplobs`. For example:

```
Directory: c:\InetPub\wwwroot\cgi-bin Alias: /cgi-bin
Directory: c:\InetPub\wwwroot\tmplobs Alias: /tmplobs
Directory: c:\cmbroot Alias: /cmbroot (Assume Content Manager is
installed at C:\CMBROOT)
```

4. Copy the Content Manager files from the Net.Data HTML directory to `<home>` alias. For example:

```
copy c:\db2www\html\*. * c:\InetPub\wwwroot
copy c:\db2www\cgi-bin\db2www.exe c:\InetPub\wwwroot\cgi-bin
```

5. Modify `c:\inetpub\wwwroot\db2www.ini` to make sure that the `HTML_PATH` and `INCLUDE_PATH` point to `c:\inetpub\wwwroot`.

Windows NT with the IBM ICS Web server

This is the Net.Data default setup for Windows NT 4.0. All you must do is make sure that the `cgi-bin`, `tmplobs`, and `cmbroot` aliases are set and that the directories exist. For example:

```
Directory: c:\db2www\cgi-bin      Alias: /cgi-bin
Directory: c:\db2www\html\tmplobs Alias: /tmplobs
Directory: c:\cmbroot            Alias: /cmbroot
```

You can verify this by using your Web browser to access:

```
<lines>: http://machine/admin-bin/cfgin/mpfrule< /lines>
```

IBM AIX with the IBM ICS Web server

This is the Net.Data default setup for AIX, so all you must do is create a directory named `/usr/lpp/internet/server_root/pub/tmplobs` for BLOBs, and ensure that you bring up the Web server as a subsystem, by entering the following command for example:

```
startsrc -s http
```

Connection manager setup

Windows NT 4.0 platform

1. Modify the `dtwcm.cnf` file in the `c:\db2www\connect` directory. Locate the section starting with `DTW_DLDPB` and modify the parameters as shown in the following example:
2.

<code>CLLETTE DTW_DLDPB:LIBSRVRN{</code>	→ Library server
<code>MIN_PROCESS=1</code>	→ Minimum start
<code>MAX_PROCESS=5</code>	→ Maximum process
<code>START_PRIVATE_PORT=7150</code>	
<code>START_PUBLIC_PORT=7350</code>	
<code>EXEC_NAME=c1d1dpb.exe</code>	
<code>DATABASE=LIBSRVRN</code>	→ Library server
<code>BINDFILE=NOT_USED</code>	
<code>LOGIN=FRNDPMIN</code>	→ DP user ID
<code>PASSWORD=PASSWORD</code>	→ DP password
<code>}</code>	

 - a. Set `MIN_PROCESS` parameter to 0 for the remaining sections in the file.
 - b. Start the library server and the object server.
 - c. Double-click the **Net.Data** icon to start the Live Connection Manager.

IBM AIX OS

Modify the `dtwcm.cnf` file in the `/usr/lpp/internet/db2www/db2` directory. Locate the section starting with `DTW_DLDPB` and modify the parameters as shown in the following example:

```
CLLETTE DTW_DLDPB:LIBSRVRX{      → Library Server
MIN_PROCESS=1                    → Minimum start
MAX_PROCESS=5                    → Maximum process
START_PRIVATE_PORT=7150
START_PUBLIC_PORT=7350
EXEC_NAME=./c1d1dpb DATABASE=LIBSRVRX → Library server
BINDFILE=NOT_USED LOGIN=FRNDPMIN   → DP user ID
PASSWORD=PASSWORD                → DP password
}
```

1. Set `MIN_PROCESS` parameter to 0 for the remaining sections in the file.
2. Start the library server and the object server

3. Start **Live Connection Manager** by typing `dtwcm -d` from the directory of `/usr/lpp/internet/db2www/db2`

Configuring sample macro files

1. Specify `SERVER`, `DATABASE`, `LOGIN`, `PASSWORD`, `QB_CATALOG`, `QB_DATABASE`, `QB_CONNECT` and `HTMLROOT` parameters in files `frndp15.d2w` and `frndp25.d2w`, which are located under `c:\db2www\macro` for Windows NT and `/usr/lpp/internet/db2www/macro` for AIX:

```
%define{
SERVER = "Enter Web server machine host name here"
DATABASE = "Enter Library server name here"
LOGIN = "Enter Library server user id here"
PASSWORD = "Enter Library server password here"
QB_CATALOG = "Enter QBIC Catalog here"
QB_DATABASE = "Enter QBIC Database here"
QB_CONNECT = "Enter QBIC_Server/UserID/Password here"
HTMLROOT = "Enter Default HTML root directory here"
%}
```

2. Specify the Web address in the file `dlheader.html`, located in `c:\InetPub\wwwroot` directory for Windows NT with IIS Web server, `c:\db2www\html` directory for Windows NT with ICS Web server, and the `/usr/lpp/internet/server_root/pub` directory for IBM AIX with ICS Web server; for example:

```
<head>
<title>IBM Content Manager Internet Connection</title>
<BASE HREF="http://xyz/">
</head>
```

xyz is your Web server address for the Internet or Web server host machine name (or IP address) for your intranet.

3. Specify web addresses in the files of `dpcolorapplet.html`, `dphistogramapplet.html`, and `dpdrawapplet.html`, which are located under `-cmbroot` (Content Manager installation directory) for Windows NT, or `/usr/lpp/internet/server_root/pub` for AIX.

Running the sample macro

Open `http://xyz/dlqbic.html` in your Web browser. *xyz* is your Web server address for the Internet or Web server host name (or IP address) for your intranet.

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
J74/G4
555 Bailey Avenue
P.O. Box 49023
San Jose, CA 95161-9023
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 information 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, 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 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:

AIX	IBM
AS/400	ImagePlus
DataJoiner	Net. Data
DB2	OS/390
DB2 Universal Database	QBIC
e-business	VisualInfo

Domino, Lotus, Lotus Notes, and Notes are trademarks of Lotus Development 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.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product or service names may be the trademarks or service marks of others.

Glossary

This glossary defines terms and abbreviations specific to the Enterprise Information Portal system, but not necessarily to this particular document. 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 individual user IDs or user groups and their associated *privileges*. You use access 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*.

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.

attribute. A characteristic that identifies and describes a managed object. The characteristic can be determined, and possibly changed, through operations on the managed object. For example, title and duration might be attributes of a video object. See *federated attribute* and *native attribute*.

B

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. An area of storage used to temporarily store objects on the client workstation. See also *LAN cache*.

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.

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 toolkit. The Content Manager toolkit enables you to build your own Internet or desktop client applications that access data and objects managed by Content Manager. Use the toolkit to develop Web-based and desktop applications that combine object catalog and text searches in a single query.

client application. An application written with the object-oriented or Internet APIs to access content servers from Enterprise Information Portal.

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.

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.

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. A logical name assigned to a file type, valid only within Content Manager. Content Manager provides a large range of predefined data formats. You can also define your own data format. For example, you can define your own data format called MYGIF and use it for GIF files.

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*.

DDO. See *dynamic data object*.

document. An *item* that can be stored, retrieved, and exchanged among Content Manager systems and users as a separate unit. It can be any multimedia digital object. A single document 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).

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.

dynamic page builder. An API in the Internet application development toolkit that is used to create applications that dynamically format the results of queries and display those results on a Web page.

E

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 in *DDOs*.

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 a *key field*, policy num, in Content Manager and to a key field, 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. A container used to organize objects, which can be other folders or documents.

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.

form element. In the dynamic page builder, an element of the user interface that can be created with HTML tags. The dynamic page builder supports several HTML form elements.

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 class. A category for storing and retrieving objects, consisting of a named set of attributes. When you create an object in the Content Manager system, your application must assign an index class and supply the *key field* values required by that class. An index class identifies the key fields, automatic processing requirements, and storage requirements for an object.

index class subset. A view of an *index class* that an application uses to store, retrieve, and display folders and objects.

index class view. 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.

interchange. The capability to import or export an image along 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. Generic term for the smallest unit of information that Enterprise Information Portal administers. An item contains one or more *item parts*. Each item has an identifier. An item can be a *folder* or a *document*.

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.” Once built, these beans can be made available for use by other software engineers or can be used in Java applications. Also, using JavaBeans, software engineers can manipulate and assemble beans in a graphical drag-and-drop development environment.

K

key field. See *attribute*.

L

LAN cache. An area of temporary storage on a local *object server* that contains a copy of objects stored on a remote object server.

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 server. The component of a Content Manager system that contains index information for the objects stored on one or more *object servers*.

list. A data structure, or electronic queue, that your application uses to temporarily insert or remove *objects* for work.

M

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++.

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 *parts* of a defined group of *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 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. The component of a Content Manager system that physically stores the *objects* or information accessed by client applications.

object server cache. The working storage area for the *object server*. Also called the *staging area*.

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. A subelement of an *item* that corresponds to a discrete multimedia data object, such as an image, video, audio, or text. An item can contain parts with diverse binary formats, for example, an image, a word processing file, and a spreadsheet.

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.

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.

S

search criteria. 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.

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.

staging area. The working storage area for the *object server*. Also referred to as *object server cache*.

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.

T

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.

TSM. See *Tivoli Storage Manager*.

TSM management class. A logical area of storage that is managed by *Tivoli Storage Manager*.

U

user. In Enterprise Information Portal, anyone who is identified in the Enterprise Information Portal administration program.

user exit. (1) A point in an IBM-supplied program at which a user exit routine can be given control. (2) A programming service provided by an IBM software product that can be requested during the execution of an application program for the service of transferring control back to the application program upon the later occurrence of a user-specified event.

user exit routine. A user-written routine that receives control at predefined user exit points.

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.

V

volume. A representation of an actual physical storage device or unit on which the objects in your system are stored.

W

work item. A work activity that is active within a *workflow*.

work packet. In Enterprise Information Portal, 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*.

workflow. A sequence of steps, and the rules governing those steps, through which a *work packet* travels while it is being processed.

For example, a process called *claims approval* would describe the steps that a *work packet* related to approving an insurance claim must follow.

workflow coordinator. 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*.

worklist. A collection of *work items* assigned to a user and retrieved from a workflow management system.

X

XDO. See *extended data object*.

Index

A

- accessing a worklist 124
- actions, creating 124
- ActiveX 331
- administration client
 - description 4
- administration database, description 4
- advanced search sample in Information Mining 200
- annotation object, adding 32
- Application Programming Interfaces (API)
 - Active/X 331
 - Installation 333
 - Multiple search 334
 - C++ 215
 - Dynamic Page Builder 393
 - configuring DPB with Net.Data 393
 - connection manager 410
 - Environment (DTW_DLDAPB) 393
 - HTML_PATH 394
 - INCLUDE_PATH 394
 - MACRO_PATH 394
 - parameters 395
 - performance 408
 - reserved terms 394
 - sample macro 1 400
 - sample macro 2 404
 - sample startup 409
 - server configuration 409
 - wizard 409
 - Java 17
 - Architecture 17
 - Differences from C++ 18
 - Multiple search 22, 218
 - Packaging 18
- attribute mapping 12
- attributes, listing 100, 103, 295, 299

B

- buffer, adding an XDO 31
- building an application, non-visual
 - beans 144
- building an application, visual
 - beans 161

C

- C++ 215
- cabinet attributes, listing 108, 306
- categorization sample in Information Mining 192
- class name terminology 60, 258
- client 17
- cmbregist71.bat 21
- cmbregist71.sh 21
- code page conversion 217
- Collections and iterators
 - C++ 243

- Collections and iterators (*continued*)
 - Memory management 244
 - Programming tips 245
 - Sequential collection 243
 - Sequential iterator 243
 - Sorting 245
- Java 46
 - Sequential collection 46
 - Sequential iterator 46
 - Sorting 47
- Combined query
 - Active/X 380
 - parametric with text 380
 - Programming tips 381
 - ranking 381
 - using a scope 381
 - C++ 290
 - parametric with text 290
 - Programming tips 292
 - ranking 291
 - using a scope 291
 - Java 91
 - parametric with text 91
 - Programming tips 92
 - using a scope 92
- common 17
- common classes, Enterprise Information Portal 126, 317
- common object model 17, 215
 - ActiveX application programming 331
- configuration string 116
- connection string 116
- connectors, creating custom 125
- connectors, description 4
- connectors, developing 125
- connectors toolkit, description 5
- console sub system, setting 217
- constants 23
- Content Manager, working with 62, 260, 357
- content provider in Information Mining 210
- content server, executing a parametric query 249
- content servers 7, 23
 - accessing options 24
 - connecting 23
 - listing servers 25
- content servers, specific 59, 258
- content servers, creating connectors 125
- content viewer 181
- cs package 18
- custom content servers, developing 316
- custom content servers, working with 316
- custom content servers
 - connectors, developing 125

D

- data access 7
- data concepts, persistent 8

- data definition classes 61
- databases, listing 110, 308, 386
- datastore 7
- datastore, evaluating a parametric query 51, 249
- datastore, evaluating a text query 53, 251
- datastore, executing a parametric query 51
- datastore, executing a text query 52, 251
- datastore registration 14
- DB2 Warehouse Manager Information Catalog Manager, working with 119
 - connecting 120
 - executing a query 121
- DDO, adding properties 27, 223
- DDO, properties 63, 260, 357
- DDO, understanding datastores 11
- DDO, XDO as a part of 30, 227
- DES 110, 308, 385
- diagnostic information 22
- DKAny
 - Assignment from 242
 - Assignment to 241
 - Destroying 242
 - Display of 242
 - Memory Management 241
 - Programming tips 243
 - type code, Getting 241
 - Typecode 241
 - Using type constructors 241
- DKAny, using 240
- dkCollection 46
- DKConstants 23
- dkDatastore 127, 318
- DKDatastorexx 60
- DKDatastoreDL
 - Active/X 334
 - Connecting 334
 - DKDatastoreDL options 335
 - List schema and schema attributes 335
 - List servers 335
 - C++ 219
 - Connecting 219
 - DKDatastoreDL options 219
 - List schema and schema attributes 220
 - List servers 220
 - Java 23
 - Connecting 23
 - DKDatastoreDL options 24
 - List schema and schema attributes 25
 - List servers 25
- DKDatastoreTS
 - Active/X 363
 - Connecting 365
 - DKDatastoreTS options 365
 - List schema 366
 - List servers 366

- DKDatastoreTS (*continued*)
 - C++ 266
 - Connecting 268
 - DKDatastoreTS options 269
 - List schema 270
 - List servers 269
 - Java 69
 - Connecting 71
 - DKDatastoreTS options 71
 - List schema 72
 - List servers 72
- DKException (Java) 22
- dkFederatedIterator 48
- dkIterator 48
- DKSequentialCollection 46
- dkSort 47
- dkWorkFlowUserExit 124
- DLL 18, 215
- document, DES 113, 311, 388
- documents 9
- documents, representing 63, 260, 358
- Domino.Doc, working with 106, 304
- Domino Extended Search, working with 110, 308, 385
- Dynamic Data Object (DDO)
 - Active/X 336
 - Adding 337
 - attribute, DKFOLDER 344
 - attribute, DKPARTS 344
 - data item values 338
 - Deleting 339
 - Displaying 338
 - Information, Digital Library 357
 - Information, Text Search Engine 364
 - PID 336
 - properties 338
 - C++ 221
 - Adding 223
 - attribute, DKFOLDER 239
 - attribute, DKPARTS 238
 - Creating 222
 - data item values 224
 - deleting 225
 - Displaying 225
 - Information, Digital Library 260
 - Information, Text Search Engine 267
 - PID 222
 - properties 224
 - Java 26
 - Adding 27
 - attribute, DKFOLDER 45
 - attribute, DKPARTS 44
 - attribute properties 28
 - Creating 26
 - data item values 27
 - Displaying 29
 - Information, Digital Library 63
 - Information, Text Search Engine 70
 - PID 27
 - properties 28
- dynamic data objects 9
- dynamic data objects, comparing 11

E

- EIP workflow services 122
 - connecting to 122
 - creating a workpacket 122
- Enterprise Information Portal
 - components 4
 - administration client 4
 - administration database 4
 - connector toolkit 5
 - connectors 4
 - image search 6
 - Information Mining feature 5
 - sample client application 5
 - text search 6
 - thin client samples 5
 - workflow builder 5
 - workflow feature 5
- Enterprise Information Portal, concepts 7
- Enterprise Information Portal database 8
- Enterprise Information Portal database infrastructure 125, 316
- entities, listing 100, 107, 295, 305
- entity mapping 12
- exception handling 22
 - DKException 22
- extended data objects 9

F

- federated collection, understanding 48, 245
- federated datastore mapping components, Enterprise Information Portal 14
- federated iterator, understanding 48, 245
- federated query processing 14
- federated query syntax 16
- federated schema mapping 14
- federated searching, DES 115, 315, 390
- federated searching, understanding 11
- FeServerDefBase 139
- fields, listing 110, 308, 386
- file, adding and XDO 31
- folders 9
- folders, representing 63, 261, 358
- folders, updating 66, 264, 360
- framework, Enterprise Information Portal 9
- function, invoking and XDO 230

G

- Generalized Query Language 111, 310, 387
- GQL 111, 310, 387

I

- image queries, working with 88, 286
- image search
 - description 6
 - image search, catalogs 79
 - image search, databases 79
 - image search, features 80, 279
 - image search, representing information 88, 286

- image search applications 82, 280
- image search catalogs, listing 87, 284
- image search concepts 79, 278
- image search databases, listing 87, 284
- image search engine, using 90, 288
- image search features, listing 87, 284
- image search servers, listing 87, 284
- image search terms 79, 278
- ImagePlus for OS/390, working with 100, 295
- images by content, searching 78, 277
- importing XML 41
- index classes 8
- index classes, listing 103, 299
- Information Catalog, working with 119
- Information Mining
 - advanced search sample 200
 - beans 191
 - building an application 191
 - categorization sample 192
 - JSP applications 212
 - location of the sample files 191
 - own content provider 210
 - summarization sample 197
 - Web Crawler sample 206
- Information Mining feature
 - description 5
- item attributes 8
- item parts 8
- items 8
- iterators 46

J

- jar files 18
- Java 17
- Java APIs 17
 - differences 18
- JavaBeans 141
- JavaBeans, understanding 141
- JavaServer Pages 150
- JSP 150
- JSP applications in Information Mining 212

L

- large objects, handling 62, 260
- Library 18, 215

M

- mapping terminology 12
- match highlighting 53, 55, 251, 253
- media object, adding an XDO 232
- media object, deleting and XDO 233
- media object, retrieving an XDO 235
- member, adding 66, 263
- member, removing 66, 263
- MIME types, DES 115, 315, 390
- multimedia data object 8
- multimedia data objects 10
- multimedia item 9
- multiple search 17, 215, 331

N

- native entity 8
- non-visual beans 141

- non-visual beans, understanding
 - events 143
- non-visual beans, understanding
 - properties 143
- non-visual beans, working with 142
- Notices 413

O

- object management
 - Active/X 358
 - creating 358
 - deleting 361
 - updating 360
 - C++ 261
 - creating 261
 - deleting 264
 - updating 263
 - Java 64
 - creating 64
 - deleting 67
 - updating 65
- OnDemand, listing information 97
- OnDemand, retrieving documents 98
- OnDemand, working with 96

P

- package hierarchy, Java 18
- Parametric query 22, 218
- parametric query, executing 50, 248
- parametric query, formulating 49, 247
- parametric query, formulating multiple criteria 50, 248
- part, indexing 179
- part information, displaying 177
- parts 9
- parts, updating 66, 263
- password mapping 14
- persistent data concepts 8
- persistent identifier 11

Q

- QBIC 283
- QBIC, connecting 86
- queries, Domino.Doc 108, 306
- Query
 - Active/X 346
 - parametric type 346
 - text type 349
 - C++ 247
 - dkResultSetCursor vs DKResults 247
 - parametric type 247
 - text type 250
 - Java 49
 - dkResultSetCursor vs DKResults 49
 - parametric type 49
 - query object types 49
 - text type 51
- query applet, understanding 170
- query string 49, 52
- query syntax, Domino.Doc 109, 307
- query syntax, ImagePlus for OS/390 102, 297

- Queryable collection
 - Active/X 354
 - evaluating 355
 - getting results 354
 - queryable vs refined 355
 - C++
 - evaluating 257
 - getting results 257
 - queryable vs refined 258
 - Java 58
 - evaluating 59
 - getting results 58
 - Programming tips 59
 - queryable vs refined 59

QUIC 78

R

- related classes 127, 318
- relational databases, working with 116
- Remote Method Invocation (RMI) 21
- Result set cursor
 - Active/X 353
 - C++ 255
 - creating a collection 256
 - open and close 255
 - set and get 255
 - Java 57
 - creating a collection 58
 - open and close 57
 - set and get 57
- Retrieval
 - Active/X 361
 - documents 361
 - folders 362
 - parts 362
 - C++ 265
 - folders 266
 - parts 265
 - Java 67
 - folders 68
 - parts 68
- RMI server 21
 - cmbregist71.bat 21
 - cmbregist71.sh 21
 - starting 21

S

- sample client application
 - description 5
- sample files, location
 - Information Mining 191
- schema, listing 117
- schema attributes, listing 117
- search engines, indexing an existing XDO 90, 289
- search template folders 98
- searchable entities 108, 306
- server 17
- Settings
 - C++ 216
 - Building on AIX 216
 - Building on NT 217
 - On AIX 216
 - On NT 216, 333
 - Java
 - Client connect and disconnect 24

- Settings (*continued*)
 - Java (*continued*)
 - Client/Server 17
 - On AIX 20
 - On NT 20
 - Programming tips 18
 - Setup environment 18, 333
 - Using sample Java applets and servlet 169
 - Connect 169
 - Dynamic Page Builder 185
 - Java application on client 175
 - Local access 175
 - Remote access 176
 - Retrieve servlet 174
 - View 173
- shared objects 18, 215
- sorting 47

Stand-alone code examples

- C++
 - annotation type 229
 - from buffer 227
 - from file 228
 - search indexed by Text Search Engine 271
- Java
 - search indexed by Text Search Engine 73
- starting a workflow 123
- storage collection 40, 237
- storage collection, adding an XDO 40, 237
- storage collection, changing 41, 238
- subentities, listing 107, 305
- summarization sample in Information Mining 197

T

- terminating a workflow 123
- Text query 22, 218
- text query, executing 52, 250
- text query, formulating 52, 250
- text query, formulating multiple indexes 52, 250
- text query, getting a particular result item 55, 253
- text query, getting each result item 53, 251
- text search
 - description 6
- Text Search Engine
 - Active/X 363
 - Boolean query 363
 - Client/Server mode 332
 - Free text query 363
 - GTR query 364
 - Hybrid query 363
 - Load and index data 367
 - Programming tip 366
 - Proximity query 364
 - C++
 - Boolean query 267
 - Exceptions 219
 - Free text query 267
 - GTR query 267
 - Hybrid query 267
 - Load and index data 272

- Text Search Engine (*continued*)
 - Programming tip 269
 - Proximity query 267
- Java 69
 - Boolean query 69
 - Exceptions 22
 - Free text query 69
 - GTR query 69
 - Hybrid query 69
 - Load and index data 74
 - MAXPIECE 62
 - Programming tip 71, 365
 - Proximity query 69
 - Setting heap size 63, 260
- Text Search Engine, parametric queries 22, 218
- Text Search Engine, text queries 22, 218
- text structured document 74, 273
- Thin client, using 187
- thin client samples
 - description 5
- Tracing information 22, 218

U

- user ID mapping 14

V

- video stream, displaying parts
 - information 184
- video stream, playing 185
- video streams, loading 182
- visual beans 141
- visual beans, common behaviors 159
- visual beans, specialized behaviors 160
- visual beans, working with 151
- VisualInfo for AS/400, working
 - with 103, 299

W

- Web Crawler sample in Information Mining 206
- workbasket, understanding 292
- workbasket, understanding in Content Manager 93
- workflow 123
 - starting 123
 - terminating 123
- workflow, understanding 292
- workflow, understanding in Content Manager 93
- workflow builder, description 5
- workflow feature
 - description 5
- workflow service 93, 292
- worklist 124
 - accessing 124
 - accessing work items 124
 - moving work items 124
- workpacket 122

X

- XDO
 - Active/X 339

- XDO (*continued*)
 - data members 339
 - DDO, part of 340
 - in datastore 340
 - indexing 366
 - Programming tips 340
 - stand-alone 341
- C++ 225
 - data members 225
 - DDO, part of 227
 - indexing 270
 - PID 225, 339
 - Programming tips 226
 - stand-alone 227
- Java 29
 - adding an annotation 32
 - adding from a buffer 31
 - adding from a file 31
 - data properties 29
 - DDO, part of 30
 - deleting 32
 - function, invoking an XDO 33
 - indexing 73
 - media object, adding 35
 - media object, deleting 37
 - media object, retrieving 39
 - PID 29
 - Programming tips 30
 - retrieving 32
 - stand-alone 31
 - storage collection, adding 40
 - storage collection, changing 41
 - updating 32
- XDO, deleting 229
- XDO, retrieving 229
- XDO, updating 229
- XML
 - dtd for import 42
 - importing 41



Program Number: 5697-G29, 5697-G31



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC27-0877-01

