IBM Content
Manager for Multiplatforms/IBM Enterprise
Information Portal for Multiplatforms

Workstation Application Programming Guide

Version 8 Release 1

SC27-1347-00

IBM Content
Manager for Multiplatforms/IBM Enterprise
Information Portal for Multiplatforms

Workstation Application Programming Guide

Version 8 Release 1

SC27-1347-00

Note
Before using this information and the product it supports, read the information in “Notices” on page 559.

First Edition (May 2002)

This edition applies to Version 8 Release 1 of IBM Enterprise Information Portal for Multiplatforms, IBM Content Manager for Multiplatforms (product numbers 5724-B43, 5724-B19), and to all subsequent releases and modifications until otherwise indicated in new editions. This edition replaces SC27-0877-00.

Contents

About this guide ix
Who should use this guide ix
Where to find more information ix
Information included in your product package. . ix
Support available on the Web. x
How to send your comments xi
What's new Enterprise Information Portal 8.1 . . . xi
What's new in Content Manager 8.1 xii

Chapter 1. Introducing Enterprise Information Portal 1
Searching for customer information. 1
The need 2
The solution 2
The Enterprise Information Portal solution 2
IBM Enterprise Information Portal for Multiplatforms components 4
What's new in Version 8.1 APIs 6
New and changed Java classes 6
New and changed C++ classes 7

Chapter 2. Enterprise Information Portal application programming concepts 11
Understanding data access through content servers 11
Understanding dynamic data object concepts . . . 12
Dynamic data objects (DDO) 12
Extended data objects (XDO) 13
Representing multimedia content 13
Understanding datastores and DDOs. 13
Comparing DDO/XDOs with attribute values and item parts 14
Understanding persistent identifiers (PID) . . . 14

Chapter 3. Working with a federated datastore and federated searching. . . 15
Federated schema mapping 17
Using federated datastore mapping components . . 17
Running federated queries 18
Federated query syntax 19
Working with system administration 21
Customizing the EIP system administration client 22

Chapter 4. Using the Java application programming interfaces (APIs) 23
Client/server architecture. 23
Differences between the Java and C++ APIs . . . 24
Packaging for the Java environment 24
Programming tips 24
Setting up the Windows, AIX, and Solaris environments. 24
Setting environment variables 25
Using Remote Method Invocation (RMI) with content servers 26
Multiple search facilities 26
Tracing and diagnostic information 27
For text queries using Text Search Engine . . . 27
For parametric queries. 27
Exception handling. 27
Constants 28
Connecting to content servers 28
Establishing a connection. 28
Connecting and disconnecting from a content server in a client. 29
Setting and getting content server options . . . 29
Listing servers 30
Listing the entities and attributes for a content server 30
Working with DDOs 31
Creating a DKDDO. 32
Adding properties to a DDO 32
Creating a persistent identifier (PID) 32
Working with data items and properties. . . . 33
Getting properties 34
Displaying the whole DDO 34
Working with XDOs 35
Using an XDO PID. 35
Understanding XDO properties. 35
Programming tips 35
Using XDO as a part of a DDO or stand-alone . 36
Examples of working with an XDO 38
Working with XML. 47
Importing XML documents 47
Exporting XML 51
Creating and using the DKPARTS attribute. . . . 52
Creating and using the DKFOLDER attribute . . . 53
Using collections and iterators 53
Using sequential collection methods 54
Using the sequential iterator. 54
Sorting the collection 55
Understanding federated collection and iterator 55
Querying a content server 56
Differences between dkResultSetCursor and DKResults. 57
Using parametric queries. 57
Using text query. 59
Using the result set cursor 65
Opening and closing the result set cursor to rerun the query 65
Setting and getting positions in a result set cursor 65
Creating a collection from a result set cursor . . 66
Querying collections 66
Getting the result of a query. 66
Evaluating a new query 67
Using queryable collection instead of combined query 67

Chapter 5. Working with Content Manager 8.1 in Java 69

Understanding the Content Manager system . . . 69
Understanding Content Manager concepts 70
Items 70
Attributes 71
Item types 71
Root and child components 71
Objects 72
Links and references 72
Documents 73
Folders 73
Versioning 73
Access control 74
The data model 78
Planning a Content Manager application 78
Determining the features of your application . . 78
Handling errors 79
Working with the Content Manager samples . . . 79
The insurance scenario sample 80
Creating a Content Manager application 80
Understanding the software components . . . 80
Representing items using DDOs 81
Connecting to the Content Manager system . . 82
Working with items 82
Working with folders 91
Defining links between items 93
Working with the access control APIs. 95
Creating a privilege 95
Creating a privilege set 96
Defining an ACL 97
Assigning an ACL to an item type. 97
Assigning an ACL to an item 98
Understanding the query language 98
Querying the Content Manager server 99
Applying the query language to the Content Manager data model 100
Understanding parametric search. 101
Understanding text search 101
Searching for object contents 102
Searching for documents 102
Making user-defined attributes text searchable 102
Understanding text search syntax. 102
Creating combined parametric and text search . . 103
Query examples 104
Query examples 107
Escape sequences in the query language 113
Using escape sequences with comparison operators (″=″, ″!=″, ″>″, ″<″, ″BETWEEN″ and others). 113
Using escape sequences with the LIKE operator 113
Using escape sequences with advanced text search (″contains″ and ″score″ functions) . . . 114
Using escape sequences with basic text search (″contains-text-basic″ and ″score-basic″ functions). 115
Using escape sequences in Java 117
The query language grammar 117
Working with the resource manager 119
Working with resource manager objects . . . 120
Managing documents in Content Manager . . . 121
Creating the document management data model 122
Creating a document item type 122
Creating a document 122
Updating a document 123
Retrieving and deleting a document 124
Versioning of parts in the document management data model 124
Working with transactions 125
Things to consider when designing transactions in your application 126
Caution when using explicit transactions . . . 126
Using check-in and check-out in transactions 126
Processing transactions 127
Routing a document through a process 128
Understanding document routing concepts . . 128
Understanding the document routing process 130
Setting up a document routing process 131

Chapter 6. Working with other content servers in Java 141

Working with earlier Content Manager 143
Handling large objects 143
Using DDOs to represent Content Manager's data 144
Creating, updating, and deleting documents or folders. 145
Retrieving a document or folder 148
Understanding text searching (Text Search Engine) 150
Searching images by content 159
Using image search applications 163
Establishing a connection in QBIC 167
Listing image search servers 168
Listing image search databases, catalogs, and features 168
Representing image search information with a DDO 169
Working with image queries 169
Using the image search engine 171
Indexing an existing XDO using search engines 171
Using combined query 172
Understanding the earlier Content Manager workflow and workbasket functions. 174
Working with OnDemand 178
Representing OnDemand servers and documents 178
Listing information on OnDemand 179
Retrieving an OnDemand document. 180
Enabling the OnDemand folder mode 181
Asynchronous search. 182
OnDemand folders as search templates. . . . 182
OnDemand folders as native entities 182
Create and Modify Annotations 183
Working with ImagePlus for OS/390 183
Listing entities and attributes 183
ImagePlus for OS/390 query syntax 185
Working with Content Manager for AS/400 . . . 186
Listing entities (index classes) and attributes . . 186
Executing a query 187
Executing a parametric query 188
Working with Domino.Doc 189
Listing entities and subentities. 190
Listing cabinet attributes 191

Using queries in Domino.Doc 191
Using query syntax 192
Working with Extended Search (ES) 193
Listing Extended Search servers 193
Listing databases and fields 194
Using Generalized Query Language (GQL) . . 195
DDO properties in ES 196
Creating PIDs in Extended Search 196
Contents of a ES document. 196
Retrieving a document 198
Retrieving a BLOB. 198
Associating MIME types with documents . . . 198
Using Extended Search in federated searching 198
Working with Panagon Image Services 199
Data modeling 199
Documents and pages in Panagon Image Services 200
Listing entities and attributes 201
Querying. 202
Working with relational databases 204
Connecting to relational databases 204
Listing entities and entity attributes 206
Executing a query 207
Working with DB2 Warehouse Manager Information Catalog Manager 208
Connecting to a DB2 Warehouse Manager Information Catalog Manager server 208
Listing entities and attributes 209
Executing a query 210
Creating custom content server connectors . . . 211
Developing custom content server connectors 211
Using the FeServerDefBase class 224

Chapter 7. Using Enterprise Information Portal workflow in Java. . 227

Connecting to workflow services 227
Starting a workflow 227
Terminating a workflow 228
Suspending a workflow 228
Resuming a workflow 228
Listing all the workflow templates 229
Listing all the workflows 229
Listing all the worklists 229
Accessing a worklist 230
Accessing work items 230
Moving items in the workflow 230
Creating your own actions 231

Chapter 8. Using non-visual and visual JavaBeans. 233

Using JavaBeans in builders 233
Using IBM VisualAge for Java. 233
Non-visual beans 235
Non-visual bean configurations 236
Non-visual bean data model classes 236
Understanding properties and events for non-visual beans 238
Building an application using non-visual beans 238
Working with visual beans 245
CMBLogonPanel bean 246
CMBSearchTemplateList bean 248
CMBSearchTemplateViewer bean 249
Validating or editing fields of the CMBSearchTemplateViewer. 249
CMBSearchPanel bean 250
CMBSearchResultsViewer bean 250
Overriding pop-up menus 251
CMBFolderViewer bean 251
CMBDocumentViewer bean 252
Viewer specifications 253
Default viewers 254
Launching external viewers 254
CMBItemAttributesEditor bean 254
Vetoing changes in the CMBItemAttributesEditor 255
CMBVersionsViewer bean 255
General behaviors for visual beans 255
Replacing a visual bean 256
Building an application using visual beans . . 257

Chapter 9. Using the C++ application programming interfaces 265

Setting up the Windows and AIX environment . . 265
Setting AIX environment variables 266
Setting Windows environment variables . . . 267
Building C++ programs 267
Setting console subsystem for code page conversion on Windows. 267
Multiple search facilities. 267
Tracing information 268
For text queries using Text Search Engine . . . 268
For parametric queries 268
Catching a DKException. 268
Constants 269
Connecting to content servers 269
Establishing a connection 269
Setting and getting content server options. . . 270
Listing content servers 270
Listing a content server's entities and attributes 271
Using DDOs. 272
Creating a DKDDO 272
Creating a persistent identifier (PID) 273
Adding data items and properties 273
Adding properties to a DDO 274
Setting and getting data item values. 274
Getting the DDKDO and attribute properties 274
Displaying the DDO 275
Deleting a DDO 275
Using XDOs. 276
Using an XDO PID 276
Understanding XDO data members 276
DB2, ODBC and DataJoiner configuration strings for C++ 276
Programming tips 277
Using XDO as a part of DDO instead of a stand-alone XDO 277
Creating and using the DKPARTS attribute . . . 288
Creating and using the DKFOLDER attribute. . . 290
Using DKAny 291
Using type code 291
Managing memory in DKAny. 291

Using constructors. 292
Getting the type code. 292
Assigning a new value to DKAny 292
Assigning a value from DKAny 292
Displaying DKAny 293
Destroying DKAny 293
Programming tips 293
Using collections and iterators. 293
Using sequential collection functions 294
Using the sequential iterator 294
Managing memory in collections 295
Sorting the collection 296
Programming tips 296
Understanding federated collection and iterator 296
Querying a content server 297
Differences between dkResultSetCursor and DKResults 298
Using parametric query 298
Using text query 300
Using the result set cursor 306
Opening and closing the result set cursor to re-execute the query 306
Setting and getting positions in a result set cursor 306
Creating a collection from a result set 307
Querying collections 307
Getting query results 307
Evaluating a new query 308
Using queryable collection instead of combined query 308

Chapter 10. Viewing and annotating documents 309
Viewer architecture 310
The document engines 310
The annotations engine 310
Creating a generic document viewer. 311
Customizing the generic document viewer . . . 311
Example applications. 312
Standalone viewer. 313
Java application 313
Thin client 313
Applet or servlet 314
Dual-mode and applet or servlet 314
Working with the annotation services 315
Using annotation services interfaces 316
Understanding annotation editing support . . 317
Building an application using the annotation services 317
Adding a custom annotation type to your application 318

Chapter 11. Using the EIP tag library and controller servlet 319
Setting up the tag library and servlet 319
Using the tag library 319
Conventions used in the tag library 320
Tag summary 320
Connection related tags 320
Schema related tags 321
Search related tags 322
Item related tags 322
Folder related tags. 323
Document related tags 323
EIP controller servlet 324
What the servlet can do 324
Servlet reference 325
Servlet toolkit function matrix 329

Chapter 12. Working with information mining 333
Building an information mining application using the beans. 333
Location of the sample files 336
Categorizing documents. 337
Summarizing documents 344
Extracting information 348
Clustering 352
Importing documents from a Web space . . . 357
Searching for documents by category 365
Building your own content provider. 370
Using the service API 371
Connecting to the information mining service API. 372
Managing the library, taxonomies, and catalogs 372
Using the information mining tools 376
Creating records and storing metadata in catalogs 379
Searching for documents 380
Running a server task 381
Example of an information mining application based on JSPs 382
Document formats supported 383

Chapter 13. Working with Content Manager 8.1 in C++ 391
Understanding the Content Manager system . . . 391
Understanding Content Manager concepts. . . . 392
Items 392
Attributes 393
Item types 393
Root and child components. 393
Objects 394
Links and references 394
Documents 394
Folders 395
Versioning 395
Access control 396
The data model 399
Planning a Content Manager application 400
Determining the features of your application 400
Handling errors 400
Working with the Content Manager samples . . . 401
The insurance scenario sample 402
Tools and wrappers 402
Creating a Content Manager application 402
Understanding the software components . . . 402
Representing items using DDOs 403
Connecting to the Content Manager system . . 403
Working with items 404

Working with folders 415
Defining links between items 417
Working with the access control APIs 419
Creating a privilege 419
Creating a privilege set 420
Displaying privilege set properties 421
Defining an ACL 421
Retrieving and displaying ACL information . . 422
Assigning an ACL to an item type 422
Assigning an ACL to an item 422
Understanding the query language 423
Querying the Content Manager server 424
Applying the query language to the Content Manager data model 424
Understanding parametric search. 426
Understanding text search 426
Searching for object contents 426
Searching for documents 427
Making user-defined attributes text searchable 427
Understanding text search syntax. 427
Creating combined parametric and text search . . 428
Query Examples 429
Query examples 432
Escape sequences in the query language 437
Using escape sequences with comparison operators (″=″, ″!=″, ″>″, ″<″, ″BETWEEN″ and others) 438
Using escape sequences with the LIKE operator 438
Using escape sequences with advanced text search (″contains″ and ″score″ functions) . . . 439
Using escape sequences with basic text search (″contains-text-basic″ and ″score-basic″ functions) 440
Using escape sequences in c++ 442
The query language grammar 442
Working with the resource manager. 444
Managing documents in Content Manager . . . 445
Working with the document model 446
Creating a document item type 446
Creating a document 447
Updating a document 448
Retrieving and deleting a document. 449
Versioning of parts in the document management data model 449
Working with transactions 450
Things to consider when designing transactions in your application 450
Caution when using explicit transactions . . . 451
Using check-in and check-out in transactions 451
Processing transactions 452
Routing a document through a process. 452
Understanding document routing concepts . . 453
Understanding the document routing process 455
Setting up a document routing process 456

Chapter 14. Working with other content servers in C++ 467

Working with earlier Content Manager 468
Handling large objects 468
Using DDOs to represent earlier Content Manager content 469
Creating, updating, and deleting documents or folders. 470
Retrieving a document or folder 474
Understanding text searching (Text Search Engine) 475
Searching images by content 486
Understanding image search terms and concepts 487
Using image search applications 489
Establishing a connection in QBIC 492
Listing image search servers 493
Listing image search databases, catalogs, and features 493
Representing image search information with a DDO 495
Working with image queries 495
Using the image search engine 497
Indexing an existing XDO using search engines 498
Using combined query 499
Understanding the workflow and workbasket functions 501
Using rights management in earlier Content Manager 505
Working with Content Manager ImagePlus for OS/390 509
Listing entities and attributes 509
ImagePlus for OS/390 query syntax 511
Working with Content Manager for AS/400 . . . 513
Listing index classes and attributes 513
Executing a query 514
Executing a parametric query 517
Working with Domino.Doc 518
Listing entities and subentities. 519
Listing cabinet attributes 520
Building queries in Domino.Doc 520
Using query syntax 521
Working with Extended Search (ES) 522
Listing ES servers 522
Listing entities (databases) and attributes (fields) 523
Using Generalized Query Language (GQL) . . 524
DDO properties in ES 525
Creating PIDs in ES 525
Contents of a ES document. 525
Retrieving a document 528
Retrieving a BLOB. 529
Associating MIME types with documents . . . 529
Using federated searching in ES 530
Working with OnDemand 530
Representing OnDemand servers and documents 531
Listing information on OnDemand 531
Retrieving an OnDemand document. 532
| Tracing 534
Working with relational databases 534
Connecting to relational databases 534
Listing entities and entity attributes 535
Executing a query 536
Working with custom content servers 538
Developing custom content servers 538

| **Chapter 15. Using Enterprise Information Portal workflow in C++ . . 553**

| Connecting to workflow services 553
| Starting a workflow 553
| Terminating a workflow 554
| Listing all the workflows 554
| Suspending a workflow 555
| Resuming a workflow 555
| Listing all the worklists 555
| Accessing a worklist 556
| Accessing work items 556
| Moving items in the workflow 556
| Listing all the workflow templates 557

Notices 559
Trademarks 561

Glossary 563

Index 571

About this guide

This guide describes how to use and modify the Java™, JavaBeans™, and C++ application programming interfaces (APIs) provided with Enterprise Information Portal Version 8 Release 1 (EIP) and Content Manager (CM) Version 8 Release 1. These APIs provide building blocks for you to create applications to query content stored in heterogeneous content servers.

In the past these two products have maintained separate application programming guides. With Version 8 Release 1 of each product, however, many of the APIs are shared by both products and much of the new functionality of EIP Version 8 Release 1 involves the new connector to CM Version 8 Release 1. As a result, the two guides are combined into one.

This guide includes:

- An introduction to EIP and CM application programming concepts, including dynamic data object concepts in the context of Java and C++
- A description of the function accessible through the CM Version 8 Release 1 connector
- Documentation on all other EIP connectors to content servers
- Updates to visual and non-visual JavaBeans
- Updates to programming information for Information Mining, IBM® Web Crawler, and workflow

Illustrations referring to Content Manager imply both pre-Version 8.1 and Version 8 Release 1 of the product.

Who should use this guide

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

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

Where to find more information

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

Information included in your product package

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

The information center

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

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

PDF publications

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

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

Table 1. Content Manager publications

File name	Title	Publication number
install	*Installing Your Content Manager System*[1]	GC27-1332-00
migrate	*Migrating to Content Manager Version 8*	SC27-1343-00
sysadmin	*System Administration Guide*	SC27-1335-00

Notes:

1. You receive a printed copy of *Installing Your Content Manager System* with IBM Content Manager for Multiplatforms.

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

Table 2. Enterprise Information Portal publications

File name	Title	Publication number
apgwork	*Workstation Application Programming Guide*[1]	SC27-1347-00
ecliinst	*Installing, Configuring, and Managing the eClient*	SC27-1350-00
eipinst	*Planning and Installing Enterprise Information Portal*[2]	GC27-1345-00
eipmanag	*Managing Enterprise Information Portal*	SC27-1346-00
messcode	*Messages and Codes*[3]	SC27-1349-00

Notes:

1. The *Workstation Application Programming Guide* contains information about programming applications for both Content Manager and Enterprise Information Portal.
2. When you separately order IBM Enterprise Information Portal for Multiplatforms, you receive a printed copy of *Planning and Installing Enterprise Information Portal* with the product.
3. *Messages and Codes* contains the messages and codes for Content Manager and Enterprise Information Portal.

Support available on the Web

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

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

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

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

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

How to send your comments

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

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

  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, include the location of the text (for example, a chapter and section title, a table number, a page number, or a help topic title).

What's new Enterprise Information Portal 8.1

The following changes have been made to the product:

Support for Sun Solaris
You can install connectors, features, and databases on Solaris systems.

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

New connectors

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

Improved connectors

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

New information mining services

- Feature extraction
- Clustering
- Language identification

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

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

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

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

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

What's new in Content Manager 8.1

The following changes have been made in Content Manager 8.1:

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

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

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

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

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

Common system administration
A single client application provides separate access to Content Manager

and Enterprise Information Portal administration. Within Content Manager, administrative domains provide a way to limit administrative access to subsections of the library server.

Full-function desktop client and enhanced eClient

Client enhancements provide users with an out-of-the-box application for rapid deployment or line of business application integration. The Client for Windows® supports integrated text search, document routing, the hierarchical data model (to a single child component level), versioning, and index during import. The eClient includes integrated text search, EIP advanced workflow, version control, and multi-valued attributes.

Easier installation

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

Information center

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

Accessibility

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

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

Comprehensive code samples

Extensive real-world code samples for the majority of the Content Manager 8.1 key features

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

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

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

Searching for customer information

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

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

The need

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

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

The solution

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

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

The Enterprise Information Portal solution

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

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

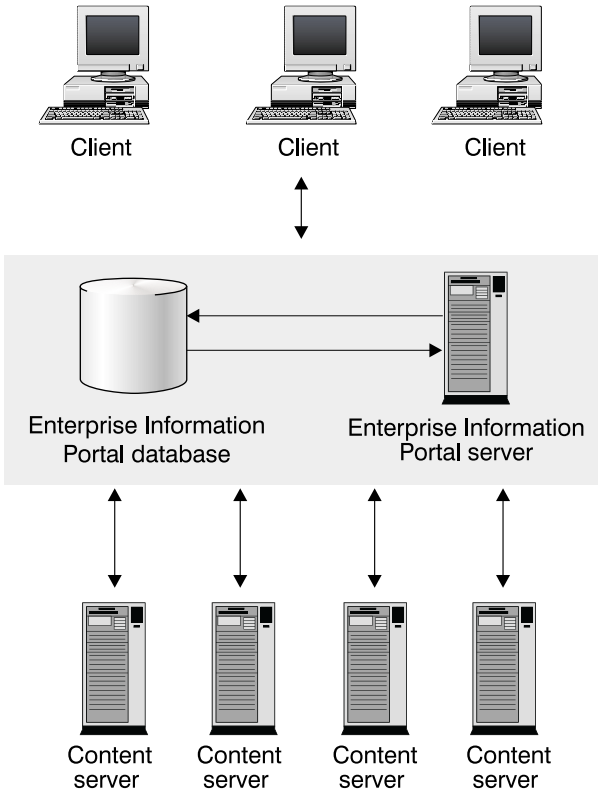

Figure 1. Multiple-tier architecture

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

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

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

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

IBM Enterprise Information Portal for Multiplatforms components

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

Administration database

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

Migrating your Enterprise Information Portal Version 7.1 database: You must migrate your data from Enterprise Information Portal Version 7.1 *before* using your Enterprise Information Portal Version 8 Release 1 administration database.

Administration client

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

- Define each content server for federated searching.
- Identify native entities and attributes on content servers and map them to federated entities.
- 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.

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

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

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

- Have DB2 Client support installed and use the DB2 Client Configuration Assistant to configure access to the system administration database on each workstation on which the administration client is installed.
- Use a Remote Method Invocation (RMI) configuration by starting the RMI server on which the IBM Enterprise Information Portal for Multiplatforms database is installed. You need to make sure that your `\CMBROOT\cmbclient.ini` file points to this server.

Connectors

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

- Relational database connectors (DB2, JDBC, ODBC)
- Federated connector (to the IBM Enterprise Information Portal for Multiplatforms database)
- Content Manager Version 8 Release 1
- Content Manager connector for earlier versions

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

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

IBM eClient

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

Information Mining

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

IBM Web Crawler

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

Workflow

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

Content Manager Text Search Engine

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

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

Content Manager image search server and client

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

Restriction: Image search server's functionality is deprecated in both EIP Version 8 Release 1 and Content Manager 8.1. Its functionality is still available using pre-Version 8.1 Content Manager.

What's new in Version 8.1 APIs

Enterprise Information Portal Version 8 Release 1 provides unprecedented access to disparate content servers. The new features and components include:

- XML import capabilities:

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

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

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

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

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

New and changed Java classes

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

New classes:

- dkAbstractAccessControlList
- dkAbstractAttrGroupDef
- dkAbstractAuthorizationMgmt
- dkAbstractConfigurationMgmt
- dkAbstractDatastoreAdmin
- dkAbstractDataObjectBase
- dkAbstractPrivilege
- dkAbstractPrivilegeGroup
- dkAbstractPrivilegeSet

- dkAbstactResultSetCursor
- dkAbstractUserDef
- dkAbstractUserMgmt
- dkAttrGroupDef
- dkAuthorizationMgmt
- dkCheckableObject
- DKChildCollection
- dkConfigurationMgmt
- DKLinkCollection
- dkPersistentCheckableObject
- dkPrivilege
- dkPrivilegeGroup
- dkUserDef
- dkUserGroupDef

Changed classes:

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

Changes in class behavior for EIP Version 8.1

Between EIP 7.1 and EIP 8.1, some classes or class components have changed their behavior. This section describes those changes. For detailed information, see the *online API reference.*

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

New and changed C++ classes

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

New classes:

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

Changed classes:

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

Changes in class behavior for EIP 8.1

Between EIP 7.1 and EIP 8.1, some classes or class components have changed their behavior. This section describes those changes. For detailed information, see the *online API reference*.

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

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

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

  count = ddo->dataCount();
  for (data_id = 1; data_id <= count; data_id++)
  {
  any = ddo->getDataId(data_id);
  if (any.typeCode() == DKAny::tc_string)
  {
  strTmp = any; // This is how to get a DKString from a DKAny
    }
  }
  ```

- dkXDO: getPidObject() and setPidObject(DKPid* pid)

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.

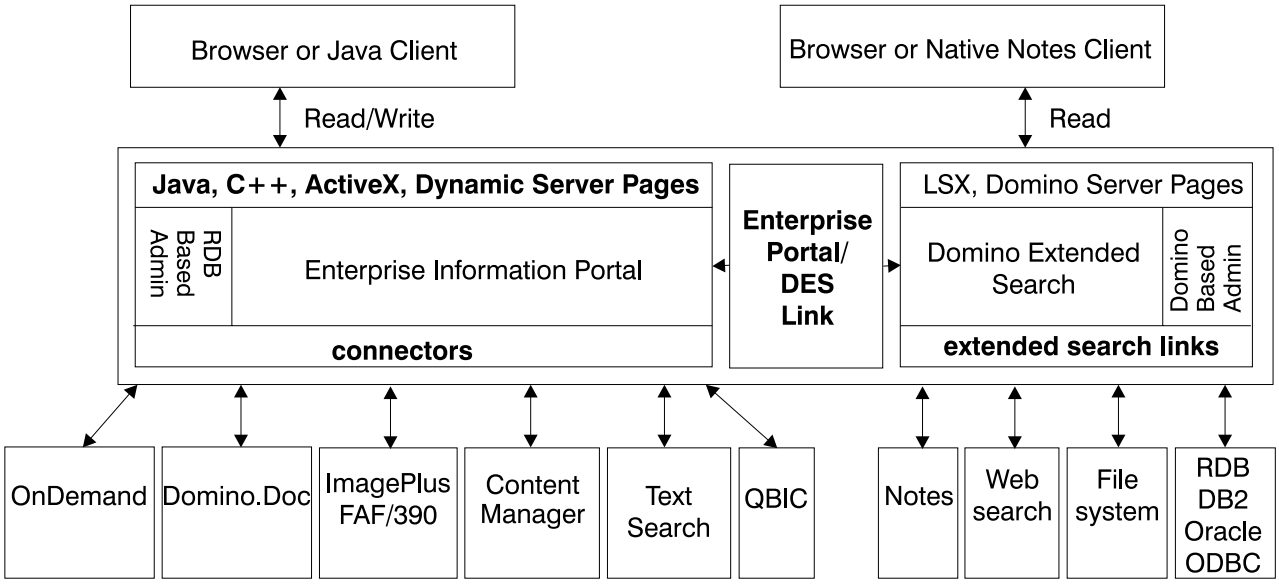

Figure 2. Enterprise Information Portal organization

Understanding data access through content servers

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

- Content Manager Version 8 Release 1
- Content Manager pre-Version 8.1
- Domino.Doc
- Extended Search
- ImagePlus for OS/390
- Content Manager OnDemand
- VisualInfo™ for AS/400
- DB2
- DB2 DataJoiner
- Information Catalog

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

Applications that use Enterprise Information Portal can create a federated 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 dynamic data object concepts

In compliance with Object Management Groups' (OMG) CORBA Persistent Object Service and Object Query Service Specification, Enterprise Information Portal provides an implementation of the dynamic data object (DDO) and its extension, the extended data object (XDO), which are part of the CORBA Persistent Data Service (PDS) protocols. The concepts of DDO and XDO are not specific to any one 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 3 on page 13 shows dynamic data objects and data items.

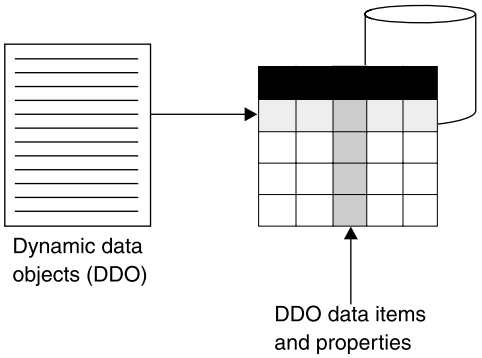

Figure 3. Dynamic data objects and items

Extended data objects (XDO)

An XDO is a representation of complex multimedia data, for example a 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 4 shows an example of XDOs.

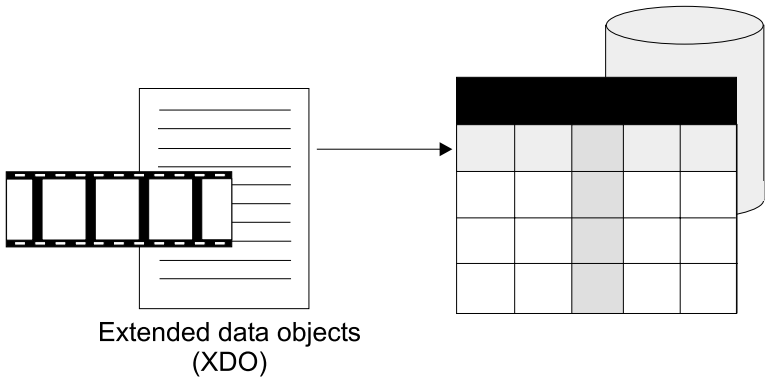

Figure 4. Extended data objects (XDOs)

Representing multimedia content

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

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.

Chapter 3. Working with a federated datastore and federated searching

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

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

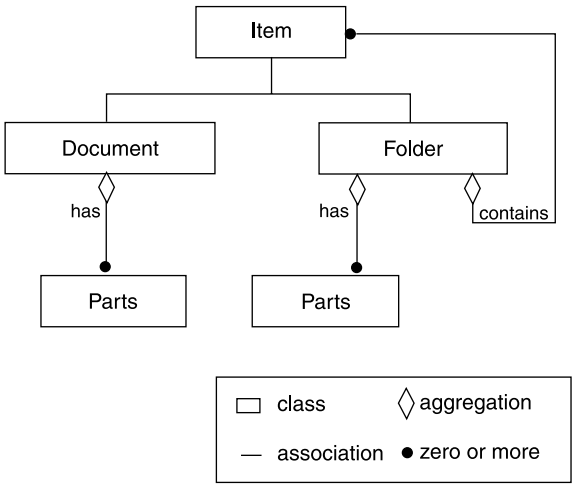

Figure 5. Federated document view

The lines in the Figure 5 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 3 relates the terminology used for various content servers to the federated model:

Table 3. Mapping terminology for each datastore

Content server	Data source	Entity	Attribute	View
Content Manager	Library server	item type	attribute	item type view
earlier Content Manager	Library server	index class	• attribute • key attribute	index class view
OnDemand	OnDemand server	• application group • search template	• field • criteria	N/A
ImagePlus	ImagePlus for OS/390 server	entity	attribute	N/A
Content Manager for AS/400	Content Manager for AS/400 server	index class	attribute	index class view
Domino.Doc	Domino server	• library • cabinet • binder	key field	N/A
Extended Search	Extended Search server	database name	database name	N/A
Relational database	IBM DB2 UDB, JDBC, ODBC, IBM DB2 DataJoiner	table	column	view
Information Catalog	DB2 Warehouse Manager Information Catalog Manager	index class	property	
Enterprise Information Portal federated datastore	mapping server	mapped federated entity	mapped federated attribute	search template

Figure 6 on page 17 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.

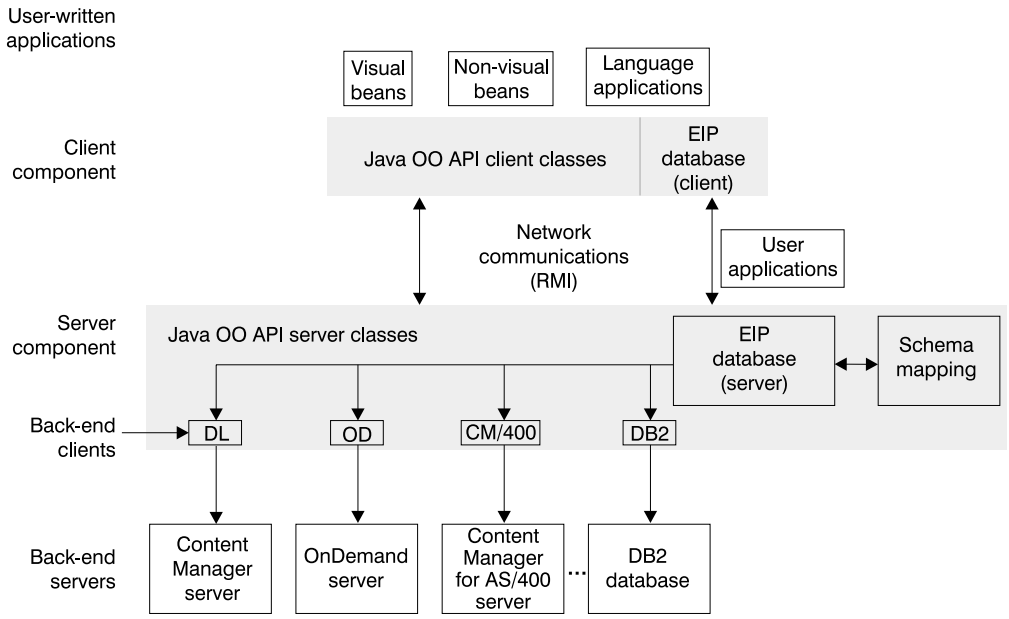

Figure 6. Structure of federated searches

RMI 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 7 on page 19.

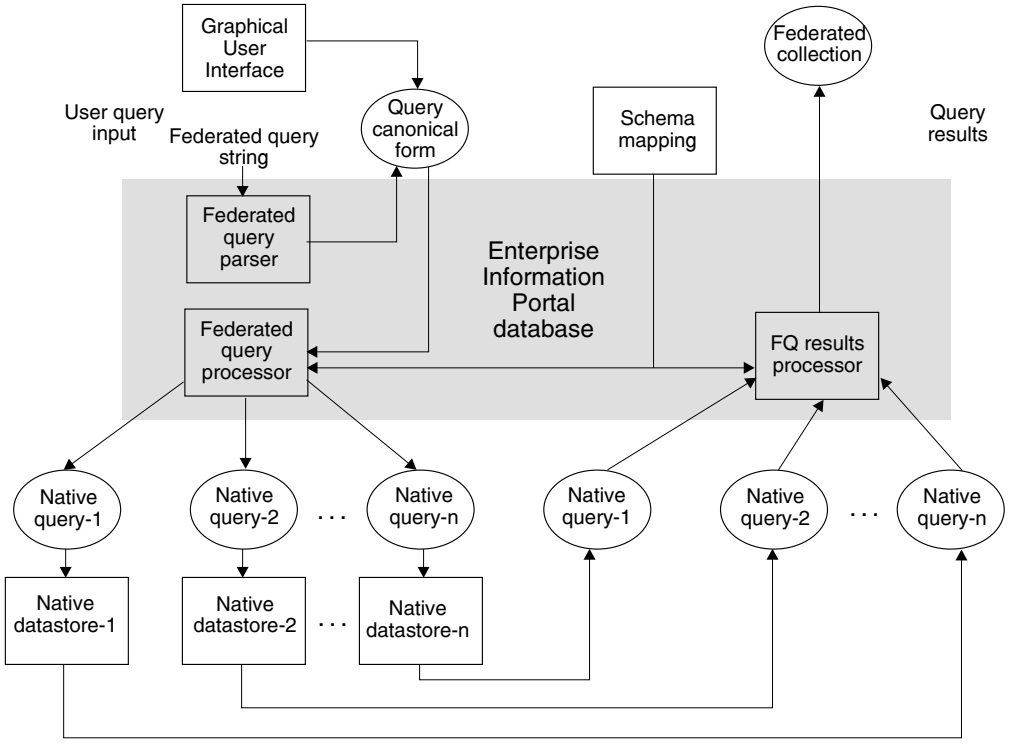

Figure 7. Federated query processing

Federated query syntax

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

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

      [and

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

The NOT operator is not supported in federated searches.

Examples of federated query strings Federated parametric query using the LIKE operator:

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

Federated parametric query using the LIKE and > operator:

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

Federated parametric query using the LIKE and < operator:

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

Federated parametric query using the BETWEEN operator: MAX_RESULTS returns all results when set to zero.

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

Federated parametric query using the NOTBETWEEN operator:

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

Federated parametric query using the IN operator:

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

Federated parametric query using the NOTIN operator:

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

Federated parametric query using the == operator:

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

Federated parametric query using the <> operator:

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

Federated parametric query using the AND and OR operators:

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

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

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

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

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

Federated text query: Searches for text in content. The text can be a word or a phrase. The ASSOCIATED_ENTITY keyword is only applicable when a federated

entity is text-searchable. This is only valid when the text-searchable federated entity (FedEntity) is mapped to a Content Manager Version 8 text-searchable item type or an Extended Search text-searchable entity.

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

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

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

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

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

Federated parametric and text query using the OR operator:

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

Federated parametric and text query using the AND operator:

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

Federated parametric and text query on attributes using the OR operator:

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

Federated parametric and text query on attributes using the OR operator:

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

Working with system administration

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

Customizing the EIP system administration client

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

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

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

Chapter 4. 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 functions, 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, the Content Manager Version 8 Release 1 text search feature.
- Client/server implementation for Java application users
- Workflow capability
- Administration functions

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 EIP 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 IBM Enterprise Information Portal for Multiplatforms Java and C++ API sets:

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

Packaging for the Java environment

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

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

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

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

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

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

Programming tips

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

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, AIX, and Solaris environments

When you set up your Windows, AIX, or Solaris 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

Shared objects for AIX

DLLs for Windows

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 8.1 → Development Window.** As an alternative, you can run `cmbenv81.bat` in a DOS command prompt to set up the environment.

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

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

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

For AIX

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

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

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

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

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

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

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

For Solaris

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

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

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

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

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

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

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

Using Remote Method Invocation (RMI) with content servers

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

An RMI server can connect to an infinite number of datastores, but each server must be connected to at least one datastore. A master RMI server can reference other RMI servers in the server pool. When an RMI client first searches for a datastore, it starts an RMI server. If the datastore is not found there, the RMI pool servers 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 `cmbregist81.bat` on Windows or `cmbregist81.sh` on AIX or Solaris. Before starting the RMI server, define the correct port number and server type. For information on configuring and administering RMI servers, refer to *Planning and Installing Enterprise Information Portal* and *Managing Enterprise Information Portal*.

Multiple search facilities

Content Manager Version 8 Release 1 offers an integrated text feature, which does not require a separate text search facility. See “Understanding text search” on page 101.

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.

Parametric query
Query requiring an exact match on the condition specified in the query

predicate and the data values stored in the datastore. You use a parametric query to search for a documents by a customer's name.

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 (TSE) and all of its functions can only be used with earlier Content Manager. Content Manager Version 8 and later offers an integrated text feature, which does not require a separate text search facility. See "Understanding text search" on page 101.

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

`CMBTMDSTREAMTRACE=`*`fileName`*

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

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

`CMBTMTRACE=`*`fileName`*

The following environment setting writes the text search terms to a specified file:
`CMBTMTERM=`*`fileName`*

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

For parametric queries

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

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

When a DKException is caught, it allows you to see any error messages, error codes, and error states that occurred while running. When an error is caught, an error message is issued along with the location of where the exception was thrown. 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 DKDatastore*xx* (where *xx* indicates a specific content server, for example, Content Manager (ICM), earlier 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 DKDatastore*xx* class provides methods for connecting to it and disconnecting from it. The following example uses an earlier Content Manager library server named LIBSRVRN, the user ID FRNADMIN and password PASSWORD. For information on Content Manger, see "Connecting to the Content Manager system" on page 82; for other content servers, see Chapter 6, "Working with other content servers in Java" on page 141. 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 earlier 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 DKDatastore*xx*. The following example shows how to set and get the option for establishing an administrative session on a earlier 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

DKDatastore*xx* provides a method to list the servers that it can connect to. The list of servers are returned in a DKSequentialCollection of DKServerInfo*xx* objects (where *xx* identifies the specific content server, for example, ICM for Content Manager, DL for an earlier 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 DKServerDef*xx* object you can retrieve the server name and server type, and use the server name to establish a connection to it.

The following example 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 attributes for a content server

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

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

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

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

```
. . .
try {
  DKSequentialCollection pCol = null;
  dkIterator pIter = null;
  DKSequentialCollection pCol2 = null;
  dkIterator pIter2 = null;
  DKServerDefICM pSV = null;
  String strServerName = null;
```

```
String strItemType = null;
DKComponentTypeDefICM itemTypeDef = null;
DKAttrDefICM attrDef = null;
DKDatastoreDefICM dsDefICM = null;
int i = 0;
int j = 0;

// ------ Create the datastore and connect (assumes the
//     parameters for the connection are previously set)
DKDatastoreICM dsICM = new DKDatastoreICM();
dsICM.connect(db,userid,pw,"");
// ----- List the item types
pCol = (DKSequentialCollection) dsICM.listEntities();
pIter = pCol.createIterator();
i = 0;
while (pIter.more() == true)
{
  i++;
  itemTypeDef = (DKComponentTypeDefICM)pIter.next();
  strItemType = itemTypeDef.getName();
  System.out.println("item type name [" + i + "] - " + strItemType);
  System.out.println("   type " + itemTypeDef.getType());
  System.out.println("   itemTypeId " + itemTypeDef.getId());
  System.out.println("   compID " + itemTypeDef.getComponentTypeId());
  // ----- List the attributes
  pCol2 = (DKSequentialCollection) dsICM.listEntityAttrs(strItemType);
  pIter2 = pCol2.createIterator();
  j = 0;
  while (pIter2.more() == true)
  {
    j++;
    attrDef = (DKAttrDefICM)pIter2.next();
    System.out.println("    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("      sequenceNo " + attrDef.getSequenceNo());
    System.out.println("      userFlag " + attrDef.getUserFlag());
  }
}

dsICM.disconnect();
}
catch(DKException exc)
{
// ----- Handle the exceptions
```

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

Working with DDOs

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

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

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

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

Creating a DKDDO

DKDDO has several constructors. In earlier versions of EIP, you can create a DKDDO by calling its constructor without any parameters.

In EIP Version 8 and later, for some datastores you can create a DKDDO by using the createDDO() method in the DKDatastore *XX* class. For example, DKDatastoreICM and DKDatastoreOD provide such methods.

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

You can use other constructors, passing various parameters; for example, to create a DKDDO by supplying content server and object type for Content Manager Version 8 Release 1:

```
DKDatastoreICM dsICM = new DKDatastoreICM(); // create a Content Manager datastore
DKDDO cddo = dsICM.createDDO("GRANDPA",  // create a DDO to hold an object type
DKConstant.DK_CM_DOCUMENT);
```

For other connectors, 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; refer to the *online API reference* for information on the constructors.

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, folder, or item. 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.

Use one of the createDDO() methods in DKDatastoreICM to create a DDO in Content Manager; refer to “Representing items using DDOs” on page 81.

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.

```
// ----- Create an earlier Content Manager datastore
DKDatastoreDL dsDL = new DKDatastoreDL();
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 4. 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 an earlier Content Manager datastore
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 a document
         ....
         break;
       case DK_CM_FOLDER:
         // --- process a folder
       case DK_CM_ITEM:
         // --- Process an item in Content Manager
         ....
         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 DKBlob*xx*, where *xx* is the suffix representing the specific server; for example, use DKBlobDL for earlier Content Manager, DKBlobOD for OnDemand, or DKBlobIP for ImagePlus for OS/390. When you create a DKBlob*xx* object, you must pass it the datastore DKDatastore*xx*. For Content Manager, you use DKLogICM to create the XDO.

Using an XDO PID

To use an XDO to locate and store data, you must supply a PID for the DKBlob*xx*, using a DKPidXDO*xx*. An item ID and part ID are required for the DKPidXDO*xx*. For relational databases (RDB), the table name, column name and data predicate are required for DKPidXDO*xx*.

Understanding XDO properties

Use the methods of the DKBlob*xx* to set the properties of an XDO where they apply; all properties are not available for all content servers. When loading, default values for the properties are set if specific values are not specified. For example, with earlier 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 earlier Content Manager correctly, you must set SearchEngine, SearchIndex, and SearchInfo in the extension object DKSearchEngineInfoDL.

For working with XDOs in Content Manger, see “Working with items” on page 82.

Programming tips

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

For earlier Content Manager, Content Manager for AS/400, and IP, you identify an XDO by the combination of item ID, part ID and the RepType. For RDB, the key to identify an XDO is combination of table, column and data predicate string. To

handle a stand-alone XDO, you provide the item ID and part ID. The RepType is optional since the system provides a default value for it.

Use the add method of DKBlob*xx* to add the current content to a 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.

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

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

```
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 in earlier Content Manager:

```
// ----- 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\dl directory. TLoadSampleDL.java shows more examples of XDO use.

Stand-alone XDO

All of the following examples are specific to earlier Content Manager. For examples for Content Manager and other content servers, see "Representing items using DDOs" on page 81, Chapter 6, "Working with other content servers in Java" on page 141, and 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
    String itemId = "CPPIORH4JBIXWIY0";    //existing itemId
    String fileName = "/u4/mmdb/images/choice.gif";
//an AIX file for add  try
    {
      DKDatastoreDL dsDL = new DKDatastoreDL();
//required datastore
       dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
```

```
//connect     DKBlobDL axdo = new DKBlobDL(dsDL);
  //create XDO DKPidXDODL  apid = new DKPidXDODL();
   //create Pid      apid.setPartId(partId);
  //set partId in Pid
 apid.setPrimaryId(itemId);
 //set itemId in Pid
axdo.setPidObject(apid);
 //setPid to XDO    axdo.setRepType("ABCD");
  //set representation type  axdo.setContentClass(DK_DL_CC_GIF);
   //set ContentClass
        axdo.add(fileName);   //add from file
       dsDL.disconnect();  //disconnect from datastore
        dsDL.destroy();      //destroy the datastore
       }
        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();
    }    }
}
```

Examples of working with an XDO

The following examples illustrate using a stand-alone XDO.

Adding an annotation object to an XDO

To add an annotation object, insert the following statements before the add method in the previous example:

```
//------ set DKAnnotationDL ------ (using extension object)
axdo.setAffiliatedType(DK_DL_ANNOTATION);
DKAnnotationDL ann = new DKAnnotationDL();
ann.setPart(14);
ann.setPageNumber(1);
ann.setX((short)5);
ann.setY((short)5);
axdo.setExtension("DKAnnotationDL", (dkExtension)ann);
```

Retrieving, updating, and deleting an XDO

To retrieve, update or delete an object in a content server, you provide the correct item ID, part ID and RepType for the XDO that represents the object. The following example illustrates working with earlier Content Manager; for Content Manager Version 8 and later, see “Retrieving items” on page 88.

```
public class txdorudDL implements DKConstantDL
{
  public static void main(String[] args)
  {
    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
         //  replace following with your library server, user ID, password
         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 an earlier 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 =  "K1A04EWBVHJAV1D7";
    if (args.length == 3)
    {
      partId = (short)Integer.parseInt(args[0], 10);
      repType = args[1];
      itemId = args[2];
      System.out.println("You enter: java txdoDelVSDL " +
       + partId + " " + repType + " " + itemId);
    }
    // ----- Check the arguments for main
    if (args.length == 2)
    {
      partId = (short)Integer.parseInt(args[0], 10);
      repType = args[1];
      System.out.println("You enter: java txdoDelVSDL " +
       + partId + " " + repType);
    }

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

    // ----- Processing
    try
    {
```

```
      DKDatastoreDL dsDL = new DKDatastoreDL();
      System.out.println("connecting to datastore...");
      // ----- replace following with your library server, userid, password
      dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD","");
      System.out.println("datastore connected");

      DKBlobDL axdo = new DKBlobDL(dsDL);
      DKPidXDODL  apid = new DKPidXDODL();
      apid.setPartId(partId);
      apid.setPrimaryId(itemId);
      apid.setRepType(repType);
      axdo.setPidObject(apid);
      boolean flag2 = axdo.isCategoryOf(DK_DL_MEDIA_OBJECT);
      System.out.println("isMediaObject?=" + flag2);
      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 DKStorageManageInfo*xx*, where *xx* is the suffix representing the specific server.

The following example uses DKStorageManageInfoDL, for an earlier Content Manager server; for Content Manager Version 8 and later, see Chapter 5, “Working with Content Manager 8.1 in Java” on page 69 .

```
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\dl 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\dl directory.

Working with XML

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

Importing XML documents

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

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

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

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

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

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

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

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

The XML Document Type Definition (DTD)

In order to import content to Content Manager to store as XML, you must store the content in XML documents that conform to ddo.dtd, which is located at `CMROOT\samples\java\dl\ddo.dtd`.

```
<!ELEMENT ddo (pid?, pidIdStrings*, propertyCount?, property*, dataCount?, dataItem*, xdoValue?)>
<!ATTLIST ddo   entityName CDATA #REQUIRED
                xmlns      CDATA #FIXED "http://www.omg.org/pub/docs/formal/97-12-12.pdf#ddo/EIP-7.1"
                   xdoType    CDATA #IMPLIED
                   dsType     CDATA #IMPLIED>
<!ELEMENT pid EMPTY>
<!ATTLIST pid      dsType          CDATA #IMPLIED
                   dsName          CDATA #IMPLIED
                   objectType      CDATA #IMPLIED
                   pidString       CDATA #IMPLIED>
<!ELEMENT pidIdStrings EMPTY>
<!ATTLIST pidIdStrings  idPosition   CDATA #REQUIRED
                        idValue      CDATA #REQUIRED>
<!ELEMENT propertyCount (#PCDATA)>
<!ELEMENT property EMPTY>
<!ATTLIST property       propertyId    CDATA #IMPLIED
                         propertyName  CDATA #IMPLIED
                         propertyValue CDATA #IMPLIED
                         propertyType  CDATA #IMPLIED>
<!ELEMENT dataCount (#PCDATA)>
<!ELEMENT dataItem (dataPropertyCount?, dataProperty+, (dataValue | dataValues))>
<!ATTLIST dataItem       dataId         CDATA #IMPLIED
                         dataName       CDATA #REQUIRED
                         dataNameSpace  CDATA #IMPLIED>
<!ELEMENT dataPropertyCount (#PCDATA)>
<!ELEMENT dataProperty EMPTY>
<!ATTLIST dataProperty      propertyId    CDATA #IMPLIED
                            propertyName  CDATA #IMPLIED
                            propertyValue CDATA #IMPLIED
                            propertyType  CDATA #IMPLIED>
<!ELEMENT dataValues (dataValueCount?, dataValue*)>
<!ATTLIST dataValues       collectionName     CDATA #IMPLIED>
<!ELEMENT dataValueCount (#PCDATA)>
<!ELEMENT dataValue (#PCDATA | ddo | xdoRef | linkRef)*>
<!ELEMENT linkRef (linkSource, linkTarget, linkItem?)>
       <!ATTLIST linkRef linkTypeName     CDATA #REQUIRED>
<!ELEMENT linkSource (ddo)>
<!ATTLIST linkSource sameAsParentDDO (yes | no) "no">
<!ELEMENT linkTarget (ddo)>
<!ATTLIST linkTarget sameAsParentDDO (yes | no) "no">
<!ELEMENT linkItem (ddo)>
<!ELEMENT xdoRef (xdoPid, xdoIdStrings*, xdoValue)>
<!-- partId deprecated it is replaced by xdoIdString on an xdoRef   -->
<!-- repType deprecated it is replaced by xdoIdString on an xdoRef  -->
<!ELEMENT xdoPid EMPTY>
<!ATTLIST xdoPid       dsType            CDATA #REQUIRED
                       dsName            CDATA #IMPLIED
                       xdoType           CDATA #REQUIRED
                       objectType        CDATA #IMPLIED
                       partId            CDATA #IMPLIED
                       repType           CDATA #IMPLIED
                       pidString         CDATA #IMPLIED>
<!ELEMENT xdoIdStrings EMPTY>
<!ATTLIST xdoIdStrings  idPosition   CDATA #REQUIRED
                        idValue      CDATA #REQUIRED>
<!ELEMENT xdoValue (contentType?, specificInfo*, searchEngineInfo?, smsInfo?, xdoContent?)>
<!ATTLIST xdoValue       refType      CDATA #REQUIRED
                         refEncoding  CDATA #IMPLIED
                         mimeType     CDATA #REQUIRED
                         XML-LINK     CDATA #IMPLIED
                         HREF         CDATA #IMPLIED>
<!ELEMENT contentType (#PCDATA)>
```

```
<!ELEMENT specificInfo EMPTY>
<!ATTLIST specificInfo          infoGroup      CDATA #REQUIRED
                                infoName       CDATA #REQUIRED
                                infoValue      CDATA #REQUIRED>
<!-- searchEngineInfo deprecated it is replaced by specificInfo   -->
<!ELEMENT searchEngineInfo EMPTY>
<!ATTLIST searchEngineInfo       searchEngine     CDATA #REQUIRED
                                 searchIndex      CDATA #REQUIRED
                                 searchInfo       CDATA #REQUIRED>
<!-- smsInfo deprecated it is replaced by specificInfo   -->
<!ELEMENT smsInfo EMPTY>
<!ATTLIST smsInfo         smsRetention          CDATA #IMPLIED
                          smsCollection         CDATA #IMPLIED
                          smsMgmtClass          CDATA #IMPLIED
                          smsStorageClass       CDATA #IMPLIED
                          smsObjServer          CDATA #IMPLIED>
<!ELEMENT xdoContent (#PCDATA)>
```

Storing content in XML documents

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

Sample for Content Manager

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

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

Sample for earlier Content Manager

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

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

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

Extracting content from different XML sources

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

Here are examples of each XML source:

XML from a file:

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

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

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

Exporting XML

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

The following example illustrates exporting XML:

```
DKNVPair xmlDestination = null;

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

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

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

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

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

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

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
// ----- Create and add some more BLOBs to the
 //         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. 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. After creating the iterator, use the setToFirst() method to point to the first item. The following example shows using an iterator:

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

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

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

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

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 a 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.

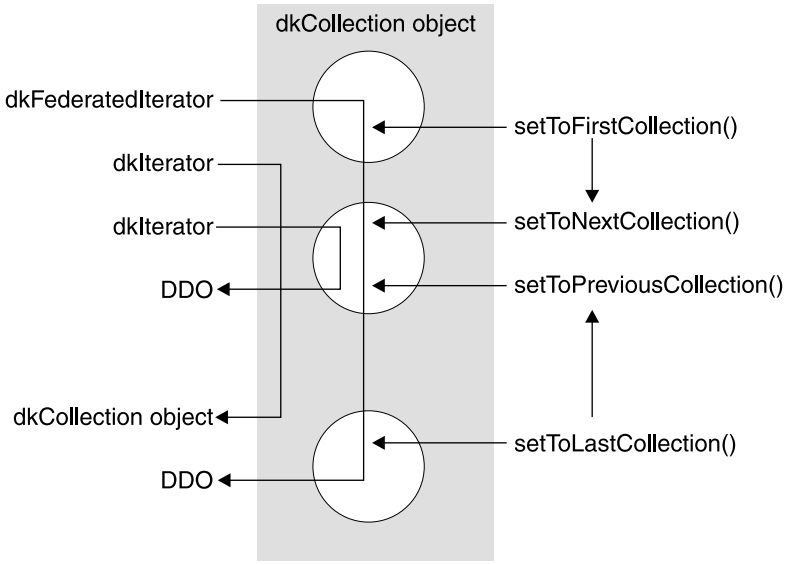

Figure 8. DKFederatedCollection structure and behavior

The rectangle 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

In EIP, you query a content server datastore and receive results in a dkResultSetCursor or DKResults object. First you create a query string that is specific to the content server you want to search. Then you call the execute or evaluate method on the datastore object to run the query. Using the APIs of the content server, the datastore performs query processing tasks, such as preparing and executing a query, monitoring the status of a query execution, and storing the results. Some datastores support using a query object as an alternative; earlier Content Manager and the federated datastore are two of these.

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

For Content Manager parametric and text queries are integrated. You should not use query objects; for information on how to query in Content Manager, see “Querying the Content Manager server” on page 99.

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 an object and add it to a collection. Not all content servers support using a result set cursor for updating and deleting.

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 65.

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 65 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.
- Leaving a dkResultSetCursor open for long periods of time may degrade performance of concurrent users for some content servers.

Using parametric queries

A parametric query is a query requiring an exact match on the condition specified in the query predicate and the data values stored in the 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 in earlier Content Manager. For examples of query string in Content Manger, see “Query examples” on page 104. The condition of the query is to search for all documents or folders where the attribute DLSEARCH_DocType is not `null`. The maximum number of results returned is limited to five, and the content is set to `YES` so that contents of the document or folder are returned.

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

You can specify more than one search criteria for a parametric query. The following example shows how to specify a query on two index classes for earlier Content Manager:

```
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 DKDatastore*xx* that represents a content server contains a method for creating a query object. You use the query object to execute the query and obtain the results. The following example shows how to create a parametric query object and execute the query on an earlier Content Manager server; you should not use a query object with Content Manger Version 8 or later. 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;
DKNVPair parms[] = null;
// ----- Connect to the datastore
dsDL.connect(libSrv,userid,pw,"");
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
                      "MAX_RESULTS=5," +
                      "COND=(DLSEARCH_DocType <> NULL));" +
                      "OPTION=(CONTENT=YES;"       +
                              "TYPE_QUERY=STATIC;" +
                              "TYPE_FILTER=FOLDERDOC)";
// ----- Create the query using the query string
pQry = dsDL.createQuery(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Execute the query
pQry.execute(parms);
// ----- Process the results
pResults = (DKResults)pQry.result();
processResults((dkCollection)pResults);
// ----- Disconnect when you are through
dsDL.disconnect();
dsDL.destroy();
```

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

Executing a parametric query from a content server

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

```
// ----- Create the datastore and cursor
DKDatastoreDL dsDL = new DKDatastoreDL();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
// ----- Connect to the content server
dsDL.connect(libSrv,userid,pw,"");
// ----- Formulate the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
                     "MAX_RESULTS=5," +
                     "COND=((DLSEARCH_DocType <> NULL)" +
                     "AND (DLSEARCH_Date >= 1995)));" +
                     "OPTION=(CONTENT=YES;"       +
                             "TYPE_QUERY=DYNAMIC;" +
                             "TYPE_FILTER=FOLDERDOC)";
...
// ----- Execute the query using the query string
pCur = dsDL.execute(cmd, DK_CM_PARAMETRIC_QL_TYPE, parms);
// ----- Process query results as you want
...
// ----- When finished with the cursor, delete it, and disconnect
pCur.destroy();
dsDL.disconnect();
dsDL.destroy();
```

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

Evaluating a parametric query from a content server

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

```
// ------ Create the query string
String cmd = "SEARCH=(INDEX_CLASS=DLSAMPLE," +
                   "COND=((DLSEARCH_Date >= \"1995\") AND " +
                    "(DLSEARCH_Date <= \"1996\")));" +
                    "OPTION=(CONTENT=NO;"       +
                          "TYPE_QUERY=DYNAMIC;" +
                          "TYPE_FILTER=FOLDERDOC)";
DKNVPair parms[] = null;
DKDDO item = null;
// ----- Create the datastore and connect
//  replace following with your library server, user ID,
password DKDatastoreDL dsDL;
dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");

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

Using text query

In Content Manager Version 8 and later, text and parametric queries are integrated; see “Creating combined parametric and text search” on page 103.

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

Formulating a text query string

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

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

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

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

Executing a text query from the datastore

The DKDatastore*xx* 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 an earlier Content Manager datastore:

```
// ----- Create the datastore; declare query and the results
DKDatastoreTS dsTS = new DKDatastoreTS();
dkResultSetCursor pCur = null;
DKNVPair parms[] = null;
```

```
// ----- Connect to the datastore
//       for example, dsTS.connect
("zebra","7502",DK_TS_CTYP_TCPIP);
dsTS.connect(srchSrv,"","","");

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

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

Evaluating a text query from the datastore

The DKDatastore*xx* that you use to represent a content server provides an evaluate method to run a query and return a DKResults collection. The following example shows how to evaluate a text query against an earlier Content Manager 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;
DKNVPair parms[] = null;
// ----- Connect to the content server
//     replace following with your library server,
user ID, password
dsTS.connect("TM","","","LIBACCESS=(LIBSRVRN,
FRNADMIN, PASSWORD)");
// ----- Formulate the query string
String cmd = "SEARCH=(COND=('UNIX operating' AND system));" +
             "OPTION=(SEARCH_INDEX=TMINDEX; MAX_RESULTS=5;
              MATCH_INFO=YES; MATCH_DICT=NO)";

...

pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);
DKDDO item = null;
DKMatchesInfoTS pMInfo = null;
DKMatchesDocSectionTS pMSect = null;
DKMatchesParagraphTS pMPara = null;
DKMatchesTextItemTS  pMText = null;
int i = 0;
int j = 0;
int k = 0;
int m = 0;
int lCCSID = 0;
int lLang = 0;
int lOffset = 0;
int lLen = 0;
int numberSections = 0;
int numberParagraphs = 0;
int numberTextItems = 0;
int numberNewLines = 0;
String strDoc = "";
String strSection = "";
String strText = "";
Object anyObj = null;
while (pCur.isValid())
{
   // ----- Get the next DKDDO
   item = pCur.fetchNext();
   if (item != null)
   {
     // ----- 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 items that met the criteria of the query.

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 podition
   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 to indicates that all items will be put into the collection.

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

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

// ----- Create and execute a query object
String query1 = "SEARCH=(INDEX_CLASS=GRANDPA,COND=(Title <> null));";
DKParametricQuery pq =
```

```
    (DKParametricQuery) dsDL.createQuery(query1,DK_CM_PARAMETRIC_QL_TYPE, null);
pq.execute();
// ----- Get the reuslt
DKResult rs = (DKResults) pq.result();
```

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.

Chapter 5. Working with Content Manager 8.1 in Java

This section describes the Content Manager Version 8.1 connector (ICM connector) application programming interfaces (APIs). The ICM connector is an extension of the Enterprise Information Portal (EIP) framework, so it is essential that you understand the EIP framework concepts described in Chapter 2, "Enterprise Information Portal application programming concepts" on page 11 before continuing with this information.

You can use the ICM connector APIs to build and deploy custom applications that access a Content Manager content server. You can also use the APIs to integrate your existing applications into a Content Manager content server.

This section contains the following information:

- Understanding the Content Manager system
- Understanding Content Manager concepts
- Planning a Content Manager application
- Creating a Content Manager application
- Controlling access to information
- Processing transactions
- Searching for items

Understanding the Content Manager system

The main components of the Content Manager system include a library server, one or more resource managers, and a set of object-oriented application programming interfaces (APIs). To administer your Content Manager system, you are also provided with a Java-based system administration client.

The library server provides you with flexible data modeling capabilities, secure access to your system, efficient managing of content, and other features. The library server manages the relationships between items in the system and controls access to all of the system information, including the information stored in the resource managers configured in the system.

The resource manager is the component that stores the actual content of any binary object, like a scanned image, an office document, or video. You can integrate other resource managers, like Content Manager VideoCharger or other non-IBM products, into your Content Manager system. With the resource manager you can complete the following tasks:

- Automatically move content from slow, costly media to high speed, less expensive media using System Managed Storage (SMS)
- Access the resource manager directly from a Web browser
- Retrieve all or part of an object
- Synchronize your data with the library server

Available in Java and C++, the APIs provide applications with access to the Content Manager system. Using the APIs, your applications can take advantage of all of the Content Manager functionality, such as data modeling, integrated parametric and text search, third-party data access and delivery, and so forth.

The diagram in Figure 9 illustrates how the system components fit together. Keep in mind that this is only one implementation of a Content Manager system. In another system configuration you might have four resource managers, for example.

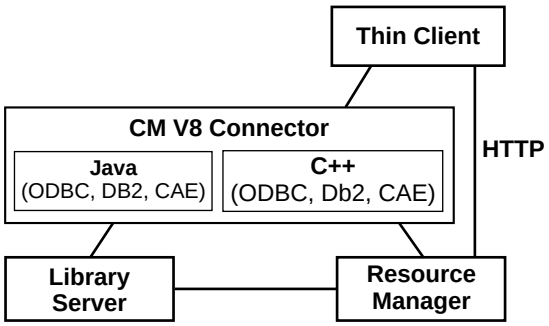

Figure 9. System configuration

Understanding Content Manager concepts

This section describes important Content Manager concepts. It is imperative that you understand the Content Manager concepts before you proceed to the programming tasks. The information described in this section includes:

- Items
- Attributes
- Item types
- Root and child components
- Objects
- Links and References
- Documents
- Folders
- Versioning
- Access control
- Data model
- Document management data model

Items

An item is the basic entity managed by the library server. Examples of items include a policy, claim, phone number, and so forth. An *item* is a generic term for an instance of an item type. If an object is a discrete piece of digital content, then an item is a representation of that object. The item is not the object, but it thoroughly identifies it and how to find it. In the system, items represent objects including documents and folders. To define business objects, like a document, you work with item definitions.

When an application creates an item, Content Manager assigns the item several system-defined attributes and allows you to define your own attributes.

The system-defined attributes include a creation time stamp and an item identifier (item ID). The item ID is unique for every item. The itemID is stored by Content Manager and used to locate the item within the library server. When writing your application, you use the itemID to access all of the data associated with the item.

Attributes

An *attribute* is a unit of data that describes a certain characteristic or property (for example, name, address, age, and so forth) of an item, and which can be used to locate that item.

You can group attributes to make attribute groups. For example, the address attribute can be made up of a group of attributes including street, city, state, and zip code.

You can also define attributes that have multiple values; such attributes are called multi-valued attributes. For example, you can store multiple addresses, home address, work address, and so forth for a policy owner.

Item types

An *item type* (index class in earlier Content Manager versions) is essentially a template for defining and later locating like items. An item type consists of a root component, zero or more child components, and a classification. An item type is the overall structure containing all the components and associated data. For example, in an insurance scenario a policy item type contains items with attributes like policy number, name, claim, and so forth.

In Content Manager, there are two kinds of item types: non-resource item types and resource item types. A non-resource item type represents entities that are not stored in a resource manager. A resource item type represents objects stored in a resource manager, like files in a file system, video clips in a video server, LOBs (large objects) in database tables, and so forth. Content Manager provides a base set of resource item types: LOB, text, image, stream, and video objects.

Root and child components

An item type is composed of components: a root component and any number of child component, which are optional.

A *root component* is the first or only level of a hierarchical item type. An item type consists of both system- and user-defined attributes. Internally, the most basic item type contains only one component.

A *child component* is an optional second or lower level of a hierarchical item type. Each child component is directly associated with the level above it. Figure 10 on page 72 shows the diagram of the Content Manager meta-model. It shows root and child components and their relationships in forming an item hierarchy. The diagram also shows links, references, resource items, and resource objects.

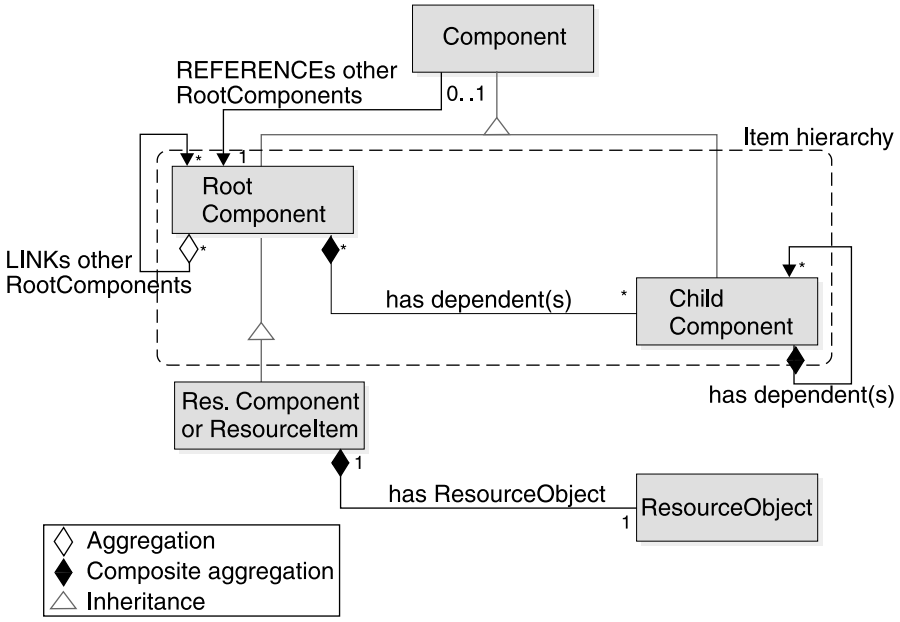

Figure 10. The Content Manager meta-model.

Figure 11 shows an example of a data model for an insurance application with policy as the root component having claim, police report, and damage estimate as child components.

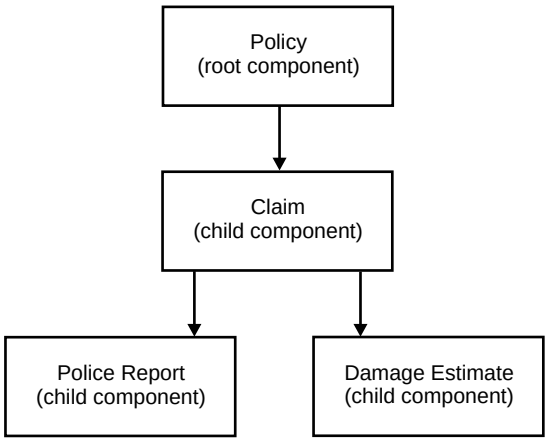

Figure 11. Components hierarchy

Objects

An *object* is any digital content that a user can store, retrieve, and manipulate as a single unit (for example, JPEG images, MP3 audio, AVI video, and a text block from a book). An object is always stored in a resource manager. Access to an object is controlled by the library server.

Links and references

A *link* is a one-to-many association between items. As shown in Figure 10, links can only be used to associate items that are root components. You can define any number of links.

You can use a link to represent the foldering or container-containee relationship. If you choose to implement foldering using links, remember that the container does not *own* the containee, which means that the items in the container are not deleted when the container is deleted.

A *reference* is a one-to-one association between components of an item to other items. A reference is represented as a reference attribute in a component. A component can have any number of reference attributes that refer to other root components. A reference usually does not indicate ownership, but you can implement an ownership relationship if necessary.

When you add a reference to a component type, items of that component type can refer to another item. In terms of the DDO, the DDO has an attribute that is identified by the name of the reference. The attribute's value can be set to another DDO. The attribute value for the DDO is the DDO that is referred to by the reference.

References can be defined in both root and child component types referring to another root component type. See Figure 10 on page 72. References also refer to a specific version of an item, whereas links refer to all versions. References are also modeled in the data definition and links are not.

Documents

A document is an object comprised of other objects containing content (scanned image, office document, video clip). Additionally, a document can be associated with objects like an annotation, a note, or other affiliated information.

You can use child components to associate objects with a document. You can use more than one object to construct a document. For example, each page in a document can be a separate object. An object number is stored in the child component for providing the order of the objects to an application. You can also use a child component to point to an object (a reference attribute) that contains additional information about the object such as size, the resource manager ID on which the object is stored, the collection name for the object, and so forth.

Folders

A *folder* is an item that contains any kind of item and can also contain other folders. In Content Manager Version 8 Release 1, the concept of folders is implemented by using link relationships between items. The document-folder relationship is conceptually a container-containee relationship in the Content Manager system. Items can contain other items to form a containment hierarchy called a folder hierarchy. For example, a policy item belongs to the policy item type, and potentially has many claims, making policy a folder that holds other items such as a photo, a social security number, and so forth. Folders are very flexible because any item can be a folder and can contain any number of other items.

Versioning

Versioning is the ability to store and maintain multiple versions of an item, including versions of the item's child components. You specify versioning when you define an item type. Every instance (item) of an item type that is enabled for versioning is also enabled for versioning. When an item is enabled for versioning, a new version of the item is created every time the item is updated and the new version of the item is stored into the datastore.

Versioning is handled by the Content Manager library server. Different versions of version-enabled objects are stored in the resource manager as separate objects. Important characteristics of versioning support include the following:

- Versioning involves a root component and its entire hierarchy.
- Item types can have one of three possible versioning policies: versioned-always, versioned- never (the default) and application-controlled versioning.
- All of the versions of an item in the CM system are searchable and retrievable.
- Any version of an item can be updated and deleted.
- For item types with application-controlled versioning, when the item is updated, the user has the option of applying the updates to the existing version or creating a new version based on the updates.
- Each version of an item has its own persistent identifier (PID). The PID has several parts of which two are relevant in the current context. The first relevant part is the ItemID which is the same across all different versions of the item. The other is the version number. Each version of the item has a different version number that can be retrieved and set as a string. Below is a sample that demonstrates how to work with version numbers.

```
DKPidICM pid = (DKPidICM)ddo.getPidObject();
String version = pid.getVersionNumber();
....
pid.setVersionNumber(version);
```

- If an update to an exceeds the maximum number of allowed, the oldest saved version is dropped and a new version is created by the system.
- If a version-enabled item is re-indexed, all previous versions of the item are automatically deleted.
- Child components of an item inherit the version of their parent component.
- The version of a child component type cannot be changed, since it follows the versioning of its parent type.
- For an item with parts, if the item type has a version policy of versioned-never, the various parts in the item type also have a version policy of versioned-never.
- Part-level versioning rules can be obtained from the item type relation object that represents the types.

Access control

The Content Manager access control model is comprised of the following fundamental elements:

- Privileges and privilege sets
- Controlled entities
- Users and user groups
- Access control lists

The various access control elements work as follows. Each Content Manager user is granted a set of user privileges. These privileges define the a maximum operation a user can perform. A user's effective access rights will never exceed the user's defined privileges.

The access control model of Content Manager is applied to the controlled entity. A *controlled entity* is a unit of protected user data. In Content Manager, the controlled entity can be at the level of item, item-type, or at the level of the entire library. For example, you can bind an ACL to an item type to enforce access control at the item

type level. Operations on controlled entities is regulated by one or more control rules, called access control lists (ACLs). Every controlled entity in Content Manager system must be bound to an ACL.

When a user initiates an operation on an item, the system checks the user's privilege and the ACL bound to the item to determine if the user has the right to do such an operation on the item. Logically, the right to access an item also requires the right to access the item type, where the item is defined. Figure 12 shows an example of how the system determines user's access rights to an item based on privileges and ACLs.

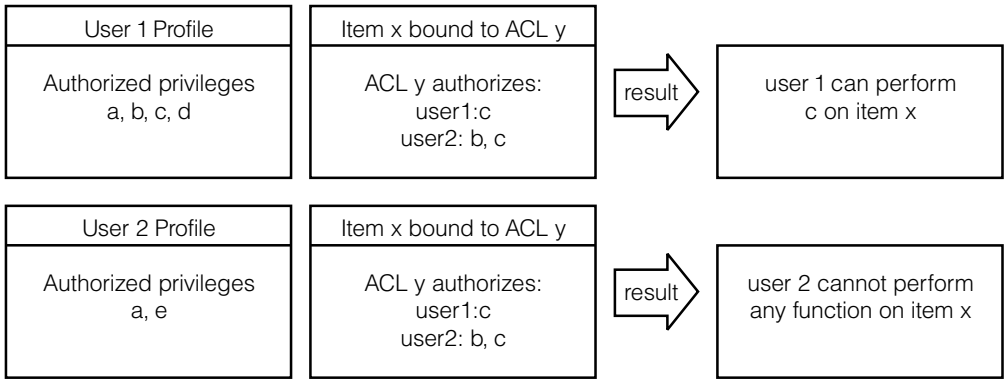

Figure 12. Access control diagram

Privileges and privilege sets

Privileges allow a user to perform a specific action on an item in the system, such as create or delete it. Every Content Manager user is granted a set of user privileges. The privileges define the maximum operations a user can perform on information in the Content Manager system. A user's access rights do not exceed the defined user privileges for the user.

Content Manager provides a number of pre-defined privileges that you cannot change, called system-defined privileges. You can also define your own privileges, called user-defined privileges. You enforce user-defined privileges in your application using user exit routines.

Every privilege has a system-generated, unique code called a privilege definition code. The privilege definition codes 0 to 999 are reserved for system-defined privileges. You can use codes of 1000 and above for user-defined privileges.

The system-defined privileges are classified into two categories: system administration privileges, and data access privileges. You can use the system administration privileges to model user data and administer and maintain the Content Manager system. You need system administration privileges to complete tasks such as configuring the system, managing the library server configuration, and managing item types. You can use the data access privileges to access and modify the system data, like items and item types.

A group of privileges assigned to a user is a *privilege set*. For example, one privilege set can contain the privileges create, update, and delete. Privilege sets allow for easier system administration. You must group privileges into a set before you can use them. There is no limitation on the number of privileges a set can contain.

The Content Manager pre-defined privilege sets include:

AllPrivSet; PrivSetCode: 1
A user with this privilege set can perform all functions on all Content Manager entities. The privileges contained in this privilege set include: All system-defined and user-defined privileges.

NoPrivSet; PrivSetCode: 2
Users with this privilege set cannot perform any functions on any Content Manager entities. The privileges contained in this privilege set include: None.

SystemAdminPrivSet; PrivSetCode: 3
Users with this privilege set can perform all Content Manager system administration and data modeling functions. The privileges contained in this privilege set include:

ItemAdminPrivSet; PrivSetCode: 4
Users with this privilege set can perform all Content Manager data modeling and item access functions. he privileges contained in this privilege set include:

Table 5. Privilege Codes

Privilege name	Code
ICMLogon	1
SystemAdmin	40
SystemDefineItemType	45
ItemSQLSelect	121
ItemTypeQuery	122
ItemQuery	123
ItemAdd	124
ItemSetUserAttr	125
ItemSetSysAttr	126

Users and user groups

Most likely, you have a group of users that require the same type of access to the system. For example, all of the underwriters in an insurance company require search, retrieve, and update privileges to the claims item type. You can group the underwriters and any other users with common access needs into a user group. You cannot, however, put one user group into another user group.

A user group is solely a convenience grouping of individual users who perform similar tasks. A user group consists of zero or more users. You do not assign a user group a privilege set. Each user in a user group has a privilege set. A user group makes it easier to create access control lists for objects in your system. A user group cannot belong to other groups.

Access control lists

When a user creates an item in the Content Manager system, that user must define the access that other users will have to that item, and what operations they can perform on that item. The list of users that have access to the item and the operations that they can perform on the item is called an *access control list* (ACL). An ACL can contain one or more individual user IDs or user groups and their associated privileges. You can associate items, item types, and worklists with an ACL. Privilege sets define an individual's maximum ability to use the system, an

ACL restricts that individual's access to an item. For example, if its ACL allows the photograph item to be deleted but John doesn't have the delete privilege in his privilege set, then John cannot delete the photograph.

A controlled entity is bound to a specific ACL through the ACL code. When associated with controlled entities, ACLs define the authorization of the bound entities and do not circumvent the user privileges. An ACL is enforced and user privileges are checked.

The users specified in access control rules can be individual users, user groups, or public. The interpretation is determined by the UserKind field of a rule. The types of rules, for illustration purposes, can be given the names ACL Rule for User, ACL Rule for Group, and ACL Rule for Public respectively. By specifying public, the ACL Rule for Public authorizes all the users to perform operations specified in the ACL Privileges on the bound entity, provided the users pass their User Privileges check. The ACL privileges on the bound entity to Public can be configured in the System level. The capability of opening a bound entity to Public can be configured system-wide. The configuration parameter is named PubAccessEnabled (defined in table ICMSTSysControl). When disabled, all the ACL Rules for Public are ignored during the access control process.

Within the same ACL, a user can be specified in more than one type of rule. The precedence of the three types, from highest to lowest, is ACL Rule for Public, ACL Rule for User, and ACL Rule for Group. When applying ACL checks, if any higher-precedence rule type passes, the authorization is resolved and the process stops. If the check for ACL Rule for Public failed, the checking process will continue on the lower-precedence rule types.

If the check for ACL Rule for User failed, however, the checking stops; i.e., the ACL Rule for Group is not checked (there is no need to continue the check on the Group type because if a user does an individual user check (which means the user has a rule in the Access List table), the user will be excluded from the group type access based on the access control algorithm. The access control check for individual User type and Group type is not a sequential process. It is an either-or situation, even though there is no harm in doing a sequential check).

If the user has failed to pass an individual user type check (or the user does not have a rule in the Access List table), the checking process will continue to the group type. If the user belongs to one of the groups and the check of the privilege passes, the authorization is resolved and the process stops. Otherwise, access is denied and the process also stops. When a user is specified in more than one ACL Rule for a Group, the user is authorized by the union of all those rules' ACL Privileges. A user is never specified in more than one ACL Rule for User.

The CM system provides the following pre-configured ACLs: SuperUserACL, NoAccessACL and PublicReadACL.

SuperUserACL
: This ACL consists of a single rule that authorizes the CM pre-configured user ICMADMIN to perform all CM functions (AllPrivSet) on the bound entities.

NoAccessACL
: This ACL consists of a single rule that specifies, for all CM users (public), no actions (NoPrivSet) is allowed.

PublicReadACL

This ACL consists of a single rule that specifies, for all CM users (ICMPUBLC) , the read capability (ItemReadPrivSet) is allowed. This is the default value assigned to a user's DfltACLCode.

The data model

This section describes two possible implementations of the data model. The data model that you implement is dictated by the needs of your business. At this point you should have already analyzed your business requirements and determined which model best suits your needs. For more information about planning your data model see *Planning Your Content Manager System*.

One implementation of the data model is an item type with attributes, represented by a single component in the system. For example, an insurance company might implement the basic data model by defining the item type policy that contains the attributes policynumber, insuredfirstname, insuredlastname, and so forth.

Another implementation of the data model is an item type with a hierarchy of child components. For example, XYZ Insurance has implemented their business model by defining a policy item type that contains the attributes policynumber, insuredfirstname, insuredlastname, and claim. The policy item type has several claims associated with it and the claims have several reports associated with them. This implementation is represented as a hierarchy of components in the Content Manager system.

To see how to create a complete data model, see the insurance data model samples in the `samples` directory.

Planning a Content Manager application

This section helps you identify requirements for creating a Content Manager application and provides information about how Content Manager operates. The following topics are covered in this section:

- Determining the features of your application
- Handling errors

Determining the features of your application

The approach you take to develop your application varies based on the needs of your organization. To produce an effective application, all interested parties in your organization should contribute to the planning and design of the application. For additional help with planning, see *Planning Your Content Manager System*.

Before you can create your application, should be able to answer all or most of the following questions:

- What types of documents does your organization use?
- What type of content is in your existing documents?
- How do you process documents?
- Can you automate your document process?
- How do you receive, display, store, and distribute documents?
- How often do you retrieve documents after they are stored?
- What is the volume of documents that your organization manages?
- What types of storage media do you want to use to store your large objects?

- Are there other applications your organization uses?

Use the answers to the questions above to help you determine which features to include in your application.

Handling errors

The most important exception to catch is the DKException class. You must also handle the Java, java.lang.Exception when working in Java.

Do not use exceptions for program logic, and do not rely on catching exceptions to detect if something exists in the datastore or for any reason other than for truly exceptional cases. Using exceptions in program logic decreases performance and can render tracing and log information useless for debugging and support.

Carefully review all of the exception information. There are numerous sub-classes of DKException and depending on the program, it may be best to handle each or some exceptions individually. Table 6 contains DKException information.

Table 6. DKException information

DKException	Description
Name	Exception Class Name. Contains sub-class name.
Message	A specific message explains the error. The message can contain a lot of information, sometimes encapsulating important variable states at the time the error was detected.
Message ID	A unique Message ID identifies this error type and matches it to a core message used above.
Error State	Might contain additional error information about the state of the OO API or library server error. If the library server detects an error, the following four pieces of information are packaged here: Return code Reason code Ext / SQL return code Ext / SQL reason code
Error Code	Might contain the library server return code.
Stack Trace	Important information indicating the failure point in the user program and exactly where the error was last detected or handled by the OOAPI.

The SConnectDisconnectICM.java sample in the `samples` directory demonstrates how to catch and print errors.

For information about logging and tracing, see *Messages and Codes*.

Working with the Content Manager samples

Content Manager provides a comprehensive set of code samples to help you complete key Content Manager tasks. The samples are located in the `cmbroot\samples\java\icm` directory. Also included in the samples directory is the samples README, which contains information to help you make the best use of the samples.

Every sample name has the prefix S, which stands for samples. For example, SItemUpdateICM. Every sample is thoroughly documented and provides in-depth

conceptual information and an explanation of each task step. Additional information contained in each sample includes:

- Detailed header information explaining the concepts shown in the sample
- A description of the sample file including prerequisite information and command line usage
- Fully commented code that you can easily cut, customize, and use in your applications

The Getting Started section in the samples README helps you to quickly learn how to complete the following general tasks:

- Data modeling
- Connecting to a server and handling errors
- Defining attributes and attribute groups
- Working with reference attributes
- Defining your data model
- Working with items
- Working with resource items
- Working with folders
- Working with links
- Defining the resource manager
- Defining an SMS collection
- Searching for items

The insurance scenario sample

Content Manager provides a code sample of one possible implementation in the real-world using an insurance company. The information used to create the insurance company sample is fabricated and created only to help explain key Content Manager features. The insurance scenario samples are also located in the samples directory, `cmbroot\samples\java\icm`. For a complete list of the samples that make up the insurance scenario, see the samples README: *README_SAMPLES_JAVA_ICM.txt*.

Creating a Content Manager application

The APIs that implement Content Manager Version 8 Release 1 functionality are grouped into what is called the ICM connector. The ICM connector APIs have an ICM suffix, as in the example `DKDatastoreICM`.

This information in this section includes:

- Understanding the software components
- Representing items using DDOs
- Connecting to the Content Manager system
- Working with items

Understanding the software components

For conceptual purposes, you can categorize the OO APIs into the following groups of services:

- Data and document modeling
- Search and retrieve
- Data import and delivery

- System management
- Document routing

The data and document modeling module contains the APIs that enable you to map your business data model to the underlying Content Manager hierarchical data model. For example, an insurance company's data model includes *policies*, which in the Content Manager data model are essentially items. The data and document modeling module APIs provide interfaces to define items that represent *policies*.

The search and retrieve module processes requests about managed items like documents and folders. The search module APIs enable you to perform combined text and parametric searches for items contained in the Content Manager system. The search results are returned to the application in the form of search result sets.

The data import and delivery module provides the APIs that enable you to import data into your system and deliver that data through various media, like a network or the Web.

The system management module provides you with the interfaces to configure and maintain an efficient, secure Content Manager system. For example, you can incorporate the system management APIs into your application to allow you to adjust the system control settings, manage users, assign users privileges, allow access to the system, and so forth.

The document routing module APIs help you to route business objects, like documents, through a process, as defined by the needs of your business.

Representing items using DDOs

Before you can create an application, you must understand the DDO/XDO protocol concepts explained in the "Understanding dynamic data object concepts" on page 12. The information in this section is specific to Content Manager Version 8 Release 1.

A DDO is essentially a container of attributes. An attribute has a name, value, and several properties. One of the most important properties of attributes is the attribute type. A DDO has a persistent ID (PID) to indicate the location where the object resides in persistent storage. A DDO has some methods to populate itself, and corresponding methods to retrieve an item's information. The DDO methods include add, retrieve, update and delete. You use these methods to move an item's data in and out of persistent storage, like Content Manager.

In memory, Content Manager items are represented as DDOs. Item attributes are represented as DDO attributes with a name, type, and a value. Links and references are represented as special types of attributes. The difference between a link attribute and a reference attribute, however, is that a reference attribute refers to another (single) DDO or XDO, and a link attribute refers to a collection (multiple) of DDOs or XDOs. XDOs are used to represent large objects (LOBs).

A reference to an item, either to an XDO or another DDO, has a name with the type property set to object reference, and value set to refer to the instance of the referenced object. Child components and links are also represented as DDO attributes with the type property set to a collection of data objects, and value set to a collection of DDOs. In the case of a child component, the attribute name is the name of the child component. The value is the collection of child components

belonging to the root component. If the root item is deleted, all of the child components of the root item are also deleted.

Connecting to the Content Manager system

One of the first things that you need to do when you build a Content Manager application is connect to the server. This section helps you with the various tasks involved in connecting to, and disconnecting from, a Content Manager server.

To access the Content Manager server, your application needs to create a datastore, which acts as a common server. To create a datastore:

1. Create a datastore object.

   ```
   DKDatastoreICM dsICM = new DKDatastoreICM();
   ```

2. Call the connect operation on the datastore. `databaseNameStr` is the name of the database you want to connect to.

   ```
   dsICM.connect(databaseNameStr,usridStr,pwStr,"");
   ```

Depending on your system configuration, you might have several library servers and resource managers that you can connect to. To see a list of the names of the library servers that you can connect to, use `DKDatastoreICM` and call the `listDataSourceNames()` method, and then the `listDataSources()` method. The `listDataSources()` method lists the library servers that are currently available to connect to.

After you connect to a library server, use the `DKRMConfiguration` and call the `listResourceMgrs()`method, to get the list of resource managers associated with that library server.

To disconnect from the system, call the `disconnect` operation in the datastore.

Changing a password

You can allow users to change their password each time they begin a new library server session. To implement the change password option, use `dkDatastore` and call the `changePassword()` method.

Working with items

This section describes the processes involved in creating, updating, and deleting items.

For additional information about working with items see the SItemCreationICM sample located in the `samples` directory, `cmbroot\samples\java\icm`.

Creating an item type

Before you can create any items, you must create item types. You must define an item type for every item you create. For example, a `claim` item is the child component of the item type `policy`.

When you create an item type, you can define a classification for the item type. An *item type classification* is a categorization within an item type that further identifies the items of that item type. All items of the same item type have the same item type classification. Content Manager supplies the following item type classifications: ICM document model, LOB, text, image, stream, and video stream. You can also define your own item type classifications. If you do not specify an item type classification when you create an item type, the item type classification defaults to item (DDO). The pre-defined item type classifications and the corresponding ID constant are listed in Table 7 on page 83.

Table 7. Item type classifications

Classification	ID Constant	Number	Description
Item	DK_ICM_ITEMTYPE_CLASS_ITEM	0	A standard item (DDO).
Resource	DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM	1	A resource item that describes and contains data that is stored in the resource manager. Video, images, documents, and other data archived in the resource manager are examples of resource items.
Document model	DK_ICM_ITEMTYPE_CLASS_DOC_MODEL	2	An item that models documents using parts. A document is composed of any number of parts, which are contained in the attribute DKConstant.DK_CM_DKPARTS.
Part	DK_ICM_ITEMTYPE_CLASS_DOC_PART	3	Items (parts) in the document model classification.

Note: Constants are located in DKConstantICM

To create an item type (see code example below):

1. Create an item type and pass it a reference to the datastore.
2. Give the item type a name.
3. Add attributes to the item type, and set any desired qualifiers, like `nullable`, `textsearchable`, and `unique`.
4. Add the item type to the persistent datastore.

Example:

1. This example creates an item type definition and names it. The item type name must be less than 15 characters in length.

```
DKItemTypeDefICM bookItemType = new DKItemTypeDefICM(dsICM);
bookItemType.setName("book");
bookItemType.setDescription("This is an example item type name.");
```

2. In the code example below, a text-searchable attribute called `book title` is added. The attribute is defined to require a unique name and also a value. The value does not have to be unique.

```
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_title");
attr.setTextSearchable(true);
attr.setUnique(true);
attr.setNullable(false);
bookItemType.add(attr);
```

3. Here, a `book_num_pages` attribute is added to the book item type with the following characteristics: text searchable, unique, and a value is not required.

```
DKAttrDefICM attr = (DKAttrDefICM)_dsDefICM.retrieveAttr("book_num_pages");
attr.setTextSearchable(false);
attr.setUnique(false);
attr.setNullable(false);
bookItemType.addAttr(attr);
bookItemType.add();
```

Listing item types

To get a list of available, defined item types:

1. Connect to a `DKDatastoreICM` datastore.

2. Get a reference to the datastore definition.
3. Call the `listEntityNames` method on the datastore definition object to get a string array of the names of the item types.
4. Use a loop to list all of the names.

Example:

```
String itemTypeName[] = dsICM.listEntityNames();
DKSequentialCollection itemTypeColl = (DKSequentialCollection) dsICM.listEntities();
dkIterator iter = itemTypeColl.createIterator();
while(iter.more()){
   DKItemTypeDefICM itemType = (DKItemTypeDefICM) iter.next();
   System.out.println(" Item type name : " + itemType.getName());
}
```

See the SitemTypeCreationICM sample, in `cmbroot\samples\java\icm`, for more information and examples.

Creating attributes

To create attributes:

1. Create an attribute definition object.
2. Describe the object that you create by setting its name, description, type, size, and so forth. Examples of types of attributes that you can create include:
   - DKConstant.DK_CM_TIMESTAMP
   - DKConstant.DK_CM_DATE
   - DKConstant.DK_CM_CHAR
   - DKConstant.DK_CM_VARCHAR
   - DKConstant.DK_CM_SHORT
   - DKConstant.DK_CM_INTEGER
   - DKConstant.DK_CM_LONG
   - DKConstant.DK_CM_FLOAT
   - DKConstant.DK_CM_DECIMAL
   - DKConstant.DK_CM_TIME
   - DKConstant.DK_CM_BLOB
   - DKConstant.DK_CM_CLOB
3. Add the new definition to the persistent datastore.

Examples:

1. This example defines an attribute for the title of a book.

```
attr = new DKAttrDefICM(dsICM);
attr.setName("book_title");
attr.setDescription("The title of the book.");
attr.setType(DKConstant.DK_CM_VARCHAR);
attr.setSize(100);
attr.add();
```

2. This example defines an attribute for the number of pages in a book.

```
attr = new DKAttrDefICM(dsICM);
attr.setName("book_num_pages");
attr.setDescription("The number of pages in the book.");
attr.setType(DKConstant.DK_CM_INTEGER;
attr.setMin((short) 0);
attr.setMax((short) 100000);
attr.add();
```

Creating, updating, and deleting attribute groups

To create an attribute group use the DKAttrGroupDefICM class, for details see the *online API reference*. When you create an attribute group you must provide a list of existing primary attributes.

To update the name and description of an attribute group, call the update() method in DKAttrGroupDefICM and provide an array of new attribute IDs.

To delete an attribute group you also work with the DKAttrGroupDefICM class. When you delete the attribute group the primary attributes that used to make up the attribute group remain in library server. The following exceptions apply when you delete an attribute group:

- An attribute group cannot be deleted if it is associated with a component type and is persistent.
- An attribute can not be removed from an attribute group if the attribute group is associated with a component type and is persistent.
- You cannot add an attribute to an attribute group if the attribute group is associated with a component type and is persistent.

Creating an item

When you create an item, you must assign it an item type property. The types that you can assign an item are document, folder, or item. You must specify the item type property as the second parameter of the datastore's `createDDO` function. The value for the property type is stored in the DDO's property named, DK_CM_PROPERTY_ITEM_TYPE. Do not confuse this item type property with the overall item type definition that describes the structure of the item. Table 8 shows a list of available item property types.

Table 8. Item type property definitions

Property type	Constant	Definition
Document	DK_CM_DOCUMENT	Item represents a document or stored data. The information contained in this item might form a document. This item can be considered a common document since it does not rigidly mean an implementation of a specific document model.
Folder	DK_CM_FOLDER	Item represents an object containing or referencing contents or objects. This item can be considered a common folder since it does not rigidly mean an implementation of a specific document model.
Item (default)	DK_CM_ITEM	Generic item. This item does not fit system defined or user defined semantic types.

Items are created as DKDDOs. Always use the DKDatastoreICM's createDDO() methods to create DKDDOs because the system uses the DKDatastoreICM's createDDO methods to automatically setup important information in the DKDDO structure.

When you create an item, you define the components that make up the item. For example, if you choose to create a hierarchical item, you must create child components.

1. Using the datastore's createDDO and createChildDDO methods, create an item DDO and set all of its attributes and other required information. This example uses the logged on, connected DKDatastoreICM object named dsICM.
2. Create a root component.

```
DKDDO myDocumentDDO = dsICM.createDDO("EmployeeDoc",
   DKConstantICM.DK_CM_DOCUMENT);
```

   OR

```
DKDDO myFolderDDO   = dsICM.createDDO("DeptFolder",
   DKConstantICM.DK_CM_FOLDER);
```

3. Create a child component under the root component ″EmployeeDoc″, for example, ″Dependent″.

```
DKDDO myDDO =dsICM.createChildDDO("EmployeeDoc","Dependent");
```

4. Use the setData method to populate the DDO with the appropriate values.

```
myDDO.setData(.....);
```

5. Save the item into the persistent store.

```
myDocumentDDO.add();
```

In the preceding example, the last step created a document in the datastore. When a document DDO is added to a datastore, all of its attributes are added, including all of the parts inside the DKParts collection.

When you define an item, you must also define its semantic type. *Semantic type* defines the usage or rules for an item. Content Manager comes with some pre-defined semantic types, but you can also define your own semantic types.

You can specify the semantic type as the second parameter of the datastore's createDDO function. The semantic type value is stored in the DDO's property DKConstantICM.DK_ICM_PROPERTY_SEMANTIC_TYPE, which can contain the same value as the item property type. The ICM connector supports folder and document semantic types. You can also create your own semantic types. Table 9 lists the available semantic types.

Table 9. Pre-defined semantic types

Semantic type	Constant	Definition
Document	DK_ICM_SEMANTIC_TYPE _Document	Indicates that this item is a document
Folder	DK_ICM_SEMANTIC_TYPE _Folder	Indicates that this item is a folder
Annotation	DK_ICM_SEMANTIC_TYPE _Annotation	Indicates that this item or part is an annotation to the base part
Container	DK_ICM_SEMANTIC_TYPE _Container	Indicates that this item is a container, which can contains other items. The container-containee relationship is represented using links.
History	DK_ICM_SEMANTIC_TYPE _History	Indicates that this item or part is a history of the base part.
Note	DK_ICM_SEMANTIC_TYPE _Note	Indicates that this item or part is a note to the base part.
Base	DK_ICM_SEMANTIC_TYPE _Base	Indicates that this item or part is the base part that may have an annotation, note, or history associated with it.

Table 9. Pre-defined semantic types (continued)

Semantic type	Constant	Definition
User defined	User defined	A user defined semantic type interpreted by the application.

Notes:

- The constants are defined in DKConstantICM.java.
- In Content Manager Version 7, semantic type is called affiliated type.

If you create an item in an item type that has been defined as a resource item type, the correct XDO is returned. For more information about resources, please see the SItemTypeCreationICM.java sample in the samples directory.

Setting and retrieving item attribute values

Attribute values are stored and retrieved as java.lang.Objects. To set or retrieve attribute values, use the DDO's setData method. Set the value using an object with the type corresponds to the attribute type. **Example code:**

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,nameOfAttrStr),valueObj);
```

The above statement sets the attribute value to the value passed in as a java.lang.Object valueObj. The nameOfAttrStr is the stringname of the attribute.This attribute was defined and specified in the item type when the item type of this DDO was defined. valueObj is the value you must set and it must be of the right type for this attribute.

Modifying item attributes

To modify an item's attributes, use the DDO's setData method and set it to the required value by specifying that value using the java.lang.Object, as shown in the example below. In the example below, nameOfAttrStr is the string name of the attribute and valueObj is the value.

```
ddo.setData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR, nameOfAttrStr),valueObj);
```

Updating an item

To update an item:

1. Retrieve an item or create an item and add it to the datastore.
2. Modify the item, its attributes, link collections, and so forth.
3. Call the DDO's update operation

   ```
   ddo.update();
   ```

If an item is enabled for versioning, you can create a new version of the item instead of updating the current item, as shown in the following example:

```
DKDDO ddo;
ddo.update(DK_CM_VERSION_NEW);
```

To update the latest version of an item, use the format shown in the following example:

```
ddo.update(DK_CM_VERSION_LATEST);
```

You can also update a DDO by calling the updateObject method on DKDatastoreICM with the appropriate options, as shown in the example below:

```
DKDatastoreICM ds;
DKDDO ddo;
int options = DK_CM_VERSION_NEW + DK_CM_CHECKIN;
ds.updateObject(ddo, options);
```

You must check out the item before calling the update method and check the item back in when your are done updating it. See “Checking in and checking out items” on page 89.

Searching for items

To find items that match a set of criterion, complete the following steps:

1. Connect to a datastoreICM object.
2. Process the correct query. Make sure that your query conforms to the query language. For more information see “Understanding the query language” on page 98.
3. Save the query results in a DKResult object.

```
DKResults results = (DKResults)dsICM.evaluate(queryStr,
DKConstantICM.DK_CM_XQPE_QL_TYPE,null);
```

See the SSearchICM sample, in the `cmbroot\samples\java\icm` directory, for additional information and examples.

Retrieving items

To retrieve a document from a DKDatastoreICM that represents a Content Manager content server, you must know the document’s item type name and item ID. You must also associate the DDO with a datastore and establish a connection.

1. After connecting to a datastoreICM object, search for the item using the appropriate item ID. For information about writing queries, see “Querying the Content Manager server” on page 99.
2. Save the query results in a DKResults object. In the code example below, `queryStr` is the search string in the correct query language format.

   ```
   DKResults results = (DKResults)dsICM.evaluate(queryStr,
   DKConstantICM.DK_CM_XQPE_QL_TYPE, null);
   ```
3. Create an iterator on DKResults and use it to get DDOs, which you can now retrieve one by one.

   ```
   ddo.retrieve();
   ```

You must have a PID string that you obtained from a DDO as shown in this example:

```
DKPidICM pid = ddo.getPidObject();
String  pidStr = pid.pidString();
```

Using the PID string, you can perform a retrieve by completing the following steps.

1. Create a DDO using the DKDatastoreICM’s createDDO method. Do not use the DKDDO constructor. In the example below, the object `dsICM` is already connected to a Content Manager datastore. Also, the PID is a string named `pidStr`.

   ```
   DKDDO ddo = dsICM.createDDO(pidStr);
   ```
2. Call the DDO’s retrieve operation.

   ```
   ddo.retrieve();
   ```

If an item is enabled for versioning, you can retrieve a specific version of the item by using the retrieve method of the DKDDO class with the appropriate options. To retrieve the latest version of an item, use the constant DK_CM_VERSION_LATEST as the retrieve option. By default (if no options are specified), ddo.retrieve() retrieves the latest version of an item.

Example:

```
DKDDO ddo = ds.createDDO(...);
   ....
 ddo.retrieve(DK_CM_VERSION_LATEST);
```

You can also retrieve a DDO by calling the retrieveObject method on the DKDatastoreICM object.

Example:

```
DKDatastoreICM ds =new DKDatastoreICM();
//connect,etc ...
DKDDO ddo =ds.createDDO(itemType,option);
//sets the PID as shown above...
ds.retrieveObject(ddo);
```

To retrieve the latest version of an item including its attributes and children, use the following format:

```
int options =DK_CM_CONTENT_ATTRONLY + DK_CM_CONTENT_CHILDREN +
                                      DK_CM_VERSION_LATEST;
ds.retrieveObject(ddo,options);
```

Deleting items

Use the delete method in the DDO to delete an item 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.

Checking in and checking out items

To prevent two users from making changes to the same item at the same time, you can require users to check out an item before updating it. Checking out an item grants exclusive update rights to the user who has the item checked out. When you check out an item you hold a persistent write lock on that item, which is released when you check in the item.

To check items in and out, use the DKDatastoreICM's checkIn and checkOut operations, as shown in example below.

1. Having connected to a DKDatastoreICM object named dsICM, and using a DDO called myDataObject, check out the item.

   ```
   dsICM.checkOut(myDataObject);
   ```

2. Check in the item.

```
dsICM.checkIn(myDataObject);
```

Defining a resource item type

A resource item is an item with additional system defined attributes that define the location, type, size, and so forth, of the object that the item represents. The object is sometimes called a ″resource″ and can be a video file, an image, a word processor document, and so forth. For additional information, see the SItemTypeCreationICM sample (in the samples directory).

The steps below take you through the process of defining a resource item type.

1. Get the datastore definition object from the connected datastore.

   ```
   DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM)dsICM.datastoreDef();
   ```

2. Create a new item type definition.

   ```
   DKItemTypeDefICM  itemType = new DKItemTypeDefICM(dsICM);
   itemType.setName("SampleResource");
   itemType.setDescription("Simple Resource Lob Item Type");
   ```

3. Add an attribute.

```
DKAttrDefICM attr = (DKAttrDefICM)dsDefICM.retrieveAttr("S_varchar");
itemType.addAttr(attr);
// Resource classification indicates that this class will contain data file
itemType.setClassification(DKConstantICM.DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
```

4. Specify XDO class and type of resource for this item type.

```
itemType.setXDOClassName(DKConstantICM.DK_ICM_XDO_LOB_CLASS_NAME);
itemType.setXDOClassID(DKConstantICM.DK_ICM_XDO_LOB_CLASS_ID);
itemType.add();
```

Creating a resource item

Creating resource items is very similar to creating regular items. XDOs extend DDOs, and depending on the type of resource item, the XDO can be extended further. Table 42 on page 411 contains the resource item types and the class hierarchy used to create them. For more information, see the SResourceItemCreationICM sample (in the samples directory).

Table 10. Resource item type class hierarchy

Type	DDO	XDO	Extension
LOB	DKDDO ->	DKLobICM	
Text	DKDDO ->	DKLobICM ->	DKTextICM
Image	DKDDO ->	DKLobICM ->	DKImageICM
Stream	DKDDO ->	DKLobICM ->	DKStreamICM

The steps below take you through the process of creating resource item.

1. Create the resource item. Note that it can be any semantic type and that the DKDatastoreICM::createDDO call is also used to create a resource item in the same way that it is used to create a regular item. The of type resource returned from DKDatastoreICM.createDDO is cast to the correct sub-class (in this case DKLobICM) based on the XDO classification defined during the creation of the item type on which this resource item is based.

```
DKLobICM    lob = (DKLobICM)dsICM.createDDO
           ("SampleResource",DKConstant.DK_CM_DOCUMENT);
```

2. Set the MIME type for the resource. In this case, the resource is a MS Word document. The MIME type describes the type of content that is being stored.

```
lob.setMimeType("application/msword");
```

3. Add the data to the datastore. In this case, a sample Word document. Note that the resource item content is stored in resource manager. Even though it is possible to specify the resource manager on which to store the item, the example below uses the default resource manager and SMS collection for the item type on which this resource is based.

```
lob.add("SResourceItemICM_Document1.doc");
```

Working with versioning properties

This section provides examples to help you work with versioning properties.

Getting the value of the version control scheme being used on an item type

```
short  versionControlPolicy;
DKItemTypeDefICM item = newDKItemTypeDefICM();
....
versionControlPolicy = item.getVersionControl();
```

The following return values are supported on this method:

- DK_ICM_VERSION_CONTROL_ALWAYS - versioned-always

- DK_ICM_VERSION_CONTROL_NEVER - versioning is not supported for this item type (default)
- DK_ICM_VERSION_CONTROL_BY_APPLICATION - application determines when to create or not to create a new version.

Setting the value of the version control scheme being used on an item type

```
DKItemTypeDefICM item = new DKItemTypeDefICM();
short versionControl = DK_ICM_VERSION_CONTROL_ALWAYS;
....
item.setVersionControl(versionControl);
```

Getting the value for the maximum number of versions allowed at any given time for an item type

```
short  versionMax;
DKItemTypeDefICM item = new DKItemTypeDefICM();
....
versionMax = item.getVersionMax();
```

Setting the value for the maximum number of versions allowed at any given time for an item type

In this example, only ten versions of an item that is based on this item type are maintained by the system.

```
short  versionMax=10;
DKItemTypeDefICM item = new DKItemTypeDefICM();
...
item.setVersionMax(versionMax);
```

Working with folders

A folder is a fully supported DDO that represents an item. A folder has the full hierarchical data structure of the item type that it is created in. A folder is a DDO of semantic type folder and has an attribute, DKConstant.DK_CM_DKFOLDER, that contains a DKFolder, regardless of the item type's classification. The DKFolder object, a DKSequentialCollection can contain other DDOs.

The procedures below include code fragments for informational purposes only. Do not copy these code fragments and paste them directly into your application program, because they will not work in their current form. For a complete folders sample that you can run, see the SFolderICM.java sample in the samples directory.

Creating a folder

The DKDatastoreICM.createDDO() method is used to create DDOs at runtime. As shown in the SItemCreationICM sample (in the samples directory), when creating folder items, the item property type or semantic type needs to be specified as DKConstant.DK_CM_FOLDER.

A folder item is created using the DKDatastoreICM.createDDO method with the DK_CM_FOLDER semantic type. You can create a folder by specifying the DKConstant.DK_CM_FOLDER as the second parameter to the createDDO method.

```
DKDDO ddoFolder  = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
```

Use setData method to populate the ddoFolder. Once the folder DDO is created, it is saved to the persistent datastore.

```
ddoFolder.add();
```

Adding items to a folder

All contents that are to be placed in the DKFolder must be persistent in the datastore before you call the add() or update() method on the folder DDO. To make the contents persistent, call the DKDDO.add() method.

To add items to a folder, use the DKFolder's addElement() function. When you have added enough members into the folder, you can call the add or update method to save the folder relationship into the persistent store. This groups any number of folder modifications and into a single call to the library server. When adding items to a folder, do not add multiple copies of the same item to DKFolder.

Follow the steps below to add items to a folder.

1. Create 3 new items. These items will be added to a folder as its contents. Folder contents can be of any semantic type.

```
DKDDO ddoDocument = dsICM.createDDO("S_simple", DKConstant.DK_CM_DOCUMENT);
```

   Note that a folder can contain another folder (sub-folder) as one of its content items.

```
DKDDO ddoFolder2  = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);
DKDDO ddoItem     = dsICM.createDDO("S_simple", DKConstant.DK_CM_ITEM);
```

2. Use the setData method to populate the DDOs. Items that are to be added as folder contents need to be persisted into the datastore.

```
ddoDocument.add();
ddoItem.add();
ddoFolder2.add();
```

3. Retrieve the dkFolder attribute of the previously created and persisted folder DDO.

```
short dataid = ddoFolder.dataId
   (DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKFOLDER);
dkFolder = (DKFolder) ddoFolder.getData(dataid);
```

4. The folder must be checked out of the datastore before it is updated (by adding contents).

```
dsICM.checkOut(ddoFolder);
```

5. Add the previously created items to the folder.

```
dkFolder.addElement(ddoDocument);
dkFolder.addElement(ddoItem);
dkFolder.addElement(ddoFolder2);
```

6. Commit the changes to the folder and explicitly check in the changes.

```
ddoFolder.update();
dsICM.checkIn(ddoFolder);
```

Retrieving a folder's contents

To retrieve folder contents, you must set the retrieval option. By default, the retrieval option is not set. If you do not set the retrieval option, the object will not contain a DKFolder or a DK_CM_DKFOLDER attribute until retrieve is called and the correct options are set.The retrieval options include the following:

- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND
- DKConstant.DK_CM_CONTENT_ITEMTREE

To retrieve a folder item with the DKFolder attribute collection with Outbound Links specified, use the following format:

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);
```

To retrieve a folder item where the item tree includes links and folder contents, use the following format:

```
ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_ITEMTREE);
```

Removing contents from a folder

Items can be removed from a folder in one of two ways. A deferred removal (the actual removal is done when the folder DDO is updated and multiple datastore

calls are combined into one) is done using the removeElementXX method(s). An immediate (and expensive since multiple calls are made to the datastore) removal can be done using the dkFolder.removeMember method. See see the SFolderICM sample (in the samples directory) for further details on working with folders.

Complete the following steps to remove contents from a folder:

1. Check out the folder DDO from which we want to remove an item.

```
dsICM.checkOut(ddoFolder);
```

2. Look for the item to remove. In this case, look for the docItem DDO created earlier. Create an iterator to iterate through the folder's content.

```
dkIterator iter = dkFolder.createIterator();
String itemPidString = ddoItem.getPidObject().pidString();
while(iter.more()) {
   // Move the pointer to next element and return that object.
   // Compare the PID trings of the DDO returned from the iterator
   // with the DDO that is to be removed.
   DKDDO ddo =(DKDDO)iter.next();
   if(ddo.getPidObject().pidString().equals(itemPidString)) {
      // The item to be removed is found.
      // Remove it (current element in the iterator)
      dkFolder.removeElementAt(iter);
   }
}
```

3. Persist the change and check in the folder DDO.

```
ddoFolder.update();
dsICM.checkIn(ddoFolder);
```

Obtaining all folders containing a specific DDO

In order to determine which folders currently contain a specific DDO (since contents can be stored in multiple places at the same time) use the example below. The following steps demonstrate how to find folders that contain the ddoDocument object, previously created. See the SFolderICM sample (in the samples directory) for additional details about working with folders.

1. Retrieve the datastore extension object.

```
DKDatastoreExtICM dsExtICM =(DKDatastoreExtICM)
                dsICM.getExtension(DKConstant.DK_CM_DATASTORE_EXT);
```

2. Call the `getFoldersContainingDDO` method on the extension object to find the folders containing the ddoDocument object. This returns a sorted collection of DDOs where each returned DDO is a folder containing the ddoDocument object.

```
DKSequentialCollection list = dsExtICM.getFoldersContainingDDO(ddoDocument);
```

3. Create an iterator to cycle through the returned collection of folders.

```
iter =list.createIterator();
while(iter.more()) {
      // Move iterator to next element and return that object.
      DKDDO ddo =(DKDDO) iter.next();
      // Print out the item Id of the folder DDO in order to identify
      // the returned folder object.
      System.out.println("-Item ID: " +
         ((DKPidICM)ddo.getPidObject()).getItemId());
   }
```

Defining links between items

You can use links to associate a source item to a target item with an optional description item. You define links in DKLink objects, where you can specify the following types of link elements:

Table 11. Link data structure

DKLink	Definition
LinkTypeName	The type of link.
Source	The source item of a link.
Target	The target item of a link.
LinkItem	The description item. Optional.

Since a link represents a one to many relationship, it is represented in a DDO by a data-item under LINK namespace, of value DKLinkCollection. For correct usage of folders, please refer to the SFolderICM.java sample in the `samples` directory.

A link itself does not belong to either the source or the target. A link just connects a source and target. For example, if source A is linked to target B, A is always the source and B is always the target, regardless of which DDO, A or B, is being referenced.

In memory, both the source and the target DDOs contain copies of the same DKLink object. Since the DKLink object contains a reference to both the source and the target, a DDO containing a link also contains a link that refers to itself as well as to another DDO. For example, if Source A is linked to Target B, both A and B contain the same link, as shown in the example in Table 12.

Table 12. DKLink definition example

DKLink Defined for A to B	DDO A	DDO B
Source is **A**	Source is **A**	Source is **A**
Target is **B**	Target is **B**	Target is **B**

When you add or remove links in the persistent store, you only have to perform the operation on one of the two items, source or target. The link is automatically updated for the other item.

For a complete links sample that you can run, see the SLinksICM.java sample in the `samples` directory.

Inbound and outbound links

When using links to reference a particular DDO, either the source or the target, you can consider the links to be *inbound* or *outbound*. If a DDO that is being referenced by other DDOs, links from itself **to** those DDOs, the links are considered to be outbound links. When a DDO is the source of the link, it is an outbound link. If the DDO is the target of a link, it is an inbound link. Inbound and outbound links allow a DDO to determine what DDOs link to it and break the links if necessary.

In the example in Table 12, the link from DDO A to DDO B is outbound link, while the same link to DDO B is the inbound link.

Link type names

Link relationships have names. Within a DDO, links are grouped into link collections. There are system defined link type names and you can also define your own. You can use any number of user-defined link type names or system-defined link type names. Table 13 on page 95 lists the system defined link type names.

Table 13. System defined link type names

Link type name	Constant
Contains	DKConstant.DK_ICM_LINKTYPENAME_CONTAINS
DKFolder	DKConstant.DK_ICM_LINKTYPENAME_DKFOLDER

The DKFolder link type name is provided and used by the system to manage folders. The DKFolder object is a simplified interface to an outbound link collection to make folders easy to work with. Therefore, you should not use the DKFolder link type name to define or delete links. You should also not use the DKFolder link type name in any way with a dKLink Collection. However, you must specify the DKFolder link type name in order to search folders using links.

See the SLinksICM sample, in the `cmbroot\samples\java\icm` directory, for additional information about links.

Retrieving linked items

When you retrieve links, you have the option of retrieving only inbound, only outbound, or both types. In order to retrieve inbound and outbound, you must set a retrieve option. You can set the following options to retrieve links:

- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND
- DKConstant.DK_CM_CONTENT_LINKS_INBOUND
- DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND + DKConstant.DK_CM_CONTENT_LINKS_INBOUND
- DKConstant.DK_CM_CONTENT_ITEMTREE

Keep in mind that before you make a call to the DDO `add()` or `update()` methods, all the items that are associated with links must already be persistent.

Working with the access control APIs

If you have access to either the Content Manager system administration client or to the Content Manager APIs for writing your own administration program, you can use the access control functions to control access to the information in your Content Manager system. The various access control APIs allow you to control access to the entire system, to a closely related set of operations on the system, or to an individual item.

Some of the tasks that you can complete using the access control APIs include:

- Creating privileges
- Associate a list of actions with information in the system
- Give permission to users to perform actions on the information in the library server

Creating a privilege

A privilege is represented by the class DKPrivilegeICM. The following the steps and code fragments show you how to create a new privilege object, make it persistent, and retrieve a privilege. To create a privilege, follow the steps below:

1. Connect to a datastore.

```
DKDatastoreICM ds = new DKDatastoreICM();
ds.connect("ICMNLSDB","icmadmin",password,"");
dkDatastoreDef dsDef =(dkDatastoreDef)ds.datastoreDef();
DKDatastoreAdminICM dsAdmin =(DKDatastoreAdminICM)dsDef.datastoreAdmin();
```

2. Retrieve a DKAuthorizationMgmtICM object. This class handles authorization management tasks.

```
DKAuthorizationMgmtICM aclMgmt = (DKAuthorizationMgmtICM)
        dsAdmin.authorizationMgmt();
```

3. Create a new privilege object.

```
DKPrivilegeICM priv = new DKPrivilegeICM(ds);
```

4. Assign a name to the privilege object.

```
priv.setName("UserPriv");
```

5. Assign a description to the privilege object.

```
priv.setDescription("This is user-defined privilege");
```

6. Add the new privilege to the authorization manager object.

```
aclMgmt.add(priv);
```

7. Retrieve the newly created privilege from the authorization manager object.

```
dkPrivilege aPriv = aclMgmt.retrievePrivilege("UserPriv");
```

8. Display information about the privilege object.

```
System.out.println("privilege name = " + aPriv.getName());
System.out.println("privilege description = " + aPriv.getDescription());
```

Creating a privilege set

A privilege set is represented by the class DKPrivilegeSetICM. Before you can begin creating privilege sets, you must be connected to a datastore. To create a privilege set, complete the following steps:

1. Create new privileges (or retrieve privileges) to add to the new privilege set.

```
DKPrivilegeICM priv_1 = new DKPrivilegeICM(ds);
priv_1.setName("ItemCheckOut");
DKPrivilegeICM priv_2 = new DKPrivilegeICM(ds);
priv_2.setName("ItemQuery");
DKPrivilegeICM priv_3 = new DKPrivilegeICM(ds);
priv_3.setName("ItemAdd");
```

2. Create a new privilege set.

```
DKPrivilegeSetICM privSet1 = new DKPrivilegeSetICM(ds);
```

3. Assign a name to the privilege set.

```
privSet1.setName("UserPrivSet");
```

4. Assign a description to the privilege set.

```
privSet1.setDescription("This is a user-defined priv set");
```

5. Add the privileges to the privilege set.

```
privSet1.addPrivilege(priv_1);
privSet1.addPrivilege(priv_2);
privSet1.addPrivilege(priv_3);
```

6. Add the newly created privilege set to the authorization manager.

```
AclMgmt.add(privSet1);
```

7. Display information about the newly created privilege set.

```
DKPrivilegeSetICM aPrivSet = (DKPrivilegeSetICM)
        aclMgmt.retrievePrivilegeSet("UserPrivSet");
System.out.println("privilege set name = " + aPrivSet.getName());
System.out.println("privilege set description = " + aPrivSet.getDescription());
dkCollection coll = aPrivSet.listPrivileges();
dkIterator iter = coll.createIterator();
while (iter.more()) {
   DKPrivilegeICM _priv = (DKPrivilegeICM) iter.next();
   System.out.println("  privilege name = " + _priv.getName());
}
```

Defining an ACL

You can create ACLs using privileges and privilege sets. You can then bind the ACLs to items or item types. The following code fragment shows the steps to defining an ACL.

There are two ACL related classes in the class library. One is the DKACLData and the other is DKAccessControlListICM. In the code fragment below, it is assumed that you have established a connection and created two privilege sets, privSet_1 and privSet_2.

```
//Define a new DKACLData object.
//A DKACLData class is used to hold ACL related data.
DKACLData aclData1 = new DKACLData();
//set the user group name for the ACL
aclData1.setUserGroupName("ICMADMIN");
//set ACL patron type.
aclData1.setPatronType(DK_CM_USER_KIND_USER);
//Set the privilege set associated with this ACL
aclData1.setPrivilegeSet(privSet_1);
//Create another DKACLData object.
DKACLData aclData2 = new DKACLData();
aclData2.setUserGroupName("ICMPUBLC");
aclData2.setPatronType(DK_CM_USER_KIND_GROUP);
aclData2.setPrivilegeSet(privSet_2);
//Create a new ACL. A DKAccessControlListICM represents a CM v8.1. ACL
DKAccessControlListICM acl1 = new DKAccessControlListICM(ds);
//Assign a new to the newly-created ACL
acl1.setName("UserACL");
//Assign a description to the newly-created ACL
acl1.setDescription("This is a user-defined ACL");
//Add the previously created ACL data objects to the ACL
acl1.addACLData(aclData1);
acl1.addACLData(aclData2);
//Add the newly-created ACL to the authorization manager
aclMgmt.add(acl1);
//Retrieve and display information about the newly created ACL
DKAccessControlListICM acl_1 = (DKAccessControlListICM)
aclMgmt.retrieveAccessControlList("UserACL");
System.out.println("ACL name  = " + acl_1.getName());
System.out.println("    desc  = " + acl_1.getDescription());
dkCollection coll = acl_1.listACLData();
dkIterator iter = coll.createIterator();
while (iter.more()) {
   DKACLData aclData = (DKACLData) iter.next();
   DKPrivilegeSetICM _privSet = (DKPrivilegeSetICM) aclData.getPrivilegeSet();
   System.out.println("  PrivSet name  = " + _privSet.getName());
   System.out.println("  PrivSet desc  = " + _privSet.getDescription());
   String usrGrpName = aclData.getUserGroupName();
   System.out.println("    UserGroupName = " + usrGrpName);
   short patronType = aclData.getPatronType();
   System.out.println("    Patron type   = " + patronType);
}
```

The complete sample program is in the samples directory.

Assigning an ACL to an item type

Once an ACL is created, you can associate it to a specific item type. Following are the steps you follow to assign an ACL to an item type:

```
//create a new item type
DKItemTypeDefICM  it = new DKItemTypeDefICM(dsICM);
//Assign a name to this item type
it.setName("TextResource1");
//Assign a description to this item type
```

```
it.setDescription("CMv8.1 Text Resource Item Type.");
//Assign an ACL code to this item type
it.setItemTypeACLCode( (int) 1 );
....
it.add();  // make the item type persistent
...
//Retrieve the item type's ACL code
int itemTypeACLCode = it.getItemTypeACLCode();
// By setting the item type's ACL flag to TRUE(1) we confirm that ACL binding is
// at the item type level. By setting the item type'sACL flag was set to FALSE(0),
// we would be saying that the ACL binding is not at the item type level
it.setItemLevelACLFlag(1); // - true
//Determine whether the ACL binding is at the item type level or not
int itemTypeACLFlag = it.getItemLevelACLFlag();
```

The complete sample program is available in the samples directory.

Assigning an ACL to an item

To enable item level access control, you must bind an ACL to the item. To do this, you must create the item, using the add() method on the DDO, as shown in the code fragment below. In the code fragment below, it is assumed that you already have a connection to the datastore and have created the ACL.

```
//Create a new DDO
DKDDO ddoItem =dsICM.createDDO("myItemType",DKConstantICM.DK_CM_ITEM);
//create a new ACL (access contrl list)
DKAccessControlListICM acl1 =new DKAccessControlListICM(ds);
//Assign a name to the ACL
acl1.setName("MyACL");
//Add a new property to the DDO.This property will be called DK_ICM_PROPERTY_ACL
int propId =ddoItem.addProperty(DK_ICM_PROPERTY_ACL);
//Set the previously created (or retrieved)ACL as the value of this property
ddoItem.setProperty(propId,"MyACL");
//Persist the DDO into the data store
ddoItem.add();
```

The complete program is in the samples directory.

Understanding the query language

A powerful XML-based query language is now available in Content Manager 8.1. When an application searches for items stored in the Content Manager system, the underlying engine performs the processing of the search based on a query. To efficiently traverse Content Manager's hierarchical data model, you use the Content Manager query language. The query language provides the following benefits:

- Supports the full data model
- Enables searches within component type view hierarchies, across linked items, and references
- Combines parametric and text search
- Provides SORTBY capabilities
- Enforces Content Manager access control
- Conforms to XQuery Path Expressions (XQPE), a subset of W3C XML Query working draft
- Executes high performance searches

The Content Manager query language is an XML-based query language that, unlike proprietary languages, conforms to XQuery Path Expressions (XQPE), a subset of W3C XML Query working draft. The query language searches

hierarchical item types and locates items quickly and easily. Before you begin to write queries, you must understand the query language concepts, syntax, and grammar.

All queries are done on component type views. Therefore, the names that you use for root components or child components in your query strings can be the names of either base component type views that are created for you by the system (e.g. Journal, Journal_Article) or the names of user-defined component type views (e.g. My_Journal, My_Book_Section).

When you submit a query for a document, folder, object, and so forth, your request is directed to the Content Manager query engine, which processes the query and translates it to the appropriate SQL. The generated SQL goes through the library server where access permission is verified by applying the query to the system ACLs. After the query passes through the ACLs, it is applied to database tables to locate the items. The search results are returned to the application in the form of DDOs and XDOs that are accessed through either a collection, a result set cursor, or a callback object. By interfacing the DDO and XDO collections with the Content Manager persistent storage, your application can create, read, update, and delete documents into and from the Content Manager datastore.

Querying the Content Manager server

To query the Content Manager library server:

1. Query its datastore by creating a query string to represent your search conditions.
2. Call the evaluate, execute, or executeWithCallback method with your query string and query options.
3. Receive the results through a DKResults object, a dkResultSetCursor, or a dkCallback object.

The query APIs perform the query processing tasks, like preparing and executing a query, monitoring the status of a query execution, and storing the results.

Logically, a query string can contain one of three types of queries: parametric, text, and combined. A parametric query uses conditions like equality and comparison. A text query uses text search functions to make the search more powerful. A combined query is composed of both text and parametric conditions.

To run a query, you can use one of the following methods: evaluate, execute, or executeWithCallback. The execute method returns a dkResultSetCursor object, which has the following characteristics:

- The dkResultSetCursor works like a datastore cursor.
- You can use it for large result sets because the DDOs it returns are retrieved in blocks as the user requests them.
- You can use dkResultSetCursor to rerun a query, by calling the close and open methods.
- You can use the dkResultSetCursor to delete and update the current position of the result set cursor.

Applying the query language to the Content Manager data model

To help you understand the query language, you can conceptually view the library server as an XML document. The XML document analogy is used only for the purpose of explaining the query language. Therefore, it is very important to remember that the XML representation of the library server is only a virtual representation and items in a library server are not XML elements, nor do you get XML elements when you perform a query. In the XML representation of the library server, Content Manager data model elements are represented as follows:

Items In general, each CM Item is represented by nested XML elements, with the top level XML element representing the root component and the nested XML elements representing the descendent components. The nesting of XML elements thus represents the component hierarchy.

Root components

A root component is represented by the first level of an XML element. A root component has the following XML attributes: `ID ITEMID`, `STRING COMPONENTID`, `INTEGER VERSIONID`, `INTEGER SEMANTICTYPE`, and any other user-defined attributes of the component. In the library server, the ITEMID is unique.

Child components

A child component is represented by a nested XML element and has the following attributes: `STRING ITEMID`, `STRING COMPONENTID`, `INTEGER VERSIONID`, and any other user-defined attributes of the component.

Note that COMPONENTID alone is only unique within a Content Manager component.The`ITEMID` and the `VERSIONID` are exactly the same as the child's root component `ITEMID` and `VERSIONID`.

User-defined attributes

Each user-defined attribute is represented by a nested XML attribute within the XML element representing the containing component.

Links Although the inbound and outbound links are not a part of an Item itself in the CM data model (i.e. they are stored separately in the Links table), for the purpose of querying it is very convenient to conceptually think of them as being a part of the XML element representing the Item. This relieves applications from writing joins explicitly in the queries. Links represent a one-to-many relationship between items. Note that this relationship is only between the root component (a link cannot originate or end in a child component). The links originating at an Item are represented by <OUTBOUNDLINK> XML elements with attributes: IDREF LINKITEMREF, IDREF TARGETITEMREF and STRING LINKTYPE. The LINKITEMREF is a reference to an Item that contains meta-data for the link. The TARGETITEMREF is a reference to the Item pointed to by the link. The LINKTYPE is the type of the link. Similarly links pointing to an Item are represented by <INBOUNDLINK> XML elements with attributes: IDREF LINKITEMREF, IDREF SOURCEITEMREF and STRING LINKTYPE. The SOURCEITEMREF is a reference to the Item where the link originates.

References (reference attributes)

Reference attributes are represented by XML attributes of type IDREF. A reference represents a one-to-one relationship between an item or a component and another item. Therefore, the target of a reference can only be a root component, not child. A reference attribute, however, can originate in either root or child components.

Reference attributes can be either system-defined (SYSREFERENCEATTRS) or user-defined (PublicationRef in sample queries below). References can be traversed in both directions.

Reverse traversal of references is performed in a way similar to reverse traversal of links, as described above. You can conceptually think of the item that is being referenced as having an XML element called REFERENCEDBY that contains an XML attribute called REFERENCER (of type IDREF), which points to the component that references this item. This is similar to the INBOUNDLINK element with the SOURCEITEMREF attribute for reverse traversal of links.

Unlike links, references remember the versions of the components they connect. References also support delete semantics (restrict, cascade, set null, or no action).

Documents

The query language offers a virtual XML element called ICMPARTS that allows access to all the ICM parts item types contained in an item type with the classification of document.

Understanding parametric search

Items are often retrieved by initiating a search on selected attributes. A single query can examine both system-defined and user-defined attributes of the items in the datastore. Simple search conditions consist of an attribute name, an operator, and a value that are combined into a clause. Content Manager provides you with many comparison operators to complete parametric searches. The operators include:

- "="
- "< "
- "<="
- ">"
- ">="
- "!="
- "LIKE"
- "NOT LIKE"
- "BETWEEN"
- "NOT BETWEEN"
- "IS NULL"
- "IS NOT NULL"

You can specify complex search conditions by combining simple search conditions into a clause using the Boolean operators AND, OR, and NOT. Refer to query examples for more details.

Understanding text search

Using the DB2 Universal Database Text Information Extender (TIE), Content Manager provides two types of text search: text search of attributes that contain text in components and text search of objects. The main difference between the two types of text search is how the content is stored. When you define an attribute is to be text searchable, you are indicating that one can search text contained in the column of that attribute. To make an attribute (column) text searchable, TIE creates

a text index. The text index holds information about the text to be searched. This information is used to perform text search efficiently. For example, Fred, a system administrator, creates an item type called `Journal` that has a child component type view `Journal_Article`, which he wants to enable for text search. One of the attributes for `Journal_Article` is `Title`, which Fred enables for text search. When Lily, an underwriter, searches for `Title` that contains the word ″Java″, the system searches the `Title` text index for any hits on ″Java″.

For information about TIE, see the *DB2 Universal Database Text Information Extender (TIE)* documentation.

Searching for object contents

Searching for contents of objects works a little differently. Instead of indexing a column directly, the system uses a reference to the object's location on a resource manager. TIE uses the reference to fetch the content when it creates a text index. An end user performing a search does not notice any difference when searching for objects stored in a resource manager. A system administrator, however, has to set up a text resource item type view in order for the search mechanism to locate the content in the resource manager. The text search is performed on the resource item type's attribute ″TIEREF″, which refers to the contents stored on the resource manager for text search purposes.

Searching for documents

You can perform text search on the contents of document parts. A virtual component type view ″ICMPARTS″ is supported in query as a child of every document in the system. The ″TIEREF″ attribute under the ″ICMPARTS″ component type view refers to the contents of all the text-searchable parts of that document for text search purposes. Refer to query examples for specific usage of this functionality.

Making user-defined attributes text searchable

You can make your user-defined attributes text searchable by using the DKAttrDefICM and DKItemTypeDefICM APIs. Default properties of the created text index can be modified by using the DKTextIndexDefICM class. For more information on the APIs, see the online API reference.

Understanding text search syntax

You can perform text search queries by using either basic or advanced text search syntax.

Basic text search

Since the majority of text searches are done by simply listing a few words one after the other, basic (simplified) text search syntax was designed specifically to make this most common case easy for users. The syntax also allows for use of ″+″ and ″-″, as well as for use of quoted phrases. Simplified text search is done by using ″contains-text-basic″ and ″score-basic″ functions. The ″contains-text-basic″ function is used to search within attributes or within content of resources or documents. The ″score-basic″ function uses the same syntax as the ″contains-text-basic″ function and is used for sorting results based on the rank of the text search results. Don't forget to equate the ″contains-text-basic″ function to 1 to check if it is true, and to equate it to 0 to check if it is false. Refer to query examples for how these functions are used.

Additional information about basic text search syntax includes the following:

- Performs case-insensitive text search (just like advanced syntax by default). See the TIE documentation for case-sensitive search options.
- Terms within quotes are assumed to be a phrase
- Use of + (plus) - (minus)
  - + (plus) = document must include this word
  - - (minus) = document must not include this word
  - When a + or - is not specified, the query engine uses an algorithm to match the words to the text.
- Boolean operators (AND, OR, NOT) are not valid and are ignored
- Parentheses in the basic syntax are not supported
- Valid wildcards
  - ? (question mark) = represents a single character
  - * (asterisk) = represents any number of arbitrary characters

Advanced text search

Advanced text search syntax is used to allow the user to specify more complex conditions for text search. The text search uses the TIE text search syntax, and allows such powerful features as proximity search and fuzzy search. Advanced text search syntax uses ″contains-text″ and ″score″ functions similar to the way the ″contains-text-basic″ and ″score-basic″ functions are used for basic text search. The strings that are supplied to the advanced functions should be in TIE syntax, except as follows: change double quotes to single quotes, and vice-versa. For example, CONTAINS (description,' ″IBM″ ')=1 condition in TIE would become contains-text(@description, ″ 'IBM' ″)=1 in CM query language. This needs to be done to support simplicity of writing queries with minimal use of escape characters. Don't forget to equate the ″contains-text″ function to 1 to check if it is true, and to equate it to 0 to check if it is false. Refer to query examples for more details on advanced text search.

Creating combined parametric and text search

Parametric and text search are performed through the same XQPE string interface. To prepare and execute a query:

1. Create an ICM datastore and connect to it.
2. Generate the combined query string.

```
String queryString =
       "//Journal_Article [Journal_Author/@LastName = \"Richardt\"" +
       " AND contains-text (@Text, \" 'Java' & 'XML' \")=1]";
```

3. If you want to use retrieve options that are different from the defaults, use the DKNVPair array to package the options.

```
DKNVPair parms[] = new DKNVPair[3];
String strMax = "5";
parms[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, strMax);
parms[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,
DKConstant.DK_CM_CONTENT_ATTRONLY | DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);
parms[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
```

4. Execute the combined query in one of three different ways: evaluate, execute, and executeWithCallback.

```
DKResults resultsCollection =
(DKResults)dsICM.evaluate(queryString,
DKConstant.DK_CM_XQPE_QL_TYPE, parms);
```

5. Process the results. The procedure for handling the results depends on which execution method you used.

Query examples

To help you better understand the query language and to get you started with writing queries, this section provides you with the following information:

- A sample data model
- An XML document representation of the data model
- Sample queries
- Query language grammar

The sample queries in the following sections are based on the data model in Figure 13 on page 105. Refer to the data model illustration when you review the sample queries.

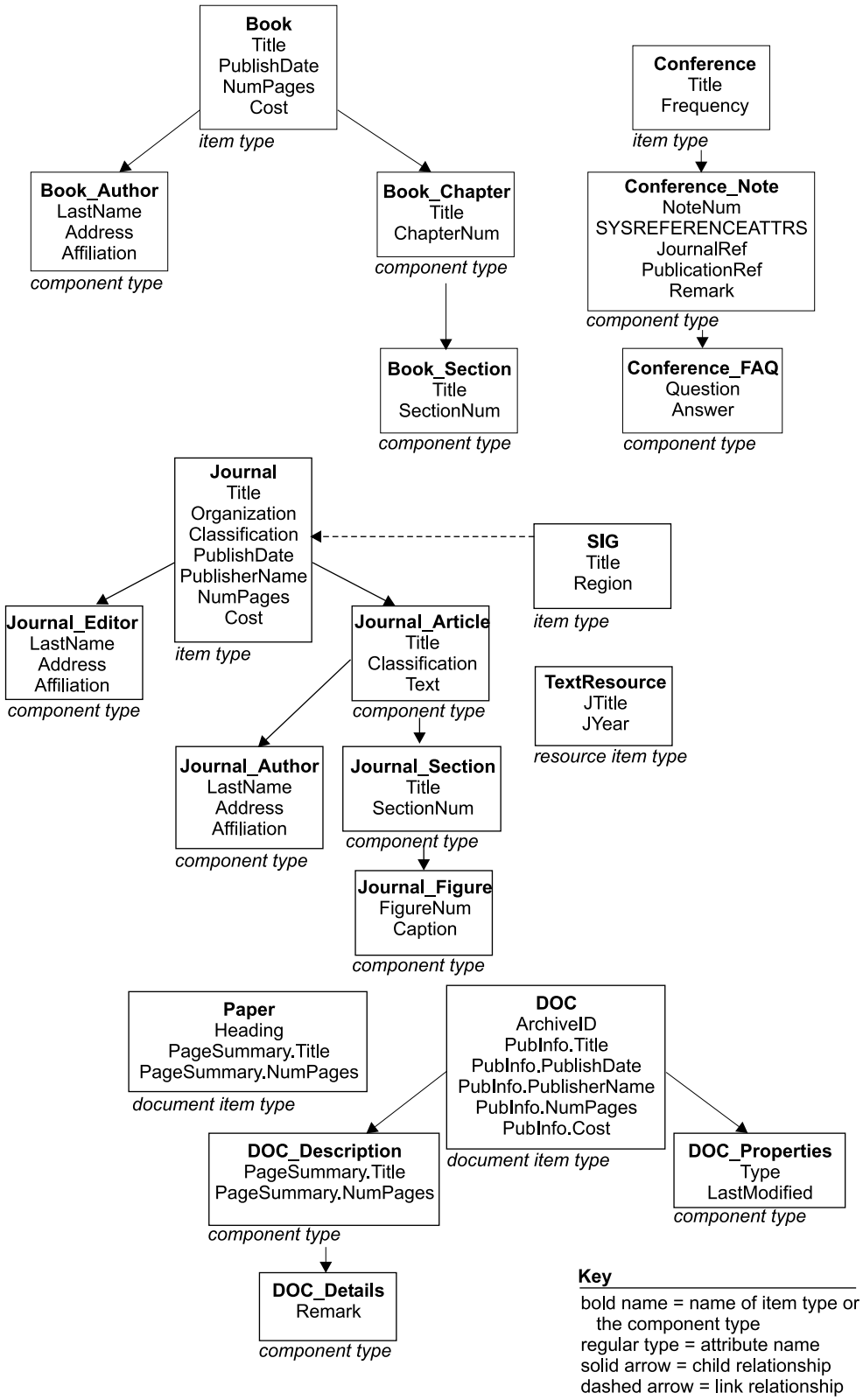

Figure 13. Query examples data model

The XML document below is a representation of the data model in figure Figure 13.

XML representation of the query examples data model:

```
<Journal (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                INTEGER SEMANTICTYPE, Title, Organization, Classification,
                PublishDate, PublisherName, NumPages, Cost)>
    <Journal_Editor (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                   LastName, Address, Affiliation)>
    </Journal_Editor>
    ... (repeating <Journal_Editor>)

    <Journal_Article (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                    Title, Classification, Text)>
        <Journal_Section (STRING ITEMID, STRING COMPONENTID,
                         INTEGER VERSIONID, Title, SectionNum)>
            <Journal_Figure (STRING ITEMID, STRING COMPONENTID,
                            INTEGER VERSIONID, FigureNum, Caption)>
            </Journal_Figure>
            ...(repeating <Journal_Figure>)
        </Journal_Section>
        ... (repeating <Journal_Section>)

        <Journal_Author (STRING ITEMID, STRING COMPONENTID,
                        INTEGER VERSIONID, LastName, Address, Affiliation)>
        </Journal_Author>
        ... (repeating <Journal_Author>)
    </Journal_Article>
    ... (repeating <Journal_Article>)

    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                   STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                  STRING  LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</Journal>
...(repeating <Journal>)

<Book (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID, INTEGER
       SEMANTICTYPE,Title, PublishDate, NumPages, Cost)>
    <Book_Author (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                  LastName, Address, Affiliation)>
    </Book_Author>
    ... (repeating <Book_Author>)

    <Book_Chapter (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                   Title, ChapterNum)>
        <Book_Section (STRING ITEMID, STRING COMPONENTID,
                       INTEGER VERSIONID, Title, SectionNum)>
        </Book_Section>
        ... (repeating <Book_Section>)
    </Book_Chapter>
    ... (repeating <Book_Chapter>)

    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                   STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                  STRING  LINKTYPE)>
    </INBOUNDLINK>
```

```
    ... (repeating <INBOUNDLINK>)


    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</Book>
... (repeating <Book>)

<SIG (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
      INTEGER SEMANTICTYPE, Title, Region)>
    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                   STRING  LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                  STRING  LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</SIG>
... (repeating <SIG>)

<TextResource (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
               INTEGER SEMANTICTYPE, JTitle, JYear)>
    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                  STRING  LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                  STRING LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</TextResource>
... (repeating <TextResource>)
```

Query examples

The sample queries provided in this section are based on the sample data model, Figure 13 on page 105, and the sample XML document on page 106. Here are some hints to help you understand the query examples:

- Follow the query string as you would follow a directory structure
- ″/″ single slash indicates a direct child relationship
- ″//″ double slash indicates either a child relationship or a descendant relationship
- ″.″ (DOT) represents the current component in the hierarchy
- ″..″ (DOT-DOT) represents the parent of the current component
- ″@″ (AT sign) denotes an attribute
- ″[ ]″ (square brackets) denote a conditional statement or a list
- ″=>″ (DEREFERENCE operator) represents linking or referencing action
- The result of the query must be a component (for example, an attribute cannot be the last thing in the path)

Example 1: access to components

This query finds all journals.

```
/Journal
```

Explanation: The "/" starts at the implicit root of the XML document, which in this case is the entire library server. Each item type is an element under this root. If LS.xml is the XML document that contains the entire model as described above, then the explicit document root is document (LS.xml).

Example 2: access to attributes

This query finds all journal articles with a total of 50 pages in them.

```
/Journal[@NumPages=50]
```

Explanation: The predicate @NumPages = 50 evaluates to true for all journals that have the Content Manager attribute "NumPages"set to 50.

Example 3: multiple item types

This query finds all books or journals that have"Williams" as one of the authors and have a section title beginning with "XML".

```
(/Book | /Journal)
[(.//Journal_Author/@LastName = "Williams"
OR .//Book_Author/@LastName = "Williams")
AND (.//Book_Section/@Title LIKE "XML%"
OR .//Journal_Section/@Title LIKE "XML%")]
```

OR

```
(/Book[.//Book_Author/@LastName = "Williams"
AND .//Book_Section/@Title LIKE "XML%"])
| (/Journal[.//Journal_Author/@LastName = "Williams"
 AND .//Journal_Section/@Title LIKE "XML%"])
```

Explanation: The above two queries produce the same result. ".//Journal_Author" means that a component Journal_Author should be found either directly under the current component in the path (which in the first case is either a Book or a Journal) or somewhere deeper in the hierarchy. Note that the LIKE operator is used in conjunction with a wildcard character, in this case" %".

Example 4: arithmetic operations in conditions

This query finds all journals with the number of pages between 45 and 200.

```
/Journal[@NumPages BETWEEN 49-4 AND 2*100]
```

Explanation: Note that you can perform arithmetic operations to calculate the resulting values to be used with the BETWEEN operator.

Example 5: traversal of links in the forward direction

This query finds all articles in journals edited by "Williams" that are contained in SIGs with title "SIGMOD".

```
/SIG[@Title = "SIGMOD"]/OUTBOUNDLINK
 [@LINKTYPE = "contains"]/@TARGETITEMREF =>
 Journal[Journal_Editor/@LastName = "Williams"]
 /Journal_Article
```

Explanation: This is an example of following links in the forward direction. The virtual XML component OUTBOUNDLINK and its attribute TARGETITEMREF are used to traverse to all Journals and then finally the underlying Journal_Articles. The last component in the path is what is

returned as the result of the query. The result can be constrained by traversing only specific link types (“contains” in this example) to a specific type of items (Journal in this example). Since the conceptual XML representation of the library server looks at inbound and outbound links as being parts of items, the dereferencing operator can be used to relieve applications from writing explicit joins.

Example 6: traversal of links in the backward direction

This query finds all items of any type that have journals which cost less than five dollars with articles by author“ Nelson”.

```
/Journal[@Cost < 5
AND .//Journal_Author/@LastName = "Nelson"]
/INBOUNDLINK[@LINKTYPE = "contains"]
/@SOURCEITEMREF => *
```

Explanation: This is an example of following links in the backward direction. The wildcard“*”, following the dereference operator “=>” ensures that items of ANY type are returned as the result.

Example 7: text search (contains-text and score functions)

This query finds journal articles with author“ Richardt” that contain the text “Java” and the text “XML”. The results are ordered by the text search score.

```
//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text(@Text, " 'Java' & 'XML' ")=1]
SORTBY(score(@Text, " 'Java' & 'XML' "))
```

Explanation: This is an example of performing text search with the contains-text function. For the syntax supported by this function, see the documentation for DB2 Text Information Extender (TIE). Note that the contains-text function should be equated with 1 to be true and 0 to be false. The score function uses the ranking information returned by TIE, which is used in this case to sort the resulting journal articles through SORTBY.

Example 8: text search (contains-text function and attribute sorting)

This query finds all journals that have either the word “Design” or the word“ Index” in their title and sorts the results in descending order by their title.

```
/Journal
[Journal_Article[contains-text(@Title, " 'Design' | 'Index'  ")=1]]
SORTBY (@Title DESCENDING)
```

Explanation: This is another example of performing text search using the contains-text function. The sorting in this case uses the DESCENDING operator on the “Title ”attribute. The default for the SORTBY is ASCENDING.

Example 9: text search (contains-tex-basic and score-basic functions)

This query finds all journal articles that contain the text “Java” and the text “JDK 1.3” but not the text “XML” using the simplified (basic) text search syntax and sort the results by the text search score.

```
//Journal_Article
[contains-text-basic(@Title, " +Java -XML +'JDK 1.3'")=1]
SORTBY (score-basic(@Title, " +Java -XML +'JDK 1.3' "))
```

Explanation: This is an example of performing text search using the simplified text search syntax. Use a“ +” to indicate the words or phrases that should be present in the attribute“ Title”, and, similarly, use a “-” to

exclude other words or phrases. The score-basic function works similarly to the score function in the previous example, but uses a simplified syntax.

Example 10: text search on resource items

This query finds text resources in a text resource item type “TextResource” that contain the text “Java” and the text “XML”.

```
/TextResource[contains-text(@TIEREF, " 'Java' & 'XML'
")=1]
```

Explanation: This is an example of performing text search inside of the resources in the resource manager. Note that the “TIEREF” attribute is used as a representation of the resource that is represented by the item of type “TextResource”. TIE syntax is used as usual in this case inside the contains-text function. For the syntax supported by this function, see the documentation for DB2 Text Information Extender (TIE).

Example 11: traversal of references in the forward direction

This query finds all the frequently asked questions for conferences, for which the conference notes refer to books with titles mentioning EIP.

```
 /Conference/Conference_Note [@PublicationRef =>
Book[@Title LIKE "%EIP%"]]
/Conference_FAQ
```

Example 12: traversal of references in the forward direction

This query finds all chapters of books referenced in the notes of conferences related to Internet.

```
/Conference[@Title LIKE "%Internet%"]
/Conference_Note/@PublicationRef =>
*/Book_Chapter
```

Example 13: traversal of references in the reverse direction

This query finds all the components that have references pointing to any books.

```
/Book/REFERENCEDBY/@REFERENCER => *
```

Example 14: traversal of references in the reverse direction

This query finds all the frequently asked questions under conference notes that refer to books about XML.

```
/Book[@Title LIKE "XML"]/REFERENCEDBY/@REFERENCER =>
Conference_Note/Conference_FAQ
```

Explanation: Note that since the reference attributes originate inside of the Conference_Note component, this is the component that must appear as the first component after the dereference operator. This query produces an empty result set if, for example, Conference follows the“ =>” operator.

Example 15: traversal of references in the reverse direction

This query finds all the components that contain XML in their remarks and that have references pointing to books.

```
/Book/REFERENCEDBY/@REFERENCER =>
*[@Remark LIKE "%XML%"]
```

Example 16: latest version function

This query finds all the journals of the latest version. By default, all versions of the indicated component type view that match the query are returned. VERSIONID is a system-defined attribute that is contained in every component type.

```
/Journal[@VERSIONID = latest-version(.)]
```

Example 17: latest version function on the target of the dereference

This query finds all the books of the latest version that are referenced in the notes of any conferences.

```
/Conference/Conference_Note/@SYSREFERENCEATTRS =>
Book[@VERSIONID = latest-version(.)]
```

Example 18: latest version function on wildcard components

This query finds all the components of the latest version that have references pointing to any books.

```
/Book/REFERENCEDBY/@REFERENCER => *
[@VERSIONID = latest-version(.)]
```

Example 19: system-defined attributes

This query finds all the root components with a specific item ID.

```
/*[@ITEMID =
"A1001001A01J09B00241C95000"]
```

Example 20: text search on document model

This query finds all documents that contain the word “XML” in any one of its parts.

```
/Doc[contains-text(.//ICMPARTS/@TIEREF, " 'XML' ")=1]
```

Explanation: The query language offers a virtual component “ICMPARTS” that allows access to all the ICM Parts item types contained under a specific item type of Document classification.

Example 21: document model (access to ICM Parts)

This query finds all the parts of the document with the storage ID of 555.

```
/Doc[@ArchiveID = 555]/ICMPARTS/
@SYSREFERENCEATTRS => *
```

Example 22: document model (access to ICM Parts)

This query finds all the parts in all of the documents in the system.

```
 //ICMPARTS/@SYSREFERENCEATTRS => *
```

Explanation: Because both the Doc and Paper item types have been defined as being Documents in the system, the ICM Parts from both of them are returned in the result.

Example 23: existence of attributes

This query finds all root components that have a title.

```
  /*[@Title]
```

Explanation: To eliminate the restriction that only root components should be returned, the query can be rewritten to start with a double-slash

```
//*[@Title]
```

Example 24: list of both literals and expressions

This query finds all journals that have a title that is equal to either its article’s title, its section’s title, or ″IBM Systems Journal″.

```
/Journal[@Title = [Journal_Article/@Title,
.//Journal_Section/@Title,"IBM Systems Journal"]]
```

Example 25: list of numeric literals

This query finds all books that cost either $10, or $20, or $30.

```
 /Book[@Cost = [10, 20, 30]]
```

Example 26: list of a result of query

This query finds all journals or all books with the title “Star Wars”.

```
[/Journal, /Book[@Title = "Star Wars"]]
```

Example 27: attribute groups

This query finds all details on documents in which the description is at least 20 pages long.

```
/Doc[Doc_Description/@PageSummary.NumPages >=
20]//Doc_Details
```

Explanation: Note that if an attribute (for example, "NumPages") is contained in an attribute group (for example, "PageSummary"), then you must refer to that attribute as GroupName.AttrName (for example, PageSummary.NumPages). The attribute "@NumPages" would not be found under Doc_Description.

Intermediate results obtained by INTERSECT/EXCEPT cannot be combined with arithmetic(unary/binary) or comparison operators. They can be combined by set operators (UNION/INTERSECT/EXCEPT) or appear by themselves.

Examples of valid usage of UNION/INTERSECT/EXCEPT:

1.
```
(/Journal/Journal_Article[@Title = "Content Management"]
EXCEPT
//Journal_Article[@Classification =
"Security"])/Journal_Section
```

   This query is valid because the result of the EXCEPT is the result of the entire query - it is not combined using any operators.

2.
```
/Journal[(Journal_Editor/@LastName
UNION .//Journal_Author/@LastName) = "Davis"]
```

   This query is valid because there is no restriction on UNION operator.

3.
```
/Journal[Journal_Article[Journal_Section/@Title INTERSECT
.//Journal_Figure/@Caption]/@Title = "Content Management"]
```

   This query is valid because the result of INTERSECT is not combined using any operator.

4.
```
/Journal[@Title = "VLDB"]
UNION /Journal[@Cost = 20]
INTERSECT /Journal[@Organization = "ACM"]
```

   This query is valid because the result of INTERSECT operator is combined using a set operator (UNION).

Examples of invalid usage of INTERSECT/EXCEPT:

1.
```
/Journal[(Journal_Editor/@LastName
INTERSECT .//Journal_Author/@LastName) = "Davis"]
```

   This query is invalid because the result of INTERSECT operator is combined using a comparison operator (=).

2.
```
/Journal[(.//Journal_Section/@SectionNum
EXCEPT .//Journal_Figure/@FigureNum) + 5 = 10]
```

   This query is invalid because the result of EXCEPT is combined using an arithmetic operator (+).

Escape sequences in the query language

To support advanced features of the query language (like the wildcards ″%″ or ″_″ inside of text strings), escape sequences are used to differentiate between the cases when wildcards are treated as regular characters versus when they are given the special meaning of wildcard characters. For the user, it is important to know which characters are used as wildcards because wildcard characters, when intended to be treated as regular characters, must be preceded by an escape character. Escape sequences are also used to handle single and double quotes.

You need to add escape sequences when the strings used in queries contain either special characters (double-quote, apostrophe) or wildcard characters (percent sign, underscore, star, question mark) or a default escape character (a backlash). This handling is the simplest for strings used in comparison conditions and becomes a bit more involved for the LIKE operator and text search functions. Proper handling of special characters will ensure successful execution of queries and correctness of query results.

Important: Use wildcard characters sparingly as using them in your queries can increase the size of your result list significantly, which can decrease performance and return unexpected search results.

Using escape sequences with comparison operators (″=″, ″!=″, ″>″, ″<″, ″BETWEEN″ and others)

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[@Title = "Analysis of ""The Time Machine"" by H.
G. Wells himself"]
```

Since the article's title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark (apostrophe) ’

You do not need to escape in this case.

Example:

```
/Book[@Title != ″Uncle Tom’s Cabin″]
```

Using escape sequences with the LIKE operator

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[@Title LIKE "Analysis of ""The Time Machine"" %"]
```

Since the article's title contains the name of the book in double quotes,″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark (apostrophe) ’

You do not need to escape in this case.

Example:

```
 /Book[@Title LIKE "Uncle Tom's Cabin"]
```

Wildcards (″%″, ″_″)

The percent sign ″%″ is a wildcard character used to represent any number of arbitrary characters in a string used with the LIKE operator. The underscore ″_″ is a wildcard character used to represent a single arbitrary character. If you want these wildcard characters to be treated as regular characters, you need to do the following:

1. Precede the wildcard character with an escape character
2. Add an ESCAPE clause with the escape character after the LIKE phrase.

Example A:

```
/Book[@Title LIKE "Plato%s%S_mposium"]
```

This example shows how wildcards ″%″ and ″_″ are used to find a book whose title's spelling is uncertain.

Example B:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ in query"
ESCAPE "!"]
```

Since the search string in this example contains the underscore ″_″ as a regular character (not a wildcard), you can escape the underscore with an exclamation point character ″!″. Any single character can be used as an escape character.

Example C:

```
//Journal_Article[@Title LIKE "_sage of underscore \_ in%" ESCAPE
"\"]
```

In this query, wildcard characters are used as both regular characters (″_″ escaped by ″\″) and as wildcards (″_″) to catch both uppercase and lowercase versions of the word ″Usage″, as well as ″%″ to catch multiple endings of the string.

Example D:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ on Yahoo!!"
ESCAPE "!"]
```

You can also use an escape character as a regular character. To do so, precede the escape character with itself, as in the example to search for ″Yahoo!″ below.

Using escape sequences with advanced text search (″contains″ and ″score″ functions)

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[contains-text (@Title, " 'Analysis of ""The Time
Machine"" %' ")=1]
```

Since the article's title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark(apostrophe) ’

Precede the apostrophe with another apostrophe. A single apostrophe is not allowed in advanced text search because a set of apostrophes is used to enclose a term or a phrase. If an apostrophe appears inside a term, then the apostrophe needs to be escaped to differentiate it from the apostrophe that ends the term or the phrase.

Example A:

```
/Book[contains-text (@Title, " 'Uncle Tom''s Cabin' ")=1] SORTBY
(score (@Title, " 'Uncle Tom''s Cabin' "))
```

Note that Tom’’s has two apostrophes.

Example B:

```
/Book[contains-text (@Title, " ('Greek' & 'Plato''s Symposium') &
NOT ' Socrates' ")=1] SORTBY (score (@Title, " ('Greek' & 'Plato''s
Symposium') & NOT ' Socrates' "))
```

Note that Plato’’s has two apostrophes.

Wildcards (″%″, ″_″)

Just as the LIKE operator, advanced syntax uses ″%″ and ″_″ as wildcards. The percent sign ″%″ is a wildcard character used to represent any number of arbitrary characters. The underscore ″_″ is a wildcard character used to represent a single arbitrary character. If you want a wildcard character to be treated as a regular character, you need to do the following:

1. Precede the wildcard character with an escape character
2. Add an ESCAPE clause after EACH term where you use the escape character

Example A:

```
/Book[contains-text (@Title, " 'Usage of underscore !_ in query'
ESCAPE '!' ")=1] SORTBY (score (@Title, " 'Usage of underscore !_ in
query' ESCAPE '!' "))
```

In this example, an exclamation mark ″!″ is used as an escape character before the underscore.

Example B:

```
/Book[contains-text (@Title, " 'Usage of underscore !_ in query'
ESCAPE '!' | 'Yahoo! For Dummies' | 'Usage of underscore !_ on
Yahoo!!' ESCAPE '!' | 'War and Peace' ")=1]
```

Note that an ESCAPE clause must be added after every term in your text search string where you escape wildcards, even if the escape character is the same in all the terms.

Using escape sequences with basic text search (″contains-text-basic″ and ″score-basic″ functions)

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[contains-text-basic (@Title, "Analysis of '""The
Time Machine""' ")=1]
```

Since the article's title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped. The book title is inclosed in apostrophes to keep it as a phrase.

Single quotation mark (apostrophe) '

Precede the apostrophe with another apostrophe. Basic text search syntax allows terms enclosed within single quotes, so that a term can contain a space. The doubling of the apostrophe is therefore necessary to differentiate the case of an apostrophe occurring within a term from the case of an apostrophe starting a new term.

Example A:

```
/Book[contains-text-basic (@Title, "Uncle Tom''s Cabin")=1]SORTBY
(score-basic (@Title, "Uncle Tom''s Cabin"))
```

Note that Tom''s has two apostrophes.

Example B:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato''s Symposium'
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek +'Plato''s
Symposium' -Socrates "))
```

Note that Plato''s has two apostrophes and 'Plato's Symposium' is enclosed in single quotes since it is a phrase.

Wildcards (″*″, ″?″ and ″\″)

Precede ″*″, ″?″, and ″\″ characters with a backslash ″\″ if these characters are not to be treated as wildcards. Star ″*″ is a wildcard character used to represent any number of arbitrary characters in basic text search for the functions contains-text-basic and score-basic. The question mark ″?″ is a wildcard character used to represent a single arbitrary character. For basic text search, the query language provides an escape character backslash ″\″ to be used when the term to be searched contains the wildcard character in it and you want to treat that wildcard character as a regular character.

Example A:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato*s*S?mposium'
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek
+'Plato*s*S?mposium' -Socrates "))
```

This example shows how to use basic text search when the spelling of a term is not certain. The ″*″ and ″?″ characters are meant to be wildcards in this case, so they are not escaped.

Example B:

```
/Book[contains-text-basic (@Title, "Why forgive\?")=1] SORTBY
(score-basic (@Title, "Why forgive\?"))
```

In this example, the title contains the question mark ″?″ as a normal character, so this character can be escaped with a backslash.

Example C:

```
//Journal_Section[contains-text-basic (@Title,
"C:\\OurWork\\IsNeverDone")=1] SORTBY (score-basic (@Title,
"C:\\OurWork\\IsNeverDone"))
```

Each backslash that naturally occurs in the search term`"C:\OurWork\IsNeverDone"` must be escaped with another backslash.

Using escape sequences in Java

Precede special characters (for example, double quotes and backslash) with a backslash.

Example:

Query:

```
/Book[contains-text-basic (@Title, "Why forgive\?")=1]
```

Java:

```
String query = "/Book[contains-text-basic (@Title, \"Why forgive\\?\")=1]";
```

Note how the internal double quotes and the backslash before the question mark are preceded by a backslash. This handling is inherent to Java and C++ programming languages. For more information, refer to the specifications for these languages.

The query language grammar

The query language formal grammar is as follows:

```
(* keywords *)
AND = ("a" | "A"), ("n" | "N"), ("d" | "D") ;
ASCENDING = ("a" | "A"), ("s" | "S"), ("c" | "C"), ("e" | "E"), ("n" | "N"), ("d"
| "D"), ("i" | "I"), ("n" | "N"), ("g" | "G") ;
BETWEEN = ("b" | "B"), ("e" | "E"), ("t" | "T"), ("w" | "W"), ("e" | "E"), ("e" |
"E"), ("n" | "N") ;
DESCENDING = ("d" | "D"), ("e" | "E"), ("s" | "S"), ("c" | "C"), ("e" | "E"), ("n"
| "N"), ("d" | "D"), ("i" | "I"), ("n" | "N"), ("g" | "G") ;
DIV = ("d" | "D"), ("i" | "I"), ("v" | "V") ;
EXCEPT = ("e" | "E"), ("x" | "X"), ("c" | "C"), ("e" | "E"), ("p" | "P"), ("t" | "T")
;
INTERSECT = ("i" | "I"), ("n" | "N"), ("t" | "T"), ("e" | "E"), ("r" | "R"), ("s" |
"S"), ("e" | "E"), ("c" | "C"), ("t" | "T") ;
 LIKE = ("l" | "L"), ("i" | "I"), ("k" | "K"), ("e" | "E") ;
 MOD = ("m" | "M"), ("o" | "O"), ("d" | "D") ;
NOT = ("n" | "N"), ("o" | "O"), ("t" | "T") ;
OR = ("o" | "O"), ("r" | "R") ;
SORTBY = ("s" | "S"), ("o" | "O"), ("r" | "R"), ("t" | "T"), ("b" | "B"), ("y" | "Y")
;
UNION = ("u" | "U"), ("n" | "N"), ("i" | "I"), ("o" | "O"), ("n" | "N") ;
IS = ("i" | "I"), ("s" | "S");
NULL = ("n" | "N"), ("u" | "U"), ("l" | "L"), ("l" | "L");
ESCAPE_KEYWORD = ("e" | "E"), ("s" | "S"), ("c" | "C"), ("a" | "A"), ("p" |
"P"), ("e" | "E");
```

KEYWORD = (AND | ASCENDING | BETWEEN | DESCENDING | DIV | EXCEPT | INTERSECT | LIKE | MOD | NOT | OR | SORTBY | UNION | IS | NULL) ;

(* literals *)

DIGIT = (″0″ | ″1″ | ″2″ | ″3″ | ″4″ | ″5″ | ″6″ | ″7″ | ″8″ | ″9″) ;

NONZERO_DIGIT = (″1″ | ″2″ | ″3″ | ″4″ | ″5″ | ″6″ | ″7″ | ″8″ | ″9″) ;

Exponent = (e | E), [″+″ | ″-″], DIGIT, { DIGIT }

INTEGER_LITERAL = ″0″ | NONZERO_DIGIT, {DIGIT} ;

FLOAT_LITERAL = DIGIT, { DIGIT }, ″.″, { DIGIT }, [Exponent] | [″.″], DIGIT, { DIGIT }, [Exponent] ;

(* UNICODE_CHARACTER is the set of all unicode characters and escape sequences. It's definition is not included in this document *) (* String literals are delimited by double quotes and can contain any character except double quote. To include a double quote as the part of the string literal, specify two consecutive double quotes i.e. a double quote is escaped by another double quote. These will be treated as one double quote character *)

STRING_LITERAL = ’″’, { (UNICODE_CHARACTER - ’″’) | (’″’, ’″’) } , ’″’ ;

(* Escape sequence is a single character delimited by double quotes. To specify a double quote itself as the escape character, specify two consecutive double quotes i.e. a double quote is escaped by another double quote. These will be treated as one double quote character. For the complete explanation of the legal values for ESCAPE_CHARACTER, please see the DB2 SQL Reference section on the LIKE Predicate. *)

ESCAPE_LITERAL = ’″’, ((ESCAPE_CHARACTER - ’″’) | (’″’, ’″’)), ’″’;

LETTER = (″a″ | ″b″ | ″c″ | ″d″ | ″e″ | ″f″ | ″g″ | ″h″ | ″i″ | ″j″ | ″k″ | ″l″ | ″m″ | ″n″ | ″o″ | ″p″ | ″q″ | ″r″ | ″s″ | ″t″ | ″u″ | ″v″ | ″w″ | ″x″ | ″y″ | ″z″ | ″A″ | ″B″ | ″C″ | ″D″ | ″E″ | ″F″ | ″G″ | ″H″ | ″I″ | ″J″ | ″K″ | ″L″ | ″M″ | ″N″ | ″O″ | ″P″ | ″Q″ | ″R″ | ″S″ | ″T″ | ″U″ | ″V″ | ″W″ | ″X″ | ″Y″ | ″Z″ | ″_″ | ″$″) ;

(* An IDENTIFIER begins with a letter (a-z, A-Z) or an underscore or a dollar character, followed by zero or more letters, underscores, dollar characters or digits (0-9). A keyword can be an IDENTIFIER only if it is enclosed within single quotes *)

IDENTIFIER = (LETTER, { LETTER | DIGIT }) - KEYWORD | ″’″, LETTER, { LETTER | DIGIT }, ″’″;

ExpressionWithOptionalSortBy = LogicalOrSetExpression, SORTBY, ″(″, SortSpecList, ″)″ | Expression ;

Expression = LogicalOrSetExpression ;

SortSpecList = SortSpec, { ″,″, SortSpec } ;

SortSpec = Expression, [ASCENDING | DESCENDING] ;

LogicalOrSetExpression = LogicalOrSetTerm | LogicalOrSetExpression, (OR | UNION | ″|″ | EXCEPT), LogicalOrSetTerm ;

LogicalOrSetTerm = LogicalOrSetPrimitive | LogicalOrSetTerm, (AND | INTERSECT), LogicalOrSetPrimitive ;

LogicalOrSetPrimitive = [NOT], SequencedValue ;

SequencedValue = ValueExpression ;

ValueExpression = Comparison ;

Comparison = ArithmeticExpression | Comparison, CompareOperator, ArithmeticExpression, ESCAPE_KEYWORD, ESCAPE_LITERAL | Comparison,

CompareOperator, ArithmeticExpression | Comparison, [NOT], BETWEEN, ArithmeticExpression, AND, ArithmeticExpression | Comparison, IS, [NOT], NULL ;

ArithmeticExpression = ArithmeticTerm | ArithmeticExpression, (″+″ | ″-″), ArithmeticTerm ;

ArithmeticTerm = ArithmeticFactor | ArithmeticTerm, (″*″ | DIV | MOD), ArithmeticFactor ;

ArithmeticFactor = ArithmeticPrimitive | (″+″ | ″-″), ArithmeticFactor ;

ArithmeticPrimitive = BasicExpression, OptionalPredicateList | PathExpression ;

PathExpression = Path | (″/″ | ″//″), Path | BasicExpression, OptionalPredicateList, (″/″ | ″//″), Path ;

Path = Step | Path, (″/″ | ″//″), Step ;

Step = NodeGenerator, OptionalPredicateList ;

NodeGenerator = NameTest | ″@″, NameTest | ″@″, NameTest, ″=>″, NameTest | ″..″ ;

OptionalPredicateList = {Predicate} ;

Predicate ::= [″, Expression, ″]″ ;

BasicExpression = Literal | FunctionName, ″(″, OptionalExpressionList, ″)″ | ″(″ Expression ″)″ | ListConstructor | ″.″ ;

FunctionName = QName ;

Literal = STRING_LITERAL | INTEGER_LITERAL | FLOAT_LITERAL ;

OptionalExpressionList = [ExpressionList] ;

ExpressionList = Expression, {″,″, Expression } ;

ListConstructor = ″[″, [ListContent], ″]″ ;

ListContent = Expression, {″,″, Expression } ;

NameTest = QName | ″*″ ;

QName = LocalPart ;

LocalPart = IDENTIFIER;

CompareOperator = ″=″ | ″< ″ | ″<=″ | ″>″ | ″>=″ | ″!=″ | [NOT] LIKE;

Working with the resource manager

A Content Manager resource manager controls a collection of managed resources (objects). It also manages the necessary storage and Hierarchical Storage Management (HSM) infrastructure, but you must first configure the resource manager to support HSM. Resource managers have facilities to support type-specific services for more than one type of object, such as streaming, zipping, unzipping, encrypting, encoding, transcoding, searching, or text mining.

A single resource manager is used exclusively by one library server. Each resource manager delivered by the Content Manager system provides a common subset of native data access APIs through which it is accessible by the controlling library server, by other Content Manager components, and by applications, either locally (on the same network node) or remotely.

Other data access APIs allow remote access to a resource manager using the resource manager's own client support or a standard network access protocol such as CIFS, NFS, or FTP. For remote access, use a client-server connection. Clients communicate with Content Manager resource managers using HTTP through the use of a standard Web server. Data delivery is based on HTTP, FTP, and FS data transfer protocols. Using HTTP, any application or Content Manager component

that needs to access Content Manager-managed content can dynamically form a triangle with a library server and resource manager. This triangle forms a direct data access path between the application and each resource manager, and a control path between the library server and the resource manager. You can map this conceptual triangle to any network configuration, ranging from a single-node configuration to a geographically distributed one.

The new architecture also accommodates resource managers that an application is not able to access directly, such as a host-based subsystem, a single-user system that does not handle access control, or a system containing highly sensitive information where direct access by an application is not allowed by business policy. In this case, access to such resource managers is indirect. Both the pull and the push paradigms of data transfer are accommodated by the Content Manager system as well as synchronous and asynchronous calls.

For information about how to configure a resource manager see *Planning Your Content Manager System* and the `SResourceMgrDefCreationICM` sample in the `samples` directory, `cmbroot\samples\java\icm`.

Working with resource manager objects

Within Content Manager, every managed entity is called an item. Items come in two types, the type that represent pure logical entities, such as documents or folders, or entities that represent physical data objects, such as the text data of a word processing document, the scanned image of a claim or the video clip of an automobile accident. Objects have a special state and behavior needed to handle the physical data associated to a logical document.

Resource objects also represent things like files in a file system, video clips in a video server, and BLOBs. At run time, resource objects are used to access the physical data they point to. For that reason, resource objects in Content Manager have a type. That is, they have a specific state and behavior. The library server and the resource manager share a schema to store the state of an object. The base object types provided by Content Manager are: generic BLOBs or CLOBs, Text, Image, andVideo content objects. You can also create sub-classes of the pre-defined types. A resource object can also have user-defined attributes, which are used for search and retrieval.

From the Content Manager system perspective, each object is represented by a unique logical identifier, its Uniform Resource Identifier (URI). The library server manages the URI name space. On request, the library server maps URIs onto Uniform Resource Locators (URL). URLs are used to gain access to the physical data. URLs do not point directly to a storage area managed by the resource manager. Instead, the resource manager uses a local name space to convert logical object names to physical file names. Object URIs are created by the specific resource manager. The library server or the end-user can suggest an object URI (its name), but the decision is made by the resource manager.

You can access an object using the Content Manager resource manager APIs (store, retrieve, update, delete, and so forth). In some cases, you can use APIs that are native to the object (stream, multicast, and stage) or file system.

For information about how to work with resource manager objects, see the SResourceItemCreationICM sample in the `samples` directory, `cmbroot\samples\java\icm`.

Managing documents in Content Manager

Content Manager implements a flexible document management data model that you can use for managing business objects. The basic elements of the data model includes folders, documents, and objects.

As mentioned earlier, documents, folders, and other objects are all represented by items in the Content Manager system. One difference between a folder and a document, however, is the semantic type attribute. A document is comprised of attributes, or metadata, that describe the document, including single valued attributes (document name, date, subject), multi-valued attributes (keywords), and collections of multi-valued attributes (address, consisting of street, city, state, and zip).

The document management data model uses a child component to associate objects (resource items) with the document. This model supports more than one part to construct a document. For example, each page could be a separate part. In order for an application to determine the order of the parts that make up a document, a part number is stored in the child component. The child component contains a pointer to the object (a reference attribute) which contains other information about the part such as MIME type, size, the resource manager ID which contains the part, the collection name on that resource manager and so forth. Every object can have different attributes. For example, an annotation might have X and Y coordinates, while a note log might have the CCSID of the text of the note.

To help you understand the document management data model, consider the following scenario of a user who imports a document using a client application:

- A window is displayed to the user.
- The user enters (or selects) the name of the file to import into the system. For example, a police report.
- The user selects the type of document (memo, claim, design).
- A new window opens, where the user can enter attributes that describe the document. The date, a claim number, and insurance policy number, for example.
- The user defines a child component and enters some values for the attributes. The police report is a child component of a claim, for example.
- The user finishes entering the document descriptions and completes the task. The police report is created.

Using either the APIs or the JavaBeans, the client application then connects to the library server and the resource manager. The system creates two items (a non-resource item and a resource item) to store the document. Two items are created because the police report contains a photograph, which is stored in the resource manager. The document is created with a single API call. The object is then stored in the resource manager and the resource manager returns the timestamp, and other metadata for the object. The resource manager creates a reference attribute for the object, and inserts the reference attribute into the child component of the document. A final call to the library server is made to store the child component and to update the attributes. The entire process is bound by a transaction so that any API failure does not result in a partially created document.

After you create a document, you can update it. You can perform two types of updates: change the metadata or change the content. The library server automatically creates a new item record with the next version number (if the item is version-enabled) and copies all of the child components associated with the item.

Creating the document management data model

This section helps you complete the main tasks associated with the document management data model:

- Creating a document item type
- Create a document
- Updating a document
- Retrieving and deleting a document
- Versioning of parts in the document management data model

Creating a document item type

1. Create a document ItemType with ItemType classification = 3, set the following.

```
docItemTypeDef.setVersionControl
    ((short)DKConstantICM.DK_ICM_VERSION_CONTROL_ALWAYS);
docItemTypeDef.setVersioningType(DKConstantICM.DK_ICM_ITEM_VERSIONING_FULL);
docItemTypeDef.setDeleteRule((short)1);
```

2. Create ItemType relation to add Parts to document. Retrieve EntityDef for each Part.

```
Parttype = (DKItemTypeDefICM) dsDef.retrieveEntity(PartName);
```

3. For each part, create a ItemType Relation and set the values.

```
DKItemTypeRelationDefICM itRel = new DKItemTypeRelationDefICM(ds);
itRel.setTargetItemTypeID(PartName);
itRel.setDefaultRMCode((short)1);
itRel.setDefaultACLCode(1);
itRel.setDefaultCollCode((short)1);
itRel.setDefaultPrefetchCollCode((short)1);
itRel.setVersionControl((short) 0);
```

4. Add the ItemType relation to the Document (Source).

```
docItemTypeDef.addItemTypeRelation(itRel);
```

5. Add the document ItemType to persistent store.

```
docItemTypeDef.add
```

Creating a document

An item with the ″Document″ semantic type can contain attributes (like items of other semantic types) and multiple ″parts″ (unlike items of other semantic types) inside it. The steps below take you through the process of creating an item (based on a pre-defined document item type) that contains one attribute and one ″part″. Note that in the steps below, it is assumed that an item type called ″s_simple,″ with one attribute, called ″S_varchar, ″ and one part (″ICMBASE″) has already been defined.

1. Create the document DDO.

```
DKDDO ddoDocument = dsICM.createDDO("S_simple",
       DKConstant.DK_CM_DOCUMENT);
short    dataId   = 0;
String attrValue  = "Test";
```

2. Set the document's attribute. In this case, we assume that the item type has only one attribute.

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
DKParts  parts    = null;
// Document's parts
```

3. Retrieve the document's parts collection.

```
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
     DKConstantICM.DK_CM_DKPARTS);
if (dataId == 0) {
    dataId = ddoDocument.addData(DKConstant.DK_CM_NAMESPACE_ATTR,
       DKConstantICM.DK_CM_DKPARTS);
    parts = new DKParts();
    ddoDocument.setData(dataId, parts);
}
else
{
    parts = (DKParts)ddoDocument.getData(dataId);
    if (parts == null)
    {
        parts = new DKParts();
        ddoDocument.setData(dataId, parts);
    }
}
```

4. Create a part of pre-defined type ″ICMBASE″. This part will be added to the created document. It is assumed that the document created below is based on an item type with only one part.

```
DKLobICM pLobPart = (DKLobICM) dsICM.createDDO("ICMBASE",
   DKConstant.DK_CM_RESOURCE);
pLobPart.setPartNumber(1);
// Set the mime type for added part
pLobPart.setMimeType("text/plain");
String partValue = "This is a base part";
pLobPart.setContent(partValue.getBytes());
```

5. Add the created part to the ″parts″ collection. Note that this is a deferred save (the change is not committed to the datastore until the document DDO is persisted).

```
parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
```

6. Persist document to the datastore.

```
ddoDocument.add();
```

Updating a document

The steps below take you through the process of updating an item of semantic type ″Document.″ In the steps below, a new part is added and the attribute value is updated.

1. Retrieve the datastore definition object.

```
DKDatastoreDefICM dsDef = (DKDatastoreDefICM )dsICM.datastoreDef();
String itemTypeName = "S_simple";
```

2. Retrieve the item type for the document item that is to be updated.

```
DKItemTypeDefICM itemType = (DKItemTypeDefICM)
  dsDef.retrieveEntity(itemTypeName);
```

3. Update the attribute value for the document item.

```
String attrValue = "New Value";
short dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"S_varchar");
ddoDocument.setData(dataId,attrValue);
```

4. Retrieve the document's parts collection.

```
DKParts  parts      = null;
// Document's parts
dataId = ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
    DKConstantICM.DK_CM_DKPARTS);
if (dataId == 0) {
    dataId = ddoDocument.addData(DKConstant.DK_CM_NAMESPACE_ATTR,
      DKConstantICM.DK_CM_DKPARTS);
    parts = new DKParts();
    ddoDocument.setData(dataId, parts);
```

```
}
else
{
    parts = (DKParts)ddoDocument.getData(dataId);
    if (parts == null)
    {
        parts = new DKParts();
        ddoDocument.setData(dataId, parts);
    }
}
```

5. Create data for the new part.

   ```
   String partValue = "This is an annotation";
   ```

6. Create a part of pre-defined type ″ICMANNOTATION″. This part will be added to the created document. Here, it is assumed that the document being created is based on an item type with only one part. Once the new part is added, the document will have two parts.

   ```
   DKLobICM pLobPart = (DKLobICM)dsICM.createDDO("ICMANNOTATION",
      DKConstant.DK_CM_RESOURCE);
   pLobPart.setContent(partValue.getBytes());
   pLobPart.setPartNumber(2);
   ```

7. Add the created part to the ″parts″ collection. Note that this is a deferred save (the change is not committed to the datastore till the document DDO is persisted).

   ```
   parts.addElement((dkDataObjectBase)((DKDDO) pLobPart));
   ```

8. Persist the changed document to the datastore.

   ```
   ddoDocument.update();
   ```

Retrieving and deleting a document

To retrieve a document, call ddo.retrieve(option). If you set option to DK_ICM_CONTENT_YES, the parts TOC list as well as the parts is retrieved. Otherwise, only the TOC list of the parts is retrieved.

To delete a document, call ddo.del(). The document and its attached parts is deleted.

Versioning of parts in the document management data model

Versioning properties of document parts are dependent on the versioning properties of the document. Versioning characteristics of document parts include the following:

- Like regular documents, parts can have one of three versioning models: versioned-always, versioned-never (default) and application-controlled versioning.
- If an item type has a version policy of versioned-never, its parts also have a versioning policy of versioned-never.
- If an item type T has a version-policy of versioned-always and an item I of item type T and you modify (by adding/deleting or updating a part) either its attributes or its parts collection, a new version of item I is created.
- Parts of documents, unlike documents themselves, do not have a maximum number of versions.
- You can obtain part-level versioning rules from the item type relation object for the part of interest (base, note, annotation etc.).

The example below shows how to get the versioning rules for an item type's base part.

```
String itemTypeName="book";  //example item type
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM item =null;
DKDatastoreICM ds = new DKDatastoreICM();
...
item = (DKItemTypeDefICM)ds.datastoreDef.retrieveEntity(itemTypeName;
DKItemTypeRelationDefICM itemRelation =
(DKItemTypeRelationDefICM)item.retrieveItemTypeRelation(partId);
versionControlPolicy = itemTypeRelation.getVersionControl();
```

Working with transactions

Transactions allow Content Manager to maintain consistency between the library server and any adjoining resource manager. A transaction is a user-determined, recoverable, unit of work, that consists of one or more consecutive API calls made through a single connection to the library server. The sequence of consecutive DKDatastoreICM method calls are made either directly or indirectly, through the DDOs and XDOs.

The scope of a transaction and the amount of work within that transaction is by default the work performed by a single API method (implicit transaction). This type of transaction is recommended and is the best performing scope of a transaction. You can, however, change the scope of a unit of work, making it larger to include multiple method calls (explicit transaction), but using this type of transaction can introduce some performance overhead.

When a transaction ends, the entire transaction is either committed or rolled back. If it is committed, all of the Content Manager server changes made by API calls within the transaction are permanent. If a transaction is rolled back or fails, all the changes made within the transaction are reversed during rollback processing.

The commit and rollback of a transaction is done automatically in the case of implicit transactions. In the case where explicit transactions are in use, the transaction commit is controlled by the application, whereas a transaction rollback can be initiated by an application or automatically by the Content Manager system. The Content Manager system initiates a rollback when a severe error occurs or when it is necessary to resolve a deadlock between the library server and the database.

Within a transaction, uncommitted resource manager changes are not visible to the application that made the changes until the transaction is committed. For example, you make changes to a resource manager item and you store it. If you retrieve that item before the transaction is committed, the item will not reflect the changes that you just made. You will not see the updated item until the transaction is committed.

Concurrent or overlapping transactions through a single library server connection are not supported. To maintain concurrent transactions, you must make multiple connections between the library server and the database, or initiate multiple client processes or threads if you are working with a client application. Applications like IBM WebSphere ® Application Server handle processes, connections, and sessions.

The execute() and executeWithCallback() methods in DKDatastoreICM automatically create an additional connection to the database when invoked. The new database connection is then used to execute the query. Since queries use a separate database connection, they also have a separate transaction scope from

other datastore operations. The connection to the database is closed (or returned to the pool, if pooling is enabled) when the DKResultSetCursor is closed.

Things to consider when designing transactions in your application

If a client node or library server fails before the transaction is committed, the database recovery function rolls back the transaction on the library server immediately. The resource manager changes made during the failure are undone immediately if the client node and resource manager are both active. If the client node itself failed, you should put the resource manager through a cycle of the Asynchronous Recovery Utility in order to restore consistency between the resource manager and the library server. Before the utility runs, the servers still have data integrity. What is affected are operations on the in-progress items that had the failure, which will be rejected until the RM is recovered. Failure during in-progress update of an object prevents another update of that same object, until the first failure is reconciled.

If the resource manager fails, you should run the asynchronous recovery utility to remove inconsistencies. On S/390, the resource manager has native transaction capabilities, such as Object Access Method (OAM), which are used to recover more expediently.

Caution when using explicit transactions

For explicit transactions, where you control the transaction scope using DKDatastoreICM.startTransaction() and DKDatastoreICM.commit(), use caution when developing an application where you work with Content Manager Documents with parts and when performing DKLobICM create, retrieve, update, and delete (CRUD) operations. When performing these operations, you should perform CRUD operations as close as possible to the end of the transaction. You should also keep a transaction as short as possible, since a long transaction increases the potential for database locking problems.

Locking problems are most apparent when updating an item, and the application chooses to not commit the transaction immediately. As long as the transaction is not committed, the item that is being updated, is still visible to other applications. When another user attempts to access or view the item, that user is locked out until the update transaction is committed. The same problem (database locking) occurs when creating new items in a folder. If the folder is visible to another user, and that user attempts to retrieve the new item, the user is locked out until the transaction is committed. The amount if time prior to the transaction commit is the amount of time the user is locked out.

The best approach to avoiding database locking is to commit transactions often and to avoid long running transactions. If you must perform CRUD operations within a transaction, it is recommended that you perform these operations when it is understood that no one else will access the items being updated.

Using check-in and check-out in transactions

Content Manager supports check-out and check-in operations on items. The check-out operation is called to acquire a persistent write lock for items. When an item is checked out by a user, other users can not update it although they can still retrieve and view it. You need to call the check-out operation prior to updating or re-indexing an item, regardless of the transaction mode (implicit or explicit) that you use. When you are done with the item, call the check-in operation to release

the persistent lock and make the item available for other users to update. After you create an item, you have the option to keep it in checked out state to prevent other users from changing it until you are completely done with the work. If you check-out (or check-in) an item using an explicit transaction, the checkout is undone if the transaction is rolled back. If you check-out an item using an implicit transaction, the checkout is committed. It is the application's responsibility to check the item back in, using checkin options or methods.

Processing transactions

The transaction scope can be controlled by a client API call, but it must be designed carefully. To group a set of API calls into a transaction, you must build it explicitly by completing the following steps:

1. Call the `startTransaction()` method of the `DKDatastoreICM` class. You work with the `DKDatastoreICM` methods to complete all the transaction steps.
2. Call all of the APIs that you want to include in the transaction in the order that you want them called.
3. Call the `commit` or `rollback` methods to end the transaction.

All of the API calls made between the startTransaction() and either commit() or rollback(), are treated as one transaction.

All APIs can be included in transactions, unless specifically noted. See the Online API Reference for details. Some administrative APIs cannot be included in explicit transactions. For example, the method to define or update item types.

Below is the list of class methods involved in Content Manager transactions in relation to item creation and update.:

DKDatastoreICM.startTransaction()
Starts an explicit transaction.

DKDatastoreICM.commit()
Commits transaction changes to the database.

DKDatastoreICM.rollback()
Rolls back or removes transaction changes from the database.

DKDatastoreICM.checkOut()
Acquires a persistent write lock on an item.

DKDatastoreICM.checkIn()
Releases a previously acquired persistent write lock.

DKDatastoreICM.add()
Creates a new item in the database.

DKDatastoreICM.updateObject
Updates an item. The item must be checked out prior to calling this method.

DKDatastoreICM.retrieveObject
Retrieves an item from the database.

DKDatastoreICM.deleteObject
Deletes an item from the database.

DKDatastoreICM.moveObject
Re-index an item. Moves an item from one item type to another item type. The item must be checked out prior to calling this method.

Routing a document through a process

Content Manager provides an integrated document routing service to help you route documents through a business process. The document routing APIs enable you to build new applications using document routing, or add document routing functionality into your existing applications. Document routing provides you with the following features:

- Synchronization of all items in a document routing process because document routing functions are included in Content Manager transactions.
- Presentation of only the work that the user has access to.
- Single audit trail that includes records for document creation, modification, and routing.
- Optional automatic routing of items. When an item is defined, it can be automatically added to an appropriate process.

Understanding document routing concepts

Before you can work with the document routing APIs, you must understand the document routing concepts and the data model. This information provides you with conceptual and data model information:

Item In the realm of document routing, folders and documents are items.

Work node

A step within a process where items wait for actions to be taken by end users or applications, or a process through which items move automatically. Work lists and collection points are the types of work nodes available. Every work node has a corresponding work list. A work node is an item with a root component and a child component, to implement work packages, that contains the following attributes:

- ID (an entry in the NLS keyword table containing name and description)
- Type (work list, collection point, decision point, split, join)
- Notify time or 0 (in hours)
- Overload limit
- Exits to be run on the server (DLL and function name)
  - Overload exit
  - entering work node exit
  - Leaving work node exit
  - Notify exit

The administrator defines the maximum time that a work package should be at a given work node. When that time is reached, the `notify` flag is set and the Notify exit is called.

Process

A series of steps, defined by an administrator, through which an item is routed. A *process* consists of a root component and a child component which define the route. The root component uses a system defined attribute for *process ID*. The *process ID* maps to a name and description in the NLS Keyword table.

The child component has the following attributes:

- **From** work node (null for the *start* node)
- **To** work Node (null for the“ end” node)

- Menu entry (ID in the NLS keyword table) for routing between **From** and **To**

Work list

One or more work nodes from which a user obtains a list of work packages or the next work package based on priority, or state (suspend or notify). Items are ordered first by priority (or date or another key) and second by work node within the list. Each work list has an ACL. A work node may be in more than one work list. Using a work list allows an administrator, or an application, to dynamically change work assignments without contacting end users. A work list is an item which defines the selection criteria for a list of work and contains the following attributes:

- ID (an entry in the NLS keyword table containing name and description)
- List of work nodes (Atomics)
- Selection order (priority, last work date ascending/descending)
- Selection filter (suspended 0/1, notify 0/1)
- Number return (0=all, 1=system assigned, >1 for a limit)
- SQL statements that implement both the *count* and *retrieve* functions

You can define a work list to span all work lists (or a selected subset) to show work packages in *notify* or *suspend* state. A user authorized to the work list can see the count of waiting work packages, and can open the list and make decisions about each one, such as increasing the priority, forcing early resume, forcing exit from a collection point, changing user assignment, and so forth.

Work package

A set of information such as priority, state, resume time, process, and `ItemID` being routed. End users are not aware of the existence of a work package. A work package is used to associate the item being routed with a work node. A work package is a child component of a work node which associates a document or folder being routed with a work node within a process, and contains work specific information such as priority. Work package attributes include:

- Priority
- Reference attribute to item being routed
- Item ID of the process
- User ID of the process
- User ID of the user who last moved the work package
- Date/time when the work package was last moved
- Suspend state (0/1)
- Notify time (UNIX time in the future to switch to notify state, 0 if not used)
- Resume time (UNIX time, 0 if not suspended for time)
- Resume list
  - Child component with one row per required item type
  - Contain item type (or 0 for *any*) and number of that type required
  - Setting number required to 0 allows continuation when any item of any required type arrives

Collection point

A collection point is created by the administrator and is associated with a work node. A collection point allows automatic suspension of a folder

when it arrives at a work node, unless all the required items have been added to the folder. When the required items arrive in the folder, the work package advances to the next step in the process. End users do not access work packages at a collection point, other than for exception processing such as “forcing” continuation even though required documents have not been received.

A collection resume list is defined in a collection point by the administrator. When the collection resume list is satisfied, the work package on the collection point advances automatically to the next work node. If the collection point has more than one branch, the first branch is selected. The suspend flag is not set by using the collection resume list. The notify flag can be set in a collection point, if the notification time is defined by the administrator. Just like a regular work node, a user can apply continue, terminate, suspend, and resume operations on a collection point.

The notify flag is based on the scheduled time in the DOCROUTINGUPDATE field in the system control table. When the scheduled time is reached, the system updates the field by adding the time that is indicated in the DOCROUTINGFREQ field. The default value for the DOCROUTINGFREQ field is 10 minutes.

Understanding the document routing process

Document routing consists of processes, work nodes, collection points, work lists, and work packages. The system administrator creates the work nodes, collection points, processes, and work lists through the system administration client. A process consists of work nodes and collection points. Each work node and collection point in the process is a separate step in the process. You can create a process that branches out in several directions. The user determines which branch the work node goes to next. The user can choose from a list of possible selections that the system administrator defines. You can define a server exit when you define a work node. You can define server exits for entering a work node, leaving a work node, and to notify the user when the overload limit is reached. When a process is started, a work package is created. The work package is the routing element and contains the attributes of the work. The attributes of the work package consist of the item PID, priority, owner, and so forth.

Collection points are work nodes with additional function. A work package at a collection point node continues to the next work node in the process once the specified number of items of a specified item type exist in the specified folder. Work lists define the work packages assigned to a user. You can have one or more work lists. Each work list can include one or more work nodes. You can specify the order of the work packages in the work list by priority, time, and owner. You can also define the the order of work nodes in the work list.

When you retrieve work lists, you can filter the results to include or exclude suspended work. Suspended work are work packages that are in *notify* state. *Notify* state is when work packages have been at the node for longer than the time specified by the administrator. Remember that a work node can be in more than one work list. The number of packages returned in a work list is defined by the system administrator.

Those basic operations you can perform using document routing include:

- Start a process
- End a process
- Continue a process

- Suspend a process
- Resume a process
- Get work from a work list
- Get the next item from a work list
- Define, update, and delete a process
- Define, update, and delete work node
- Define, update, and delete a work list

Setting up a document routing process

There are nine APIs that you can use to implement document routing functionality into your application. You can find the details about these APIs and methods in the *online API reference*. The nine document routing APIs include:

DKDocRoutingServiceICM
This class provides the methods for managing a process: start, terminate, continue, suspend, and resume.

DKDocRoutingServiceMgmtICM
This class provides the methods to manage the helper classes: DKProcessICM, DKWorkNodeICM, and DKWorkListICM. You can access the DKDocRoutingServiceMgmtICM object from the DKDocRoutingServiceICM object.

DKProcessICM
This class represents a process in the library server.

DKWorkNodeICM
This class represents a work node in the library server.

DKWorkListICM
This class represents a work list in the library server.

DKRouteListEntryICM
This class defines the route that a process can take (from, to). A process object (that is, DKProcessICM) contains a collection of route entry objects (that is, DKRouteListEntryICM).

DKCollectionResumeListEntryICM
This class represents an entry in the resume list for the work nodes. A work node can contain a collection of DKCollectionResumeListEntryICM objects.

DKWorkPackageICM
This class represents a work package in the library server. When a process is started, a work package is created.

DKResumeListEntryICM
This class represents a resume list. A work package can contain a collection of resume lists.

Creating document routing service objects

The example below shows you how to create a document routing object.

```
String database = "ICMNLSDB";
String userName = "icmadmin";
String password = "password";
 // Create new datastore object.
DKDatastoreICM dsICM = new DKDatastoreICM();
// Connect to the datastore.
dsICM.connect(database,userName,password,"");
//The DKDocRoutingServiceMgmtICM object is a helper class that provides
```

```
// methods to manage DKProcessICM, DKWorkNodeICM, and DKWorkListICM
// and the meta-data required to define these objects
DKDocRoutingServiceMgmtICM routingMgmt = new
        DKDocRoutingServiceMgmtICM(dsICM);

//The DKDocRoutingServiceICM class provides the core routing services
//like starting, terminating, continuing, suspending, and resuming a process.

DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);
```

Defining a new regular work node

A work node is a step in a document routing process definition. It may be exited by the user application at any time. The user application is responsible for validating its own exit criteria.

```
// Create new Work Node Object.
 DKWorkNodeICM workNode1 = new DKWorkNodeICM();
//Choose a Name that is 15 characters or less length
workNode1.setName("S_fillClaim");
//Choose a Description for more information than the name.
workNode1.setDescription("Claimant Fills Out Claim");
// Sets the value of the maximum time that an item can spend at this
// work node (in minutes).
workNode1.setTimeLimit(100);
// Specify max number of work packages that can be at this node
//at any given time
workNode1.setOverloadLimit(200);
// Set the type to be a regular work node
workNode1.setType(0);

//Add the new work node definition to the document routing
//managment object
routingMgmt.add(workNode1);
```

Listing work nodes

The listWorkNodeNames method lists all work node names in the library server, and the listWorkNodes method returns a collection of DKWorkNodeICM objects representing a work node in the library server.

```
// Obtain the document routing  management object.
 // Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt = new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain all Work Nodes in the System.
dkCollection workNodes = routingMgmt.listWorkNodes();
System.out.println("Work Nodes in System:  ("+workNodes.cardinality()+")");
dkIterator iter = workNodes.createIterator();
while(iter.more())
{
        DKWorkNodeICM workNode = (DKWorkNodeICM) iter.next();
        if(workNode.getType()==0)
               System.out.println("  Normal Node - "+workNode.getName()+":
                 "+workNode.getDescription());
        else
                System.out.println(" Collection Pt - "+workNode.getName()+":
                  "+workNode.getDescription());
}
```

Defining a new collection point

A collection point is a work node with system-defined exit criteria, specifically applicable to routing folders. A set of requirements can be specified that must be met before the process can resume (advance) past this point.

```
// Create a new Work Node Object. This will be
//the collection point
DKWorkNodeICM collectionPoint = new DKWorkNodeICM();
```

```
// Choose a Name with 15 characters or less.
collectionPoint.setName("S_gatherAll");

// Choose a Description for more information than the name.
collectionPoint.setDescription("Gather Claim, Police Report, Policy, & Photos");

// Sets the value of the maximum time that an item can spend at this
// work node (in minutes).
 collectionPoint.setTimeLimit(100);

// Specify max number of work packages that can be at this node.
collectionPoint.setOverloadLimit(200);

// Set the type of node to be a collection point.
collectionPoint.setType(1);

// Create the "Resume" List, which is the list of document types
//that the process must  wait for before moving on to the next node.
//A list will be created to  hold "resume entries" which are descriptions
//of requirements that must be met before the process can move on.

// Create a List/Collection to hold all Resume Entries.
dkCollection resumeList = new DKSequentialCollection();

// Create as many requirements, or "Resume List Entries", which
//specify what Item Types it must wait for.  The process cannot
//pass this collection point unless the specified number of Item
//(DDO) of the specified Item Type reaches this collection point.

DKCollectionResumeListEntryICM resumeRequirement = new
     DKCollectionResumeListEntryICM();
// Set the Item Type Name Folder Item that is being routed.
resumeRequirement.setFolderItemTypeName("S_simple");

// Make the collection wait for an Item of the specified Item Type
//to be added to the folder before proceeding.
resumeRequirement.setRequiredItemTypeName("S_autoClaim");

// Specify the number of Items of the specified Item Type that it
//must wait for.
resumeRequirement.setQuantityNeeded(1);

// Add the requirement (Entry) to the List of Requirements (Resume List).
resumeList.addElement(resumeRequirement);

resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement.setFolderItemTypeName("S_simple");
resumeRequirement.setRequiredItemTypeName("S_policeReport");
resumeRequirement.setQuantityNeeded(1);

// When all requirements (resume list entries) have been added to the
//list of requirements (resume list), set the resume list in the
//collection point.
collectionPoint.setCollectionResumeList(resumeList);

// Add the new collection point definition to the document routing
// managment object
routingMgmt.add(collectionPoint);
```

Defining a work list

A work list consists of one or more work nodes from which a user obtains a list of work packages or the ″next″ work package. A work node can be in more than one work list. Using a work list allows an administrator or a user application to dynamically change work assignments without contacting end users.

```
// Create a new work list.

 DKWorkListICM workList = new DKWorkListICM();
// Choose a name of 15 characters or less.
workList.setName("S_fillClaimWL");

workList.setDescription("Work List Covering Fill/Submit Claim Work Node.");
//Specify that work packages returned by the work list will be sorted by time
 workList.setSelectionOrder(DKConstantICM.DK_ICM_DR_SELECTION_ORDER_TIME);

//Specify that the work packages returned will be the one that are not
// in the suspend state
workList.setSelectionFilterOnSuspend
    (DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that work packages returned are not the ones in the notify state
workList.setSelectionFilterOnNotify
     (DKConstantICM.DK_ICM_DR_SELECTION_FILTER_NO);

// Specify that at most 100 work packages should be listed in
// this work list
workList.setMaxResult(100);

String[] wnNames = {"S_fillClaim"};

workList.setWorkNodeNames(wnNames);

//Add the new work list definition to the document routing
//management object
routingMgmt.add(workList);
```

Listing work lists

The listWorkListNames method lists all work list names in the library server, and the listWorkLists method returns a collection of DKWorkListICM objects representing a work list in the library server.

```
// Obtain the document routing management object.
// Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt = new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain all Work Lists in the System.
dkCollection workLists = routingMgmt.listWorkLists();
System.out.println("Work Lists in System:  ("+workLists.cardinality()+")");
dkIterator iter = workLists.createIterator();
while(iter.more())
      {
           DKWorkListICM workList = (DKWorkListICM) iter.next();
           System.out.println("     - "+workList.getName()+":
              "+workList.getDescription());
       }
```

Defining a new process and associated route

A document routing process is the defined routes that a work package being routed will follow. Multiple routing processes can re-use the same nodes and multiple routes between nodes can be used.

```
// Create a new Process Definition
DKProcessICM process = new DKProcessICM();
process.setName("S_claimProcess");
process.setDescription("Process for an Insurance Claim");

// Define all possible Routes.

// Create a list of all possible routes between nodes.
dkCollection routes = new DKSequentialCollection();

// Connect the Work Nodes using Route List Entries.  A simple route
//between two work nodes is specified by associating a 'From' work
```

```
//node and a 'To' work node.

// A Route List Entry simply connects two nodes with an implied direction.
// Multiple routes may exist between nodes.  A specific route may be selected
// by a user-defined "selection" keyword.  Examples might be "Continue", "Go",
// "Accept", "Reject", "Complete", etc.

// Create a new connection between two nodes.
DKRouteListEntryICM nodeRoute = new DKRouteListEntryICM();

// Every process must start with the start node.
nodeRoute.setFrom(DKConstantICM.DK_ICM_DR_START_NODE);
nodeRoute.setTo("S_fillClaim");
// Choose any user-defined name for an action that will make the
//transition from the 1st node to the second
nodeRoute.setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_fillClaim");
nodeRoute.setTo("S_gatherAll");

// Choose any user-defined name for an action that will make the
// transition take place.
nodeRoute.setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute.setFrom("S_gatherAll");
nodeRoute.setTo(DKConstantICM.DK_ICM_DR_END_NODE);

// Choose any user-defined name for an action that will make
//the transition take place.
nodeRoute.setSelection("Complete");

// Add the individual route to the collection of all possible routes.
routes.addElement(nodeRoute);

// Set the route in the process.
process.setRoute(routes);

// Add the process to the routing Management.
routingMgmt.add(process);
```

Starting a document routing process

The example below shows you how to start a document routing process. **Example**:

```
//First create a document or folder that will be routed.
//An item type of name "s_simple" must be pre-defined before a
// DDO of that name can be created.

DKDDO ddoFolder = dsICM.createDDO("S_simple", DKConstant.DK_CM_FOLDER);

// Save the created folder to the persistent datastore.
ddoFolder.add();


//Create the core document routing service object.
DKDocRoutingServiceICM routingService = new DKDocRoutingServiceICM(dsICM);

//Start a process with the name "S_claimProcess" (which must be pre-defined.
```

```
//The PID string of the work package that will be routed is returned by this call.
String workPackagePidStr = routingService.startProcess
    ("S_claimProcess", ddoFolder.getPidObject().pidString(),1,"icmadmin");
```

Ending a process

A process can be explicitly terminated before it reaches the end node. The work package being routed will be removed from the system.

Example:

```
//Terminating the process requires the PID string of the work package
//being routed by this process instance
routingService.terminateProcess(workPackagePidStr);
```

Continuing a process

The continueProcess() method routes the item referenced by the item PID in the specified work package from the current work node to the next work node that is determined by the selection. The specified work package is removed from the library server, and a new work package is created for the specified owner. The item referenced by the item PID is checked in, if it has been checked out. The PID of the new work package is returned. A null is returned if the process has ended.

```
//In this code snippet, the name of the selection that will cause
//the transition from the current work
//node to the next work node is "Continue". The work package PID string
// of the current work package is specified in the method call
//This method call returns the PID string of the new work package
workPackagePidStr = routingService.continueProcess
     (workPackagePidStr, "Continue", "icmadmin");
```

Suspending a process

An instance of a document routing process can be suspended for a period of time (in minutes) or pending a set of requirements that must be met before it is resumed. This does not relate to processes and threads in a programming environment. The thread or process in the C++ runtime environment will not be stopped.

```
dkCollection  requirements = new DKSequentialCollection();

//Process will be suspended for 2 minutes.
routingService.suspendProcess(workPackagePidStr, 2, requirements);
```

Resuming a process

A suspended process (process in the suspended state) can be resumed explicitly (or implictly after expiration of a specified period of time or after some defined requirements have been met). This takes the process out of the suspended state, resuming it to normal operation.The resumeProcess method will reset the suspend flag of the specified work package to false before the suspension reaches the specified duration and the resume list is satisfied. No routing or checkout of the associate work item will be performed.

```
routingService.resumeProcess(workPackagePidStr);
```

Listing document routing processes

The following example shows you how to list document routing processes.

```
//The listProcessNames method lists all process names in the
//Library Server, and the listProcesses method returns a collection
//of DKProcessICM objects representing a process in the Library Server.

// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM routingMgmt = new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the list of all running document routing processes
```

```
 // Obtain the Routing Management object.
DKDocRoutingServiceMgmtICM routingMgmt = new DKDocRoutingServiceMgmtICM(dsICM);
// Obtain list of all routing processes running.
dkCollection processes = routingMgmt.listProcesses();
System.out.println("Running Processes:  ("+processes.cardinality()+")");

dkIterator iter = processes.createIterator();
while(iter.more())
{
// Move pointer to next element and obtain that next element.
DKProcessICM proc = (DKProcessICM) iter.next();
     System.out.println(" - "+proc.getName()+": "+proc.getDescription());
}
```

Retrieving work package information

When an instance of a document routing process is in progress, a work package is the vehicle through which an item (instance of an item type) moves along through the routing process. A work package contains all necessary information about the process and the item that it is transporting. The work package is the object that the user's application will use and manipulate as required.

The retrieveWorkPackage method will return the DKWorkPackageICM object referenced by the specified work package PID (wpPidStringStr).

```
//Use an established document routing service
//Specifying false in this method call makes sure that the work package
// is not checked out
DKWorkPackageICM workPackage =
        routingService.retrieveWorkPackage(workPackagePidStr,false);
System.out.println("----------------------------------------");
System.out.println("               Work Package");
System.out.println("----------------------------------------");
System.out.println(" Process Name:  " + workPackage.getProcessName());
System.out.println("  work Node Name:  " + workPackage.getWorkNodeName());
System.out.println("           Owner:  " + workPackage.getOwner());
System.out.println("        Priority:  " + workPackage.getPriority());
System.out.println(" User Last Moved:  " + workPackage.getUserLastMoved());
System.out.println(" Time Last Moved:  " + workPackage.getTimeLastMoved());
System.out.println("   Suspend State:  " + workPackage.getSuspendState());
System.out.println("    Notify State:  " + workPackage.getNotifyState());
System.out.println("     Notify Time:  " + workPackage.getNotifyTime());
System.out.println("     Resume Time:  " + workPackage.getResumeTime());
System.out.println("Work Package Pid:  " + workPackage.getPidString());
System.out.println("     Item Pid:  " + workPackage.getItemPidString());
```

Ad hoc routing

Below is an ad hoc routing example procedure. In the example, the system administration client is used to set up the work nodes, processes, and work lists.

1. Create two work nodes, N1 and N2 for example.
2. Create two one-node processes, P1 and P2 such that P1 has one work node, N1 and P2 has one work node, N2.

   P1 looks like this:

From:	Action:	To:
START	Continue	N1
N1	Continue	END

P2 looks like this:

From:	Action:	To:
START	Continue	N2
N2	Continue	END

3. Create two work lists, WL1 and WL2 such that WL1 has one work node, N1 and WL2 has one work node, N2.

At run time, complete the following steps to implement ad-hoc routing:

1. Start process P1 with a document PID (Example, ABC). A work package, WP1, is created. The work list WL1 displays the work package WP1 at work node N1.
2. To move the document ABC from process P1 to process P2, terminate work package WP1 and start process P2 with the same document (ABC). Work package WP2 is created.

The work list WL2 shows the work package WP2 at work node N2.

Document routing example queries

This sections contains example queries. For more information about writing queries, see “Understanding the query language” on page 98.

Example 1

Finds car documents, and returns only the associated workpackages that are active.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@SUSPENDFLAG =
0]
```

Example 2

Finds car documents and returns the associated workpackages that are in the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]
```

Example 3

Finds car documents where the name is ″Honda″, and returns the associated workpackages that are in the ″AccidentInvesigation″ process.

```
/Car[@Name = "Honda" AND @VERSIONID = latest-version(.) AND
@SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER =>
WORKPACKAGE[@PROCESSITEMID = /ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]
```

Example 4

Finds car documents and returns the associated workpackages that are in the ″UnderReview″ step of the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]/@ITEMID
AND ../@WORKNODENAME =  "UnderReview"]
```

Example 5

Finds car documents and returns only the suspended workpackages that are in the ″UnderReview″ step of the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]
/@ITEMID AND ../@WORKNODENAME = "UnderReview" AND
@SUSPENDFLAG = 1]
```

Granting privileges for document routing

In order for a user to perform document routing operations, the user must have the appropriate privileges. The privileges associated with document routing are listed in the following table. The general privileges for items are applicable to processes, work nodes, and work lists.

Table 14. Document routing privileges

Privilege	Description	Related API
ICM_PRIV_ITEM _UPDATE_WORK	Used to see if the user is authorized to do the following for a work package: set the priority set the owner set the resume list set the duration for suspension	suspendProcess resumeProcess setWorkPackagePriority setWorkPackageOwner
ICM_PRIV_ITEM _ROUTE_START	Used to see if the user is authorized to start a process.	startProcess
ICM_PRIV_ITEM _ROUTE_END	Used to see if the user is authorized to terminate a process.	terminateProcess
ICM_PRIV_ITEM _GET_WORKLIST	Used to see if the user is authorized to get the count or work packages from a work list.	getCount listWorkPackagePidStrings
ICM_PRIV_ITEM _GET_WORK	Used to see if the user is authorized to get a work package.	getNextWorkPackagePidString getNextWorkPackage checkOutItemInWorkPackage retrieveWorkPackage
ICM_PRIV_ITEM _GET_ASGN_WORK	Used to see if the user is authorized to get a work package that is owned by a different a different user.	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings
ICM_PRIV_ITEM _ROUTE	Used to see if the user is authorized to route a work package.	continueProcess

Working with access control lists for document routing

When an ACL is defined for a document routing entity such as a process, work node, and work list, the operations allowed on the entity are impacted. The effect of ACLs on Content Manager document routing entity and their associated privileges are listed in the following table.

Table 15. Access control lists and document routing

Privilege	Description	Related API
Process	startProcess	ICM_PRIV_ITEM_ROUTE_START

Table 15. Access control lists and document routing (continued)

Privilege	Description	Related API
Work node	continueProcess suspendProcess resumeProcessterminateProcess setWorkPackagePriority setWorkPackageOwner	ICM_PRIV_ITEM_ROUTE ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_ROUTE_END ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_UPDATE_WORK
Work list	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings checkOutItemInWorkPackage	ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORKLIST ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORKLIST ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORK

Document routing constants

You define document routing constants in DKConstantICM. Following is the list of document routing constants:

```
public final static int DK_ICM_DR_SELECTION_FILTER_NO = 0;
public final static int DK_ICM_DR_SELECTION_FILTER_YES = 1;
public final static int DK_ICM_DR_SELECTION_FILTER_EITHER = 2;
public final static int DK_ICM_DR_SELECTION_ORDER_PRIORITY = 0;
public final static int DK_ICM_DR_SELECTION_ORDER_TIME = 1;
public final static int DK_ICM_DR_MAX_RESULT_ALL = 0;
public final static String DK_ICM_DR_START_NODE = ″START″;
public final static String DK_ICM_DR_END_NODE = ″END″;
```

Chapter 6. Working with other content servers in Java

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 DKDatastore*xx* classes, where *xx* is the suffix of the specific content server, to create a datastore. Table 16 shows these classes.

Table 16. Server type and class name terminology

Content server	Class name
Content Manager	DKDatastoreICM
earlier Content Manager	DKDatastoreDL
Content Manager OnDemand	DKDatastoreOD
Content Manager for AS/400 (VisualInfo for AS/400)	DKDatastoreV4
Content Manager ImagePlus for OS/390	DKDatastoreIP
Domino.Doc	DKDatastoreDD
Extended Search	DKDatastoreDES
Panagon Image Services (FileNET)	DKDatastoreFN
Relational databases	DKDatastoreDB2, DKDatastoreJDBC (for Java) DKDatastoreDJ
DB2 Warehouse Manager Information Catalog Manager (Information Catalog)	DKDatastoreIC

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 14:

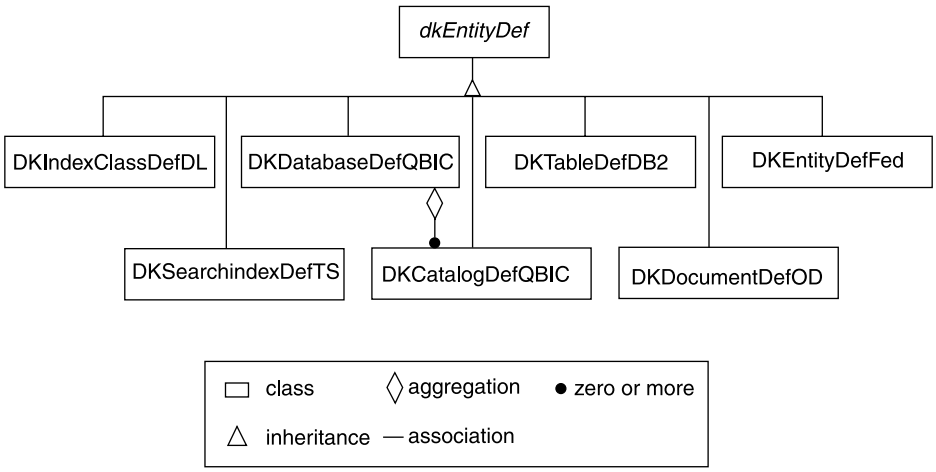

Figure 14. Class hierarchy

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 15 on page 143:

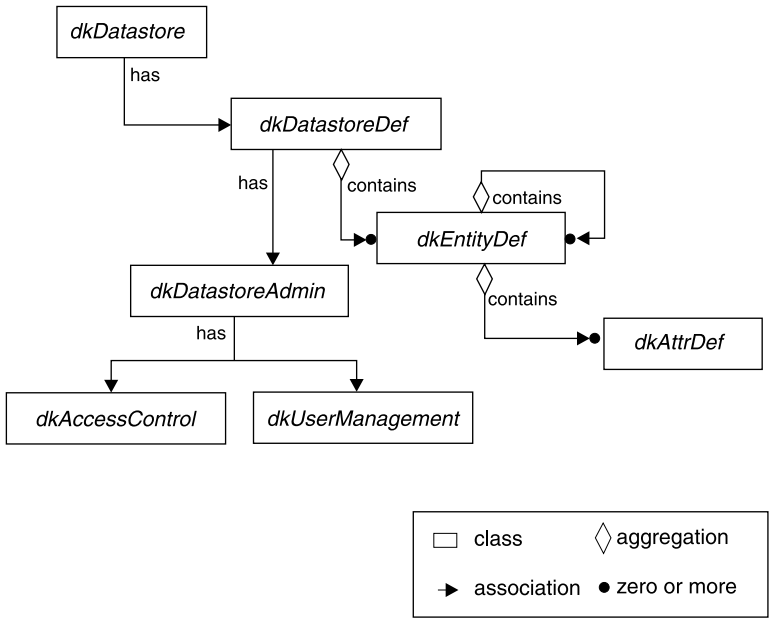

Figure 15. Data definition class hierarchy

For more information on dkDatastore and other common classes, see "Developing custom content server connectors" on page 211.

Working with earlier Content Manager

This section describes how to access data in Content Manager servers, and how to perform the following tasks:

- Handle large objects
- Use DDOs
- Use XDOs in a search engine
- Use combined query
- Use Text Search Engine
- Use image search (QBIC)
- Use workflows and workbaskets

Handling large objects

When working with large objects in earlier 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.

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.

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.

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”, and “Retrieving a document or folder” on page 148.

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 index class
pid.setPrimaryId("");               // make the item ID blank
pid.setDatastoreName("LIBSRVRN");   // set the new datastore name
ddo.setPidObject(pid);              // update the PID
ddo.setDatastore(dsN);              // re-associate the DDO with dsN
ddo.add();                          // add the DDO
```

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 145 for more information.

Retrieving a document or folder

To retrieve a document from a DKDatastoreDL representing an earlier Content Manager content server, you must know 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.

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

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 an earlier Content Manager server.

Use a DKDatastoreTS object to represent the Text Search Engine. Text Search Engine does not actually store the data, it merely indexes the data stored in earlier Content Manager to support a text search on them. The result of a text search is an item identifier describing the location of the document in Content Manager. Use these identifiers to retrieve the document.

The DKDatastoreTS object does not support add, update, retrieve, and delete methods. You can query this datastore. Refer to “Loading data to be indexed by Text Search Engine” on page 155 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 ({ƀ}). 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 without the other.

Representing Text Search Engine information using DDOs

You use a DDO associated with a DKDatastoreTS object to represent the results from text searches.

DKDastastoreTS does not have a property item type as a DKDatastoreDL object. 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\dl 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_CCSID_00850);
Integer output_option = null;

dsTS.setOption(DK_TS_OPT_CCSID, input_option);
output_option = (Integer) dsTS.getOption(DK_OPT_TS_CCSID);
```

The ouput_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_CCSID 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 151.
- 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\dl 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 is 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\dl directory.

Indexing XDOs by search engine

Before you can search data items 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 items 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 system administration client and a search index named TMINDEX associated with it, the value for the SearchIndex is TM-TMINDEX.

For an object that is to be indexed by Text Search Engine, the value of SearchEngine must be SM, for a data item to be indexed by query by image search, the value of SearchEngine must be QBIC (for more on image search, see “Understanding image search terms and concepts” on page 160).

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

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\dl 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("SAMPTITLE");
```

```
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPCORPBODY");
docSection2.setTag("BODY");
docModel.addDocSection(docSection2);

dsTS.connect("TMMUF","","","");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

// ----- Create the document model
dsAdmin.createDocModel("",docModel);

dsTS.disconnect();
dsTS.destroy();
```

Refer to TCreateDocModelTS.java and TCreateStructDocIndexTS.java in the CMBROOT\Samples\java\dl directory for more examples.

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("SAMPTITLE");
docSection.setTag("TITLE");
docModel.addDocSection(docSection);
docSection2.setName("SAMPCORPBODY");
docSection2.setTag("BODY");
docModel.addDocSection(docSection2);

// ----- Connect to the datastore
dsTS.connect("TMMUF","","","");

dsDef = (DKDatastoreDefTS)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS)dsDef.datastoreAdmin();

DKSearchIndexDefTS pEnt = new DKSearchIndexDefTS((dkDatastore)dsTS);
pEnt.setName("TSTRUCT");
pEnt.setIndexType(DK_TS_INDEX_TYPE_PRECISE);
pEnt.setIndexProperty(DK_TS_PROPERTY_SECTIONS_ENABLED);
pEnt.setLibraryId("LIBSUM");
pEnt.setLibraryClientServices("IMLLSCDL");
pEnt.setLibraryServerServices("IMLLSSDL");
String strIndexFileDir = "e:\\tsindex\\index\\tstruct";
// ----- For AIX us the following form for the file
```

```
//    String strIndexFileDir = "/home/cltadmin/tsindex/tstruct";
pEnt.setIndexDataArea(strIndexFileDir);
String strWorkFileDir =  "e:\\tsindex\\work\\tstruct";
// ----- For AIX us the following form for the file
//     String strWorkFileDir = "/home/cltadmin/work/tstruct";
pEnt.setIndexWorkArea(strWorkFileDir);

// ----- Associate document model with index
pEnt.addDocModel(docModel);

// ----- Create text search index that supports sections
dsDef.add((dkEntityDef)pEnt);

indexRules.setIndexName("TSTRUCT");
indexRules.setDefaultDocFormat(DK_TS_DOCFMT_HTML);
indexRules.setDefaultDocModel(docModel2);

dsAdmin.setIndexingRules(indexRules);

dsTS.disconnect();
dsTS.destroy();
```

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

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\dl directory.

Refer to TCheckStatusTS.java in the CMBROOT\Samples\java\dl 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;
DKNVPair parms[] = null;
// ----- Connect
dsTS.connect("TMMUF","","","");
// ----- Generate the query string
String cmd = "SEARCH=(COND=($CCSID=850," +
             "DOCMOD=(DOCMODNAME=SAMPCORPMOD," +
             "SECLIST=(SAMPCORPTITLE,SAMPCORPBODY))$ Corp));" +
             "OPTION=(SEARCH_INDEX=TMSTRUCT;MAX_RESULTS=5)";
// ----- Execute the query
pCur = dsTS.execute(cmd,DK_CM_TEXT_QL_TYPE,parms);

// ----- Process the results
.....
dsTS.disconnect();
dsTS.destroy();
```

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

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 16 on page 160 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.

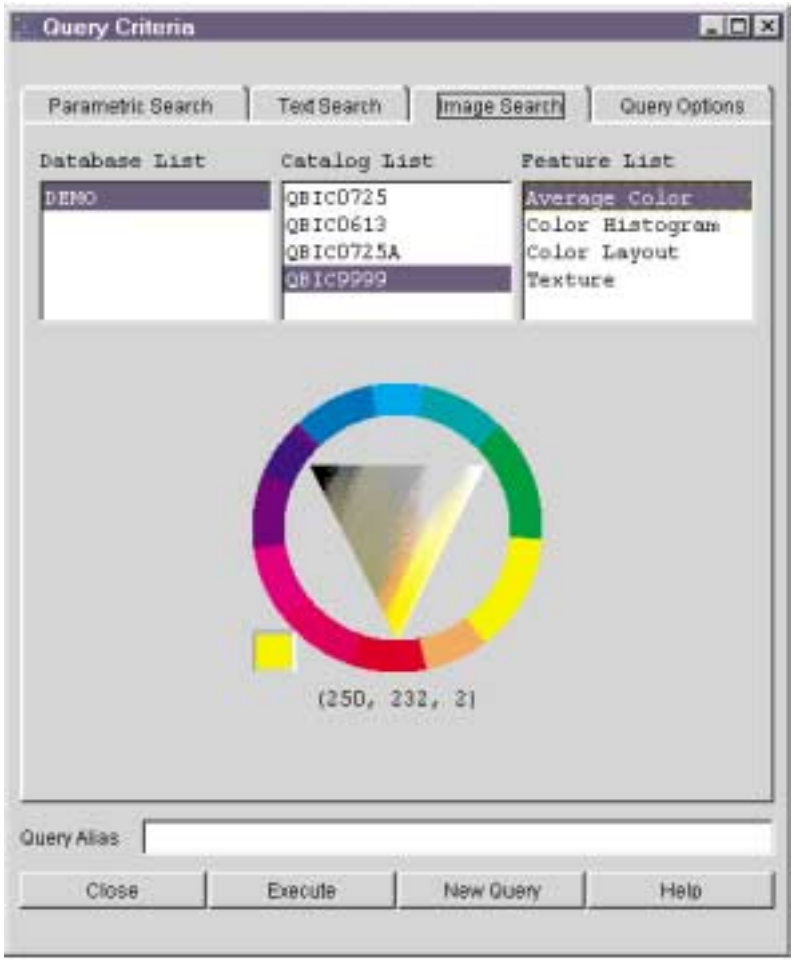

Figure 16. Image search sample client

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 earlier Content Manager.

Understanding image search servers, datastores, and catalogs: Earlier 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 17 shows an example of an image search server.

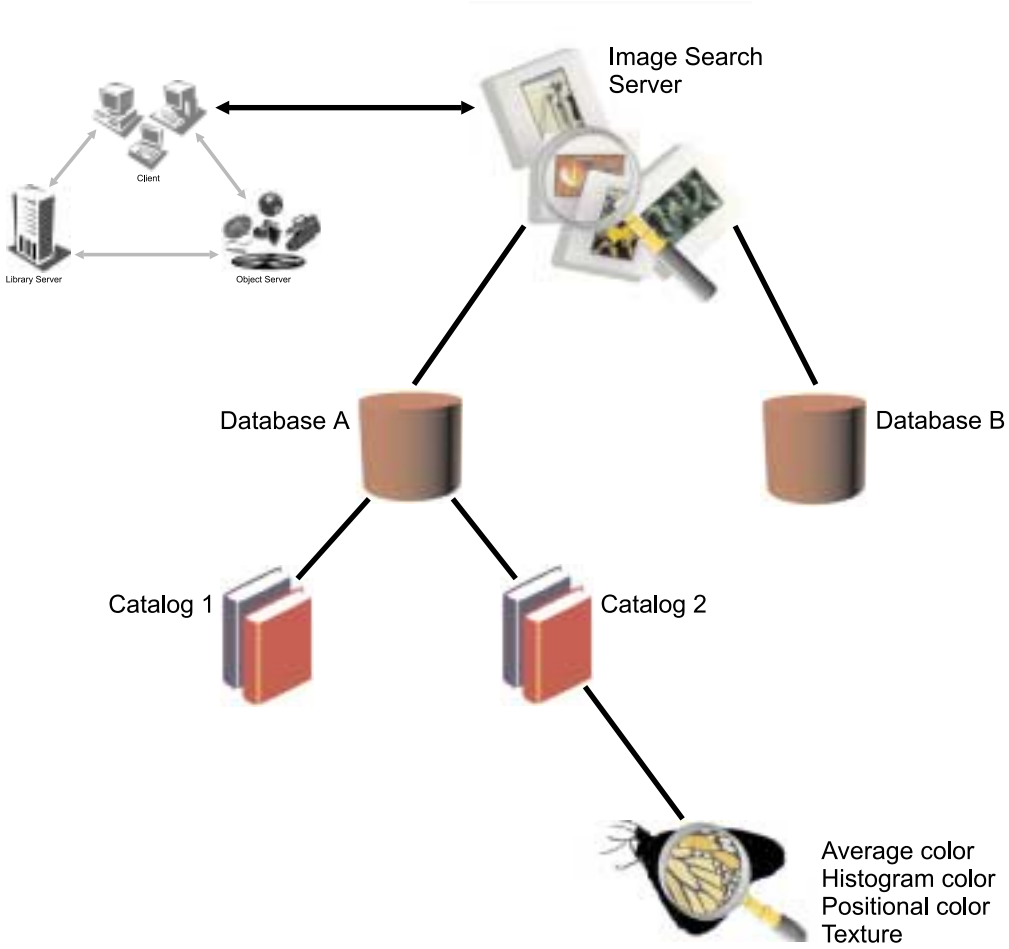

Figure 17. An image search server in a Content Manager system. The image search server communicates with the other earlier Content Manager components through the clients.

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% green, and 40% blue.

```
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% green, and 40% blue:

```
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 18 shows an example of the client and retrieve images.

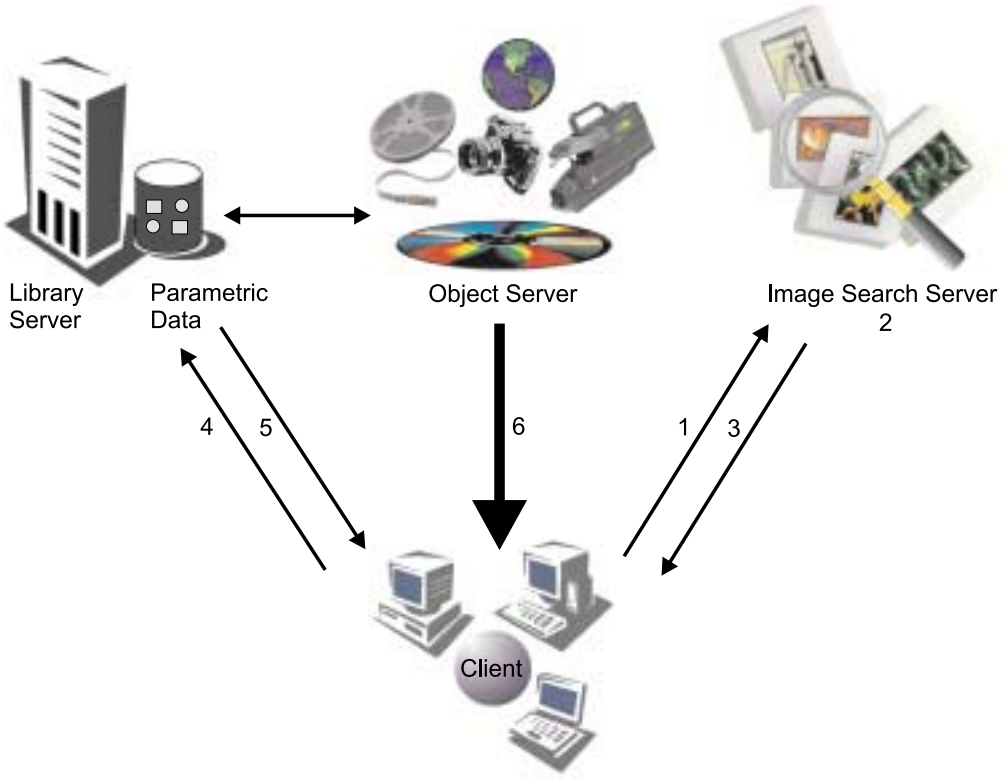

Figure 18. How image search clients search and retrieve images

To perform an image search:

1. A client builds a QBIC query string and sends it to an image search server.

2. Image search server receives the query string and searches the cataloged images for matches.
3. Client receives the matches as a list of identifiers. The identifier for each matching image consists of the item ID, part number, RepType, 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 17 describes image search query feature names and values.

Table 17. Image search query valid feature names and values

Feature name	Values
QbColorFeatureClass or QbColor	**color = < rgbValue , rgbValue , rgbValue >** where rgbValue is an integer from 0 to 255. **file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.
QbColorHistogramFeatureClass or QbHistogram	**histogram = < histList >** where histList consists of one or more `histClause` separated by a comma (,). A `histClause` is specified as (`histValue`, `rgbValue` , `rgbValue` , `rgbValue`), where `histValue` is an integer from 1 to 100 (a percentage value), and `rgbValue` is an integer from 0 to 255. **file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

Table 17. Image search query valid feature names and values (continued)

Feature name	Values
QbDrawFeatureClass or QbDraw	**description = < ″ descString ″ >** where descString is a special encoded string describing a picker file. Format of the description string: 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. **file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.
QbTextureFeatureClass or QbTexture	**file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

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 18 on page 166 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 18 describes feature names and values.

Table 18. Image search query: valid feature values

Feature name	Values
QbColorFeatureClass or QbColor	**color = < rgbValue , rgbValue , rgbValue >** where rgbValue is an integer from 0 to 255. **file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides. For example, you can search using an average color and a texture value. The texture value is provided by an image in a client file. The weight of the texture is twice that of the average color: `QbColorFeatureClass color= <50, 50, 50> and QbTextureFeatureClass file=<client, "\patterns\pattern1.gif"> weight=2.0`
QbColorHistogramFeatureClass or QbHistogram	**histogram = < histList >** where histList consists of one or more `histClause` separated by a comma (,). A `histClause` is specified as (`histValue`, `rgbValue` , `rgbValue` , `rgbValue`), where `histValue` is an integer from 1 to 100 (a percentage value), and `rgbValue` is an integer from 0 to 255. **file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

Table 18. Image search query: valid feature values (continued)

Feature name	Values
QbDrawFeatureClass or QbDraw	**description = < ″ descString ″ >** where descString is a special encoded string describing a picker file. Format of the description string: 1. Dw,h specifies the outer dimensions of the image itself with width w and height h. 2. Rx,y,w,h,r,g,b specifies that a rectangle of width w and height h is to be positioned with its upper left corner at the coordinates (x,y)—with respect to an origin in the upper left corner of the image rectangle—and this rectangle should have color values r (red), g (green), and b (blue). 3. Use the colon character (:) is used as a separator. For example, you can search for color layout (QbDrawFeatureClass) described by the description string: `QbDrawFeatureClass description= <"D100,50:R0,0,50,50,255,0,0"` **file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.
QbTextureFeatureClass or QbTexture	**file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

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\dl` 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\dl` 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\dl` directory.

Representing image search information with a DDO

A DDO associated with DKDatastoreQBIC contains specific information for representing image search results. A DDO resulting from an image query corresponds to an image part inside an item; it has the following set of standard attributes:

DKDLITEMID
: The item ID for the item to which this image part belongs. Use the item ID to retrieve the whole item from the content server.

DKPARTNO
: An integer part number of this image part. Use this with the item ID to retrieve this part from the content server.

DKREPTYPE
: A string for representation type (RepType). The default value is `FRN$NULL`. This attribute is reserved.

DKRANK
: An integer rank indicating the relevance of this part to the results set of the image query. The rank is within the range 0 to 100. A higher rank means a better match.

The PID for an image search DDO has the following information:

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>";
DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
 // ----- Open the catalog
dsQBIC.openCatalog("DEMO", "qbic0725");

  ...   // Process as appropriate

// ----- Create the query and run it
dkQuery pQry = dsQBIC.createQuery(cmd, DK_IMAGE_QL_TYPE, parms);
pQry.execute(parms);
// ----- Get the results and process
DKResults pResults = (DKResults)pQry.result();
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
   item = (DKDDO)pIter.next();
     // Process the DKDDO
 }
 dsQBIC.closeCatalog();
 dsQBIC.disconnect();
 dsQBIC.destroy();
 ...
```

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

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>";

DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");
 // ----- Execute the query from the datastore
 dkResultSetCursor pCur = dsQBIC.execute(cmd, DK_IMAGE_QL_TYPE, parms);
 while (pCur.isValid())
 {
    item = pCur.fetchNext();
    ....     // Process the DKDDO
  }
  pCur.destroy();
  dsQBIC.closeCatalog();
  dsQBIC.disconnect();
  dsQBIC.destroy();
```

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

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>";
DKNVPair parms[] = null;
DKDDO item = null;
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "qbic0725");

// ----- Use evaluate to run the query
DKResults pResults = (DKResults) dsQBIC.evaluate(cmd, DK_IMAGE_QL_TYPE, parms);
dkIterator pIter = pResults.createIterator();
while (pIter.more())
{
    item = (DKDDO)pIter.next();
     ...   // Process the DKDDO
}
dsQBIC.closeCatalog();
dsQBIC.disconnect();
dsQBIC.destroy();
```

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 earlier 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 pre-Version 8.1 Content Manager datastore and connect
DKDatastoreDL dsDL = new DKDatastoreDL();
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
// ------ Create a text search datastore and connect
DKDatastoreTS dsTS;
dsTS.connect("TM", "", ' ');  // TM is a local alias for
...                           //     the Text Search Engine server

// ----- Generate the parametric query string and create the query
String pquery = "SEARCH=(INDEX_CLASS=GRANDPA, COND=(DLSEARCH_Date > 1994));";
DKParametricQuery pq =
   (DKParametricQuery) dsDL.createQuery(pquery, DK_CM_PARAMETRIC_QL_TYPE, null);

// ----- Generate the text query string and create the query
String tquery = "SEARCH=(COND=(Tivoli)); OPTION=(SEARCH_INDEX=TMINDEX)";
DKTextQuery tq =
    (DKTextQuery) dsTS.createQuery(tquery, DK_CM_TEXT_QL_TYPE, null);

// ----- Create the combined query
```

```
DKCombinedQuery cq = new DKCombinedQuery();

// ----- Package the queries in DKNVPair as input parameters
DKNVPair par[] = new DKNVPair[3];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].setName(DK_PARM_END);    // to signal the end of parameter list

// ----- Execute the combined query
cq.execute(par);

// ----- Get the results
DKResults res = (DKResults) cq.result();
if (res != null) {
    ...  // process the results
}
```

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 if DKNVPairs as input parameters
DKNVPair par[] = new DKNVPair[4];
par[0].set(DK_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);

// ----- Execute the combined query
cq.execute(par);
....
```

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 earlier Content Manager workflow and workbasket functions

This section describes the earlier Content Manager workflow and workbasket functions.

Understanding the earlier Content Manager workflow service

An earlier 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\dl 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\dl 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\dl 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\dl 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\dl 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 (pIter1.more())
      {
         pwb1 = (DKWorkFlowDL)pIter1.next();
         pwb1->retrieve();
         ...                        // do some work
      }
   }
dsDL.disconnect();
dsDL.destroy();
```

Listing items in an earlier 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\dl directory.

Executing an earlier 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("EP8L8OR9MHH##QES");
DKString itemIDWF = new String("HI7MOPALUPFQ1U47");
DKString itemIDWB = new String("E3PP1UZOZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkFlowItem(itemID,                  // itemID
                       itemIDWF,                // itemIDWB
                       NULL,                    // default (the first workbasket)
                       TRUE,                    // overload
                       DK_WIP_DEFAULT_PRIORITY  // initial_priority
                       );
...                                          // do some work
wfDL.routeWipItem(itemID,                    // itemID
                  itemIDWF,                  // itemIDWB
                  TRUE,                      // overload
                  DK_NO_PRIORITY_CHANGE      // initial_priority
                  );
...                                          // do some work
wfDL.completeWorkFlowItem(itemID);
dsDL.disconnect();
dsDL.destroy();
```

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

Working with OnDemand

Enterprise Information Portal provides a connector and related classes for accessing content on Content Manager OnDemand servers. The OnDemand classes and APIs allow you to:

- Connect to and disconnect from OnDemand servers
- Listing application groups and application group fields
- Listing OnDemand folders
- Querying an application group
- Search and retrieve documents using the folder or the application group interface
- The ability to treat OnDemand folders as native entities in federated searches
- Search synchronously and asynchronously in either the application group or folder mode
- Retrieving entire OnDemand documents or document segments
- 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
- The ability to create and modify annotations

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

Representing OnDemand servers and documents

You represent a Content Manager OnDemand content server using a DKDatastoreOD in an Enterprise Information Portal application, and you represent an OnDemand document as a DDO using a DKDDO. OnDemand DDOs contain the following information:

- Document attribute names and their values
- Document data and annotations (represented as DKParts)
- Collection of logical views for a document
- Resource group data

An OnDemand document's attributes are stored in a DKDDO as properties. An OnDemand document's segments and notes are stored as DKParts.

All other document data (resource group and views, both fixed and logical), are stored as special properties in an OnDemand DDO with the following property names are reserved for the OnDemand:

DKViews
: A collection of logical views

DKFixedView
: Contains fixed view information

DKResource
: Contains resource group data

Notes™:

1. An EIP administrator must properly define the OnDemand connector by specifying the string in the **Additional Parameters** field on the **Initialization Parameters** page. The string should look like this: `ENTITY_TYPE=TEMPLATES;;` Be sure to include the two semicolons.
2. In EIPVersion 8.1, `DKViews`, `DKFixedView`, and `DKResource` replace `DKViewDataOD`, `DKFixedViewDataOD`, and `DKResouceGrpOD` repsectively.

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 OnDemand 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 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_PROPERTY_TYPE));
     System.out.println("       nullable: " +
                         p.getDataPropertyByName (j,DK_PROPERTY_NULLABLE));
     if (strDataName.equals(DKPARTS) == false &&
         strDataName.equals("DKResource") == false &&
         strDataName.equals("DKViews") == false &&
         strDataName.equals("DKLargeObject") == false &&
         strDataName.equals("DKFixedView") == false &&
         strDataName.equals("DKAnnotations") == false)
{
 System.out.println("       attribute id: "    +
                    p.getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID));
}
//-----Check for the type of the attribute
if (a instanceof String)
{
  System.out.println("      Attribute Value: " + a);
}
else if (a instanceof Integer)
{
  System.out.println("      Attribute Value: " + a);
}
else if (a instanceof Short)
{
  System.out.println("      Attribute Value: " + a);
```

```
}
else if (a instanceof DKDate)
{
  System.out.println("      Attribute Value: " + a);
}
else if (a instanceof DKTime)
{
  System.out.println("       Attribute Value: " + a);
}
else if (a instanceof DKTimestamp)
{
  System.out.println("       Attribute Value: " + a);
}
else if (a instanceof dkCollection)
{
  System.out.println("       Attribute Value is collection");
  pCol = (dkCollection)a;
  pIter = pCol.createIterator();
  i = 0;
  while (pIter.more() == true)
  {
    i++;
    a = pIter.next();
    pDO = (dkDataObjectBase)a;

    if (pDO.protocol() == DK_XDO)
    {
      System.out.println("     dkXDO object " + i + " in collection");
      pXDO = (dkXDO)pDO;
      DKPidXDO pid2 = pXDO.getPidObject();
      System.out.println("           XDO pid string: " +
                                    pid2.pidString());
             //----- Retrieve and open instance handler for an XDO
             pXDO.retrieve();
             // pXDO.open();
          }
        }
       }
       else if (a != null)
       {
         System.out.println("       Attribute Value: " + a.toString());
         if (strDataName.equals("DKResource")  ||
             strDataName.equals("DKFixedView") ||
             strDataName.equals("DKLargeObject"))
         {
           pDO = (dkDataObjectBase)a;

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

Enabling the OnDemand folder mode

To enable the OnDemand folder mode, the string

`ENTITY_TYPE=TEMPLATES`

must be passed to the OnDemand connector as part of the connection string or the configuration string. An sample configuration string would look like this:

```
DKDatastoreOD dsOD = new DKDatastoreOD("ENTITY_TYPE=TEMPLATES");
```

A sample connect string would look like this:

```
DKDatastoreOD dsOD = new DKDatastoreOD(); dsOD.connect(hostname, userid,
password, "ENTITY_TYPE=TEMPLATES");
```

Asynchronous search

The OnDemand connector supports both federated and direct asynchronous searches. An asynchronous search does not tie up the main thread and allows the search to be cancelled at any time; press the **Stop Search** button on the Search Template Viewer to terminate the search. The max hits property on the Search Template Viewer limits the maximum number of results returned.

The OnDemand connector also supports synchronous and asynchronous searches in the application group mode from an AIX client.

If you use a search criterion such as `WHERE userid LIKE '%'`, the resulting number of documents returned to the client can consume all available memory on the client's machine. By issuing an asynchronous search using the executeWithCallback() method, you can set the value for the maximum number of documents returned and cancel the search at any time.

You may also have to increase the default Java Virtual Machine (JVM) stack size if your result set is too large. The default stack size for each Java thread is 400k, which allows 3920 return items before the stack overflows. Increasing the JVM stack size to 800k doubles the capacity to 7840 items. If necessary, you can further increase the JVM stack size.

To raise the JVM stack size, use the Java command line option `-oss` followed by `nnnK` or `nnM`, where `K` stands for Kilobytes and `M` for Megabytes.

For examples of using asynchronous search, see the `TRetrieveWithCallbackOD`, `TRetrieveFolderWithCallbackOD` and `TCallbackOD` sample programs.

OnDemand folders as search templates

Three of the EIP visual JavaBeans, `CMBSearchTemplateList`, `CMBSearchTemplateViewer`, and `CMBSearchResultsView`, use EIP search templates. You can use these beans for federated searches by setting the CMBConnection dsType to Fed.

You can use these beans for direct searches on OnDemand servers as well. Set the following properties before login:

```
connection.setDsType("OD");
connection.setServerName(<odserver>);
connection.setConnectString("ODENTITY=TEMPLATES");
```

OnDemand folders as native entities

The OnDemand connector can also map OnDemand folders as native entities by specifying the connect string `"ODENTITY=TEMPLATES"`. Using OnDemand folders as entities can be useful in federated searches, where folder definitions may be easier to work with than OnDemand application groups, the default native entity for OnDemand.

For federated searches, specify the server definition in EIP administration. You can then define and map federated entities to the folders on the OnDemand server.

Create and Modify Annotations

When using the OnDemand viewer, which is launched by the CMBDocumentViewer bean, you can create, modify, and delete annotations for OnDemand documents by using the CMBDataManagement bean and the associated CMBAnnotation class.

Working with ImagePlus for OS/390

Enterprise Information Portal APIs support the following features when working with Content Manager ImagePlus for OS/390 content servers:

- Connecting and disconnecting from one or more ImagePlus servers
- Retrieving categories
- Retrieving attribute fields
- Retrieving folders
- Retrieving documents

You represent an ImagePlus for OS/390 content server using DKDatastoreIP in your application.

Restriction: ImagePlus for OS/390 does not support Text Search Engine and QBIC search 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\dl` 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 Content Manager for AS/400

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

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

Listing entities (index classes) and attributes

You represent a Content Manager for AS/400 content server as a DKDatastoreV4. After creating the datastore and connecting to it, you can list the entities (index classes) and attributes for the Content Manager 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\dl directory.

Executing a query

The following example runs a query in Content Manager 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\dl 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 19 on page 190 represents the Domino.Doc object model.

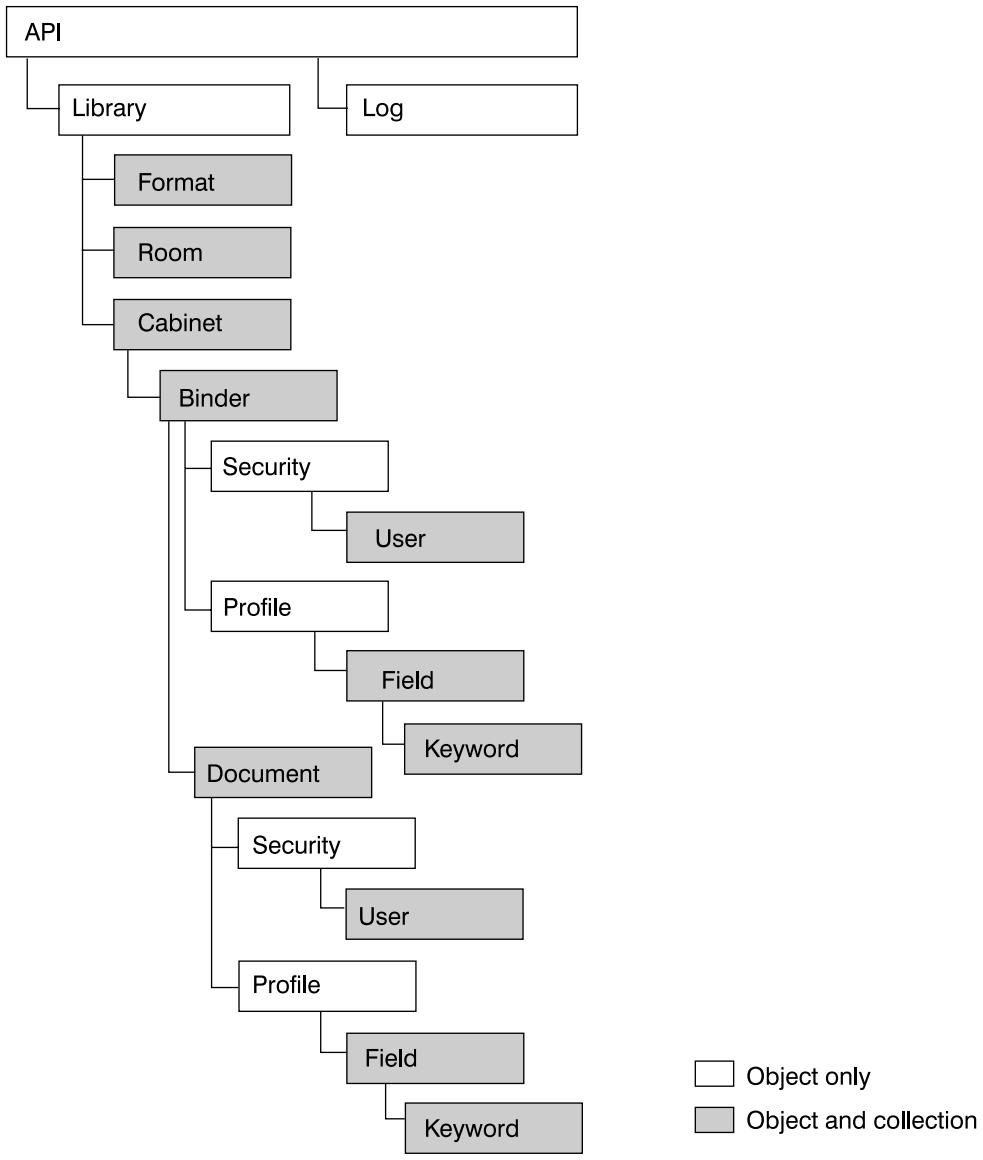

Figure 19. Domino.Doc object model

The elements contained in Domino.Doc and the APIs to represent them are arranged such that:

- The library contains rooms (DKRoomDefDD objects) and cabinets (DKCabinetDefDD objects)
- Each cabinet contains binders (DKBinderDefDD object)
- Each binder contains a profile (DKAttrProfileDefDD object) and security
- Each binder contains documents (DKDocumentDefDD objects)
- Each document contains a profile (DKAttrProfileDefDD object) and security
- Each profile contains fields (DKAttrFieldDefDD objects)
- Each field can contain keywords (DKAttrKeywordDefDD objects)

Listing entities and subentities

The following example lists the 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\dl 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\dl 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 Extended Search (ES)

Enterprise Information Portal supports working with IBM Extended Search 7.1, while deprecating support for ES 3. Extended Search allows you to query and retrieve documents from:

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

Enterprise Information Portal provides classes and APIs to support the following features when working with IBM Extended Search:

- Connecting and disconnecting from one or more ES servers
- Listing ES servers
- Listing databases and fields
- Ability to use Generalized Query Language (GQL) to perform searches
- Ability to retrieve documents
- Use the ES C++ classes and APIs on AIX
- Parametric text searches are now supported from the federated layer and through a direct ES connection
- CONTAINS_TEXT and CONTAINS_TEXT_IN_CONTENT text search operators are supported from the federated layer
- The ES classes and APIs support all EIP JavaBeans.

Restriction: ES does not support:

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

All ES features are accessed and controlled by the ES configuration database. Use the configuration database to assign database definitions for data sources to be searched, network addresses, access control information, and other related information.

Listing Extended Search servers

In order to provide access to multiple ES servers, you can create a file named `cmbdes.ini` that contains the server information. 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 ES 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 ES 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\dl directory.

Using Generalized Query Language (GQL)

Extended Search uses the Generalized Query Language (GQL) to perform searches. Table 19 contains examples of valid GQL expressions.

Table 19. GQL expressions

GQL expression	Description
″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

ES uses the query type DK_DES_GQL_QL_TYPE. This query type has the following syntax:

```
SEARCH=(DATABASE=(db_name | db_name_list | ALL);
                 COND=(GQL expression));
[OPTION=([SEARCHABLE_FIELD=(fd_name, ...);]
                  [RETRIEVABLE_FIELD=(fd_name, ...);]
                  [MAX_RESULTS=maximum_results;]
                  [TIME_LIMIT=time)]
```

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 ES. 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\dl directory.

DDO properties in ES

A DDO in ES 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 Extended Search

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)* and *Creating a persistent identifier (PID)*. 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 ES 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\dl 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 the document ID
pid.setPrimaryId("Notes Help" + DK_DES_ITEMID_SEPARATOR + "215e");
pid.setObjectType("Notes Help");
DKDDO ddo = new DKDDO(dsDES, pid);
ddo.retrieve();
```

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

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 ES 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);
DKPidXDODES pid = new DKPidXDODES();
pid.setDocumentId("D:\\desdoc\\README.html");
pid.setDatabaseName("DES files");
pid.setFieldName("Doc$Content");
xdo.setPidObject(pid);
xdo.retrieve("c:\\DESReadme.html");
```

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

Associating MIME types with documents

ES 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 ES 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 Extended Search in federated searching

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

Table 20 summarizes the SQL to GQL conversion of the supported comparison and logical operators.

Table 20. SQL and GQL operators

SQL operator	GQL operator
AND	AND
OR	OR
NOT	not supported
IN	not supported
BETWEEN	BETWEEN
EQ	EQ
NEQ	not supported
GT	GT
LT	LT
LIKE	not supported
GEQ	GTE
LEQ	LTE
NOTLIKE	not supported
NOTIN	not supported
NOTBETWEEN	not supported

Working with Panagon Image Services

You can use The Enterprise Information Portal API classes provided for Panagon Image Services (FileNET) to access content on Panagon Image Services servers. Support for Panagon Image Services is available only as a special feature of EIP.

Restriction: Panagon Image Services does not support Text Search Engine and QBIC search or combined queries.

Data modeling

Enterprise Information Portal has some data modeling concepts that need to be mapped to the corresponding objects in the content server that a connector supports. The table below shows the EIP data models in the FileNET environment.

EIP Concept	FileNET
datastore	Panagon Image Services Server
server	server
entity	document class
attribute	index
DDO	document (folder) Folders in FileNET are used to organize documents temporarily; they are not currently modeled in the connector.
XDO	page content, annotation
PID for a DDO	document_id, as the primary ID

PID for page content XDO	System_serial_num + document_id + page#
PID for annotation XDO	System_serial_num + document_id + page# + annotation_id

FileNET connector only returns user-defined indexes or attributes in FileNET, which can be used in federated entity and attribute mappings and search result attribute display lists. The following table shows the mapping of FileNET supported index or attribute data types to the corresponding EIP attribute data types in the FileNET Connector:

FileNET index data type	EIP attribute data type
numeric	double
date	varchar with max_length set to the maximum length defined for the FileNET index
menu	varchar with max_length = 14 (maximum for menu names)

Every XDO in EIP has a MIME type. EIP clients use the MIME type to determine how to display the document. FileNET Connector assumes that all XDOs of the same document are of the same MIME type. The MIME type of the XDO is set according to the F_DOCFORMAT FileNET system attribute value of the belonging document. If the value is null, the MIME type is then set to `image\tiff` if the F_DOCTYPE FileNET system attribute indicates the image type; otherwise, it is set to the default MIME type of `application/octet-stream`.

Documents and pages in Panagon Image Services

To use a DDO to represent a document in Panagon Image Services, four fields in the DDO's PID need to be set properly:

1. DatastoreType – this is the constant DK_FN_DSTYPE defined in DKConstantFN. You can get this value from the datastoreType() method of the DKDatastoreFN.
2. DatastoreName – this is the FileNET domain-organization name. You can get this value from the datastoreName() method of the DKDatastoreFN.
3. ObjectType – this is the FileNET document class (native entity name in EIP) name to which the document belongs.
4. PrimaryID – this is the FileNET document number, assigned by FileNET.

Once the PID is set, application programs can get attributes or contents of the DDO by calling the retrieve() method either in DKDatastoreFN or DKDDO class.

A document DDO in EIP has a special attribute with a reserved name DKPARTS, whose value is a DKParts object. The DKParts object is a collection of binary large objects (BLOBs). The FileNET parts (pages/annotations) within a document are represented as DKBlobFN objects (a subclass of DKXDO) which are elements in the DKParts collection.

A FileNET page is not a conventional page; a single page (or single part) document in FileNET may contain multiple physical pages but they are stored in a single page within a FileNET document. A multiple-page document in FileNet refers to a document that contains either multiple parts or multiple files that are created using the Panagon Capture Software or the Batch Entry System.

Listing entities and attributes

You represent a Panagon Image Services server as a DKDatastoreFN. After creating the datastore and connecting to it, you can list the entities (document x classes) and attributes for the Panagon Image Services server. The following example illustrates doing this:

```
DKDatastoreFN dsFN = null;

    try {
      DKSequentialCollection pCol = null;
      dkIterator pIter = null;
      DKSequentialCollection pCol2 = null;
      dkIterator pIter2 = null;
      DKServerDefFN pSV = null;
      String strServerName = null;
      String strServerType = null;
      String strEntity = null;
      DKEntityDefFN entityDef = null;
      DKAttrDefFN attrDef = null;
      int i = 0;
      int j = 0;
      dsFN = new DKDatastoreFN();
// ----- Connect to the server using the server name, user ID , and password
      dsFN.connect(libSrv, userid, pw, "");
      System.out.println("Datastore connected libSrv " + libSrv + " userid "
        +  userid );
      System.out.println("List entities in server " + libSrv);
      pCol = (DKSequentialCollection) dsFN.listEntities();
      pIter = pCol.createIterator();
      i = 0;
      while (pIter.more() == true)
      {
        i++;
        entityDef = (DKEntityDefFN)pIter.next();
        strEntity = entityDef.getName();
        System.out.println("\nEntity name [" + i + "] - " + strEntity +
          ", id = " + entityDef.getId() + ", type = " + entityDef.getType());
        System.out.println("  List attributes for the " + strEntity + " entity");
        pCol2 = (DKSequentialCollection) dsFN.listEntityAttrs(strEntity);
        pIter2 = pCol2.createIterator();
        j = 0;
        while (pIter2.more() == true)
        {
         j++;
         attrDef = (DKAttrDefFN)pIter2.next();
         System.out.println("    Attribute name [" + j + "] - " +
             attrDef.getName());
         System.out.println("      datastoreType = " + attrDef.datastoreType());
         System.out.println("      attributeOf = " + attrDef.getEntityName());
         System.out.println("      type = " + attrDef.getType());
         System.out.println("      size = " + attrDef.getSize());
         System.out.println("      id = " + attrDef.getId());
         System.out.println("      nullable = " + attrDef.isNullable());
         System.out.println("      precision = " + attrDef.getPrecision());
         System.out.println("      scale = " + attrDef.getScale());
         System.out.println("      stringType = " + attrDef.getStringType());
        }
       System.out.println("  A total of " + j + " attributes listed for the "
                + strEntity + " entity");
      }
      System.out.println("\nA total of " + i + " entities listed for the server "
            + libSrv);
     // ----- Disconnect and clean up
```

```
        dsFN.disconnect();
        dsFN.destroy();
      }
      catch(DKException exc)
```

Sample applications for working with Panagon Image Services servers are available in the CMBROOT\Samples\java\fn directory.

Querying

EIP supports three basic forms of query for Panagon Image Services:

- query object
- command string
- DKCQExp

Both query object and command string forms of query are typically used in applications that are accessing the Panagon Image Services server directly. The EIP federated datastore and query uses only the DKCQExp form of the query to interface with individual datastores.

Sample applications for querying Panagon Image Services servers are available in the CMBROOT\Samples\java\fn directory.

Query String Syntax and Parameters

FileNET Connector supports the use of DKParametricQuery to submit and execute a query. The DKParametricQuery can be constructed with a command string. The command string represents the query specification for the FileNET filter condition. Other query options (ENTITY_NAME, MAX_RESULTS, CONTENT) and FileNET key condition (KEY) will be accepted as parameters to the evaluate() or execute() methods in (name, value) pairs (e.g., DKNVPair). (Please note that the values are always of String type and the parameter names are defined in DKConstant and DKConstantFN.) The return results are a collection of document DDOs for evaluate() and the result set cursor pointing to the results for execute().

The conditional expression syntax for the command and the KEY parameter follow the FileNET query syntax supported by FileNET WAL PRS_Parse() function. It accepts the following operators:

Operator	Use
AND	logical and
OR	logical or
NOT	logical not
<	less than
<=	less than or equal to
=	equal to
>	greater than
>=	greater than or equal to
!=	not equal to
+	addition
-	subtraction
*	multiplication
/	division

IN RANGE	a range is specified with the syntaxIN RANGE *op constant op constant*, where *op* is <, <=, >, or >=, and *constant* is a numeric or string constant
LIKE	Similar to. The pattern on the right hand side of the LIKE operator must be a single quoted character string that can contain ? as a wild character (to indicate match any character), and * as a wild card..
DEFINED(*x*)	The defined function returns non-zero if the given column name *x* is non-null, and zero if the value is null.

The operands are numeric constants, string constants and attribute names. Since command string is used for searching FileNET datastore directly, the attribute names are all local FileNET attribute/index names. No federated attribute names and no schema mapping would be supported nor required

Numbers are in the format *+ddd.dddE+eee* where *ddd* stands for 0 or more numeric digits, + stands for + or - (+ is optional), *eee* stands for 0 to three digits of the exponent, and . is a decimal point (optional), and E is the beginning of the exponent (optional).

String constants are surrounded by single quotes, and an embedded single quote within a string is represented by two consecutive single quotes.

The conditional expression for the command string may contain any number of operands and operators. Parenthesis may be also used in the filter to override precedence. Examples of filter conditions are as follows:

```
PROD_PRICE > 100 AND (PROD_DATE > '10/15/92' OR PROD_NAME LIKE 'comp*')
PROD_PRICE >= 100 AND (NOT (PROD_ID < 30))
```

The conditional expression for the KEY parameter must contain just one attribute name, which is defined as a retrieval key in FileNET, and one condition. Note that key names defined in FileNET must be used, and a non-inverted column name may not be used. Examples of key strings are:

```
PROD_PRICE = 100000
PROD_PRICE IN RANGE >= 100000 <= 200000
```

Additional search options

The other supported optional parameters for evaluate() and execute() are:

(ENTITY_NAME, entity_Name):

Specify the entity name (FileNET document class name) as the scope of query. Sample:

```
parms[1] = new DKNVPair (DK_FN_PARM_ENTITY_NAME, entity_Name);
```

The actual query string submitted to the Panagon Image Services server is appended with AND (F_DOCCLASSNUMBER = doc_cls#), where = doc_cls# is the matching document class number of the specified entity name. If not specified, documents in all FileNET document classes are searched.

(MAX_RESULTS, Max_results)

Only the specified maximum number of results are returned in the result collection. If the value is zero or not specified, the maximum number of results allowed by FileNET is returned. Sample:

```
parms[0] = new DKNVPair(DK_CM_PARM_MAX_RESULTS, Max_results);
```

(CONTENT, YES / NO / ATTRONLY)

This parameter will overwrite the DK_CM_OPT_CONTENT option set at the datastore level.

- YES - the resulting documents will have their contents retrieved (default)
- NO - only the document IDs are set in the result DDOs
- ATTRONLY - the document IDs, the data attributes and their values are set in the result DDOs

Sample:

```
dsFN.setOption(DK_CM_OPT_CONTENT, DK_CM_CONTENT_YES);
```

For example, content value YES would cause the resulting document DDOs to have their PID, object type, properties and all attributes set. This option also causes the DKPARTS attribute to be set to a collection of content parts (XDOs for parts and their annotations) with PID information set but no actual content. If the CONTENT value is ATTRONLY, the resulting document DDOs PID, object type, properties, and all data attributes are set. If the CONTENT value is NO, the resulting document DDOs PID, object type, and properties are set. The document part or XDO content can be retrieved explicitly when needed using the retrieve() method of the DKBlobFN class.

Exception handling and messages

The Panagon Image Services connector employs the same set of Java exception classes defined in EIP. The text message included in an exception also uses one of the commonly defined messages in DKMessageID whenever is possible. For example, the DKUsageError exception includes a message as (DKMessage.getMessage(DK_CM_MSG_NOTIMP) + this.getClass().getName() + *XXXX* , DK_CM_MSG_NOTIMP); where *XXX* is the method name that the program is attempting to invoke.

DKDatastoreAccessError exception will include the FileNET specific message text in the following format:

WAL_*function_name* [IMS_SESSION= *sss*, ERR_CAT=*ccc*, ERR_FUN=*fff*, ERR_NUM=*nnn*]

You can use the FileNET command msg (accessible via the c:\fnsw\client\bin directory) to further display the native FileNET error message text using the three numbers *ccc*, *fff* and *nnn* (e.g., msg *ccc*,*fff*,*nnn*.

The EIP message IDs used uniquely for the Panagon Image Services servers are from 3401 to 3600.

Working with relational databases

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

Connecting to relational databases

To represent a relational database, create a DKDatastore*xx* object, where the *xx* is DB2 for a DB2 database, 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.

`JDBCSERVERSURL=(URL)`

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

`JDBCSERVERSFILE=(filename)`

Specifies the cmbjdbcsrvs.ini file that contains the list of JDBC servers as a filename.

`JDBCDRIVER=(JDBC driver)`

Specifies the JDBC driver that you want to use. This is automatically set when you use the system administration client client program.

`DSNAME=(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\dl 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\dl 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());
 ....
```

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
- 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 `cmbcm81.jar` file in order to access the common framework

Enterprise Information Portal database infrastructure

The primary interface of each content server to an 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 DKDatastore*xx*, where *xx* could be the name or type of the specific content server. Table 21 lists the content servers provided in Enterprise Information Portal.

Table 21. Content servers provided in Enterprise Information Portal

Server type	Class name
Content Manager	DKDatastoreICM
OnDemand	DKDatastoreOD
Content Manager for AS/400	DKDatastoreV4
ImagePlus for OS/390	DKDatastoreIP
Domino.Doc	DKDatastoreDD
Extended Search	DKDatastoreDES
DB2 Universal Database	DKDatastoreDB2
JDBC	DKDatastoreJDBC
IBM DB2 DataJoiner	DKDatastoreDJ

Table 21. Content servers provided in Enterprise Information Portal (continued)

DB2 Warehouse Manager Information Catalog Manager	DKDatastoreIC
earlier Content Manager	DKDatastoreDL

You must specify the DKDatastore class you create for your content server in the EIP system administration client when you create your server definition.

Common classes in Enterprise Information Portal

dkDDO

The dkDDO class provides a representation and a protocol to define and access an object's data, independent of the object's type. The DDO protocol is implemented as a set of 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.

This diagram represents the hierarchy for the dkDDO class:

```
java.lang.Object
   |
   +----com.ibm.mm.sdk.common.dkDataObjectBase
           |
           +----com.ibm.mm.sdk.common.dkDataObject
                   |
                   +----com.ibm.mm.sdk.server.DKDDOBase
                           |
                           +----com.ibm.mm.sdk.server.DKDDO
```

dkXDO

The dkXDO class represents complex user defined types (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:

```
java.lang.Object
   |
   +----com.ibm.mm.sdk.server.dkDataObjectBase
           |
```

```
                +----com.ibm.mm.sdk.server.dkXDOBase
                        |
                        +----com.ibm.mm.sdk.server.dkXDO
```

dkCollection

The dkCollection class is a 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 22 contains examples of concrete classes for the dkDatastoreDef interface.

Table 22. Concrete classes for dkDatastoreDef

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

The main methods in the dkDatastoreDef interface are:

listEntities()
Lists entities

listEntityAttrs()
Lists entity attributes

addEntity()
Adds an entity

getEntity(name)
Gets an entity

Each concrete class can also have its own 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 23 contains examples of concrete classes for the dkEntityDef class.

Table 23. Concrete classes for dkEntityDef

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

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 24 contains examples of concrete classes for the dkAttrDef class.

Table 24. Concrete classes for dkAttrDef

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

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 25 contains examples of concrete classes for the dkServerDef class.

Table 25. Concrete classes for dkServerDef

Server type	Class name
Content Manager	DKServerDefICM
OnDemand	DKServerDefOD
Content Manager for AS/400	DKServerDefV4
Domino.Doc	DKServerDefDD
Extended Search	DKServerDefDES
DB2 Universal Database	DKServerDefDB2
JDBC	DKServerDefJDBC
ODBC	DKServerDefODBC
DB2 DataJoiner	DKServerDefDJ
DB2 Warehouse Manager Information Catalog Manager	DKServerDefIC
earlier Content Manager	DKServerDefDL

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 26 contains examples of concrete classes for the dkBlob class.

Table 26. Concrete classes for dkBlob

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

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 start positions

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 afileName)
Unloads the object contents to the file `afileName` 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 DKBlob*xx* class, where the dkBlob object is the representation of the binary annotation data and the DKParts collection.

dkDatastoreExt

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

Table 27 contains examples of concrete classes for the dkDatastoreExt class.

Table 27. Concrete classes for dkDatastoreExt

Server type	Class name
Content Manager	DKDatastoreExtICM
OnDemand	DKDatastoreExtOD
Content Manager for AS/400	DKDatastoreExtV4
ImagePlus for OS/390	DKDatastoreExtIP
Domino.Doc	DKDatastoreExtDD
Extended Search	DKDatastoreExtDES
DB2 Universal Database	DKDatastoreExtDB2
JDBC	DKDatastoreExtJDBC
DB2 DataJoiner	DKDatastoreExtDJ
earlier Content Manager	DKDatastoreExtDL

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 28 contains examples of concrete classes for the DKPidXDO class.

Table 28. Concrete classes for DKPidXDO

Server type	Class name
earlier Content Manager	DKPidXDODL
OnDemand	DKPidXDOOD
Content Manager for AS/400	DKPidXDOV4
ImagePlus for OS/390	DKPidXDOIP
Domino.Doc	DKPidXDODD
Extended Search	DKPidXDODES
DB2 Universal Database	DKPidXDODB2
JDBC	DKPidXDOJDBC
ODBC	DKPidXDOODBC
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 29 contains examples of concrete classes for the dkUserManagement class.

Table 29. Concrete classes for dkUserManagement

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

DKConstant

All common constants are defined in the DKConstant class. Each content server will have its own DKConstant*xx* 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 DKMessageId*xx* 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.

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 7. Using Enterprise Information Portal workflow in Java

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 workflow assigned with a content item, 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();
```

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 PID 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
DKWorkFlowFed WF = new DKWorkFlowFed(svWF);
WF.setName("wfl");
// ----- Create an instance of a workflow with the workflow template name
WF.add("WD1");
// ----- Refresh the workflow object
WF.retrieve();
```

```
  // ----- Construct the container object for the workflow
  DKWorkFlowContainerFed con = WF.inContainer();
  // ----- Retrieve the container data
  con.retrieve();
  // ----- Add a PID string referring to an Extended Search document
  con.setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
  con.setPriority(100);
  // ----- Update the container
  con.update();
  // ----- Start the workflow
  WF.start(con);
```

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_RUNNING ||
    state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDED ||
    state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDING)
{
    WF.terminate();
}
if (state ==DKConstantFed.DK_FED_FMC_PS_READY ||
    state ==DKConstantFed.DK_FED_FMC_PS_FINISHED||
    state ==DKConstantFed.DK_FED_FMC_PS_TERMINATED)
{
    WF.del();
}
```

Suspending a workflow

You can suspend a running workflow until a specific time or indefinitely. The following example illustrates how to suspend a workflow until the specified timestamp. If you provide a null DKTimestamp, the workflow is suspended indefinitely.

```
// ----- Construct a DKWorkFlowFed object
DKWorkFlowFed WF = new DKWorkFlowFed(svWF, "Test");
WF.retrieve();
// ----- Call the suspend method if the workflow is in the running state
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_RUNNING)
{
  // ----- Suspended until 2000-07-27-16.30.00.000000
  // -----   The timestamp uses the base year 1900; months are
  // -----   numbered 0 to 11
  DKTimestamp suspension = new DKTimestamp(100, 6, 27, 16, 30, 0, 0);
  WF.suspend(suspension);
}
```

Resuming a workflow

You can resume a suspended workflow using the resume() method. The following example shows resuming a workflow:

```
// ----- Construct a DKWorkFlowFed object
DKWorkFlowFed WF = new DKWorkFlowFed(svWF, "Test");
WF.retrieve();
// ---- Call resume() if the workflow is in the suspended state
```

```
if (WF.state() == DKConstantFed.DK_FED_FMC_PS_SUSPENDED)
{
  WF.resume();
}
```

Listing all the workflow templates

You can list all the workflow templates in a workflow service by calling the listWorkFlowTemplates() method. The following example lists the name and description of all the workflow templates in a workflow service referenced by the DKWorkFlowSerivceFed object svWF:

```
// ----- Call the listWorkFlowTemplates method
DKSequentialCollection collWT =
            (DKSequentialCollection)svWF.listWorkFlowTemplates();
DKWorkFlowTemplateFed WT = null;
if (collWT != null)
{
  dkIterator iterWT = collWT.createIterator();
  while (iterWT.more() == true)
  {
    WT = (DKWorkFlowTemplateFed)iterWT.next();
    WT.retrieve();
    System.out.println("name = " + WT.name() + " description = "
            + WT.description());
  }
  iterWT = null;
 }
```

Listing all the workflows

You can list all the workflows in a workflow service using the listWorkFlows() method. The following example lists the name and description of all the workflows in a workflow service referenced by the DKWorkFlowSerivceFed object svWF:

```
// ----- Call the listWorkFlows method
DKSequentialCollection collWF = (DKSequentialCollection)svWF.listWorkFlows();
DKWorkFlowFed WF = null;
if (collWF != null)
{
  dkIterator iterWF = collWF.createIterator();
  while (iterWF.more() == true)
  {
    WF = (DKWorkFlowFed)iterWF.next();
    WF.retrieve();
    System.out.println("name = " + WF.getName() + " description = "
                                  + WF.getDescription());
  }
  iterWF = null;
}
```

Listing all the worklists

You can list all the worklists in a workflow service by calling the listWorkLists() method. The following example lists the name and description of all the worklists in a workflow service referenced by the DKWorkFlowServiceFed object svWF:

```
// ----- Call the listWorkLists method
DKSequentialCollection collWL = (DKSequentialCollection)svWF.listWorkLists();
DKWorkListFed WL = null;
if (collWL != null)
{
  dkIterator iterWL = collWL.createIterator();
  while (iterWL.more() == true)
  {
```

```
    WL = (DKWorkListFed)iterWL.next();
    WL.retrieve();
    System.out.println("name = " + WL.getName() + " description = "
                    + WL.getDescription());
  }
  iterWL = null;
}
```

Accessing a worklist

You can access a worklist by creating a DKWorkListFed object that refers to the worklist you created using system administration client. 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 ();
     System.out.println ("workitem node = " + nodename +
               "  workflow name = " + workflowname);
  }
}
iter = null;
```

Moving items in the workflow

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

```
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.

Chapter 8. Using non-visual and visual JavaBeans

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

Enterprise Information Portal provides non-visual JavaBeans for use in Java applications. The function in the non-visual beans covers what is typically needed in building end-user applications. Administrative API's, such as creating item types, are not available through the beans. A common use of the non-visual beans is as the data model in Web applications. In this case, servlets would create instances of the beans, that are then referred to in JavaServer Pages (JSPs).

The non-visual beans can be used in command line or windowed applications. They also serve as the data model 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`, `com.ibm.mm.beans.util`, `com.ibm.mm.beans.workflow` packages contain the non-visual beans.

Enterprise Information Portal provides visual JavaBeans to build Swing-based Java applications using the Enterprise Information Portal federated search capabilities. They can also be used in building applications that connect directly to content servers (not federated). 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,IBM Websphere Studio Application Developer, 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\cmb81.jar
   - \CMBROOT\lib\cmbview81.jar
   - \CMBROOT\lib\cmbsdk81.jar
   - \CMBROOT\lib\lotuskms.jar
   - \CMBROOT\lib\eclisrv.jar
   - \SQLLIB\java\db2java.zip

   The cmb81.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 IBM Websphere Studio Application Developer

You can use the non-visual beans to build servlets and JSP pages in Webphere Studio Application Developer by completing the following steps:

1. Create a Web project for your Web application.
2. In the properties for the project, underJava Build Path | Libraries, specify the following JAR files:

   \CMBROOT\lib\cmb81.jar

   \CMBROOT\lib\cmbview81.jar

   \CMBROOT\lib\cmbsdk81.jar

   \CMBROOT\lib\lotuskms.jar

   \CMBROOT\lib\eclisrv.jar

   \SQLLIB\java\db2java.zip
3. If you plan on using the EIP servlets and JSP taglib, you must also specify the following files:

   \CMBROOT\lib\cmbservlet81.jar

   \CMBROOT\lib\cmbtag81.jar

   For the tag library, you also need to import the taglib.tld file for EIP's JSP taglib into your web application:

   - Copy \CMBROOT\lib\taglib.tld to the webApplication\WEB-INF directory in your web application.
   - Configure the taglib in the webApplication\WEB-INF\web.xml file in you web application by adding the following:

   ```
   <taglib>
               <taglib-uri>cmb</taglib-uri>
               <taglib-location>/WEB-INF/taglib.tld</taglib-location>
   </taglib>
   ```
4. Since the JARs listed above contain J2EE classes, you need to include the J2EE JAR, usually located in: `\Program Files\IBM\Application Developer\plugins\com.ibm.etools.websphere.runtime\lib\j2ee.jar`

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 cmb81.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 beans.

CMBConnection
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.

CMBConnectionPool
This bean maintains a pool of CMBConnection 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 CMBConnection instance for the same user ID is reused.

CMBDataManagement
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 CMBConnection.

CMBDocumentServices
This bean provides rendering and conversion of documents for thick and thin clients (see Document Viewing Services below). It also provides pagination and annotations manipulation on documents.

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 CMBConnection, in which case it returns schema information about the server or database connected to by CMBConnection. Alternatively, you can create an instance separate from CMBConnection, 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.

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.

CMBObjectManagemet
This bean provides functions to more efficiently manage CMBItem and CMBObject objects.

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

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

CMBDocRoutingDataManagementICM
This bean works with document routing data. An instance of this bean can be obtained from CMBConnection.

CMBDocRoutingQueryServiceICM
This bean performs queries on document routing information. You can obtain an instance of this bean from CMBConnection.

Non-visual bean configurations

Non-visual beans have local, remote and dynamic configurations.

local Connects directly to the content server

remote
Connects to a content server using an RMI server

dynamic
Enables an application that dynamically switches between local and remote based on the `cmbcs.ini` file. The `cmbcs.ini` file specifies whether the content server is local or remote.

Non-visual bean data model classes

The non-visual beans provide a uniform data model onto all backend-servers regardless of type. The following classes are used to represent the model:

CMBSearchTemplate
This class represents a federated search template. It also represents a Folder when directly connected to OnDemand. The class provides methods to perform a search using the template. Instances of CMBSearchTemplate can be obtained from CMBSchemaManagement.

CMBSTCriterion
This class represents a single search criterion within a search template.

CMBEntity
This class represents the definition of an entity on the server. The entity varies depending on server type. See the table below for how this is mapped onto different types of servers.

CMBAttribute
This class represents the definition of an attribute of an entity.

CMBItem
This class represents a document, folder, or other item returned from a search.

CMBObject
This class represents a resource part on a document. This is either content (the actual document itself) or a note or annotation on the document.

CMBAnnotation
This class represents one annotation part. This provides additional methods when dealing with OnDemand notes annotations (notes).

CMBViewData, CMBFixedViewData, CMBResources
These classes represent logical and physical view data, and document resources for OnDemand documents.

CMBPrivilege
This class represents the privileges for an entity or item. Privileges define the possible operations on an item or class of items.

CMBWorkflow, CMBWorkflowContainer, CMBWorkflowTemplate, CMBWorkItem, CMBWorklist
These classes represent workflows, worklists, and work items in advanced workflow.

CMBProcessICM, CMBResumeListEntryICM, CMBRouteListEntryICM, CMBWorkListICM, CMBWorkNodeICM, CMBWorkPackageICM
These classes represent structures used in Content Manager V8 document routing.

Table 30 relates the non-visual bean classes to the objects they represent for several content servers.

Table 30. Content server class object representations

Class	Federated	CM V8.1	CM V7	OD	Relational
CMBSearchTemplate	Federated search template	N/A	N/A	Folder	N/A
CMBSTCritierion	Federated search template search criteria	N/A	N/A	Folder criterion	N/A
CMBEntity	Federated entity	Item type, or child component type	Index class	Application group	Table
CMBAttribute	Federated attribute	Attribute definition	Attribute	Attribute definition	Column
CMBItem	Item on content server	Item, document, folder	Document or folder	Document	Row
CMBObject	Content part, as defined for the server	Document part or BLOB attribute	Part	Document content or resource	BLOB attribute value
CMBAnnotation	Annotation, as defined for the server	Annotation part	Annotation part	Document note	N/A

CM V8.1 - Content Manager Version 8 **CM V7** - Content Manager Version 7 **OD** - OnDemand

Child Components
Content Manager Version 8 has the concept of child components. Child components are complex attributes of an item. For the definition of child

components, CMBEntity class is used. From the CMBEntity for the root component, CMBEntity.getSubEntities returns the CMBEntity objects describing the child components. The actual child components themselves are accessible from an item, using CMBItem.getAttrItems, and are represented using CMBItems.

Annotations

Annotations parts are represented using the CMBAnnotation class. Depending on the content server, a single annotation part may correspond to one or more actual annotations. For Content Manager V7 and V8, one annotation part contains all of the annotations for a document. In order to work with the individual annotations on the document, use the CMBAnnotationServices in the Java viewer package, com.ibm.mm.viewer. For OnDemand, each annotation part is a separate annotation (note) on the document. The properties of the annotation are available on the CMBAnnotation object itself.

Relational Database Blobs and Clobs

A relational database may contain BlOB and CLOB attributes. These attributes are treated as content parts, with their content type defined using the cmbcc2mime.properties file. All values of a particular BLOB and CLOB are considered to be of the same content type.

Understanding properties and events for non-visual beans

Each non-visual bean provides the following:

- Imported properties, vetoable or not

  The property value is determined by other beans at run time by PropertyChange or VetoableChange events. Beans that have import properties must listen to PropertyChange or VetoableChange events.

- Exported properties, vetoable or not

  A non-visual bean may have a constrained property and some other beans might have interest in its value. Whenever its value is changed, the bean is responsible for generating a PropertyChange or VetoableChange event.

- Stand-alone properties

  No other beans have interest in this property value.

- Events generated by this bean
- Events in which this bean is interested

Building an application using non-visual beans

A sample non-Graphical User Interface (GUI) application

The example in this section uses non-visual beans to create a sample non-GUI application. The sample application includes every bean except the CMBUserManagement bean. The complete sample application from which this example was taken (`DemoSimpleAppl.java`) is available in the `Cmbroot/Samples/java/beans` directory. The sample application shows how to:

1. Connect to the Enterprise Information Portal (federated) server
2. Get a list of search template names
3. Use the search template name to get a list of search criteria names
4. Select a search template and gets its search criteria
5. Complete the search values and submits a query
6. Print the result using the search results bean
7. Select a result row and displays it

8. Disconnect from the server

```
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 DemoSimpleAppl
{
    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 DemoSimpleAppl()
    {
      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 CMBConnection 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, using the EIP administration database (server) name, user ID, 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;
                        case CMBBaseConstant.CMB_OP_LESS:
                            System.out.println("\t\t\t.LT."); break;
                        case CMBBaseConstant.CMB_OP_GREATER_EQUAL:
                            System.out.println("\t\t\t.GEQ."); break;
                        case CMBBaseConstant.CMB_OP_GREATER:
                            System.out.println("\t\t\t.GT."); break;
                        case CMBBaseConstant.CMB_OP_BETWEEN:
                            System.out.println("\t\t\t.BETWEEN."); break;
                        case CMBBaseConstant.CMB_OP_NOT_BETWEEN:
                            System.out.println("\t\t\t.NOTBETWEEN."); break;
                        case CMBBaseConstant.CMB_OP_IN:
                            System.out.println("\t\t\t.IN."); break;
                        case CMBBaseConstant.CMB_OP_NOT_IN:
                            System.out.println("\t\t\t.NOTIN."); break;
                        case CMBBaseConstant.CMB_OP_LIKE:
```

```
                        System.out.println("\t\t\t.LIKE."); break;
                    case CMBBaseConstant.CMB_OP_NOT_LIKE:
                        System.out.println("\t\t\t.NOTLIKE."); break;
                    default:  break;
                }
            }
        }
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

The Schema Management bean contains all information about search templates, their names, and criteria. The listTemplates function retrieves and prints out the list of all template names. getSearchCriteria is used by listSTCriteria, which lists information about all the search criteria for a single specified template.

```
           catch (CMBConnectFailedException ex) {
        System.out.println("Connection to back-end server failed...");
        ex.printStackTrace();
    }
    catch (CMBNoConnectionException ex) {
        System.out.println("There is no current connection, please logon..");
        ex.printStackTrace();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}
private void getSearchCriterion(CMBSearchTemplate stObj)
{
    try {
        String critName  = ui.getString("Please enter criterion name: " );
        Vector critList  = stObj.getSearchCriteria();
        CMBSTCriterion critObj = null;
        for (int i=0; i<critList.size(); i++) {
            critObj = (CMBSTCriterion)critList.elementAt(i);
            if (critObj.getName().equals(critName))
                break;
        }

        if (critObj == null) {
            System.err.println("Invalid criterion name, please try again...");
            return;
        }

        int    critOp    = ui.getInteger("Please enter search operator: " +
            "\nEQ: 1\t\tNEQ: 2\t\tLEQ: 3\t\tLT: 4" +
            "\nGEQ: 5\t\tGT: 6\t\tBETWEEN: 7\tNOTBETWEEN: 8" +
            "\nIN: 9\t\tNOTIN: 10\tLIKE: 11\tNOTLIKE: 12\n");
        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[l] + ": " + values[l]);
                    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);  //borowed 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.

Information Mining beans

For more information about Information Mining beans, see “Building an information mining application using the beans” on page 333 and “Example of an information mining application based on JSPs” on page 382.

Working with visual beans

Visual beans allow you to integrate the functionality of EIP or other content servers into Java applications based on Swing. Visual beans perform basic tasks that are common to many applications, such as logging on, searching, displaying and viewing results, updating documents, and viewing version information.

Each visual bean has a Connection property. This property must reference an instance of CMBConnection, the nonvisual bean that maintains the connection to the content servers. Any application built with the EIP visual beans must also contain an instance of the CMBConnection nonvisual bean.

CMBLogonPanel
This bean displays a panel to login to EIP or to content servers such as Content ManagerVersion 8.1 (CM 8.1). It also provides the window where federated users can modify the UserIDs and passwords on the content servers.

CMBSearchResultsViewer
This bean displays search results. When the search result returns folders, use CMBSearchResultsViewer bean to ″drill-down″ into the folder to see its contents. Items in the search results or folders can be selected and opened for viewing in a Windows Explorer style window

CMBSearchTemplateList
For servers that support search templates, this bean displays a list of available search templates and allows selection of a template.

CMBSearchTemplateViewer
For servers that support search templates, this bean displays a search template and provides fields for users to enter search criteria. It performs a search based on those criteria.

CMBSearchPanel
For all servers, the search panel displays a list of available entities, and provides fields for users to enter search criteria. It performs a search based on those criteria. The CMBSearchPanel is useful for performing searches on content servers that do not support search templates.

CMBFolderViewer
Displays the contents of one or more folders in a Windows Explorer style window

CMBItemAttributesEditor
Displays a window where users can update the index class and indexing attributes for an item

CMBDocumentViewer
Displays one or more documents by launching the appropriate viewer

CMBVersionsViewer
Displays version information for a document, if versioning is enabled.

CMBLogonPanel bean

The CMBLogonPanel bean (see Figure 20 on page 247) displays a window that lets users login to a content server, update user mappings, and change a password.

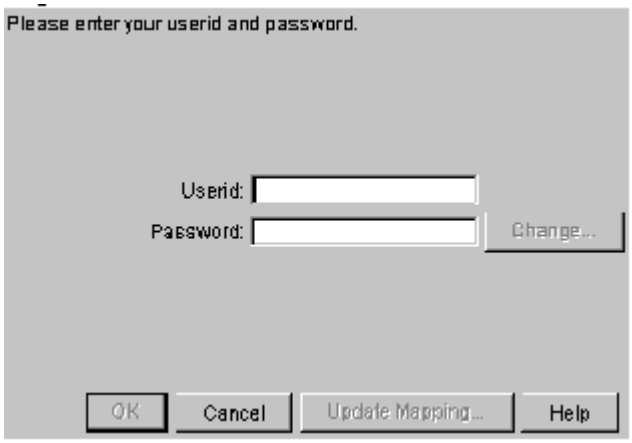

Figure 20. CMBLogonPanel bean

In the CMBLogonPanel bean, when a user clicks **Change**, the **Change Password** window appears (see Figure 21.) The user enters the old password, and enters the new password twice.

Figure 21. Change Password window

In the CMBLogonPanel bean, when a user clicks **Update Mapping** in the Logon wndow, the **Update Userid Mapping** window is displayed (see Figure 22 on page 248). When you Update Mapping, you update the user ID and password specified for a server. This function is available only when logging on to the EIP federated database.

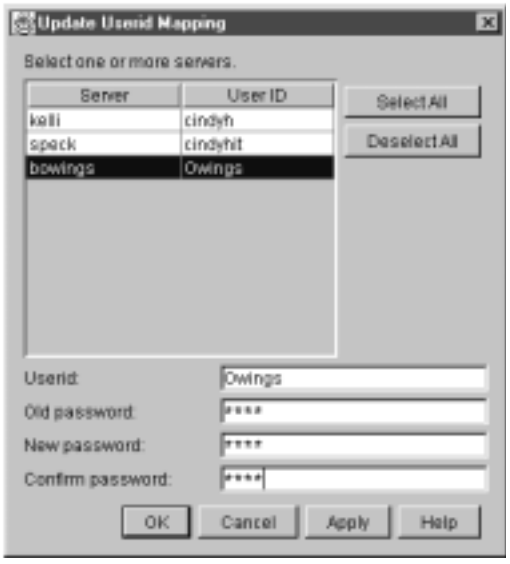

Figure 22. Update Userid Mapping window

At the top of the window is a list of all servers and a corresponding user ID. Users can select one or more servers from the list. Click **Select All** to select all servers. Users can specify a new user ID and (optionally) password after you select one or more servers. If you select one server, the user ID appears in the **Userid** field. If users select more than one user ID, the **Userid** field is blank.

Deselect All
Removes all server selections.

Apply Click to apply mapping and password changes without closing the window.

OK Click to accept changes and close the window.

Cancel
Click to close window without making changes.

CMBSearchTemplateList bean

The CMBSearchTemplateList bean has three styles. The image style, shown in Figure 23, uses one image for the backgrounds of the selected items and another for the unselected items. Figure 24 on page 249 shows the simple template list style. Figure 25 on page 249 shows the drop-down template list style.

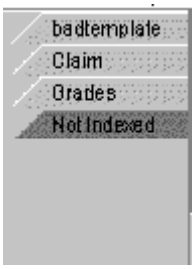

Figure 23. Image template list style

Figure 24. Simple template list style

Figure 25. Drop-down template list style

CMBSearchTemplateViewer bean

The CMBSearchTemplateViewer bean (see Figure 26) displays a window where users can specify search criteria according to the search template defined by the system administrator. The CMBSearchTemplateViewer bean launches a search and generates the CMBSearchResults event to return the search results.

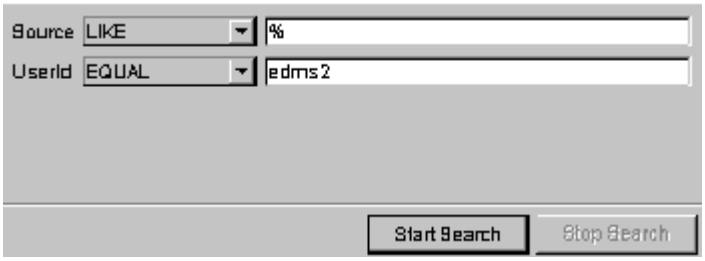

Figure 26. CMBSearchTemplateViewer bean

The CMBSearchTemplateViewer bean lists search criteria such as Source or Userid. Each search criteria has a label, an operator drop-down box, and a text field. The `BETWEEN` or `NOTBETWEEN` operator display has two text fields. The `IN` or `NOTIN` operators have a multi-line text area. Each value should be entered on a separate line.

Text search areas

The CMBSearchTemplateViewer bean can contain areas that allow users to perform a search on full text or index attributes. A full text search area on the template can be as simple as a text field with a label.

Users must match the query syntax for a free or boolean text search when they enter the query string in the text field (see the DKDatastoreTS class). Turn to the online API reference for details.

Validating or editing fields of the CMBSearchTemplateViewer

You can provide validation logic for the CMBSearchTemplateViewer bean to modify search criteria entered by the user. Do this by providing a handler for the CMBTemplateFieldChangedEvent. The current values of the search criteria are stored in the CMBTemplate returned by the `getTemplate` method prior to this event being called. You can examine and change the criteria. After the event handling is complete, the new values display.

CMBSearchPanel bean

The CMBSearchPanel bean displays a window where users can specify search criteria according to the entities available on the current content server. The CMBSearchPanel bean launches a search and generates the CMBSearchResultsEvent to return the search results. The CMBSearchPanel lists all the available entities in the drop-down list at the top of the window. When an entity is selected, the CMBSearchPanel displays the attributes of the entity. Each attribute has a label, an operator drop-down box, and a text field. A range operator display, such as BETWEEN or NOTBETWEEN, has two text fields. An operator that takes multiple values, such as the IN or NOTIN operator, has a multi-line text area. Each value should be entered on a separate line in the multi-line text area.

CMBSearchResultsViewer bean

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

Figure 27 shows the CMBSearchResultsViewer bean with the **Search Results** folder selected.

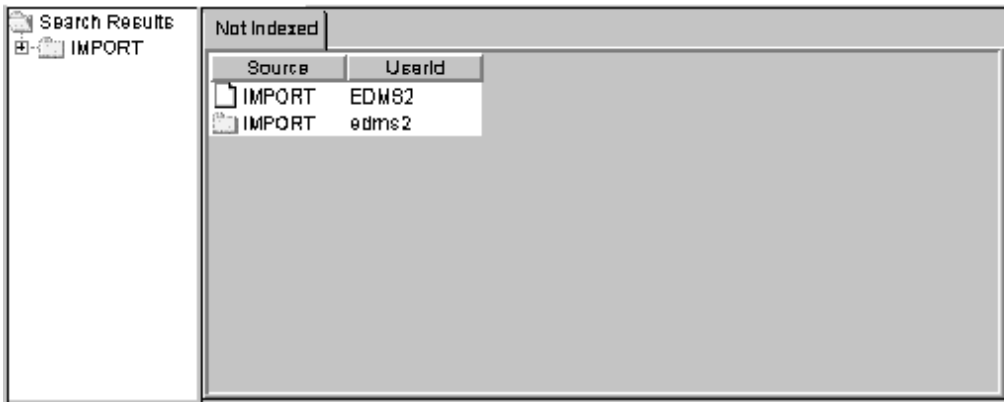

Figure 27. CMBSearchResultsViewer bean

CMBSearchResultsViewer Tree pane

The tree pane (on the left) contains a main folder labeled **Search Results**. Beneath that folder is each folder found in the search. The tree pane is optional. Remove it by setting the TreePaneVisible property: `setTreePaneVisible(false)`.

CMBSearchResultsViewer Details pane

The details pane displays the contents of the folder selected in the tree pane. When users select the **Search Results** folder, a tab appears on the notebook containing the search template name. When users select a different folder within **Search Results**, one or more tabs display: one for each index class in the folder. The tab names have the form:

index class @ server

where *index class* is the index class or item type name and *server* is the content server name. The table columns change to display the attributes according to the index class or item type. Multiple selection is supported in the details pane. Turn off Multiple selection by setting the MultiSelectEnabled property: `setMultiSelectEnabled(false)`. If an item type is hierarchical, the attribute values of the children are displayed in the

table with column headers of the form: child component name / attribute name, where child component name is the name of the child component, and attribute name is the name of the child component's attribute. For example, if an item type called `Journal` has a child component called `Author`, and the `Author` child component has an attribute called `Last Name`, the column header is: `Author/Last Name`.

Pop-up menus

A pop-up menu offering Sort options appears when a user right-clicks on a table column heading. Users click **Sort Ascending** to sort the items in the table in ascending order. Users click **Sort Descending** to sort the items in descending order. Another pop-up menu appears when a user right-clicks a folder other than the **Search Results** folder in the tree pane, or right-clicks a document or folder in the details pane. The pop-up menu lets users View folder details in the tree pane, or Edit Attributes for folders.

Optional: Use the CMBViewFolderEvent rather than show the details of the folder within the CMBSearchResultsViewer bean. Use the event to make the CMBFolderViewer bean display the selected folder's contents.

Double-click action

Double-clicking a folder in the tree pane or an item in the details pane performs the same action as clicking in the **View** pop-up menu item. If you suppress the default item pop-up menu, a CMBItemActionEvent occurs.

Overriding pop-up menus

You can override the pop-up menus on the CMBSearchResultsViewer and CMBFolderViewer with either a different pop-up menu or no pop-up menu. To turn off the default menus, use `setDefaultPopupMenu(false)`.

When the user right-clicks a folder in the tree pane, a CMBFolderPopupEvent is generated. When the user right-clicks an item in the details pane, a CMBItemPopupEvent is generated. You can use a handler to provide a different pop-up menu.

CMBFolderViewer bean

The CMBFolderViewer bean displays a tree pane that looks like the CMBSearchResultsViewer bean. There is no main **Search Results** folder. Figure 28 on page 252 shows the tree and details panes of the CMBFolderViewer bean.

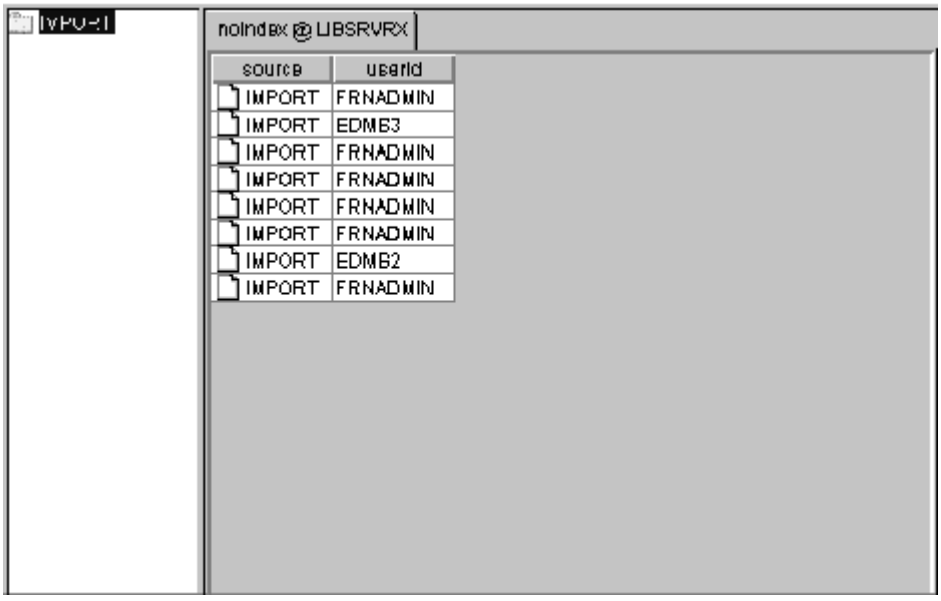

Figure 28. CMBFolderViewer bean

The CMBFolderViewer bean displays a tree of folders on the left pane. The right pane displays a notebook of tables of the documents contained by folder selected in the tree pane. A resizeable splitter separates the tree and notebook panes.

CMBFolderViewer Tree pane
The tree pane contains folders. Nested folders appear beneath each folder.

CMBFolderViewer Details pane
The details pane contains the contents of the folder that is selected in the tree pane. The contents display in a notebook with a tab for each entity (index class, item type, or other) and server, that the items in the table are indexed under. The tab names have the form:

```
index class @ server
```

where *index class* is the name of the index class and *server* is the name of the server. Within each notebook page is a table displaying the documents and folders within the selected folder. The table columns change to display the attributes according to the index class.

Pop-up menus

The behavior of the pop-up menus for the folder viewer is identical to that of the search results viewer.

Double-click action

Double-clicking in the folder viewer is identical to that of the search results viewer.

CMBDocumentViewer bean

The CMBDocumentViewer bean provides capabilities to view documents by either launching or embedding content-type specific document viewers. There are two types of viewers supported:

1. Java-based viewers. These viewers must extend the class CMBJavaDocumentViewer.

2. Non-Java viewers. Any executable may be launched as a viewer for a particular content-type.

If the Visible property is set to false, the viewer is always displayed in a separate window. If the Visible property is true, the viewer will be displayed within the display region of the CMBDocumentViewer bean if possible. (Currently, this is only possible for Java-based viewers.)

CMBJavaDocumentViewer is an abstract class extended by providers of Java-based document viewers that plug into the CMBDocumentViewer bean. These viewers can display the documents in the visible space of the CMBDocumentViewer bean or in separate windows on the screen.

A call to CMBDocumentViewer terminate() waits until all document closed events are processed. If you call terminate() from within the document closed event handler, deadlock may occur and the program hangs. To avoid this problem, when calling terminate() from within the onDocumentClosed(CMBDocumentClosedEvent) event handler, call the CMBDocumentViewer.terminate() method using SwingUtilities.invokeLater(Runnable). This adds the terminate() call to the end of the event queue and continues with the other events in the queue (such as handling the other document closed events) before calling the terminate method. Here is an example:

```
...
CMBDocumentViewer viewerBean = new CMBDocumentViewer();
...
...
public void onDocumentClosed( CMBDocumentClosedEvent event )
{
...
SwingUtilities.invokeLater( new Runnable() {
 public void run(){
   try
  {
       viewerBean.terminate();
   }catch( Exception e){
     System.out.println(e);
   }
  }
});
...
}
```

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 `wangimg.exe`, for viewing TIFF documents on Windows:

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

CMBItemAttributesEditor bean

The CMBItemAttributesEditor bean (see Figure 29) displays a window for viewing and modifying the index class and indexing attributes of a folder or document.

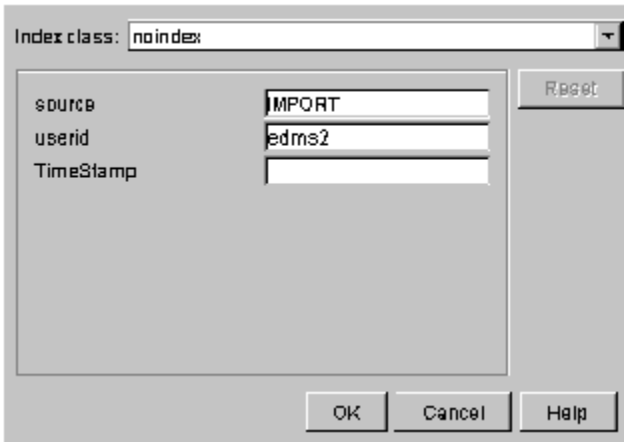

Figure 29. CMBItemAttributesEditor bean

A list containing all available entities appears at the top of the window. The current entity is selected by default. A list of attributes for that entity appears beneath the entity. The text fields (first name, last name, and so forth) initially contain the current values for the item.

If users select a new entity, any attributes with the same names as the previously-selected entity have their values propagated to the like-named attributes in the new entity.

Clicking **Reset** returns the entity and attributes to their original values.

Clicking **OK** updates the entity and attributes and triggers events before and after the update. You can use the event before the update to validate fields or complete missing fields before the update is performed. This event can veto the specified update.

Vetoing changes in the CMBItemAttributesEditor

You can provide additional validation logic to the CMBItemAttributesEditor that verifies attribute values entered by the user and modifies them, or rejects an update, if the values are not valid. Do this by providing a handler for the CMBEditRequestedEvent.

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;
        }
    }
});
```

CMBVersionsViewer bean

The CMBVersionsViewer bean displays a table of versioning attributes for a single document or item. The versioning attributes displayed are: the version number, the userid of the creator, the timestamp when the version was created, the userid of the latest updater, and the timestamp of the last update. From the versions viewer, you can view the different versions of an item or update the attributes of an item.

General behaviors for visual beans

The following sections describe properties and behaviors that are common among visual beans.

Properties

This section describe three properties shared by visual beans.

Connection

Each bean has a Connection property, which refers to an instance of the CMBConnection non-visual bean. You must set the Connection property for the visual bean to operate correctly.

CollationStrength

All beans that perform sorting have a CollationStrength property. The values defined for CollationStrength property are the same values defined for the java.text.Collator class of Java.

Hiding/Showing buttons

You can hide or show the Push buttons that appear on all visual beans. Use the set*name* ButtonVisible 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 31.

Table 31. Visual beans and key events

Visual bean	Key events
CMBSearchTemplateList	CMBTemplateSelectedEvent
CMBSearchTemplate Viewer	CMBSearchStartedEventCMBSearchResults Event
CMBSearchResultsViewer	CMBViewDocumentEvent CMBViewFolderEvent-CMBEditItemAttributesEvent
CMBFolderViewer	CMBViewDocumentEvent CMBEditItem AttributesEvent

Table 31. Visual beans and key events (continued)

CMBDocumentViewer	CMBDocumentOpenedEvent CMBDocument Closed Event
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.addCMBConnectionReplyListener(new
CMBConnectionReplyListener() {
   public void onCMBConnectionReply(CMBConnectionReplyEvent 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("cmbdb");
   } 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, CMBConnection 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, CMBConnection 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();
    }
}
```

InFigure 30 on page 262, 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);
```

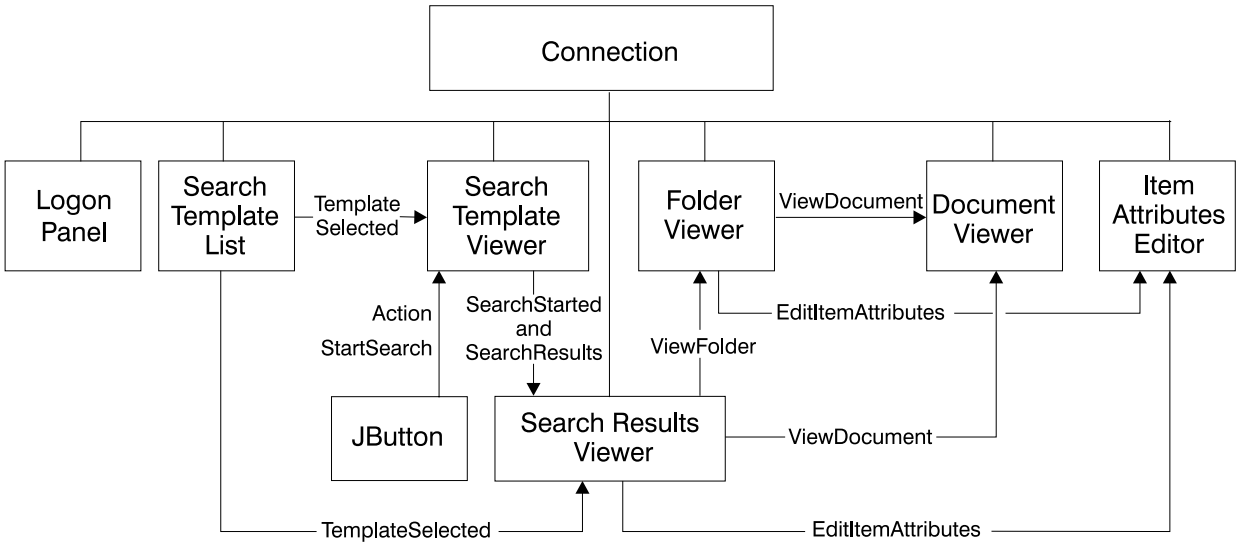

Figure 30. Visual bean connections

Figure 30shows nine beans. A JFrame or other container bean would be the parent of all of these beans. One possible order of events during run time might be:

1. The user enters a user ID and password into the logon window and clicks **OK**. The CMBLogonPanel bean invokes the `connect` method of the CMBConnection bean to establish the connection to the server.
2. The connection bean establishes the connection. The CMBSearchTemplateList bean retrieves and displays the list of search templates for that user ID. (No methods need to be invoked to cause this to happen. The CMBSearchTemplateList bean is listening to the appropriate events of the CMBConnection bean. CMBSearchTemplateList sets up the listeners when a CMBConnection bean associated itself with it using the `setConnection` method.)
3. The user selects a search template from the list. The CMBSearchTemplateList bean generates a CMBTemplateSelectedEvent. Both the CMBSearchTemplateViewer and the CMBSearchResultsViewer are listening for the event. The CMBSearchTemplateViewer displays the appropriate template. The CMBSearchResultsViewer clears and displays columns in the details pane as defined by the template.
4. The user completes the template, and either presses Enter or clicks **Search**. If the user clicks **Search**, the action event handler invokes the `startSearch` method. If the user presses Enter, the `startSearch` method is invoked implicitly.
5. The CMBSearchTemplateViewer bean validates the template fields to determine whether a search can begin. If the search can begin, a CMBSearchStartedEvent is generated. CMBSearchResultsViewer listens for a CMBSearchStartedEvent and clears the results in preparation for new search results.
6. As the search progresses, CMBSearchResultsEvents are generated to provide partial search results to the CMBSearchResultsViewer. (When the search is completed the CMBSearchCompleted event is generated. This event can be used to enable the **Search** 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 9. Using the C++ application programming interfaces

Enterprise Information Portal provides a set of C++ classes and APIs you can use to access and manipulate locally or remotely stored content.

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, Content Manager Version 8 Release 1 text search feature.

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**).

Important: The information in this section, except *Setting up the Windows and AIX environment*, about Content Manager applies to earlier versions of Content Manager (pre-Version 8.1).

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 32 lists Library, AIX shared and DLL requirements.

Requirement: To use C++, you must install DB2 Client support and the Client Configuration Assistant on all remote servers running the Enterprise Information Portal database. The your 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*.

Note: `cmbcm816.lib` is for release build and `cmbcm816d.lib` is for debug build.

Table 32. Library, shared objects and DLL environment information

Library	Shared objects for AIX	Windows DLLs
cmbcm816.lib	libcmbcm81366.a	cmbcm816.dll
cmbcm816d.lib	libcmbdb281366.a	cmbcm816d.dll
cmbdl816.lib, cmbdl816d.lib	libcmbdl81366.a libcmbdlfac81366.so	cmbdb2816.dll cmbdb2fac816.dll
cmbip816.lib		cmbdb2816d.dll cmbdb2fac816d.dll
cmbip816d.lib	libcmbdb2fac81366.so	cmbdl816.dll
cmbv4816.lib	libcmbdj81366.a	cmbdl816d.dll
cmbv4816d.lib	libcmbdjfac81366.so	cmbdlfac816.dll
cmbdd816.lib		cmbdlfac816d.dll
cmbdd816d.lib		cmbodbc816.dll cmbodbcfac816.dll

Table 32. Library, shared objects and DLL environment information (continued)

Library	Shared objects for AIX	Windows DLLs
cmbdes816.lib		cmbodbc816d.dll cmbodbcfac816d.dll
cmbdes816d.lib		cmbfed816.dll cmbfedfac816.dll
cmbdb2816.lib		cmbfed816d.dll cmbfedfac816d.dll
cmbdb2816d.lib		cmbdd816.dll cmbddfac816.dll
cmbdj816d.lib		cmbdd816d.dll cmbddfac816d.dll
cmbdj816.lib		cmbip816.dll cmbipfac816.dll
		cmbip816d.dll cmbipfac816d.dll
		cmbv4816.dll cmbv4fac816.dll
		cmbv4816d.dll cmbv4fac816d.dll
		cmbdes816.dll cmbdesfac816.dll
		cmbdes816d.dll cmbdesfac816d.dll
		cmbdj816.dll cmbdjfac816.dll
		cmbdj816d.dll cmbdjfac816d.dll
		de_db2.dll
		de_db2_d.dll
		de_ora.dll
		de_ora_d.dll

Setting AIX environment variables

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

Set the following environment variables:

NLS path

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

PATH

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

LIBPATH

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

INCLUDE

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

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 8.1 → Development Window.** As an alternative, you can run `CMBenv81.bat` in a DOS command prompt to set up the environment.

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

PATH

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

where *x* is your drive

INCLUDE

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

where x is your drive

Building C++ programs

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

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

Content Manager Version 8 Release 1 offers an integrated text feature, which does not require a separate text search facility. See “Understanding text search” on page 426.

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 Chapter 14, “Working with other content servers in C++” on page 467.

Tracing information

You can use any of the following environment variable settings to get tracing information.

For text queries using Text Search Engine

Content Manager Version 8 Release 1 offers an integrated text feature, which does not require a separate text search facility. See “Understanding text search” on page 426.

The Text Search Engine and all of its functions can only be used with Content Manager Version 6 or Version 7 servers.

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

Content Manager Version 8 Release 1 offers an integrated text feature, which does not require a separate text search facility. See “Understanding parametric search” on page 426.

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

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

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;
}
```

Constants

The constants provided for use with the Enterprise Information Portal APIs are defined in DKConstant.h and DKConstant2.h. Additional constants are defined for specific datastore in header files of the form DKConstant*XX*.h; for example, additional constants unique to Content Manager are in DKConstantICM.h

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

A DKDatastore*xx* (where *xx* is the suffix representing the specific content server, for example, Content Manager (ICM), 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

Each datastore provides functions for connecting to the content server associated with it and disconnecting from it. The following example shows how to connect to an earlier version of 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 functions. 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 datastore provides a function for listing the content servers the datastore can connect to.

Restriction: The Domino.Doc datastore does not provide such a function.

The list of servers is returned in a sequential collection of DKAny objects containing DKServerInfo*xx* (where *xx* is the datastore suffix in which you want to work, for example, an earlier version of Content Manager (DL), ImagePlus for OS/390 (IP), and so forth) objects. After you obtain a DKServerDef*xx* object you can retrieve the server name and server type, and use the server name to establish a connection. 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 entities and attributes

Each content server provides functions for listing its entities and attributes available for that content server.

The list of entities is returned in a DKSequentialCollection object of dkEntityDef objects. The list of attributes for an entity is returned in a DKSequentialCollection object of dkAttrDef objects. After you obtain a dkAttrDef object, you can retrieve information about the attributes, such as its name and type, and use the information to form a query. For specific information about these two functions, see the *online API reference*.

The following example shows how to retrieve the list of index classes and attributes from an earlier version of 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` function to delete the attribute definition inside of the collection. See "Managing memory in collections" on page 295 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 earlier versions of Content Manager or item types in Content Manager Version 8 Release 1, 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` functions 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` function.

Every DKDDO has a persistent identifier (PID). The PID contains information for locating the attributes in the datastore. For example, in earlier versions of Content Manager, a DKDDO represents an item, which could be a document or a folder.

Creating a DKDDO

The simplest way to create a DKDDO is to call its constructor without any parameters.

```
DKDDO addo;
```

If you know the number of DDO attributes you want, you can use the constructor, passing it the number as shown in the following example:

```
DKDDO bddo(10);
```

The DKDDO bddo is constructed and can hold up to ten attributes. Creating the DDO with a set number of attributes is more efficient, because The DDOaddo must grow dynamically to accommodate more attributes.

You can create a DKDDO by supplying a content server and an object type as input.

```
// create an earlier 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 functions. For the add function, 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
// ----- Create a DDO with the PID and associate it with dsDL
DKDDO* ddo = new DKDDO(&dsDL, pid);
```

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

Adding data items and properties

Suppose the item type GRANDPA has the attributes shown in Table 33.

Table 33. 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 4 on page 33 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 he 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 273.

```
// 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 function 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.

You delete a DKDDO by calling its destructor. The DDO is deleted in memory, but the persistent copy in the content server is unchanged.

You use the del function 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 DKBlob*xx*, (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. DKBlob*xx* requires the datastore DKDatastore*xx* 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 DKPidXDO*xx* 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 functions of the DKBlob*xx* 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). 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). 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 functions (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 function, 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 teh 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;
  partId = 0;                          //let system decide the part ID
  itemId = "CPPIORH4JBIXWIY0";         //existing item ID
  fileName = "g:\\test\\cheetah.gif";  //a Windows  file to be added
  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->setItemId(itemId);                   //set item ID
    axdo->setPidObject(apid);                  //set Pid to XDO
    axdo->setContentClass(DK_DL_CC_GIF);       //set ContentClass
    axdo->setContentFromClientFile(fileName);  //set file content to buffer area
    axdo->add();                               //add from buffer
    //display the part ID after add
    cout<<"after add partId="<<((DKPidXDODL*)
         (ablobDL->getPidObject()))->getPartId()<<endl;
    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
```

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

```
void main(int argc, char *argv[])
{
  DKDatastoreDL dsDL;
  DKString itemId, fileName;
  unsigned long partId;
  partId = 37;                              // part ID 37 not being used yet
  itemId = "CPPIORH4JBIXWIY0";              // existing item ID
  fileName = "/u4/mmdb/test/choice.gif";  // file(in AIX system) to be add
  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->setRepType("ABCD");               // set representation type
    axdo->setContentClass(DK_DL_CC_GIF);   // set ContentClass
    axdo->add(fileName);                    // add from file
    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
```

Adding an annotation object to an XDO: To add an annotation object, you must insert the following statements in your program before the add function.

```
//------ 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, user ID, password
 dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
 cout << "datastore connected" << endl;

 // *** create xdo and pid
 DKBlobDL* axdo = new DKBlobDL(&dsDL);
 DKPidXDODL*  apid = new DKPidXDODL;
 apid ->setPartId(partId);
 apid ->setPrimaryId(itemId);
 axdo ->setPidObject(apid);
 // *** you must use the content class DK_DL_CC_IBMVSS for a media object
 axdo ->setContentClass(DK_DL_CC_IBMVSS);
 cout <<"itemId= "<<axdo->getItemId()<<endl;
 cout <<"partId= "<<axdo->getPartId()<<endl;
 cout <<"repType= "<<axdo->getRepType()<<endl;
 cout <<"content class="<< axdo->getContentClass()<<endl;

 // *** 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, user ID, password
    dsDL.connect("LIBSRVRN","FRNADMIN","PASSWORD");
    cout << "datastore connected" << endl;

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

    DKBoolean flag2 = axdo->isCategoryOf(DK_DL_MEDIA_OBJECT);
    cout <<"isMediaObject? = "<<flag2<<endl;
    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 function.

```
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 function, which returns a `TypeCode` object, that is, `tc_null` for null, `tc_short` for short, and so forth. Refer to the *online API reference* for a complete listing of type codes.

Managing memory in DKAny

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

Using constructors

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

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

Getting the type code

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

```
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 functions. DKSequentialCollection provides the concrete implementation of those functions. 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 functions

DKSequentialCollection provides functions for adding, retrieving, removing, and replacing its members. In addition, it has the apply and sort functions. 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` functions; and its subclass DKSequentialIterator, which contains more functions. An iterator is created by calling the `createIterator` function on the collection. This function creates a new iterator and returns it to you. Use the following code to iterate over a collection:

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

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

DKSequentialIterator provides additional functions 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 functions.

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

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

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

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

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

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

Sorting the collection

Use the `sort` function to sort collection members in either ascending or descending order based on a specified key. You must pass a sort object and the desired order. The interface for sort objects is defined in `dkSort.hpp`; so you can write your own sort function for sorting your specific collection. The following example illustrates how to sort a collection of DDOs based on each DDO's item ID:

Programming tips

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

Understanding federated collection and iterator

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 31 on page 297 shows the structure and behavior of DKFederatedCollection.

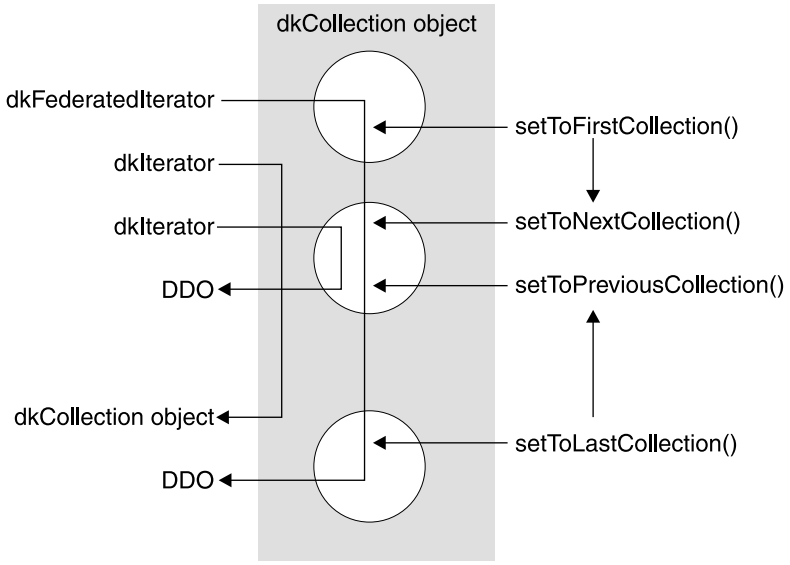

Figure 31. Structure and behavior of DKFederatedCollection

In Figure 31, 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` function in dkFederatedIterator sets the position to the first DDO of DKFederatedCollection. In this case, it is the first element of the first DKResults collection object. At this point, if the `setToNextCollection` function is invoked, it sets the iterator position to the first DDO of the second DKResults collection.

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

Querying a content server

You can search a content server and receive results in a dkResultSetCursor or DKResults object. You can create a query object to represent your query, then invoke the `execute` function or `evaluate` function of the query object. With the help of its content servers, the query object performs query processing tasks, such as preparing and executing a query, monitoring the status of a query execution, and storing the results.

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

You can also use dkResultSetCursor to run a query again by calling the `close` and `open` functions. This is described in "Using the result set cursor" on page 306.

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 307 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 for earlier versions of Content Manager 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 this earlier version of a Content Manager server uses 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 (for earlier versions of Content Manager).

```
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 function 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 pre-Version 8.1 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 293 for more information.

Executing a parametric query from the content server

Each content server provides a function 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 (TExecuteDL.cpp) is available in the Cmbroot/Samples/cpp/dl directory.

Evaluating a parametric query from the content server

Each content server provides a function 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 function 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/dl directory.

Executing a text query from the datastore

Each content server provides a function 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/dl directory.

Evaluating a text query from the datastore

Each content server provides a function 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 *pDOBase = 0;
DKMatchesInfoTS *pMInfo = 0;
DKMatchesDocSectionTS *pMSect = 0;
DKMatchesParagraphTS *pMPara = 0;
DKMatchesTextItemTS  *pMText = 0;
long i = 0;
long j = 0;
long k = 0;
long m = 0;
long lCCSID = 0;
long lLang = 0;
long lOffset = 0;
long lLen = 0;
long numberSections = 0;
long numberParagraphs = 0;
long numberTextItems = 0;
long numberNewLines = 0;
DKString strDoc;
DKString strSection;
DKString strText;
DKString strDID;
DKString strXNAME;
DKString strDataName;
DKPid pid;
while (pCur->isValid())
 {
   item = pCur->fetchNext();
```

```
   if (item != 0)
   {
    pid = item->getPid();
    // Process the DKDDO
    for (i = 1; i <= item->dataCount(); i++)
    {
     anyObj = item->getData(i);
     strDataName = item->getDataName(i);
     if (strDataName == "")
     {
      strDID = pid.getId();
     }
     if (strXNAME == "")
     {
      strXNAME = p->getObjectType();
     }
     switch (anyObj.typeCode())
     {
      ...
     }
    }
    // 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 function 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 function 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` function 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.

Chapter 10. Viewing and annotating documents

You can use a document viewer to access and annotate documents contained in your content servers. You can create a custom document viewer using the EIP Java viewer toolkit. You can also create custom viewer applets and applications to integrate into EIP or standalone applications.

The Java viewer toolkit classes include action objects that provide the following function:

- Page viewing options
  - Rotate documents: 90 degrees clockwise, 90 degrees counter clockwise, and 180 degrees
  - Zoom: in and out
  - Scale: 25%, 50%, 100%, 150%, 200%, and 400%
- Invert
- Enhance
- Print
- Close the current document
- Close all the documents
- Create and edit annotations
  - Write
  - Highlight
  - Draw a box
  - Draw a circle
  - Draw a line
  - Draw an arrow
  - Add text
  - Stamp
  - Add a note
  - Erase
  - Hide or show
  - Default cursor mode
  - Bring annotation to the front
  - Bring annotation to the back
  - Change annotation properties
  - Undo and redo annotation operations
  - Cut, copy, paste, and delete annotations
  - Save document (only annotations are saved)
- Navigate documents or a pages within a document

  Navigate pages: first page, previous page, go to a page, next page, and last page

  Navigate documents: first document, previous document, go to a document, next document, and last document
- Thumbnails
  - Hide or show thumbnails

– Page navigation using thumbnails
– Panning and zooming

Viewer architecture

The Java viewer toolkit contains a Document Viewer and Document Services bean. The beans provide a mechanism for integrating the viewer into EIP based applications.

The toolkit also contains the Generic Doc Viewer, Streaming Doc Services, and Annotation Services classes. The classes enable you to use the viewer in applications where you cannot use the beans such as in standalone viewing or in distributed process situations where the connection to content stores is not local.

Streaming Doc Services manages a set of document processing engines that parse documents, render pages and provide you the capability to manipulate document pages. The engines provide parse image-page documents, such as TIFF and IOCA, as well as text and rich text documents and office formats. You can also write additional document engines and plug them into the viewer toolkit architecture to support additional formats or alternative rendering for document formats.

The Annotation Services class enables you to manipulate annotations. The annotation engine parses Content Manager specific annotation formats. You can write additional annotation engines.

The document engines

There are four document engines provided with EIP:

- MS-Tech Document Engine: This engine handles content types typically found in IBM Content Manager, rendering pages as images. Document types supported include TIFF, MO:DCA. This engine also supports GIF, JPEG, and plain text.
- INSO Document Engine: This engine supports Microsoft Office, Lotus SmartSuite, and other office document formats.
- AFP2Web Document Engine: This engine understands AFP and converts the AFP documents to HTML or PDF.
- Java Document Engine: This engine converts documents that are URL's to HTML with a forwarding link to the URL. This engine also converts XML to HTML by invoking XSLT.

The engines are public interfaces, but you should not program directly to them. Instead, use the interfaces provided by the Document Services bean or the Streaming Doc Services class.

Some of these engines are not pure Java and have portability limitations. This can restrict use of the toolkit on some platforms. The MS-Tech and Java engines are pure Java. The other engines contain platform specific logic that restricts their use to the Windows platform.

The annotations engine

EIP provides the MS-Tech annotation engine to handle Content Manager annotations. The MS-Tech engine supports Content Manager Version 8.1, Content Manager Version 7, and VI/400 annotations.

Creating a generic document viewer

To create a generic document viewer, you work primarily with the CMBGenericDocViewer class, which uses the CMBStreamingDocServices interface. CMBStreamingDocServices loads and renders documents. CMBStreamingDocServices uses a set of document engines to convert different document formats like TIFF and MO:DCA. To load, edit, and save annotations in documents use the CMBAnnotationServices interface.

Customizing the generic document viewer

You can customize the default configuration file, CMBViewerConfiguration.properties, located in the cmbview81.jar file, or you can create a new configuration file. Whether you create a configuration file or customize CMBViewerConfiguration.properties, you must keep the same file name and place it before cmbview81.jar in the class path.

To create a configuration file, complete the following steps:

1. You must specify the following entry for each toolbar specified by the toolbar name. This step specifies the toolbar's position on the main frame. The default position is NORTH. Specify the position for each toolbar that is listed.

   ```
   Toolbars=[<toolbar_name>[,<toolbar2_name>][,<toolbar3_name>]...]
   <toolbar_name>.position={NORTH|SOUTH|EAST|WEST}
   ```

2. Specify the actions that are to be added to the specified toolbar. Use the word 'separator' to insert a separator between the actions on the toolbar.

   ```
   <toolbar_name>.tools=[<action_name>[,<action2_name>]
   [,<action3_name>]...]

   <action_name>.label=<action_label>
   <action_name>.tooltip=<action_tooltip>
   <action_name>.icon=<icon_file_name>
   <action_name>.key=<key code>
   <action_name>.cursor=<cursor_file>
   <action_name>.hotspot=<x,y>
   ```

3. You cannot add new popup menus. However, you can add submenus and menu items to the three pre-defined popup menus.

   ```
   <popup_menu_name>.items=[<menuitem_name>[,<menuitem2_name>]
   [,<menuitem3_name>]...]
   <popup_menu_name>.submenu=[<submenu_name>[,<submenu2_name>]
   [,<submenu3_name>]...]
   <submenu_name>.label=<submenu_label>
   ```

If you have specified the appropriate configuration file, you are ready to build a standalone viewer application or applet. See the *Online API Reference* as you complete the following steps:

1. Create a private class that implements CMBStreamingDocServicesCallbacks.
2. Create CMBStreamingDocServices with the callbacks implemented in the first step.
3. Create a private class that implements CMBAnnotationServicesCallbacks.
4. Create CMBAnnotationServices with the callbacks implemented.
5. Create an instance of the CMBGenericDocViewer and initialize it with the CMBStreamingDocServices, CMBAnnotationServices, and the configuration properties file. Pass null for the properties file if you just want to us the default configuration file.

6. Call loadDocument() in CMBGenericDocViewer to load a document. This returns a CMBDocument instance.
7. Call loadAnnotationSet() on CMBGenericDocViewer to load any annotations. A CMBAnnotationSet object is returned. Use the method setItemHandle(CMBAnnotationSet, CMBItem) in CMBAnnotationServices.
8. Customize the look and feel of the viewer using the different methods provided by the viewer for the position, size of thumbnails, MDI or SDI view, and so forth.
9. Add the viewer to the application's or applet's main frame.
10. Prepare the menu bar by retrieving the actions from the generic document viewer and add the menu to the applications main frame.
11. Call showDocument() in the generic document viewer to display the document.
12. Call saveAnnotations(CMBDocument) on the CMBGenericDocViewer to save annotations on the document.
13. Call closeDocument(CMBDocument) to close a document or call closeAllDocuments() to close all the documents.

Figure 32 is an example of a generic document viewer that uses all of the default settings.

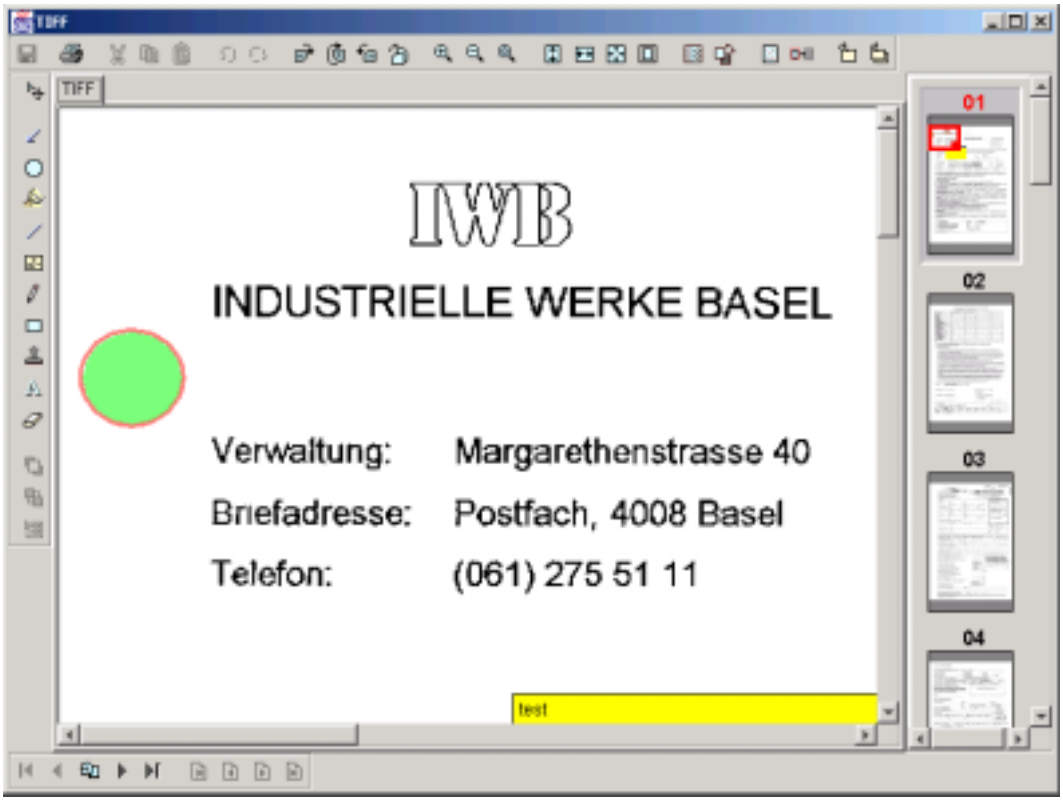

Figure 32. Generic document viewer

Example applications

To help you understand the Java Document Viewer toolkit, this sections describes five examples of applications that you can create. The examples below are not the only ways that you can use the toolkit.

Standalone viewer

You can use the Generic Document Viewer to implement a standalone viewer. You can use the standalone viewer to view files or documents that you retrieve from URL's. You can use the standalone viewers as pallets on web pages or for viewing documents that you obtained through e-mail, for example. Figure 33 illustrates a standalone viewer's architecture.

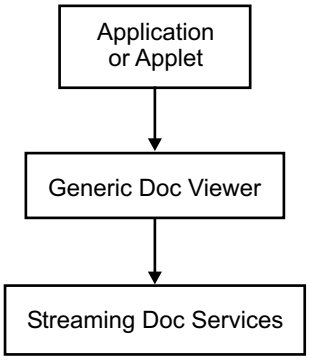

Figure 33. Standalone viewer

Java application

To create a production Java application use the EIP visual beans, which use the CMBDocumentViewer visual bean. This bean uses the Generic Document Viewer internally to display documents. The CMBDocumentViewer bean can also launch other viewers to view documents. Remember, however, that these viewers might have platform dependencies. Figure 34 illustrates the architecture you can use to build a Java application.

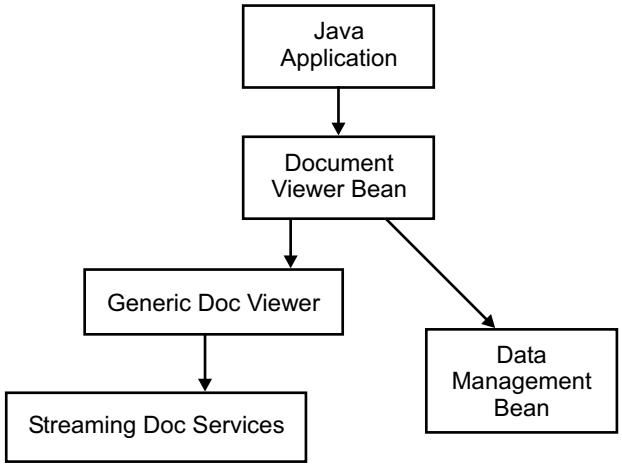

Figure 34. Java application

Thin client

You can use the CMBDocumentServices bean to perform server-side document conversions in a web based application. You can convert documents from content types that are not handled by the browser (documents that require a plugin or native application launch) to content types that are handled by the browser natively, such as HTML, GIF, JPEG, or for which plugins are readily available, such as PDF. Figure 35 on page 314 illustrates a thin client architecture.

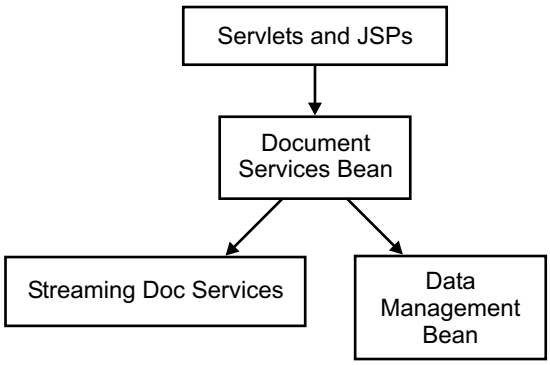

Figure 35. Thin client

Applet or servlet

A web based application can also provide document viewing and annotation editing capabilities using an applet or servlet approach. The applet can use the Generic Document Viewer to view the document. Some document types are stored in several parts on the content server to make sharing of common information, such as background forms, more efficient. The additional parts are requested by the Generic Document Viewer when needed. The applet containing the viewer satisfies the requests by sending HTTP requests to the servlet. On the servlet side, the content server using the CMBDataManagement bean obtains the requested information. Figure 36 illustrates an applet or servlet architecture.

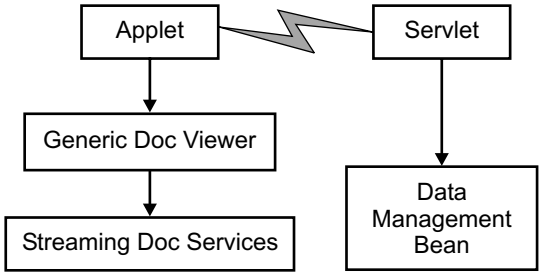

Figure 36. Applet or servlet

Dual-mode and applet or servlet

You can use a variation of the examples provided for web based viewing applications. Use CMBDocumentServices on the server and in the applet. This approach is useful when documents are not rendered in the applet but are converted on the server. This might happen when the underlying document engines on the applet and on the server have different capabilities.

For example, if the server is Windows NT or 2000 and the applet is running on OS/2, the applet might not be able to render all document types. The applet renders document formats that it supports, like TIFF. For types that the applet cannot render, such as Office formats, it requests conversion from the servlet. From the application user's perspective, the same interface is presented with the same functionality. However, server performance of server side converted documents might be slower than for locally rendered documents.

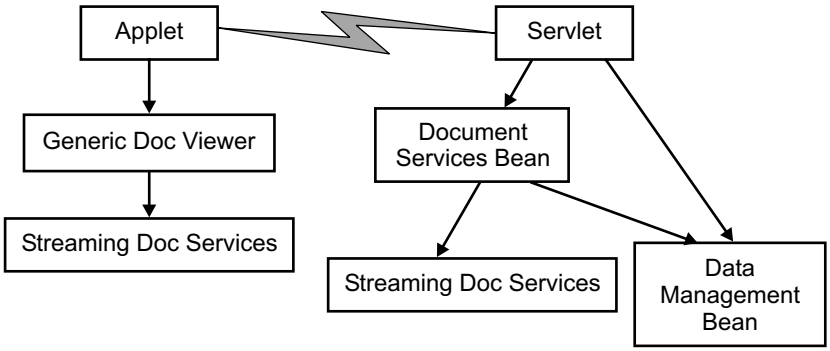

Figure 37. Dual-mode viewer

Working with the annotation services

The EIP 8.1 Java viewer toolkit provides capabilities for rendering and converting document annotations. Similar to the document services, pluggable annotation engines provide additional facilities that you can use in your applications to interpret different types of annotations. Figure 38 on page 316 illustrates how the generic document viewer and document services and annotation services fit together.

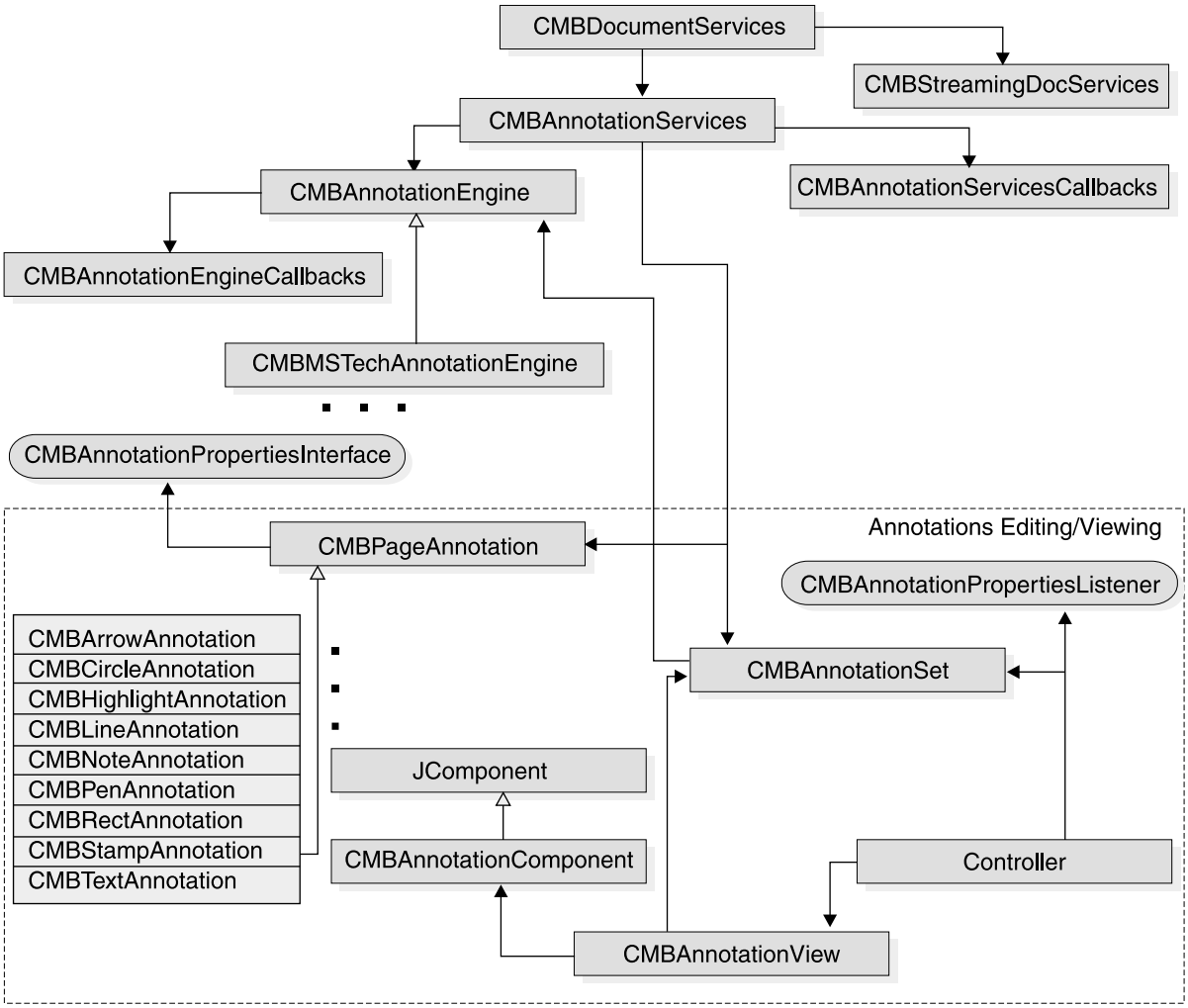

Figure 38. Annotation services and generic document viewer association

Using annotation services interfaces

CMBAnnotationServices provides the main interfaces used in the Java viewer toolkit. The annotation services enables you to load, manipulate, and save annotation objects using the annotation services, independently of backends. To work with annotations, you need to pass in annotation data as a stream and plug in a suitable annotation engine, which converts the annotation objects to CMBPageAnnotation instances. You can then manipulate and edit the annotations and save the annotations back into the original backend in the original format.

Figure 39 on page 317 shows the annotation services class diagram.

To implement the annotation engine, you must extend the abstract class CMBAnnotationEngine. The annotation engine uses CMBAnnotationServicesCallbacks and CMBAnnotationEngineCallbacks interfaces to communicate with an application and annotation services. The annotation engine in EIP 8.1 understands only the Content Manager annotation format. This annotation format is used by Content Manager Version 7, Content Manager Version 8. 1, and VI/400 backends.

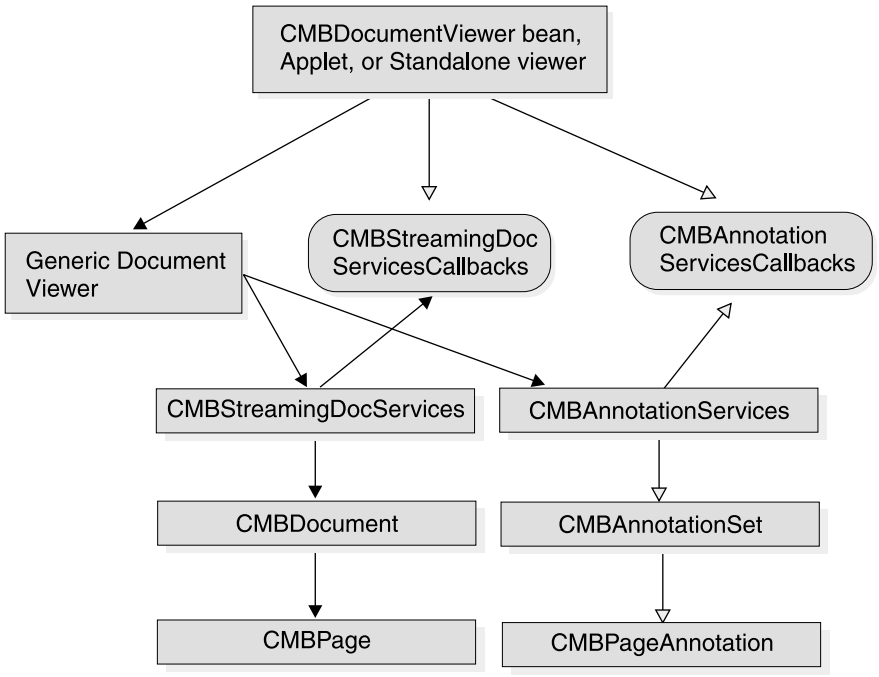

Figure 39. Annotation services class diagram

Understanding annotation editing support

The Model View Controller (MVC) design pattern is used to implement the annotations editing functionality. MVC acts as the model that represents the annotation data. CMBAnnotationSet has methods that operate on the data, but has no user interface. The CMBAnnotationSet class maintains the list of CMBPageAnnotation objects. Each document is associated with a CMBAnnotationSet object that represents its annotations. CMBAnnotationView acts as the view that presents the data from the model to the user. CMBAnnotation handles all the drawing of the annotations on the view component (a JComponent). CMBAnnotationComponent is a helper class that can be used as the view component on which the annotations are drawn. The controller is internal to the viewer toolkit and handles the mouse and keyboard events for annotation creation and editing.

CMBPageAnnotation is the base class that describes a single annotation on a page of a document. If you need to define additional types of graphical annotations, you must extend the CMBPageAnnotation class. There are nine types of Content Manager annotation types that you can create: CMBArrowAnnotation, CMBCirlceAnnotation, CMBHighlightAnnotation, CMBLineAnnotation, CMBNoteAnnotation, CMBPenAnnotation, CMBRectAnnotation, CMBStampAnnotation, and CMBTextAnnotation.

Building an application using the annotation services

This section includes the steps to follow and APIs to use to build an annotation services application. Refer to the online API reference for details about API usage.

1. Create a subclass of CMBAnnotationServicesCallbacks to implement the abstract methods to handle annotation callbacks.
2. Create an instance of CMBAnnotationServices.

```
CMBAnnotationServices annoServices = new
CMBAnnotationServices(annoServicesCallbacks);
```

3. Get an instance of CMBAnnotationSet by loading an annotation stream.

```
CMBAnnotationSet annotationSet = annoServices.loadAnnotationSet(annoStream,
 format, documentResolution, annotationPartNumber );
```

4. Prepare the annotation view.

```
CMBAnnotationComponent annoComponent = new CMBAnnotationComponent();
CMBAnnotationViewer annoView = annoServices.prepareAnnotationView(
annoComponent, annotationSet );
annoView.refreshEntireDrawingArea();
```

5. Add the annotation component to a Swing container like Jframe or Jpanel of the application.
6. You can now use the annotation services API to add new annotations, edit, or delete existing annotations.

```
annoServices.prepareToAddAnnotation();
annoServices.addAnnotation()
annoServices.removeAnnotation();
annoServices.reorderAnnotation();
...
```

7. Save the modified annotations.

```
annoServices.saveAnnotationset(annotationSet);
```

An Annotation Services sample is provided in the <CMBROOT>\Samples\java\viewer directory(TAnnotationEditor.java) directory.

Adding a custom annotation type to your application

To add a custom annotation type to the annotation services, complete the steps below. Refer to the online API reference for details about API usage.

1. Create a subclass of CMBPageAnnotation.`public class TImageAnnotation extends CMBPageAnnotation`
2. Define a constant for the custom annotation with a value larger than 100. The range of 1 to 99 is reserved.

```
public static final int ANN_IMAGE = 101;
```

3. Override the following methods of CMBPageAnnotation:

```
public void draw(Graphics2D g2);
public void drawOutline(Graphics2D g2);
public CMBPropertiesPanel getAnnotationPropertiesPanel();
```

4. Implement the CMBAnnotationPropertiesInterface interface to create the properties panel. The properties panel appears when a user chooses to edit the custom annotation properties.
5. Provide set and get methods specific to the custom annotation properties.
6. Add the custom annotation type.

```
annoServices.prepareToAddAnnotation(TImageAnnotation.ANN_IMAGE,
"TImageAnnotation",1);
```

The annotation type sample TImageAnnotation is included in the <CMBROOT>\Samples\java\viewer directory. The sample demonstrates how to add a custom annotation type to the annotation services.

Chapter 11. Using the EIP tag library and controller servlet

Enterprise Information Portal includes a JavaServer Pages tag library and a servlet that you can use when writing JSP or servlets for Web applications. Using the tag library reduces the need for Java scriptlets in JSP written to the EIP JavaBeans.

This tag library works with the servlet which can act as a controller of a model-view-controller design Web application and performs bean initialization and other actions.

Setting up the tag library and servlet

You must install the tag library and servlet on a Web server with IBM WebSphere® Application Server and configure the Web server to use them. For information about installing the tag library and servlet and configuring them, see *Planning and Installing Enterprise Information Portal.*

Using the tag library

The following JSP sample shows using the search templates tag to get a list of search templates:

```
<%@ taglib uri="cmb" prefix="cmb" %>
<%@ page import="com.ibm.mm.beans.*" %>
<jsp:useBean id="connection" scope="session"
                           class="com.ibm.mm.beans.CMBConnection" />
<%
    CMBSchemaManagement schema = connection.getSchemaManagement();
    CMBSearchTemplate[] searchtemplates = schema.getSearchTemplate();
    request.setAttribute("searchtemplates", searchtemplates);
%>
<html>
  <head>
  <title>Search Templates Tag Test</title>
  </head>

<body bgcolor="white">
<table border=2 cellspacing=3 cellpadding=3>
  <tr>
    <td><b>Available Search Templates</b></td>
  </tr>
  <cmb:searchtemplates>
  <tr>
    <td><%= searchtemplate.getName() %></td>
  </tr>
  </cmb:searchtemplates>
</table>
</body>
</html>
```

The `taglib` directive declares that the page uses the EIP tag library and associates the `cmb` prefix with it. Then the `searchtemplates` tag is called and the `getName()` method returns the name of each search template.

Conventions used in the tag library

The JSP tags have attributes to specify the beans that they use or generate. When those attributes are not specified, default values are used. These parameters are optional. Their values are picked up from local variables, and attributes in the request and session scope. When used in conjunction with the servlet toolkit, local variables, request attributes, or session attributes contain appropriate defaults. Table 34 shows the default beans and the scope in which they are assumed or placed. These conventions are also followed by the servlet; follow these conventions in any other servlets that you write to work with the tag library.

Table 34. Tag library conventions

Scope	Name	Type	Description
application	connectionPool	CMBConnectionPool	The connection pool bean that is shared across sessions
session	connection	CMBConnection	The instance of CMBConnection for the session
session	schema	CMBSchemaManagement	Schema management bean
session	data	CMBDataManagement	Data management bean
session	user	CMBUserManagement	User management bean
session	query	CMBQuesryService	Query service bean
session	traceLog	CMBTraceLog	Trace log bean; all of the other beans send their trace to this bean
session	docservices	CMBDocumentServices	Document services bean
request	item	CMBItem	The last item that you operated on
request	item	CMBItem[ ]	Collection of the items that you last operated on
request	workpacket	CMBWorkpacket	Workpacket bean
request	searchTemplate	CMBSearchTemplate	The selected search template
request	searchResutls	CMBSearchResults	The results from the last search

Tag summary

Connection related tags

<cmb:datasources″ **connection=**″*connection*″**>datasource ... </cmb:datasources>**

This tag iterates through the available data sources.

connection

Specify the name of a variable of type CMBConnection that contains the connection.

datasource

A string variable to contain the data source name as a string.

Schema related tags

<cmb:searchtemplates searchTemplates=*"searchTemplate"***> ...**
</cmb:searchTemplates>

This tag iterates through the available search templates.

searchTemplates
Specify the name of an array of type CMBSearchTemplate[] to contain the search templates.

searchTemplate
A variable of type CMBSearchTemplate to contain the search template.

<cmb:searchcriteria searchTemplate=*"searchtemplate"***>criterion ...**
</cmb:searchcriteria>

This tag iterates through the search criteria of a search template.

searchTemplate
Specify the name of the search template.

criterion
A variable of type CMBSTCriterion to contain the search criterion.

<cmb:displaycriteria searchTemplate=*"searchTemplate"***>criterion ...**
</cmb:displaycriteria>

This tag iterates through the search criteria that can be displayed for a search template.

searchTemplate
Specify the name of the search template.

criterion
A variable of type CMBSTCriterion to contain the search criterion.

<cmb:allowedoperators criterion=*"criterion"***>operator ... </cmb:allowedoperators>**

This tag iterates through the operators allowed for a search criterion.

criterion
Specify the name of a variable of type CMBSTCriterion to contain the search criterion.

operator
A string to contain the value of the operator.

<cmb:predefinedvalues criterion=*"criterion"***> value... </cmb:predefinedvalues>**

This tag iterates through the predefined values of a search criterion.

criterion
Specify the name of a variable of type CMBSTCriterion to contain the search criterion.

value A string to contain the predefined value of the search criterion.

<cmb:entities schema=*"schema"***>entity ... </cmb:entities>**

This tag iterates through the available federated entities.

schema Specify the name of a variable of type CMBSchemaManagement containing the schema.

entity A string to contain the name of the entity.

<cmb:attributes entity=*"entity"* **schema=***"schema"***>attribute ... </cmb:attributes>**

This tag iterates through the federated attributes of a federated entity.

schema Specify the name of a variable of type CMBSchemaManagement to contain the schema.

entity Specify the name of the entity.

attribute
A string to hold the name of a variable of type CMBAttribute that contains the attribute.

Search related tags

<cmb:searchresults searchresults=*"searchResults"***>item ... </cmb:searchresults>**
This tag iterates through the search results.

searchResults
Specify the name of a variable of type CMBSearchResults that contains the search results.

item A string to contain the name of a variable of type CMBItem to contain the item resulting from the search

Item related tags

<cmb:itemattributes item=*"item"***> attrname ... attrtype... attrvalue... </cmb:itemattributes>**
This tag iterates through the attributes of an item.

item Specify the name of a variable of type CMBItem that contains the item.

attrname
A string variable to contain the name of the attribute.

attrtype
A string variable to contain the attribute type.

attrvalue
A string variable to contain the value of the attribute.

<cmb:itemcontents *"* **data=***"data"* **item=***"item"***> content... </cmb:itemcontents>**
This tag iterates through the contents of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

content
A variable of type CMBObject for the item's contents.

<cmb:itemnotelogs *"* **data=***"data"* **item=***"item"***>notelog ... </cmb:itemnotelogs>**
This tag iterates through the note logs of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

notelog
A variable of type CMBObject to contain the item's note log.

<cmb:itemprivileges data=*"data"* **item=***"item"***>privilege ... </cmb:itemprivileges>**
This tag iterates through the privileges of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

privilege
A variable of type CMBPrivilege to contain the item's privilege.

<cmb:itemresources data=*"datamanagement"* **item=***"item"***> resource... </cmb:itemresources>**
This tag iterates through the resources of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

resource
A variable of type CMBResources to contain the item's resource.

<cmb:unmappeditem data=*"data"* **item=***"item"***>unmappeditem...</cmb:unmappeditem>**
This tag returns an unmapped item from the given mapped item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the mapped item.

unmappeditem
A variable of type CMBItem to contain the unmapped item.

<cmb:viewdata data=*"data "* **item=***"item"***>viewdata... </cmb:viewdata>**
This tag returns a view of an item.

data Specify the name of a variable of type CMBDataManagement.

item Specify the name of a variable of type CMBItem that contains the item.

viewdata
A variable of type CMBViewData to contain the viewable data.

Folder related tags

<cmb:folderitems folder=*"folder"***> item... </cmb:folderitems>**
This tag iterates through the contents of a folder.

folder Specify the name of a variable of type CMBItem to contain the folder contents.

item A variable of type CMBItem that represents the folder.

Document related tags

<cmb:viewerdocuments docservices=*"docservices"***>document ... </cmb:viewerdocuments>**
This tag iterates through the documents that are currently loaded.

docservices
Specify the name of a variable of type CMBDocumentServices.

document
A variable of type CMBDocument to contain the document.

<cmb:documentpages document=*"document"***> docpage... </cmb:documentpages>**
This tag iterates through the pages of a document.

document
Specify the name of a variable of type CMBDocument to contain the document.

docPage
A variable of type CMBPage to contain the page.

EIP controller servlet

To accompany the JSP tag library, EIP provides a contoller servlet. The URL of the servlet is accessed by the `cmb:cmbServlet` tag. This servlet acts as a controller of a model-view-controller design Web application, performing actions and initializing the beans (the model) which are then accessed in the JSP's (the views) either directly or indirectly by using the JSP tags.

Actions are provided for typical application tasks:

- log on and log off
- search
- create, retrieve, modify, and delete documents
- create folders, and add documents to or remove documents from folders

In addition, the servlet performs common tasks before and after the action, such as management of the connection to the content server. After every action, a JSP is invoked to format the results and send them back to the browser.

You can customize the servlet, to add new actions and associate JSPs with the actions.

What the servlet can do

Here are some of the aspects of the servlet that you can use:

Connection pooling

The controller servlet uses EIP connection pooling to provide high performance connection management. The time in which a connection is allocated to a session may be either for the request or for the time the session is logged on. Currently connection pooling is at the application scope.

Logon of Timed-Out Sessions
If a session has timed-out, and a request comes into the servlet, the logon JSP is displayed, allowing the user to logon again. Then, the original request is performed after successful logon.

Clean up on session termination
The servlet cleans up the session properly when a session is terminated, either by logging off or by a time-out. This means that the connection is destroyed or returned to the pool. All other EIP beans created by the servlet are terminated and their resources are freed without waiting for a garbage collection cycle to occur.

Locale The servlet insures that the locale is set correctly on the underlying beans, so messages and character strings are locale sensitive.

Using different JSP sets
A properties file, named `cmbservletjsp.properties`, by default, describes the JSPs to use for responses to servlet actions. The location of the

properties file is an application parameter. Therefore, several different web applications could be written using different sets of JSPs.

Extending the servlet

All actions known to the servlet are defined in a properties file named cmbservlet.properties (default). You can add, modify, or delete servlet actions by changing this file. To add a new action, follow these steps:

1. Implement a class to perform the action. The class must extend com.ibm.mm.servlets.CMBServletAction.
2. Add the name of the class and the action name to the cmbservlet.properties file. This has the following syntax:

   actions = *list of actions*action.<action_name>.class = *class_name*

   actions lists the actions understood by the servlet. For each action, a line within the properties file defines the class for the action. For example, to add an action named replay, in a class named ReplayAction:

   ```
   actions =... replay
   action.replay.class = ReplayAction
   ```

   You can also replace an action, or provide your own action to precede or follow any predefined action. For example, to precede logon with your own action, to perform additional validation:

   ```
   action.logon.class = MyLogonAction com.ibm.mm.servlets.CMBLogonAction
   ```

   The naming convention used for all predefined actions is com.ibm.mm.servlets.CMB*action*Action, where *action* is the name of the action, with the first letter in uppercase.

Servlet reference

You use a set of application parameters, request parameters, and a properties file to use the controller servlet in your applications.

Conventions

The servlet defines the following session and request values, which can be used in other JSP's or servlets. These conventions are followed by the JSP tag library. These conventions are the same as those for the EIP tag library.

Application parameters

The servlet understands the following application parameters (an alternative is to place these in the cmbservlet.properties file).

application parameter	values	description
servletPropertiesURL	URL	The location of the cmbservlet.properties file
defaultServerType	Fed, ICM, OD, DL, DES, V4, IP, DD, ...	Default logon information. This, along with defaultServer, defaultUserid, and defaultPassword can be used in situations of shared user ID. Rather than prompting with a login page, the default logon information will be used to perform the logon.
defaultServer		Default logon information.
defaultUserid		Default logon information.

application parameter	values	description
defaultPassword		Default logon information.
connectionpool	Boolean: true | false	To enable connection pooling
maxfreeconnection	integer	Maximum number of connections available in a connection pool.
minfreeconnection	integer	Minimum number of connection available in a connection pool
timeout	integer	The time duration (in milliseconds) after which a free connection will be disconnected and destroyed.
noSessionPage	URL	This is a page to display for logon, if the servlet is invoked without an established session or connection. This can be used to prompt for logon and chain back to the original action, allowing bookmarked links into EIP to work even if the user must log on.
timedOutPage	URL	This is a page to display if the session has timed out due to inactivity.
serverErrorPage	URL	This is a page to display if an error has occurred in accessing a server.
connectFailedPage	URL	This is a page to display if an error has occurred in connecting to a server. A prompt could be displayed to enter the correct userid/password for the server and retry can be performed.
tracelevel	0, 1, *or* 2	To indicate the level of tracing, as follows: • **0** - log nothing • **1** - log exceptions (default) • **2** -log exceptions, alert messages, WebSphere Application Server headers and attributes, EIP ini files, JVM system properties, EIP internal trace information
connectiontype	0, 1, *or* 2	The location of the EIP database and content server runtimes: • **0** - local (default) • **1** - remote • **2** -dynamic
cmbclient	URL	Location of `cmbclient.ini`
cmbcs	URL	Location of `cmbcs.ini`
serviceconnectiontype	0, 1, *or* 2	Location of services runtimes • **0** - local (default) • **1** - remote • **2** -dynamic
cmbsvclient	URL	Location of `cmbsvclient.ini`
cmbsvcs	URL	Location of `cmbsvcs.ini`
cmbcc2mime	URL	Location of `cmbcc2mime.ini`
cachedir	name of a directory	Directory to cache documents during document conversion

application parameter	values	description
jnitrace	name of a file	File to which to write the JNI trace information for the JNI logic used in document conversion (for IBM diagnostic purposes)
conversion	Boolean: true *or* false	If `true`, documents are converted to formats that can be displayed in a browser on middle tier, if possible. If `false`, the original document, unconverted, is sent to the browser.
maxresults	integer	Maximum hits returned; -1 (default) means all hits.
valuedelimiter	character	Defines the character that will delimit values in search criteria. The default is locale dependent and is comma (**,**) for US English.
conversion.<mimetype>	<none | document | page >	Conversion options for viewing documents of a specific mimetype. This affects the behavior of the viewDocument servlet. Page means attempt to paginate the document. Document means convert the document to a form readable in a browser. None means perform no conversion -- return the document in its native form.
nameseparator	character	Defines the character that will separate child component attribute from the parent component attribute in qualified names. The default is locale dependent, and is a forward slash (/) for US English.

Properties File

The servlet looks for a properties file, `cmbservlet.properties`. This file defines the actions that the servlet can use, including the actions defined here. It also defines the names of the JSP files that are used.

You can also define the servlet properties on the Web application server (servlet engine). The syntax is the same as used in the file.

The content of cmbservlet.properties is stored in a Properties object by the control servlet. It can be accessed through the application attribute ″cmbServletProperties,″ as shown in the following example.

```
// check to see if connection pooling is enabled
String name = "connectionpool";
Properties props = (Properties) application.getAttribute
    ("cmbServletProperties");
String value = props.getProperty(name);
// "true" if enabled, "false" otherwise
```

Request parameters

The servlet understands the following request parameters. Additional parameters can be specified, for use in the reply JSP.

General

action=*action*

The action to be performed. Additional parameters allowed are based on the action and are described below.

This parameter is optional. If it is not specified, the reply page will be executed by the servlet, after performing standard setup, such as checking the connection for time-out and logon.

reply=<URL>

Optional. Forwards to the JSP specified in this parameter rather than the JSP defined as the reply for the action in the cmbservlet.properties file. If the action parameter is not specified, and reply is specified, the reply page is executed by the controller servlet after performing standard setup, such as checking the connection for timeout and logon.

Connection Related

action=logon serverType=<> server=<> userid=<> password=<> [connstring=<>] [configstring=<>]

Login to a server. You must specify the server type, server name, user ID, and password. The connect string and init string are optional and are different depending on the type of server.

action=logoff [endSession=<true|false>]

Logout of the server. The session is also ended by default.

Search related

action=searchTemplate template=<> {<criterianame>.op=<> <criterianame>=<>} [usecache=<yes|no>]

Perform™ a search using the specified search template and criteria values.

action=searchEntity entity=<> {attribute.<*attrname***>.op=<> attribute.<***attrname***>=<>} [conjunction=<and|or>]**

Perform a search using an entity. The attributes values and operators can also be specified. Multiple values in the attribute value are separated with the value delimiter as specified in the application parameters. The attributes are combined together to form a query using `and` (default) or `or`, as specified by the conjunction parameter.

action=searchQuery queryString=<> {queryParameter.<parametername>=<>}

Perform a search using the specified query string. The query syntax depends on the server being searched.

Any number of additional query parameters may be specified. These are also server dependent.

Item related

action=lock itemId=<>

Lock an item, typically for exclusive access while updating.

action=unlock itemId=<>

Unlock a locked item.

action=createItem type=<document|folder> entity=<> {attribute.<*attrname***>=<***attrvalue***>}**

Create an item. If posted, content may be provided.

action=retrieveItem itemId=<>
Retrieve an item's attributes and content. This is useful to insure the latest content as stored on the server is used.

action=updateItem itemId=<> [entity=<>]
{attribute.<*attrname*>=<*attrvalue*>}
Update the attributes of an item. If entity is specified, the item is reindexed. Content is also updated if the servlet is invoked through a post.

action=deleteItem itemId=<>
Delete an item.

action=addContent itemId=<>
Add a content part to an item. The content data is posted.

action=getContent itemId=<> contentIndex=<>
Gets the content part and returns it to the browser.

action=updateContent itemId=<> contentIndex=<>
Update a content part on an item; the content data is posted. If no content exists, a content part is added.

action=deleteContent itemId=<> contentIndex=<>
Delete a content part for the specified item.

action=addNoteLog itemId=<>
Modifies the notelog on an item. The text of the notelog is posted.

action=updateNoteLog itemId=<> notelogIndex=<>
Modifies the note log of an item; the text of the note log is posted. If a note log does not exist, it is added.

action=deleteNoteLog itemId=<> notelogIndex=<>
Deletes the note log text of an item. The text of the note log is posted.

Folder related

action=addItemToFolder itemId=<> folderId=<>
Adds the specified item to the specified folder.

action=removeItemFromFolder itemId=<> folderId=<>
Removes the specified item from the specified folder.

Document related

action=viewDocument itemId=<>
Retrieves the document and views it. If the document is paginated, this action forwards to a JSP which generates the viewer frameset. If the document is not paginated, this action returns the actual content of the document.

action=viewPage itemId=<> page=<> scale=<> rotation=<>
annotations=<yes|no>
Retrieves a page of the document.

Servlet toolkit function matrix

Table 35. Servlet function matrix

Action	CMv8	CMv7	VI/400	IP/390	OD/Wkstn	OD/390	DD
logon	Y	Y	Y	Y	Y	Y	Y

Table 35. Servlet function matrix (continued)

Action	CMv8	CMv7	VI/400	IP/390	OD/Wkstn	OD/390	DD
logoff	Y	Y	Y	Y	Y	Y	Y
searchTemplate	N/A	N/A	N/A	N/A	Y	Y	N/A
searchEntity	Y	Y	Y	Y	Y	Y	N/A
searchQuery	Y	Y	Y	Y	Y	Y	Y
lock	Y	Y	Y	Y	N/A	N/A	N/A
unlock	Y	Y	Y	Y	N/A	N/A	N/A
createItem	Y	Y	N/A	N/A	N/A	N/A	N/A
retrieveItem	Y	Y	Y	Y	Y	Y	Y
updateItem	Y	Y	Y	Y	N/A	N/A	N/A
deleteItem	Y	Y	Y	Y	N/A	N/A	N/A
addContent	Y	Y	N/A	N/A	N/A	N/A	N/A
getContent	Y	Y	Y	Y	Y	Y	Y
updateContent	Y	Y	Y	Y	N/A	N/A	N/A
deleteContent	Y	Y	Y	Y	N/A	N/A	N/A
addNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
updateNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
deleteNoteLog	Y	Y	Y	N/A	N/A	N/A	N/A
addItemToFolder	Y	Y	Y	N/A	N/A	N/A	N/A
removeItemFromFolder	Y	Y	Y	N/A	N/A	N/A	N/A
viewDocument	Y	Y	Y	Y	Y	Y	Y
viewPage	Y	Y	Y	Y	Y	Y	Y

Y -- Function is supported **N/A** -- Function is not supported **Note:** The logon, logoff, getContent, retrieveitem,viewDocument and viewPage actions are supported on all the content servers.

Table 36. Servlet function matrix Continued

Action	DES	DB2	JDBC	IC	Fed	FileNet
logon	Y	Y	Y	Y	Y	Y
logoff	Y	Y	Y	Y	Y	Y
searchTemplate	N/A	N/A	N/A	N/A	Y	N/A
searchEntity	N/A	Y	Y	N/A	Y	N/A
searchQuery	Y	Y	Y	Y	Y	Y
lock	N/A	N/A	N/A	N/A	Y	N/A
unlock	N/A	N/A	N/A	N/A	Y	N/A
createItem	N/A	Y	Y	N/A	N/A	N/A
retrieveItem	Y	Y	Y	Y	Y	Y
updateItem	N/A	Y	Y	N/A	Y	N/A
deleteItem	N/A	Y	Y	N/A	Y	N/A
addContent	N/A	N/A	N/A	N/A	Y	N/A
getContent	Y	Y	Y	Y	Y	Y
updateContent	N/A	N/A	N/A	N/A	Y	N/A

Table 36. Servlet function matrix Continued (continued)

Action	DES	DB2	JDBC	IC	Fed	FileNet
deleteContent	N/A	N/A	N/A	N/A	Y	N/A
addNoteLog	N/A	N/A	N/A	N/A	Y	N/A
updateNoteLog	N/A	N/A	N/A	N/A	Y	N/A
deleteNoteLog	N/A	N/A	N/A	N/A	Y	N/A
addItemToFolder	N/A	N/A	N/A	N/A	N/A	N/A
removeItemFromFolder	N/A	N/A	N/A	N/A	N/A	N/A
viewDocument	Y	Y	Y	Y	Y	Y
viewPage	Y	Y	Y	Y	Y	Y

Y -- Function is supported **N/A** -- Function is not supported **Note:** The logon, logoff, getContent, retrieveitem,viewDocument and viewPage actions are supported on all the content servers.

Chapter 12. Working with information mining

This section begins with a description of how to build an application using the information mining beans, followed by sections on building your own content provider, using the information mining service API, and working with the JSP sample.

Building an information mining application using the beans

Information mining enables you to create powerful applications based on state-of-the-art text analysis and mining technology. The information mining beans offer:

- Text summarization and categorization, that is, assign a summary to a document
- Categorization, that is, assign a category to a document
- Information extraction, that is, find and extract relevant information units from a document
- Language identification, that is, determine the language a document is written in
- Clustering, that is, group similar documents in a document collection
- Text search that can be restricted to documents of a certain category
- Access to documents that are continually retrieved from the Web using the IBM Web Crawler

Note

Before using the Information Mining beans, it is important to understand the differences between the Service API and the JavaBeans.

- The Information Mining JavaBeans are software components for rapid application development and are JavaBeans conform. Certain elements of the code are 'bolted together' and cannot be changed by the user.
- The Java Service API contains the full Information Mining functionality as individual building blocks for creating applications. For more information, refer to "Using the service API" on page 371.

Before you can use the information mining beans, you must install and configure the Enterprise Information Portal information mining component on the machine that will run the beans.

The Information Mining beans are:

- **CMBSummarizationService**

  This is a non-visual bean that can be used to create summaries for documents. It can be run in three different modes and you can determine the length of the summary.

- **CMBCategorizationService**

  This is a non-visual bean that can be used to determine document categories. It runs on the results obtained using the Information Structuring Tool, namely the created taxonomy and metadata constituting terms that have been identified as being distinctive of the category and rules that describe each category. A document can be assigned to one or more categories. The categorization result consists of one or more categories and confidence values which shows how well

the document fits each category. You can set this confidence value as well as the maximum number of results to be returned.

- **CMBInformationExtractionService**

  This is a non-visual bean that can be used to extract names, terms, and expressions from a document.

- **CMBLanguageIdentificationService**

  This is a non-visual bean that can be used to determine the language in which a document is written.

- **CMBClusteringService**

  This is a non-visual bean that can be used to partition a document collection into similar sets of documents or clusters.

- **CMBAdvancedSearchService**

  The CMBAdvancedSearchService is a non-visual bean that performs a search. An advanced search can only find items that have been stored in the infromation mining data store.

- **CMBCatalogService**

  The CMBCatalogService maintains the persistent information mining data store. In order to use this service, you need to create a catalog using the Information Structuring Tool. You can use the bean to store results created by any of the information mining services, and of course to look up and delete this information.

- **CMBWebCrawlerService**

  The CMBWebCrawlerService monitors the Web space, that is, a directory where the Web crawler, that is running asynchronously, places all crawled HTML documents. CMBItems (an EIP helper class that represents a document) are created from these files that can then be used with any of the information mining services, or simply imported into the information mining data store using the CMBCatalogService.

- **CMBInfoMiningAdapter**

  The CMBInfoMiningAdapter is a switch between different information mining events. In addition, it supports the CMBResultEvent, which enables the Information Mining beans to work together with other Enterprise Information Portal beans. The CMBInfoMiningAdapter allows you to combine the beans to build various scenarios, as shown in the following diagram.

Figure 40 on page 335 demonstrates the use of the information mining beans.

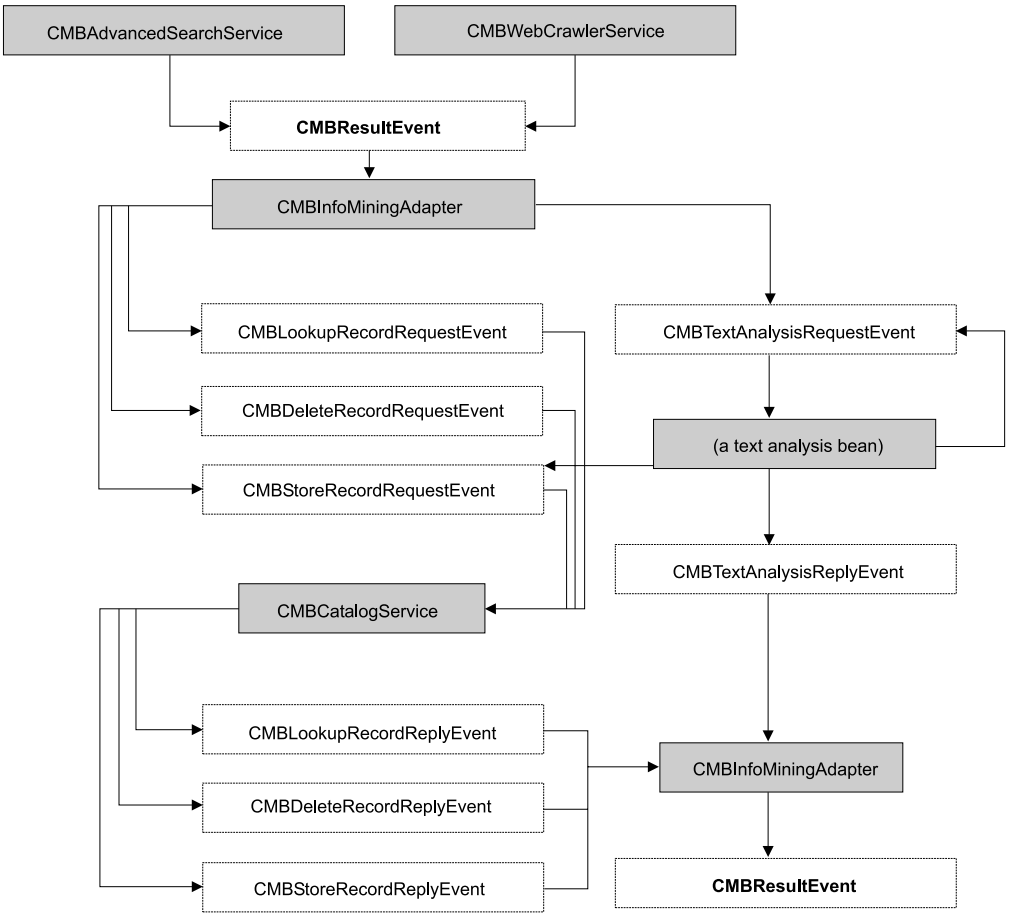

Figure 40. The information mining beans

Figure 41 lists the text analysis beans.

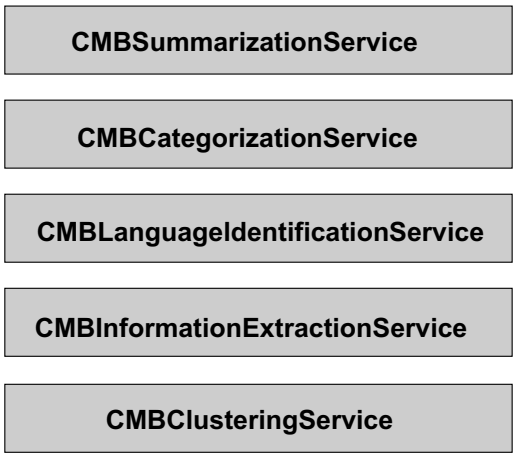

Figure 41. The text analysis beans

The following examples demonstrate how you can use the information mining beans to build applications:

- **Categorization sample:** Gathering information in preparation for information mining is a typical step for a librarian. See Figure 42 on page 337. In this sample, you begin by making a standard Enterprise Information Portal (EIP) search to

find documents in the EIP content server. (Gathering documents from the Web is in the Web Crawler sample.) Then the language of the documents is determined and the English documents are categorized. The category information is stored in the metadata store.

- **The summarization sample:** This is another typical step for a librarian. As in the categorization sample, you begin by making a standard EIP search to find documents in the EIP content server, identify the English documents, and then summarize these documents, storing the summaries in the metadata store. See Figure 44 on page 344.
- **The information extraction sample:** This is an information mining step. In this sample, you begin by making a standard EIP search to find documents using EIP. Then the English documents are determined, and names and terms extracted. The extracted information is stored in the metadata store.

  This is shown in Figure 45 on page 348.
- **The clustering sample:** This is an information mining step. You begin by making a standard EIP search to find documents using EIP. Then the English documents are determined and the clustering service is triggered manually. The results are displayed on the screen. Clustering information is not stored in the metadata store.

  This is shown in “Clustering” on page 352.
- **The Web Crawler sample:** Get documents using the Web Crawler and make the information available for search within the categories (the advanced search sample). This is also a librarian step, similar to the categorization and summarization sample, except that you gather the documents from the Internet or an intranet rather than from the EIP content server. To restore the category and summary information from the catalog to the search results, you retrieve them using the catalog service.

  See Figure 49 on page 358.
- **The advanced search sample:** This is an information mining step. You make an advanced search, which allows you to search for documents using a flexible query limited to specific categories. The search results contain the found documents, but not the categories from which the documents originated. To restore the category and summary information from the catalog to the search results, you retrieve them using the catalog service.

  This is shown in Figure 51 on page 365.

Location of the sample files

The samples described in this chapter are provided in the following directories:

Sample	Location
Categorization application	`...\samples\java\beans\infomining\categorization`
Summarization application	`...\samples\java\beans\infomining\summarization`
Information extraction application	`...\samples\java\beans\infomining\informationExtraction`
Clustering application	`...\samples\java\beans\infomining\clustering`
Advanced search application	`...\samples\java\beans\infomining\advancedSearch`

Web Crawler application
`...\samples\java\beans\infomining\webcrawler`

Each of these directories contains a `compile.bat` file to enable you to compile the source code. The application sample directories also contain a `run.bat` file to enable you to run the sample applications.

Categorizing documents

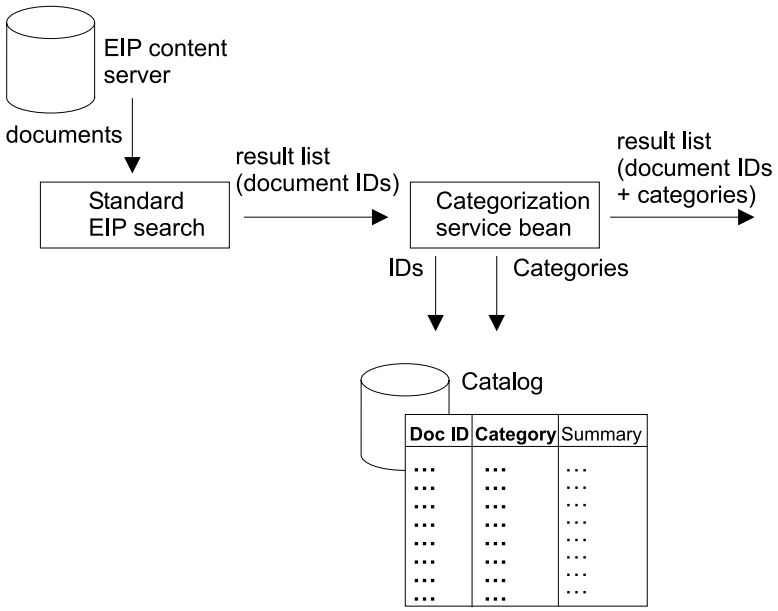

Figure 42. The categorization sample

This sample demonstrates how you can categorize documents that have been retrieved by a standard EIP search. The analysis results are stored, and the appropriate documents are made available for search within the categories.

In Figure 42, a standard EIP search is made on documents accessed by EIP. The language identification service bean determines the language in which the documents are written. The bean takes the document IDs to retrieve the document content and then analyses the content to determine the language. The categorization service bean categorizes documents and stores the results 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)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBCategorizationService
- CMBCatalogService

For this sample, the application:

1. Creates the beans.
2. Customizes the beans.
3. Connects the beans so that the language identification service can analyze the documents. The results get stored in the catalog.
4. The categorization service analyses the English documents. The results get stored in the catalog.
5. Runs the standard EIP query.
6. Displays the search results (lists of documents) and categories for verification.

The Java source is as follows.

Complete source for Categorization.java

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBInfoMiningUtilities;
import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBCategorizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class Categorization implements CMBResultListener, CMBExceptionListener
{
 String CMDBNAME         = "icmnlsdb";
 String CMDBUSER         = "icmadmin";
 String CMDBPASSWORD     = "password";
 String CATALOG          = "Sample";

 String SEARCH_TEMPLATE  = "SearchLongBySource";
 String SEARCH_VALUE     = "ibmpress";
 String SEARCH_CRITERION = "source";

 public Categorization() throws Throwable
  {
// ----- Creating beans
   CMBConnection connection = new CMBConnection();
   CMBQueryService queryService = new CMBQueryService();
   CMBSearchResults searchResults = new CMBSearchResults();
   CMBLanguageIdentificationService languageIdentificationService =
                      new CMBLanguageIdentificationService();
   CMBCategorizationService categorizationService = new
                                     CMBCategorizationService();
   CMBCatalogService catalogService = new CMBCatalogService();
   CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
   CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();
```

```
// ----- Reading parameters
   BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

   System.out.print("Database  [" + CMDBNAME + "] : ");
   String dbName = din.readLine();
   if (dbName.trim().equals("")) dbName = CMDBNAME;

   System.out.print("User name [" + CMDBUSER + "] : ");
   String userName = din.readLine();
   if (userName.trim().equals("")) userName = CMDBUSER;

   System.out.print("Password  [" + CMDBPASSWORD + "] : ");
   String passwd = din.readLine();
   if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

   System.out.print("Catalog   [" + CATALOG + "]   : ");
   String catalog = din.readLine();
   if (catalog.trim().equals("")) catalog = CATALOG;

   System.out.println("\n\nRunning Categorization with the following settings:");
   System.out.println("Database = " + dbName);
   System.out.println("User     = " + userName);
   System.out.println("Password = " + passwd);
   System.out.println("Catalog  = " + catalog + "\n");

   int key;
   do {
     System.out.print("Continue (y/n)? ");
     InputStreamReader inReader = new InputStreamReader(System.in);
     key = inReader.read();
     if (key == 'n') System.exit(0); }
   while (key != 'y');

// ----- Customizing beans
   connection.setServerName(dbName);
   connection.setUserid(userName);
   connection.setPassword(passwd);
   connection.setConnectToIKF(true);

   adapter1.setContentProvider(new CMBDefaultContentProvider());
   adapter1.setCatalogName(catalog);
   categorizationService.setCatalogName(catalog);
   catalogService.setCatalogName(catalog);
   adapter2.setCatalogName(catalog);
// ----- Running query
   connection.connect();
   catalogService.setDefaultCategoryPath(
             catalogService.getTaxonomy().getRootCategory().getPathAsString());
   CMBSchemaManagement schema = connection.getSchemaManagement();
   CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
   String[] searchValues = { SEARCH_VALUE };
   searchTemplate.setSearchCriterion(SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL,
                                searchValues);

   CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent(this,
         CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
   queryService.onCMBSearchRequest(searchRequest);

   connection.disconnect();
   }
 // ----- Implementing com.ibm.mm.beans.CMBResultListener
 public void onCMBResult(CMBResultEvent e)
  {
   if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
    return;

   Vector cmbItemVector = (Vector)e.getData();
```

```
   if(cmbItemVector == null)
    return;

   try {
     for(int i = 0; i < cmbItemVector.size(); i++)
      {
       CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

       CMBRecord currentRecord = currentItem.getInfoMiningRecord();

       System.out.println("\n\nPID       : " + currentRecord.getPID());
       System.out.println("Categories    : " +
                   currentRecord.getValue("IKF_CATEGORIES"));
      } /* for */
   }
   catch (CMBNoSuchKeyException nske)
     { nske.printStackTrace();
     }
  }

// ----- Implementing com.ibm.mm.beans.CMBExceptionListener
 public void onCMBException(CMBExceptionEvent e)
  {
   ((Exception)e.getData()).printStackTrace();
  }

 public static void main(String[] args)
  {
   try
    {
     new Categorization();
     System.exit(0);
    }
   catch(Throwable t)
    {
     t.printStackTrace();
    }
  }
}
```

Creating the beans

You need a connection to a content server to perform a search. The connection can be established using the CMBConnection bean. The beans CMBQueryService and CMBSearchResults are required to perform a standard EIP search. The language of the documents is determined using the CMBLanguageIdentificationService and stored in the attribute IKF_LANGUAGE of the corresponding record. The documents are assigned to one or more categories using the CMBCategorizationService, which also stores the category information in the appropriate records. The CMBCatalogService is used to store the item's category information in the catalog and make the appropriate documents available for advanced search. The two adapters are required to convert search result events to text analysis request events and then store item reply events back to search result events.

The code that creates the required beans is:

```
   CMBConnection connection = new CMBConnection();
   CMBQueryService queryService = new CMBQueryService();
   CMBSearchResults searchResults = new CMBSearchResults();
   CMBLanguageIdentificationService languageIdentificationService =
                           new CMBLanguageIdentificationService();
   CMBCategorizationService categorizationService =
                            new CMBCategorizationService();
```

```
CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();
```

Customizing the beans

The code shown in the customizing section has to be adapted according to your installation. On the connection bean you need to specify the name of the content server to connect, a user ID, and the appropriate password.

Before you can run the categorization service bean and the catalog service bean, you have to associate them with an existing catalog. In addition, the catalog service bean needs a default category to which items are assigned that do not contain any category information.

The code that does the customization for the sample is:

```
connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);

adapter1.setContentProvider(new CMBDefaultContentProvider());
adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);
```

Connecting the beans

Figure 43 on page 342 illustrates the flow of events among the beans.

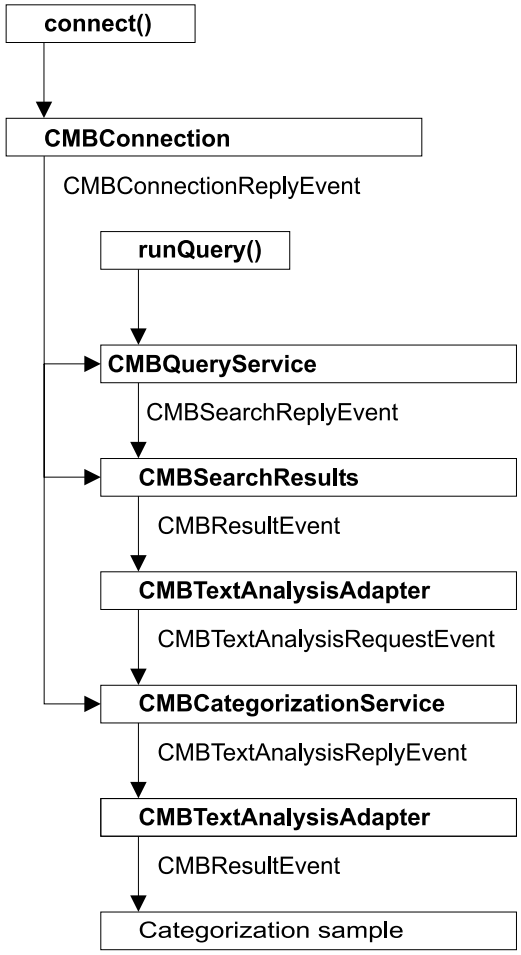

Figure 43. The categorization sample: Event flow

Five of the beans used in this sample listen to the CMBConnectionReplyEvent to get the connection handle. The CMBQueryService bean initiates a search that results in an event which then starts the event flow through the other beans.

The code that connects the beans is:

```
connection.addCMBConnectionReplyListener(queryService);
connection.addCMBConnectionReplyListener(searchResults);
connection.addCMBConnectionReplyListener(adapter1);
connection.addCMBConnectionReplyListener(languageIdentificationService);
connection.addCMBConnectionReplyListener(categorizationService);
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(adapter2);

queryService.addCMBSearchReplyListener(searchResults);
searchResults.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.addCMBTextAnalysisRequestListener
            (categorizationService);
categorizationService.addCMBStoreRecordRequestListener(catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);
adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
queryService.addCMBExceptionListener(this);
searchResults.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
```

```
categorizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);
```

Because exceptions are also sent as events, the Categorization class has to handle the appropriate event by implementing the CMBExceptionListener interface and is connected to the beans to be notified about exceptions.

Tip

You can combine the use of the categorization service bean and the summarization service bean, one after the other, in any sequence, to analyze documents. Refer to “Importing documents from a Web space” on page 357 for information on how to do this.

Running the query

Before you can run the query you need to establish the connection to the content server by calling the connect method on the CMBConnection bean:

```
connection.connect();
```

To start the standard EIP search, you need to specify a search template, a criterion of the template, the compare operator, and, of course, a search value.

You need to adapt the following code according to your configuration:

```
catalogService.setDefaultCategoryPath
   (catalogService.getTaxonomy().getRootCategory().getPathAsString());
CMBSchemaManagement schema = connection.getSchemaManagement();
CMBSearchTemplate searchTemplate =
           schema.getSearchTemplate(SEARCH_TEMPLATE);
String[] searchValues = { SEARCH_VALUE };
searchTemplate.setSearchCriterion(SEARCH_CRITERION,
     CMBBaseConstant.CMB_OP_EQUAL, searchValues);
CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent(this,
     CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
queryService.onCMBSearchRequest(searchRequest);
```

To close the current connection, call the disconnect() method:

```
connection.disconnect();
```

Displaying text analysis results

The Categorization class implements the CMBResultListener interface to be able to list the documents that have been found during the search and to display the category information created for each document. The CMBResultEvent, received as argument to the CMBResult method, contains a vector of CMBItem objects where each CMBItem object represents a document:

```
Vector cmbItemVector = (Vector)e.getData();
```

A CMBItem object encapsulates the PID of the document:

```
CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);
CMBRecord currentRecord = currentItem.getInfoMiningRecord();
System.out.println("\n\nPID       : " + currentRecord.getPID());
```

as well as the results of text analysis:

```
System.out.println("Categories    : " +
currentRecord.getValue("IKF_CATEGORIES"));
```

Summarizing documents

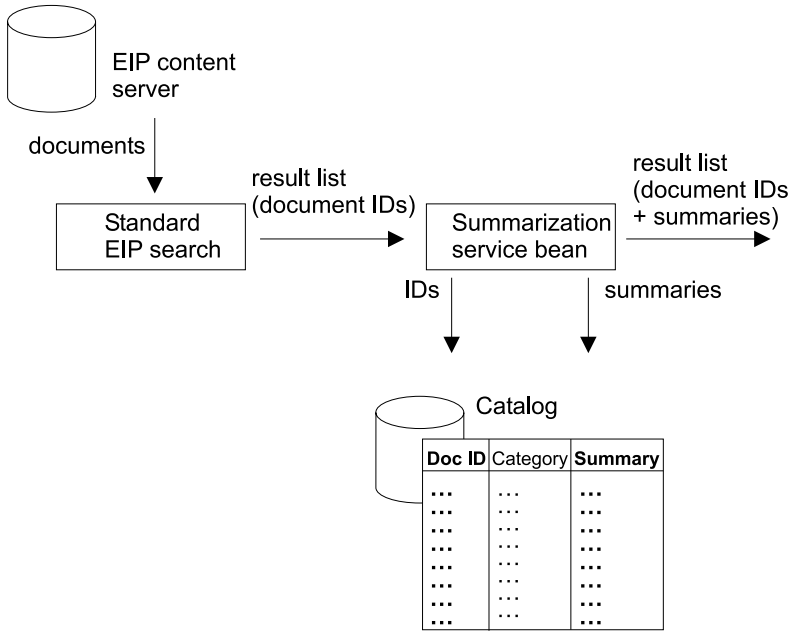

Figure 44. The summarization sample

This sample demonstrates how you can summarize documents that have been retrieved by a standard EIP search. The analysis results are stored, and the appropriate documents are made available for advanced search.

In Figure 44, a standard EIP search is made on documents accessed through EIP. The language identification service bean determines the language in which the documents are written. The bean takes the document IDs to retrieve the document content and then analyses the content to determine the language. The summarization service bean takes the English document IDs to retrieve the content of the found documents using the content provider and to create summaries of them. The catalog service bean then stores the summarization information in the metadata store. This information, the document IDs and the summaries, is then made available for further processing. The following beans are used in this sample:

- CMBConnection
- CMBQueryService (to perform standard EIP search)
- CMBSearchResults (to perform standard EIP search)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBSummarizationService
- CMBCatalogService

Complete source for Summarization.java

Here is the complete source of the summarization sample. As the summarization and categorization beans operate in a similar fashion, see “Categorizing documents” on page 337 for further detail.

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
```

```
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBInfoMiningUtilities;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBSummarizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class Summarization implements CMBResultListener, CMBExceptionListener
{
 String CMDBNAME          = "icmnlsdb";
 String CMDBUSER          = "icmadmin";
 String CMDBPASSWORD      = "password";
 String CATALOG           = "Sample";

 String SEARCH_TEMPLATE   = "SearchLongBySource";
 String SEARCH_VALUE      = "ibmpress";
 String SEARCH_CRITERION  = "source";

 public Summarization() throws Throwable
  {
// ----- Creating beans
   CMBConnection connection = new CMBConnection();
   CMBQueryService queryService = new CMBQueryService();
   CMBSearchResults searchResults = new CMBSearchResults();
   CMBLanguageIdentificationService languageIdentificationService = new
                                    CMBLanguageIdentificationService();
   CMBSummarizationService summarizationService = new
                                    CMBSummarizationService();
   CMBCatalogService catalogService = new CMBCatalogService();
   CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
   CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

// ----- Reading parameters
   BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

   System.out.print("Database  [" + CMDBNAME + "] : ");
   String dbName = din.readLine();
   if (dbName.trim().equals("")) dbName = CMDBNAME;

   System.out.print("User name [" + CMDBUSER + "] : ");
   String userName = din.readLine();
   if (userName.trim().equals("")) userName = CMDBUSER;

   System.out.print("Password  [" + CMDBPASSWORD + "] : ");
   String passwd = din.readLine();
   if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

   System.out.print("Catalog   [" + CATALOG + "]   : ");
   String catalog = din.readLine();
   if (catalog.trim().equals("")) catalog = CATALOG;

   System.out.println("\n\nRunning Summarization with the following settings:");
   System.out.println("Database = " + dbName);
```

```
   System.out.println("User     = " + userName);
   System.out.println("Password = " + passwd);
   System.out.println("Catalog  = " + catalog + "\n");

   int key;
   do {
     System.out.print("Continue (y/n)? ");
     InputStreamReader inReader = new InputStreamReader(System.in);
     key = inReader.read();
     if (key == 'n') System.exit(0); }
   while (key != 'y');


// ----- Customizing beans
   connection.setServerName(dbName);
   connection.setUserid(userName);
   connection.setPassword(passwd);
   connection.setConnectToIKF(true);

   adapter1.setContentProvider(new
                    CMBDefaultContentProvider());
   adapter1.setCatalogName(catalog);
   catalogService.setCatalogName(catalog);
   adapter2.setCatalogName(catalog);

// ----- Connecting beans
   connection.addCMBConnectionReplyListener(queryService);
   connection.addCMBConnectionReplyListener(searchResults);
   connection.addCMBConnectionReplyListener(adapter1);
   connection.addCMBConnectionReplyListener
                              (languageIdentificationService);
   connection.addCMBConnectionReplyListener(summarizationService);
   connection.addCMBConnectionReplyListener(catalogService);
   connection.addCMBConnectionReplyListener(adapter2);

   queryService.addCMBSearchReplyListener(searchResults);
   searchResults.addCMBResultListener(adapter1);
   adapter1.addCMBTextAnalysisRequestListener
                              (languageIdentificationService);
   languageIdentificationService.addCMBTextAnalysisRequestListener
                              (summarizationService);
   summarizationService.addCMBStoreRecordRequestListener(catalogService);
   catalogService.addCMBStoreRecordReplyListener(adapter2);

   adapter2.addCMBResultListener(this);

   connection.addCMBExceptionListener(this);
   queryService.addCMBExceptionListener(this);
   searchResults.addCMBExceptionListener(this);
   languageIdentificationService.addCMBExceptionListener(this);
   summarizationService.addCMBExceptionListener(this);
   catalogService.addCMBExceptionListener(this);
   adapter1.addCMBExceptionListener(this);
   adapter2.addCMBExceptionListener(this);
// ----- Running query
   connection.connect();
   catalogService.setDefaultCategoryPath
            (catalogService.getTaxonomy().getRootCategory().getPathAsString());

   CMBSchemaManagement schema = connection.getSchemaManagement();
   CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
   String[] searchValues = { SEARCH_VALUE };
   searchTemplate.setSearchCriterion(SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL,
                                     searchValues);

   CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent
            (this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
```

```
   queryService.onCMBSearchRequest(searchRequest);

   connection.disconnect();
  }
// ----- Implementing com.ibm.mm.beans.CMBResultListener
 public void onCMBResult(CMBResultEvent e)
  {
   if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
    return;

   Vector cmbItemVector = (Vector)e.getData();

   if(cmbItemVector == null)
    return;

   try {
     for(int i = 0; i < cmbItemVector.size(); i++)
      {
       CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

       CMBRecord currentRecord = currentItem.getInfoMiningRecord();

       System.out.println("\n\nPID       : " + currentRecord.getPID());
       System.out.println("Summary   : " + currentRecord.getValue("IKF_SUMMARY"));
      } /* for */
    }
    catch (CMBNoSuchKeyException nske)
      { nske.printStackTrace();
      }
  }

// ----- Implementing com.ibm.mm.beans.CMBExceptionListener
 public void onCMBException(CMBExceptionEvent e)
  {
   ((Exception)e.getData()).printStackTrace();
  }

 public static void main(String[] args)
  {
   try
    {
     new Summarization();
     System.exit(0);
    }
   catch(Throwable t)
    {
     t.printStackTrace();
    }
  }
 }
```

Extracting information

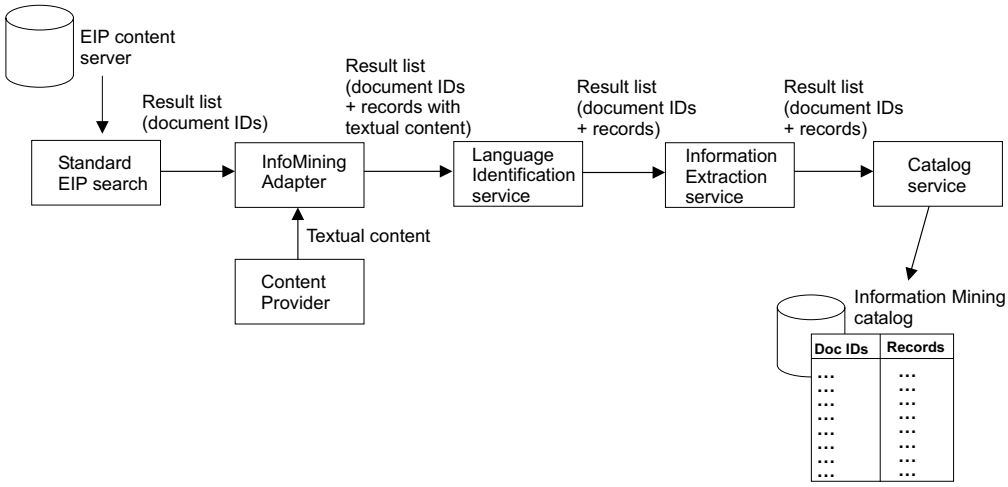

Figure 45. The extracting information sample

This sample demonstrates how you can extract information, like names, terms and expressions, from documents that have been retrieved by a standard EIP search. The analysis results are stored, and can be used to retrieve related documents or extract important information from individual documents.

In Figure 45, a standard EIP search is made on documents held in the EIP content server. The language identification service bean determines the language in which the documents are written. The bean takes the document IDs to retrieve the document content and then analyses the content to determine the language. The information extraction service bean takes the English documents, extracts information from these documents and stores the results in the metadata store. This information, the document IDs and the extracted information, is then made available for further processing.

The following beans are used in this sample:

- CMBConnection
- CMBQueryService (to perform standard EIP search)
- CMBSearchResults (to perform standard EIP search)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBInformationExtractionService
- CMBCatalogService

For this sample, the application:

1. Creates the beans.
2. Customizes the beans.
3. Connects the beans so that the language identification service can analyze the documents. The results get stored in the catalog.
4. The information extraction service analyzes the documents. The results get stored in the catalog.
5. Displays the search results (lists of documents) and extracted information for verification.

The Java source is as follows. As the information extraction and categorization beans operate in a similar fashion, see “Categorizing documents” on page 337 for further detail.

Complete source for InformationExtraction.java

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBInfoMiningUtilities;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBInformationExtractionService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

public class InformationExtraction implements CMBResultListener,
                          CMBExceptionListener
{
    String CMDBNAME          = "icmnlsdb";
    String CMDBUSER          = "icmadmin";
    String CMDBPASSWORD      = "password";
    String CATALOG           = "Sample";

    String SEARCH_TEMPLATE   = "SearchLongBySource";
    String SEARCH_VALUE      = "ibmpress";
    String SEARCH_CRITERION  = "source";

 public InformationExtraction() throws Throwable
  {
// ----- Creating beans
   CMBConnection connection = new CMBConnection();
   CMBQueryService queryService = new CMBQueryService();
   CMBSearchResults searchResults = new CMBSearchResults();
   CMBLanguageIdentificationService languageIdentificationService =
           new CMBLanguageIdentificationService();
   CMBInformationExtractionService informationExtractionService =
           new CMBInformationExtractionService();
   CMBCatalogService catalogService = new CMBCatalogService();
   CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
   CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

// ----- Reading parameters
   BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

   System.out.print("Database  [" + CMDBNAME + "] : ");
   String dbName = din.readLine();
   if (dbName.trim().equals("")) dbName = CMDBNAME;

   System.out.print("User name [" + CMDBUSER + "] : ");
   String userName = din.readLine();
```

```
    if (userName.trim().equals("")) userName = CMDBUSER;

    System.out.print("Password  [" + CMDBPASSWORD + "] : ");
    String passwd = din.readLine();
    if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

    System.out.print("Catalog   [" + CATALOG + "]   : ");
    String catalog = din.readLine();
    if (catalog.trim().equals("")) catalog = CATALOG;

    System.out.println("\n\nRunning InformationExtraction with
                                the following settings:");
    System.out.println("Database = " + dbName);
    System.out.println("User     = " + userName);
    System.out.println("Password = " + passwd);
    System.out.println("Catalog  = " + catalog + "\n");

    int key;
    do {
      System.out.print("Continue (y/n)? ");
      InputStreamReader inReader = new InputStreamReader(System.in);
      key = inReader.read();
      if (key == 'n') System.exit(0); }
    while (key != 'y');

  // ----- Customizing beans
    connection.setServerName(dbName);
    connection.setUserid(userName);
    connection.setPassword(passwd);
    connection.setConnectToIKF(true);

    adapter1.setContentProvider(new
                    CMBDefaultContentProvider());
    adapter1.setCatalogName(catalog);
    catalogService.setCatalogName(catalog);
    adapter2.setCatalogName(catalog);

  // ----- Connecting beans
    connection.addCMBConnectionReplyListener(queryService);
    connection.addCMBConnectionReplyListener(searchResults);
    connection.addCMBConnectionReplyListener(adapter1);
    connection.addCMBConnectionReplyListener
                            (languageIdentificationService);
    connection.addCMBConnectionReplyListener
                            (informationExtractionService);
    connection.addCMBConnectionReplyListener(catalogService);
    connection.addCMBConnectionReplyListener(adapter2);

    queryService.addCMBSearchReplyListener(searchResults);
    searchResults.addCMBResultListener(adapter1);
    adapter1.addCMBTextAnalysisRequestListener
                              (languageIdentificationService);
    languageIdentificationService.addCMBTextAnalysisRequestListener
                              (informationExtractionService);
    informationExtractionService.addCMBStoreRecordRequestListener
                              (catalogService);
    catalogService.addCMBStoreRecordReplyListener(adapter2);

    adapter2.addCMBResultListener(this);

    connection.addCMBExceptionListener(this);
    queryService.addCMBExceptionListener(this);
    searchResults.addCMBExceptionListener(this);
    languageIdentificationService.addCMBExceptionListener(this);
    informationExtractionService.addCMBExceptionListener(this);
    catalogService.addCMBExceptionListener(this);
    adapter1.addCMBExceptionListener(this);
```

```
   adapter2.addCMBExceptionListener(this);

// ----- Running query
   connection.connect();
   catalogService.setDefaultCategoryPath
           (catalogService.getTaxonomy().getRootCategory().getPathAsString());

   CMBSchemaManagement schema = connection.getSchemaManagement();
   CMBSearchTemplate searchTemplate =
        schema.getSearchTemplate(SEARCH_TEMPLATE);
   String[] searchValues = { SEARCH_VALUE };
   searchTemplate.setSearchCriterion(SEARCH_CRITERION,
         CMBBaseConstant.CMB_OP_EQUAL, searchValues);

   CMBSearchRequestEvent searchRequest = new
     CMBSearchRequestEvent(this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH,
       searchTemplate);
   queryService.onCMBSearchRequest(searchRequest);

   connection.disconnect();
  }
// ----- Implementing com.ibm.mm.beans.CMBResultListener
  public void onCMBResult(CMBResultEvent e)
  {
   if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
    return;

   Vector cmbItemVector = (Vector)e.getData();

   if(cmbItemVector == null)
    return;

   try {
     for(int i = 0; i < cmbItemVector.size(); i++)
      {
       CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

       CMBRecord currentRecord = currentItem.getInfoMiningRecord();

       System.out.println("\n\nPID       : " + currentRecord.getPID());
       System.out.println("Features    : " + currentRecord.getValue
                                                ("IKF_FEATURES"));
      } /* for */
   }
   catch (CMBNoSuchKeyException nske)
     { nske.printStackTrace();
     }
  }


// ----- Implementing CMBExceptionListener
 public void onCMBException(CMBExceptionEvent e)
  {
   ((Exception)e.getData()).printStackTrace();
  }

 public static void main(String[] args)
  {
   try
    {
     new InformationExtraction();
     System.exit(0);
    }
   catch(Throwable t)
    {
```

```
        t.printStackTrace();
      }
    }
}
```

Figure 46 illustrates the flow of events among the beans.

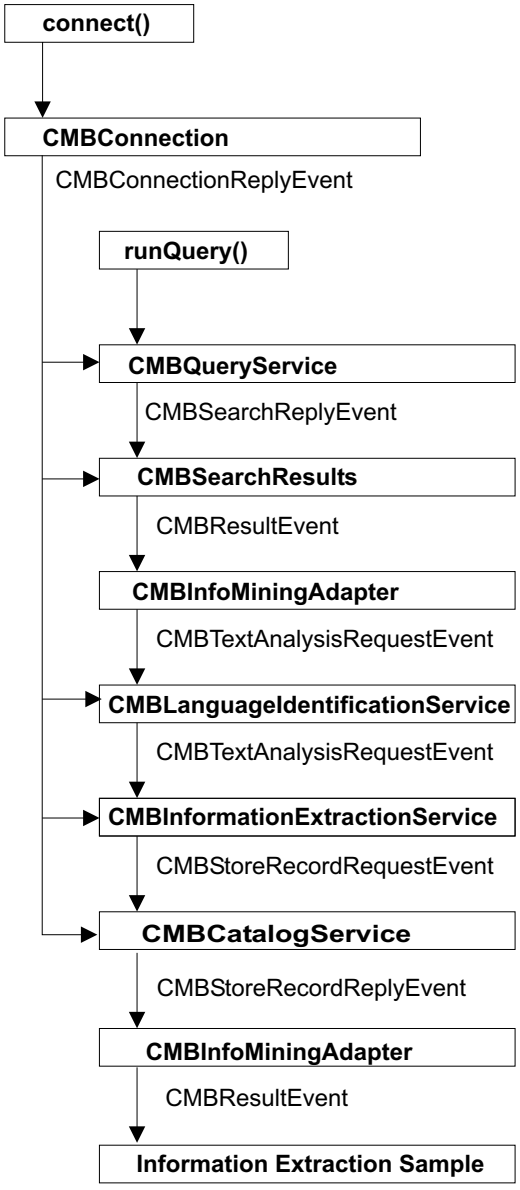

Figure 46. The information extraction sample: Event flow

Clustering

This sample demonstrates how you can cluster documents that have been retrieved by a standard EIP search. Before clustering, it is necessary to identify the language of the documents. The clustering service only automatically collects documents returned from an EIP search. The actual clustering process has to be triggered manually by calling the cluster() method.

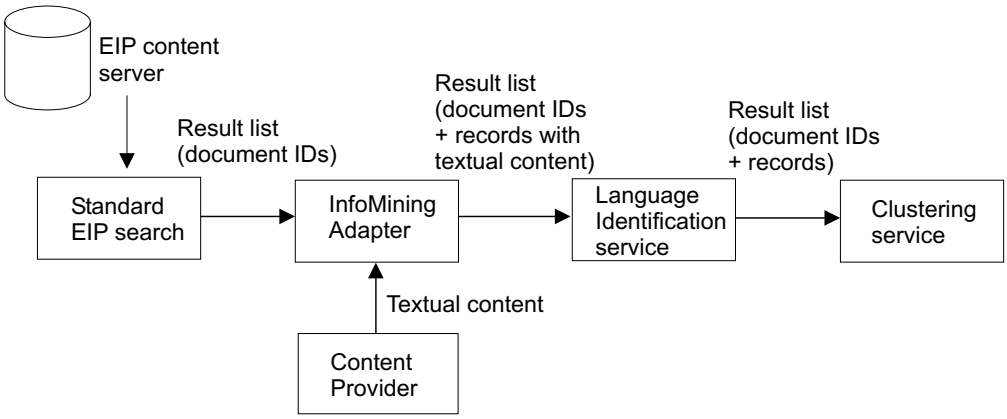

Figure 47. The clustering sample

The following beans are used in this sample:

- CMBConnection
- CMBQueryService (to perform standard EIP search)
- CMBSearchResults (to perform standard EIP search)
- CMBInfoMiningAdapter
- CMBLanguageIdentificationService
- CMBClusteringService

For this sample, the application:

1. Creates the beans.
2. Customizes the beans.
3. Connects the beans so that the Language Identification service can analyze the documents and identify the document language, and the Clustering service can collect the documents ready for subsequent clustering.
4. Triggers clustering by calling the cluster() method. The result (of class CMBClusterResult) is then printed out on the screen in a readable form.

The beans for the Clustering service are similar to the other Information Mining beans execept that:

- The clustering service only collects documents returned from an EIP search. The actual clustering process has to be triggered manually.
- The catalog is not designated to store clustering results which means that the catalog service is not needed in this scenario.

Refer to “Categorizing documents” on page 337 for details on how the beans operate.

The Java source is as follows.

Complete source for Clustering.java

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBQueryService;
import com.ibm.mm.beans.CMBSearchResults;
import com.ibm.mm.beans.CMBSchemaManagement;
```

```
import com.ibm.mm.beans.CMBSearchTemplate;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBBaseConstant;
import com.ibm.mm.beans.CMBSearchRequestEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBInfoMiningUtilities;
import com.ibm.mm.beans.infomining.CMBDefaultContentProvider;
import com.ibm.mm.beans.infomining.
                          CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBClusteringService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBClusterResult;
import com.ibm.mm.beans.infomining.CMBClusterNode;

public class Clustering implements CMBResultListener, CMBExceptionListener
{
 String CMDBNAME         = "icmnlsdb";
 String CMDBUSER         = "icmadmin";
 String CMDBPASSWORD     = "password";
 String CATALOG          = "Sample";

 String SEARCH_TEMPLATE  = "SearchLongBySource";
 String SEARCH_VALUE     = "ibmpress";
 String SEARCH_CRITERION = "source";

 public Clustering() throws Throwable
  {
// ----- Creating beans
   CMBConnection connection = new CMBConnection();
   CMBQueryService queryService = new CMBQueryService();
   CMBSearchResults searchResults = new CMBSearchResults();
   CMBLanguageIdentificationService languageIdentificationService = new
                                     CMBLanguageIdentificationService();
   CMBClusteringService clusteringService = new CMBClusteringService();
   CMBInfoMiningAdapter adapter = new CMBInfoMiningAdapter();

// ------ Reading parameters
   BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

   System.out.print("Database  [" + CMDBNAME + "] : ");
   String dbName = din.readLine();
   if (dbName.trim().equals("")) dbName = CMDBNAME;

   System.out.print("User name [" + CMDBUSER + "] : ");
   String userName = din.readLine();
   if (userName.trim().equals("")) userName = CMDBUSER;

   System.out.print("Password  [" + CMDBPASSWORD + "] : ");
   String passwd = din.readLine();
   if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

   System.out.print("Catalog   [" + CATALOG + "]   : ");
   String catalog = din.readLine();
   if (catalog.trim().equals("")) catalog = CATALOG;

   System.out.println("\n\nRunning Clustering with the
                                 following settings:");
   System.out.println("Database = " + dbName);
   System.out.println("User     = " + userName);
   System.out.println("Password = " + passwd);
   System.out.println("Catalog  = " + catalog + "\n");

   int key;
   do {
```

```
      System.out.print("Continue (y/n)? ");
      InputStreamReader inReader = new InputStreamReader(System.in);
      key = inReader.read();
      if (key == 'n') System.exit(0); }
    while (key != 'y');

// ----- customizing beans
    connection.setServerName(dbName);
    connection.setUserid(userName);
    connection.setPassword(passwd);
    connection.setConnectToIKF(true);

    adapter.setContentProvider
                (new CMBDefaultContentProvider());
    adapter.setCatalogName(catalog);
    clusteringService.setMinClusterCount(2);
    clusteringService.setMaxClusterCount(6);
    clusteringService.setClusterFeatureCount(5);

// ----- Connecting beans
    connection.addCMBConnectionReplyListener(queryService);
    connection.addCMBConnectionReplyListener(searchResults);
    connection.addCMBConnectionReplyListener(adapter);
    connection.addCMBConnectionReplyListener
                               (languageIdentificationService);
    connection.addCMBConnectionReplyListener(clusteringService);

    queryService.addCMBSearchReplyListener(searchResults);
    searchResults.addCMBResultListener(adapter);
    adapter.addCMBTextAnalysisRequestListener(languageIdentificationService);
    languageIdentificationService.addCMBTextAnalysisRequestListener
                                         (clusteringService);

    connection.addCMBExceptionListener(this);
    queryService.addCMBExceptionListener(this);
    searchResults.addCMBExceptionListener(this);
    languageIdentificationService.addCMBExceptionListener(this);
    clusteringService.addCMBExceptionListener(this);
    adapter.addCMBExceptionListener(this);

// ----- Running query
    connection.connect();

    CMBSchemaManagement schema = connection.getSchemaManagement();
    CMBSearchTemplate searchTemplate = schema.getSearchTemplate(SEARCH_TEMPLATE);
    String[] searchValues = { SEARCH_VALUE };
    searchTemplate.setSearchCriterion
                 (SEARCH_CRITERION, CMBBaseConstant.CMB_OP_EQUAL, searchValues);

    CMBSearchRequestEvent searchRequest = new CMBSearchRequestEvent
             (this, CMBSearchRequestEvent.CMB_REQUEST_SEARCH_SYNCH, searchTemplate);
    queryService.onCMBSearchRequest(searchRequest);

// ----- Running clustering and printing results
    CMBClusterResult clusterResult = clusteringService.cluster();

    System.out.println(clusterResult.getClusterCount() +
                       " clusters found for " +
                       clusterResult.getDocumentCount() +
                       " documents:");
    CMBClusterNode[] clusterNodes = clusterResult.getClusterNodes();
    for(int i = 0; i < clusterNodes.length; i++) {
      CMBClusterNode node = clusterNodes[i];
      String[] features = node.getClusterFeatures();
      String[] names    = node.getDocumentNames();
      String label      = node.getLabel();
      System.out.println("Cluster " + label);
```

```
     System.out.print("  Cluster Features : ");
     for(int j = 0; j < features.length; j++)
             System.out.print(features[j] + " ");
     System.out.println();
     System.out.println("  Documents in cluster :");
     for(int j = 0; j < names.length; j++)
               System.out.println("  " + names[j]);
   }

// ----- Disconnecting
   connection.disconnect();
  }

// ----- Implementing the com.ibm.mm.beans.CMBResultListener
 public void onCMBResult(CMBResultEvent e)
  {
   return;
  }

//----- Implementing com.ibm.mm.beans.CMBExceptionListener
 public void onCMBException(CMBExceptionEvent e)
  {
   ((Exception)e.getData()).printStackTrace();
  }

 public static void main(String[] args)
  {
   try
    {
     new Clustering();
     System.exit(0);
    }
   catch(Throwable t)
    {
     t.printStackTrace();
    }
  }
}
```

Figure 48 on page 357 illustrates the flow of events among the beans.

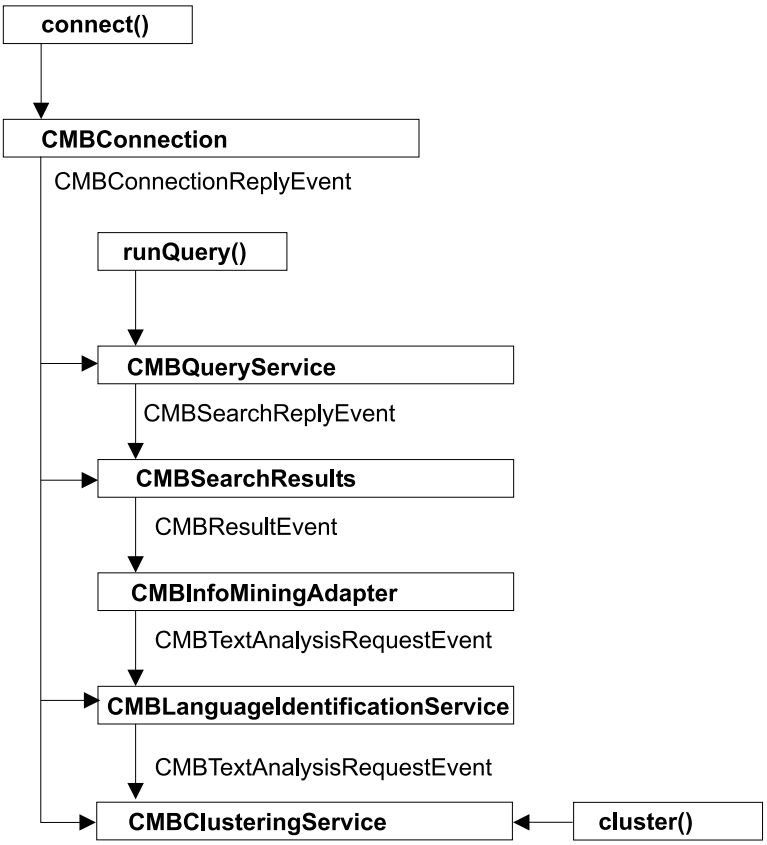

Figure 48. The clustering sample: Event flow

Importing documents from a Web space

This sample demonstrates how to import crawled documents into the information mining component and conduct text analysis (language identification, categorization and summarization) on the documents that have been imported. This can only be done if the Web Crawler has run or is running, and new or changed objects are saved within the defined Web space.

Once the Web Crawler has placed a copy of the web-page documents that it has been monitoring into a named catalog, you use the CMBWebCrawlerService bean to import the crawled documents into Enterprise Information Portal information mining.

The Web Crawler and CMBWebCrawlerService bean can be run as persistent processes to monitor and import documents as they change. A CMBResultEvent can be triggered when the number of imported documents exceeds a certain value or when all monitored files have been imported. The CMBWebCrawlerService bean notifies all the attached listeners. The event contains the imported files as a vector of CMBItems.

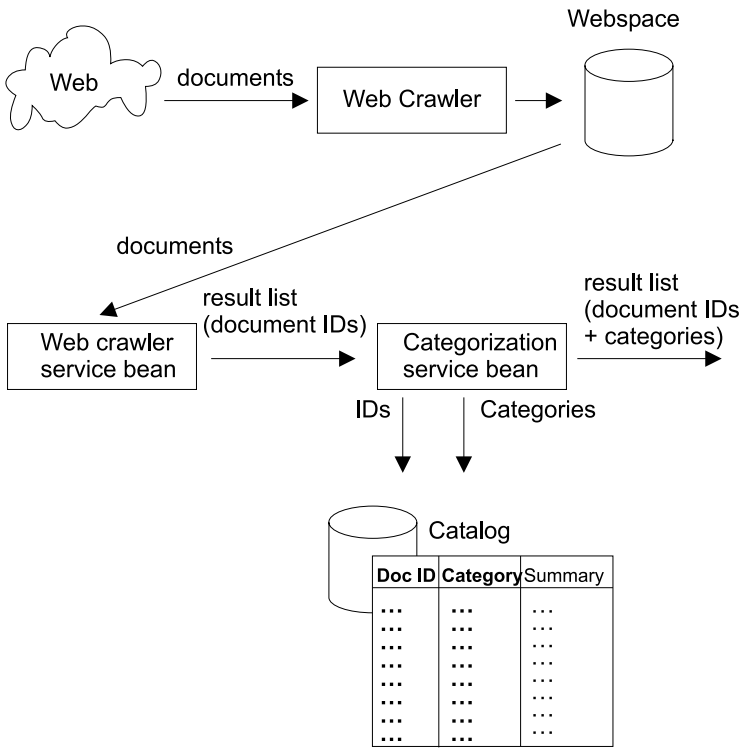

Figure 49. The Web Crawler sample

Modifying the import operation. You can change some aspects of the import operation on the CMBWebCrawlerService bean. Initially, the program looks for crawled documents every 30 minutes, but you can change this value. It also throws a CMBResultEvent notification to all listeners after every 100 imported documents. You can change this to a different number of documents, or cause a notification to be thrown when all crawled files are imported to the Enterprise Information Portal set.

Properties of CMBWebCrawlerService Java bean:

pollCycles
Number of times to poll.

pollMinutes
Number of minutes to sleep before next poll. The default is 30.

rootDir
To crawl %IMY_WEBSPACE% on local host.

archive
Keep files in *rootDir/webspace/archives*. The default is false.

pageSize
Number of imported items until a CMBResultEvent is thrown to the CMBResultListeners.

webSpace
The Web space that is monitored by the Web Crawler.

As with the other text analysis beans, the results of the Web Crawler search bean can be made available for a subsequent advanced search.

The following beans are used in this sample:

- CMBConnection
- CMBWebCrawlerService
- CMBInfoMiningAdapter
- CMBLanguageIdentification
- CMBCategorizationService
- CMBSummarizationService
- CMBCatalogService

For this sample the application:

1. Creates the beans
2. Customizes the beans
3. Connects the beans
4. Starts the Web Crawler service
5. Displays the crawler results and text analysis results

An explanation of each of the preceding steps follows in the source for WebCrawler.java.

Complete source for WebCrawler.java

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBItem;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBWebCrawlerService;
import com.ibm.mm.beans.infomining.
                      CMBLanguageIdentificationService;
import com.ibm.mm.beans.infomining.CMBCategorizationService;
import com.ibm.mm.beans.infomining.CMBSummarizationService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBInfoMiningAdapter;
import com.ibm.mm.beans.infomining.CMBNoSuchKeyException;

pblic class WebCrawler implements CMBResultListener, CMBExceptionListener
{
 String CMDBNAME          = "icmnlsdb";
 String CMDBUSER          = "icmadmin";
 String CMDBPASSWORD      = "password";
 String CATALOG           = "Sample";

 String WEBSPACE_DIR      = "";       // set this to the directory in which
                                      // the web spaces reside
 String WEBSPACE_NAME     = "";       // set this to the name of your web space

 public WebCrawler() throws Throwable
  {
// ----- Creating beans
   CMBConnection connection = new CMBConnection();
   CMBWebCrawlerService crawlerService = new CMBWebCrawlerService();
   CMBLanguageIdentificationService languageIdentificationService = new
                                 CMBLanguageIdentificationService();
   CMBCategorizationService categorizationService =
```

```
                                    new CMBCategorizationService();
   CMBSummarizationService summarizationService =
                                    new CMBSummarizationService();
   CMBCatalogService catalogService = new CMBCatalogService();
   CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
   CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();

// ----- Reading parameters
   BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

   System.out.print("Database  [" + CMDBNAME + "]      : ");
   String dbName = din.readLine();
   if (dbName.trim().equals("")) dbName = CMDBNAME;

   System.out.print("User name [" + CMDBUSER + "]      : ");
   String userName = din.readLine();
   if (userName.trim().equals("")) userName = CMDBUSER;

   System.out.print("Password  [" + CMDBPASSWORD + "]     : ");
   String passwd = din.readLine();
   if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

   System.out.print("Webspace directory [" + WEBSPACE_DIR + "] : ");
   String webspaceDir = din.readLine();
   if (webspaceDir.trim().equals("")) webspaceDir = WEBSPACE_DIR;

   System.out.print("Webspace name [" + WEBSPACE_NAME + "]     : ");
   String webspaceName = din.readLine();
   if (webspaceName.trim().equals("")) webspaceName = WEBSPACE_NAME;

   System.out.print("Catalog   [" + CATALOG + "]        : ");
   String catalog = din.readLine();
   if (catalog.trim().equals("")) catalog = CATALOG;

   System.out.println("\n\nRunning Summarization with the
                                 following settings:");
   System.out.println("Database           = " + dbName);
   System.out.println("User               = " + userName);
   System.out.println("Password           = " + passwd);
   System.out.println("Webspace directory = " + webspaceDir);
   System.out.println("Webspace name      = " + webspaceName);
   System.out.println("Catalog            = " + catalog + "\n");

   int key;
   do {
     System.out.print("Continue (y/n)? ");
     InputStreamReader inReader = new InputStreamReader(System.in);
     key = inReader.read();
     if (key == 'n') System.exit(0); }
   while (key != 'y');

// ----- Customizing beans
   connection.setServerName(dbName);
   connection.setUserid(userName);
   connection.setPassword(passwd);
   connection.setConnectToIKF(true);
   crawlerService.setRootDirectory(webspaceDir);
   crawlerService.setWebSpace(webspaceName);

   adapter1.setCatalogName(catalog);
   categorizationService.setCatalogName(catalog);
   catalogService.setCatalogName(catalog);
   adapter2.setCatalogName(catalog);

// ----- Connecting beans
   connection.addCMBConnectionReplyListener(crawlerService);
   connection.addCMBConnectionReplyListener(adapter1);
```

```
   connection.addCMBConnectionReplyListener
                                    (languageIdentificationService);
   connection.addCMBConnectionReplyListener(categorizationService);
   connection.addCMBConnectionReplyListener(summarizationService);
   connection.addCMBConnectionReplyListener(catalogService);
   connection.addCMBConnectionReplyListener(adapter2);

   crawlerService.addCMBResultListener(adapter1);
   adapter1.addCMBTextAnalysisRequestListener
                               (languageIdentificationService);
   languageIdentificationService.addCMBTextAnalysisRequestListener
                               (categorizationService);
   categorizationService.addCMBTextAnalysisRequestListener
                               (summarizationService);
   summarizationService.addCMBStoreRecordRequestListener(catalogService);
   catalogService.addCMBStoreRecordReplyListener(adapter2);
   adapter2.addCMBResultListener(this);

   connection.addCMBExceptionListener(this);
   crawlerService.addCMBExceptionListener(this);
   languageIdentificationService.addCMBExceptionListener(this);
   categorizationService.addCMBExceptionListener(this);
   summarizationService.addCMBExceptionListener(this);
   catalogService.addCMBExceptionListener(this);
   adapter1.addCMBExceptionListener(this);
   adapter2.addCMBExceptionListener(this);

// ----- Running query
   connection.connect();
   catalogService.setDefaultCategoryPath
       (catalogService.getTaxonomy().getRootCategory().getPathAsString());
   crawlerService.start();
   connection.disconnect();
// ----- Implementing com.ibm.mm.beans.CMBResultListener
 public void onCMBResult(CMBResultEvent e)
  {
   if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
    return;

   Vector cmbItemVector = (Vector)e.getData();

   if(cmbItemVector == null)
    return;

   try {
     for(int i = 0; i < cmbItemVector.size(); i++)
      {
       CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

       CMBRecord currentRecord = currentItem.getInfoMiningRecord();

       System.out.println("\n\nPID       : " + currentRecord.getPID());
       System.out.println("Summary   : " + currentRecord.getValue("IKF_SUMMARY"));
       System.out.println
                   ("Categories   : " + currentRecord.getValue("IKF_CATEGORIES"));
      } /* for */
    }
    catch (CMBNoSuchKeyException nske)
     { nske.printStackTrace();
     }
  }
// ----- Implementing com.ibm.mm.beans.CMBExceptionListener
 public void onCMBException(CMBExceptionEvent e)
  {
   ((Exception)e.getData()).printStackTrace();
  }
```

```
 public static void main(String[] args)
  {
   try
    {
     new WebCrawler();
     System.exit(0);
    }
   catch(Throwable t)
    {
     t.printStackTrace();
    }
  }
}
```

Creating the beans

You build a connection to a content server for security reasons; information mining needs to know who is working with the system. This ensures that only those who are registered in the system are allowed to use it. The connection can be established using the CMBConnection bean. The bean CMBWebCrawlerService is used to import the previously crawled documents into the information mining component and to throw a CMBResultEvent notification. Language identification, summary and category information are created using the beans CMBLanguageIdentification, CMBSummarizationService and CMBCategorizationService. The two adapters convert result events to text analysis request events, and then store record reply events back to search result events.

The code that creates the beans is:

```
CMBConnection connection = new CMBConnection();
CMBWebCrawlerService crawlerService = new CMBWebCrawlerService();
CMBLanguageIdentificationService languageIdentificationService =
              new CMBLanguageIdentificationService();
CMBCategorizationService categorizationService = new CMBCategorizationService();
CMBSummarizationService summarizationService = new CMBSummarizationService();
CMBCatalogService catalogService = new CMBCatalogService();
CMBInfoMiningAdapter adapter1 = new CMBInfoMiningAdapter();
CMBInfoMiningAdapter adapter2 = new CMBInfoMiningAdapter();
```

Customizing the beans

The code shown in the customizing section has to be adapted according to your installation. 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.

On the Web Crawler service bean, the root directory of your local host, where the Web Crawler saved or changed the crawled objects within the Web space, must be specified together with the previously defined Web space name. Before you can run the text analysis service beans and the catalog service bean, you need to associate them with an existing catalog.

The code that does the customization for the sample is:

```
connection.setServerName(dbName);
connection.setUserid(userName);
connection.setPassword(passwd);
connection.setConnectToIKF(true);
crawlerService.setRootDirectory(webspaceDir);
crawlerService.setWebSpace(webspaceName);

adapter1.setCatalogName(catalog);
categorizationService.setCatalogName(catalog);
catalogService.setCatalogName(catalog);
adapter2.setCatalogName(catalog);
```

Connecting the beans

Figure 50 illustrates the flow of events among the beans in this sample.

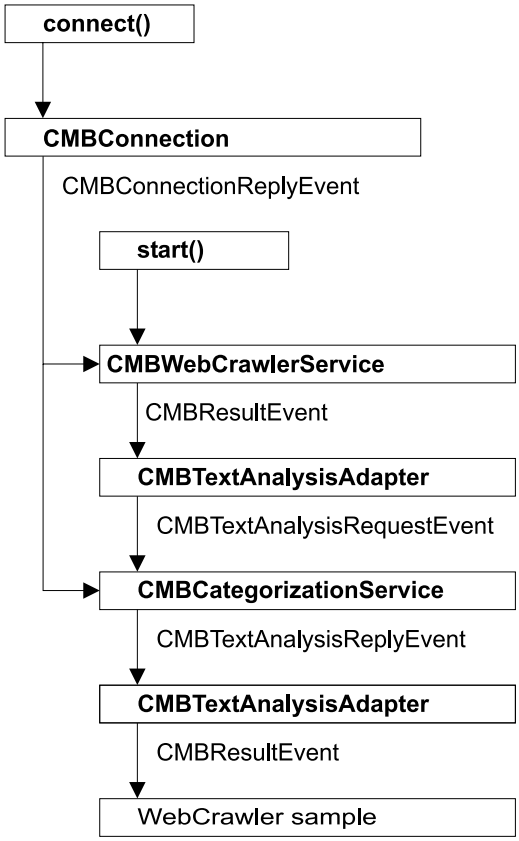

Figure 50. The Web Crawler sample: Event flow

Five of the beans used in this sample listen to the CMBConnectionReplyEvent to get the connection handle. The CMBWebCrawlerService initiates the crawl service that results in an event which then starts the event flow through the other beans.

The code that connects the beans is:

```
connection.addCMBConnectionReplyListener(crawlerService);
connection.addCMBConnectionReplyListener(adapter1);
connection.addCMBConnectionReplyListener(languageIdentificationService);
connection.addCMBConnectionReplyListener(categorizationService);
connection.addCMBConnectionReplyListener(summarizationService);
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(adapter2);

crawlerService.addCMBResultListener(adapter1);
adapter1.addCMBTextAnalysisRequestListener(languageIdentificationService);
languageIdentificationService.
           addCMBTextAnalysisRequestListener(categorizationService);
categorizationService.
           addCMBTextAnalysisRequestListener(summarizationService);
summarizationService.addCMBStoreRecordRequestListener(catalogService);
catalogService.addCMBStoreRecordReplyListener(adapter2);
adapter2.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
crawlerService.addCMBExceptionListener(this);
languageIdentificationService.addCMBExceptionListener(this);
```

```
categorizationService.addCMBExceptionListener(this);
summarizationService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
adapter1.addCMBExceptionListener(this);
adapter2.addCMBExceptionListener(this);
```

Because exceptions are also sent as events, the WebCrawler class has to handle the appropriate event by implementing the CMBExceptionListener interface, and is connected to the beans to be notified about exceptions.

Starting the Web Crawler service

Before you can start the Web Crawler service, you need to establish the connection to the content server by calling the connect method on the CMBConnection bean:

```
connection.connect();
```

The code to start the Web Crawler service is:

```
catalogService.setDefaultCategoryPath
   (catalogService.getTaxonomy().getRootCategory().getPathAsString());
crawlerService.start();
```

To close the current connection, call the disconnect() method:

```
connection.disconnect();
```

Displaying text analysis results

The WebCrawler class implements the CMBResultListener interface to be able to list the documents that have been found during the search and to display the category and summary information created for each document. The CMBResultEvent, received as argument in the onCMBResult method, contains a vector of CMBItem objects, where each CMBItem object represents a document:

```
Vector cmbItemVector = (Vector)e.getData();
```

A CMBItem object encapsulates the PID of the document:

```
CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);
CMBRecord currentRecord = currentItem.getInfoMiningRecord();
System.out.println("\n\nPID       : " + currentRecord.getPID());
```

as well as the results of text analysis.

Then you get the summary and category information:

```
    System.out.println("Categories   : " +
            currentRecord.getValue("IKF_CATEGORIES"));
    System.out.println("Summary    : " +
               currentRecord.getValue("IKF_SUMMARY"));
```

Searching for documents by category

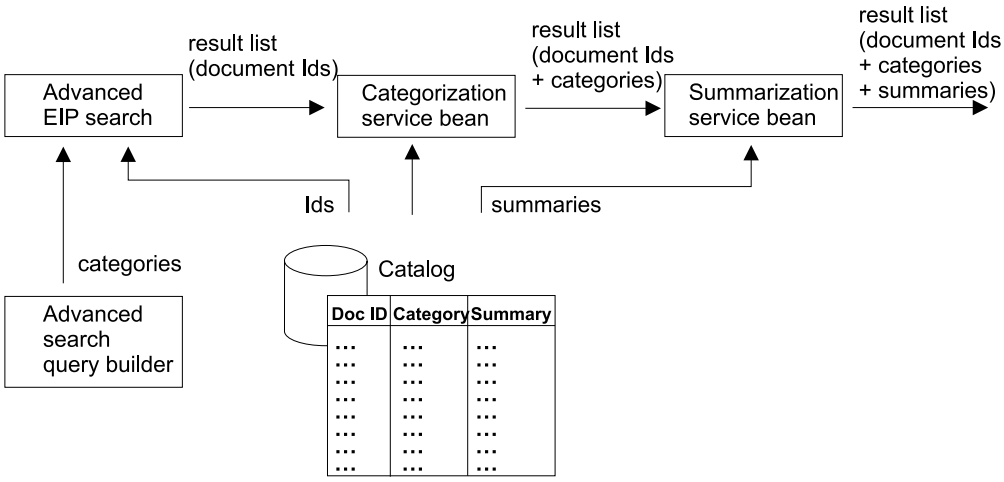

Figure 51. The advanced search sample

This sample demonstrates how to run an advanced search and look up category information in the documents that have been found. An advanced search can only find documents that have been made available for that kind of search as in the categorization sample. See “Location of the sample files” on page 336. In contrast to the previous samples where a standard EIP search is carried out on the entire EIP content server, this sample searches metadata in the data store.

When the advanced search has produced a result list, the IDs of the found documents can be used by the catalog service bean to retrieve the metadata previously stored in the data store.

The following beans are used in this sample:

- CMBConnection
- CMBAdvancedSearchService
- CMBInfoMiningAdapter
- CMBCatalogService

For this sample the application:

1. Creates the beans
2. Customizes the beans
3. Connects the beans
4. Runs the advanced search
5. Displays the search and text analysis results for verification

An explanation of each of the preceding steps follows the source for AdvancedSearch.java.

Complete source for AdvancedSearch.java

```
import java.util.Vector;
import java.io.BufferedReader;
import java.io.InputStreamReader;

import com.ibm.mm.beans.CMBConnection;
import com.ibm.mm.beans.CMBItem;
```

```
import com.ibm.mm.beans.CMBException;
import com.ibm.mm.beans.CMBResultEvent;
import com.ibm.mm.beans.CMBExceptionEvent;
import com.ibm.mm.beans.CMBResultListener;
import com.ibm.mm.beans.CMBExceptionListener;

import com.ibm.mm.beans.infomining.CMBAdvancedSearchService;
import com.ibm.mm.beans.infomining.CMBCatalogService;
import com.ibm.mm.beans.infomining.CMBRecord;
import com.ibm.mm.beans.infomining.CMBCategory;

public class AdvancedSearch implements CMBResultListener, CMBExceptionListener
{

 String CMDBNAME     = "icmnlsdb";
 String CMDBUSER     = "icmadmin";
 String CMDBPASSWORD = "password";

 String CATALOG      = "Sample";
 String SEARCH_TERM  = "Monitor";

 CMBCatalogService  catalogService = null;

  public AdvancedSearch() throws Exception
  {
// ----- Creating beans
   CMBConnection connection = new CMBConnection();
   CMBAdvancedSearchService advancedSearchService = new
                             CMBAdvancedSearchService();
   catalogService = new CMBCatalogService();

// ----- Reading parameters
   BufferedReader din = new BufferedReader(new InputStreamReader(System.in));

   System.out.print("Database    [" + CMDBNAME + "] : ");
   String dbName = din.readLine();
   if (dbName.trim().equals("")) dbName = CMDBNAME;

   System.out.print("User name   [" + CMDBUSER + "] : ");
   String userName = din.readLine();
   if (userName.trim().equals("")) userName = CMDBUSER;

   System.out.print("Password    [" + CMDBPASSWORD + "] : ");
   String passwd = din.readLine();
   if (passwd.trim().equals("")) passwd = CMDBPASSWORD;

   System.out.print("Catalog     [" + CATALOG + "]   : ");
   String catalog = din.readLine();
   if (catalog.trim().equals("")) catalog = CATALOG;

   System.out.print("Search Term [" + SEARCH_TERM + "]  : ");
   String searchTerm = din.readLine();
   if (searchTerm.trim().equals("")) searchTerm = SEARCH_TERM;

   System.out.println("\n\nRunning AdvancedSearch with
                            the following settings:");
   System.out.println("Database    = " + dbName);
   System.out.println("User        = " + userName);
   System.out.println("Password    = " + passwd);
   System.out.println("Catalog     = " + catalog);
   System.out.println("Search Term = " + searchTerm + "\n");

   int key;
   do {
     System.out.print("Continue (y/n)? ");
     InputStreamReader inReader = new InputStreamReader(System.in);
     key = inReader.read();
```

```
     if (key == 'n') System.exit(0); }
   while (key != 'y');

// ----- Customizing beans
   connection.setServerName(dbName);
   connection.setUserid(userName);
   connection.setPassword(passwd);
   connection.setConnectToIKF(true);

   catalogService.setCatalogName(catalog);
   advancedSearchService.setCatalogName(catalog);
   String[] recordKeys = {"IKF_SUMMARY"};
   advancedSearchService.setKeysToBeFetched(recordKeys);

// ----- Connecting beans
   connection.addCMBConnectionReplyListener(catalogService);
   connection.addCMBConnectionReplyListener(advancedSearchService);
   advancedSearchService.addCMBResultListener(this);

   connection.addCMBExceptionListener(this);
   advancedSearchService.addCMBExceptionListener(this);
   catalogService.addCMBExceptionListener(this);

// ----- Running query
   connection.connect();
   advancedSearchService.setCatalogName(catalog);
   advancedSearchService.setQueryString
           ("(\"IKF_CONTENT\" CONTAINS \"'" + searchTerm + "'\")");
   advancedSearchService.runQuery();
   connection.disconnect();
  }
// ----- Implementing com.ibm.mm.beans.CMBResultListener
 public void onCMBResult(CMBResultEvent e)
  {
   if(e.getID() == CMBResultEvent.CMB_RESULT_CLEARED)
    return;

   Vector cmbItemVector = (Vector)e.getData();

   if(cmbItemVector == null)
    return;

  try {
     for(int i = 0; i < cmbItemVector.size(); i++)
      {
       CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);

       CMBRecord currentRecord = currentItem.getInfoMiningRecord();

       String summary = (String)currentRecord.getValue("IKF_SUMMARY");

       CMBCategory[] categories =
                 catalogService.getCategoriesForRecord(currentItem);
       String categoryString    = categories[0].getPathAsString();
       for(int j = 1; j < categories.length; j++)
        {
         categoryString += ", " + categories[j].getPathAsString();
        }

       System.out.println("\n\nPID    : " + currentRecord.getPID());
       System.out.println("Categories : " + categoryString);
       System.out.println("Summary    : " + ((summary == null)?"n/a":summary));
      } /* for */
   }
   catch (CMBException ex)
     { ex.printStackTrace();
     }
```

```
  }

// ---- Implementing com.ibm.mm.beans.CMBExceptionListener
 public void onCMBException(CMBExceptionEvent e)
  {
   ((Exception)e.getData()).printStackTrace();
  }

 public static void main(String[] args)
  {
   try
    {
     new AdvancedSearch();
     System.exit(0);
    }
   catch(Exception e)
    {
     e.printStackTrace();
    }
  }
}
```

Creating the beans

You need a connection to a content server to perform a search. The connection can be established using the CMBConnection bean. The CMBAdvancedSearchService bean is required to perform an advanced search. Category information can be retrieved using the CMBCatalogService bean. The two adapters are required to convert search request events to lookup record request events, and then lookup record reply events back to search result events.

The code that creates the beans is:

```
  CMBConnection connection = new CMBConnection();
  CMBAdvancedSearchService advancedSearchService =
       new CMBAdvancedSearchService();
  catalogService = new CMBCatalogService();
```

Customizing the beans

The code shown in the customizing section has to be adapted according to your installation. 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.

Before you can run the advanced search service and the catalog service bean, you need to associate them with an existing catalog.

The code that does the customization for the sample is:

```
 connection.setServerName(dbName);
 connection.setUserid(userName);
 connection.setPassword(passwd);
 connection.setConnectToIKF(true);

 catalogService.setCatalogName(catalog);
 advancedSearchService.setCatalogName(catalog);
 String[] recordKeys = {"IKF_SUMMARY"};
 advancedSearchService.setKeysToBeFetched(recordKeys);
```

Connecting the beans

Figure 52 on page 369 illustrates the flow of events among the beans in this sample.

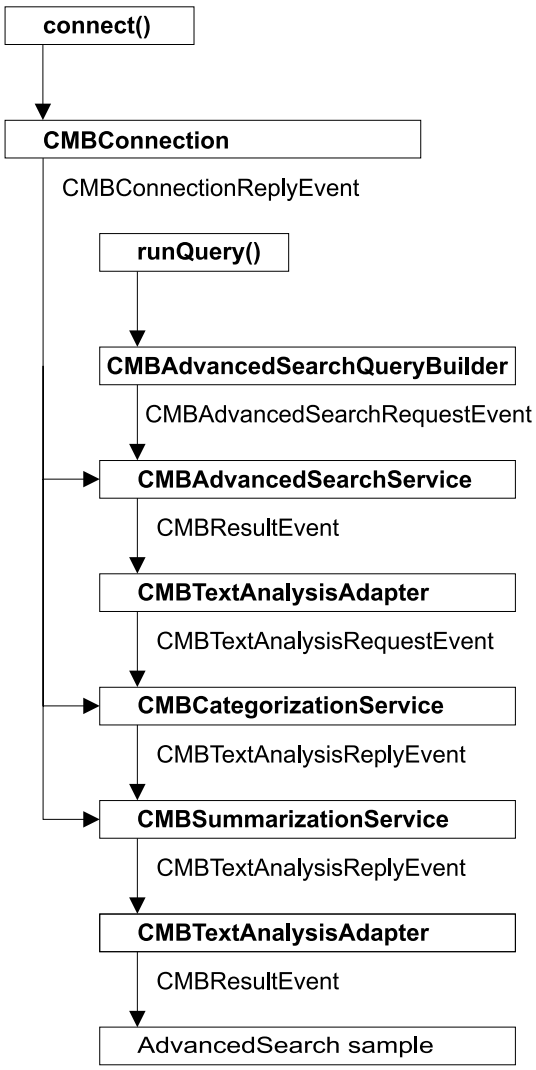

Figure 52. The advanced search sample: Event flow

Two of the beans used in this sample listen to the CMBConnectionReplyEvent to get the connection handle. The CMBAdvancedSearchService bean initiates a search that results in an event which then starts the event flow through the other beans.

The code that connects the beans is:

```
connection.addCMBConnectionReplyListener(catalogService);
connection.addCMBConnectionReplyListener(advancedSearchService);
advancedSearchService.addCMBResultListener(this);

connection.addCMBExceptionListener(this);
advancedSearchService.addCMBExceptionListener(this);
catalogService.addCMBExceptionListener(this);
```

Because exceptions are also sent as events, the AdvancedSearch class has to handle the appropriate event by implementing the CMBExceptionListener interface and is connected to the beans to be notified about exceptions.

Running the query

Before you can run the query you need to establish the connection to the content server by calling the connect method on the CMBConnection bean:

```
connection.connect();
```

To start the advanced search, you need to specify a search query string and the metadata to search in.

You need to adapt the following code according to your configuration:

```
advancedSearchService.setCatalogName(catalog);
advancedSearchService.setQueryString("(\"IKF_CONTENT\" CONTAINS \"'"
                + searchTerm + "'\")");
advancedSearchService.runQuery();
```

To close the current connection, call the disconnect() method:

```
connection.disconnect();
```

Displaying text analysis results

The AdvancedSearch class implements the CMBResultListener interface to be able to list the documents that have been found during the search and to display the category information retrieved for each document. The CMBResultEvent, received as argument in the onCMBResult method, contains a vector of CMBItem objects where each CMBItem object represents a document:

```
Vector cmbItemVector = (Vector)e.getData();
```

A CMBItem object encapsulates the PID of the document:

```
CMBItem currentItem = (CMBItem)cmbItemVector.elementAt(i);
CMBRecord currentRecord = currentItem.getInfoMiningRecord();
String summary = (String)currentRecord.getValue("IKF_SUMMARY");

CMBCategory[] categories =
          catalogService.getCategoriesForRecord(currentItem);
String categoryString = categories[0].getPathAsString();
for(int j = 1; j < categories.length; j++)
{
   categoryString += ", " + categories[j].getPathAsString();
}

System.out.println("\n\nPID       : " + currentRecord.getPID());
```

as well as the results of text analysis beans if there were some in the event flow.

Then you get the category information:

```
System.out.println("Categories    : " +
               currentRecord.getValue("IKF_CATEGORIES"));
```

The vector returned by the getCategories() method of class CMBItem contains objects of class CMBCategory that can be used to determine the absolute path to the current category.

Building your own content provider

You may want to build your own content provider when you need to apply proprietary filters to retrieve text from the binary part of a proprietary format.

This section describes how you can write your own content provider that can deal with a user-defined object model, or with proprietary formats inside the parts of a CMBItem. To support you in this work, Information Mining provides:

- Interface **CMBContentProvider** which defines the interface for classes that know how to determine the text to be used for text analysis.

- Method **setContentProvider(CMBContentProvider)** in CMBInfoMiningUtilities which sets a ContentProvider.

The CMBContentProvider interface defines one method `getContent()` which returns the text of the specified item to be used for text analysis. For example:

```
public CMBTextAnalysisDocument getContent(CMBConnection connection,
   CMBItem item) throws CMBContentProviderException;
```

- Parameter `connection`: an open connection to the server.
- Parameter `item`: the current item to be processed.
- Exception `CMBContentProviderException`: if an error occurs while processing the current item.
- Returns the text as an object of class CMBTextAnalysisDocument.

To tell the system which ContentProvider to use, use the method `setContentProvider(CMBContentProvider)` in the `CMBInfoMiningUtilities` class to specify an object that has this interface. Here's an example:

```
CMBInfoMiningUtilities.setContentProvider
        (new MyCompaniesLatestGreatestContentProvider());
```

To develop your own content provider, you can use the sample content provider (SimpleContentProvider) as a starting point. The content provider sample is located at: `\samples\java\beans\infomining\contentprovider`

A default content provider is provided withinformation mining. See "Document formats supported" on page 383 for a list of the supported formats. It extends the CMBContentProvider interface to allow selecting individual parts for processing, and offers a mechanism for preventing processing objects that are too large for in-memory processing, such as video streams.

To register the default content provider, use:

```
CMBInfoMiningUtilities.setContentProvider(new
                          CMBDefaultContentProvider());
```

Using the service API

The information mining service API is a Java API that integrates information mining functionality as an EIP service. Using the information mining service API you can write applications that can:

- Filter the textual content from different document formats, for example, pdf, Microsoft® Word and HTML
- Perform text analysis on the text documents to create metadata, such as summarization and categorization
- Store metadata for a document in persistent records
- Search for records

Note

Before using the Information Mining beans, it is important to understand the differences between the Service API and the JavaBeans.

- The JavaBeans provide a facility for rapid application development. Certain elements of the code are 'bolted together' and cannot be changed by the user. For more information, refer to "Building an information mining application using the beans" on page 333.
- The Information Mining Service API provides more flexibility for building information mining applications. The code can be seen more as individual 'building blocks' which can be put together to meet your specific requirements.

Connecting to the information mining service API

Before you can use the information mining service API, you need to create a service object using class `DKIKFServiceFed`. Depending on what type of application you want, you have to import the class from one of the following three packages:

- If you want to create an application on a server host, use the following import statement: `import com.ibm.mm.sdk.server.DKIKFServiceFed;`
- If you want to create an application on the client host, use the following import statement: `import com.ibm.mm.sdk.client.DKIKFServiceFed;`
- If you want to remain flexible and create an application that can run on the server as well as the client, use the following import statement: `import com.ibm.mm.sdk.cs.DKIKFServiceFed;`

After using the appropriate import statement, create the `ikfservice` object by using:

```
DKIKFService ikfService = DKIKFServiceFed.create();
```

After creating the service object, you can connect to the database using name, user ID and password.

```
ikfService.connect("databaseName", "userID", "password", null);
```

You now have a connection to the information mining service API and can use classes from the following two packages:

- **com.ibm.mm.sdk.common.infomining** to manage library, catalog and record tasks.
- **com.ibm.mm.sdk.common.infomining.analysis** to use the different text analysis tools.

For an explanation of these key functions, see the following sections.

To disconnect, use:

```
ikfservice.disconnect();
```

Managing the library, taxonomies, and catalogs

The classes to manage the library, catalog, and taxonomy tasks are in the **com.ibm.mm.sdk.common.infomining** package.

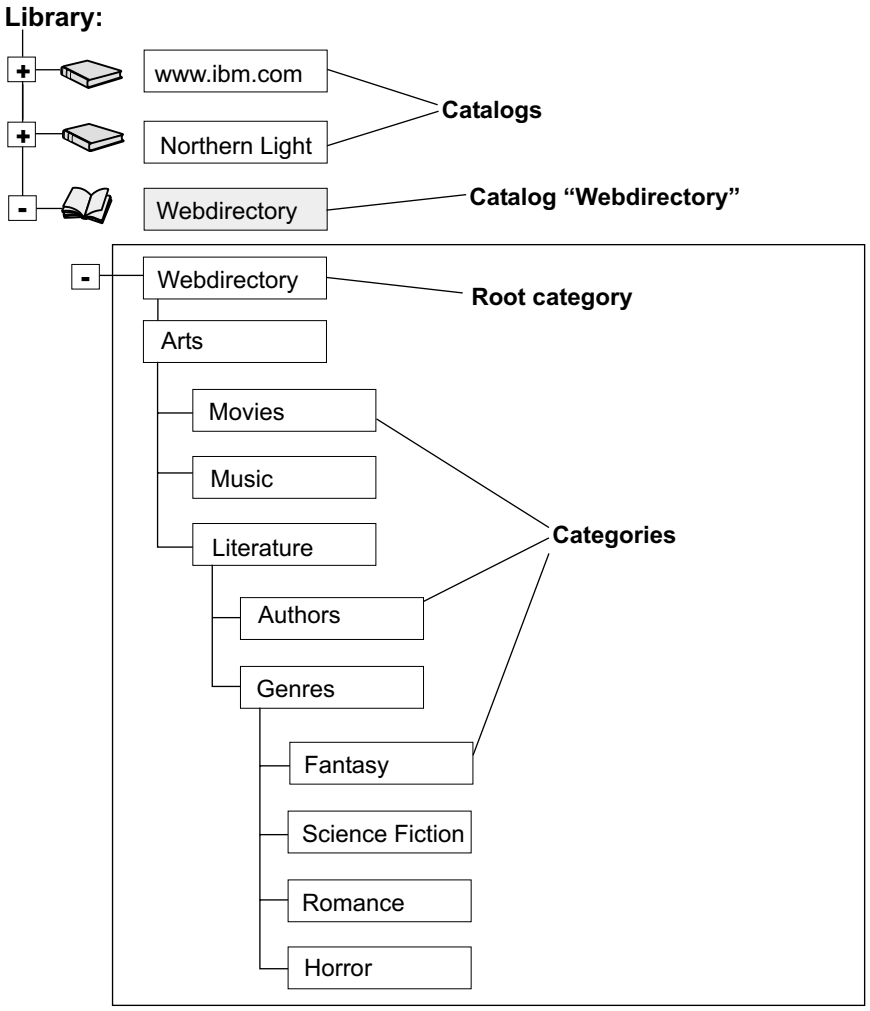

Figure 53. An example of a catalog

In the catalog example, the Library contains three catalogs namely, www.ibm.com, Northern Light, and Webdirectory. Webdirectory has the category tree structure shown in the diagram.

Catalogs can only be created by the Information Structuring Tool (IST).

Note

Working with the Information Mining Service API there are two types of object. There are objects that read the database and store changes on the database. They are known as **persistent** objects, for example, the library or catalog objects.

Likewise, there are objects that are retrieved from the database, but are used independently from the database, they are copies. These are known as **transient** objects, for example, record, schema, category and taxonomy objects.

The tasks determine whether you store information in either a persistent or transient state.

- **Transient** means that the database does not store the data, so information does not automatically update (until update methods are used). This reduces the level of network traffic where data does not frequently change. (EXAMPLE)
- **Persistent** means that the database does store the data, so where data frequently changes, the information automatically updates. (EXAMPLE & CHECK).

This is mostly relevant when working in parallel on the Service API, as it is important that only the most up-to-date information is being worked on and that work being completed by one user is not rendered useless by another. For example, where one user deletes a record whilst another user continues to add and change the same record metadata.

Returning a library

To obtain a library object, use:

```
DKIKFLibrary library = ikfService.getLibrary();
```

Listing all the catalogs in the library

To get the names of all the catalogs in the library, use:

```
String[] catalogNames = library.getCatalogNames();
```

This returns an array listing all the catalog names. The example returns the following catalog names in no specific order: `Webdirectory`, `www.ibm.com` and `Northern Light`.

Returning a specific catalog

To obtain a specific catalog, use:

```
DKIKFCatalog catalog = library.getCatalog(catalogNames[0]);
```

This sample returns the catalog with the first name of the list returned in the previous step. In this case, the `Webdirectory` catalog returns.

Returning the taxonomy of a catalog

To get the taxonomy of a catalog, use:

```
DKIKFTaxonomy taxonomy = catalog.getTaxonomy();
```

The taxonomy object has various methods to work with the categories.

Note

The taxonomy object is read from the database, and all further actions are carried out on this copy.

Listing all the categories in a taxonomy

To get all the categories in a taxonomy, use:

```
DKIKFCategory[] categories = taxonomy.getCategories();
```

This returns an array listing all the categories, including the root category. The example returns the following categories in no specific order: Horror, Author, Music, Literature, Webdirectory, Genres, Science Fiction, Fantasy, Movies, Romance, and Arts.

Note

The category objects are read from the database, and all further actions are carried out on the copy.

Returning the root category

To get the root category, use:

```
DKIKFCategory rootCategory = taxonomy.getRootCategory();
```

This sample returns the root category from the list returned in the previous step. In this case, the Webdirectory root category returns.

Returning a specific category

To return a specific category, use:

```
DKIKFCategory category = taxonomy.getCategory("Webdirectory/Literature/Genres");
```

Using the example, the category Genres returns.

Also use the same separator character ″/″ that is currently set in the taxonomy object.

Note

Three getCategory methods are available in the taxonomy object, each requiring different parameters:

- The path is a string with category names separated using a specified character
- The path is an array of category names
- Another category object

See the Javadoc for further information.

Ensuring that the current taxonomy is the most recent

To ensure that the current taxonomy is the most recent, use:

```
if(taxonomy.getTimestamp().before(catalog.getTaxonomyLastModified()))
  {
  taxonomy = catalog.getTaxonomy();
  }
```

Note

The taxonomy object is read from the database, and all further actions are carried out on this copy. Therefore, ensure that the taxonomy object is the most recent.

Returning the children of a category

To return the children of a category, use:

```
DKIKFCategory[] children = category.getChildren();
```

This returns an array listing all the children of the category. The example based on Genres category, returns the following children in no specific order: Fantasy, Science Fiction, Romance, and Horror.

Note

There can be multiple children for a category, but does not return any further sub-categories from these children.

Returning the parent of a category

To return the parent of a category, use:

```
DKIKFCategory parent = category.getParent();
```

The example based on the Genres category, only returns the Literature category.

Returning a specific catalog schema and list all the attributes in the schema

To get the specific catalog schema, use:

```
DKIKFSchema schema = catalog.getSchema();
Iterator keys = schema.keySet().iterator();

while(keys.hasNext())
  {
  String key = (String)keys.next();
  System.out.println("key: " + key + "type: " + schema.getType(key).getName());
  }
```

The example returns the schema of the Webdirectory catalog.

You can then return the keys (attribute names) from the schema. See the *online API reference* for additional information.

Note

The schema object is read from the database, and all further actions are carried out on this copy.

Using the information mining tools

The classes to use the information mining tools are in the **com.ibm.mm.sdk.common.infomining.analysis** package. Use the tools to:

- Generate a document summary
- Determine the category of the document
- Determine the language of the document

- Extract information, such as names, terms, and expressions from the documents
- Cluster sets of documents

The tools process text documents, which are objects of class DKIKFTextDocument. There are two methods of creating a text document object, namely:

- Create methods of the class DKIKFTextDocument if the document content already exists as a Java string
- DKIKFDocumentFilter class, if the document content is in a formatted form

Both classes are in the **com.ibm.mm.sdk.common.infomining** package.

Text Document

To create a text document object if the document content exists as a Java string, use:

```
DKIKFTextDocument document = DKIKFTextDocument.create("the document content");
```

Note

Three different create methods are available to:

- Create a text document with specified content
- Create a text document with specified content and name
- Create a text document with specified content, name, and language

See the Javadoc for further information.

Filtering the document

To create a text document object from a formatted document, use:

```
DKIKFDocumentFilter documentFilter = new DKIKFDocumentFilter(ikfService);
byte[] documentBytes = ... //set the bytes of the document from a data source
DKIKFTextDocument document2 = documentFilter.getTextDocument(documentBytes);
```

Note that the original text document can be in any one of the supported format, for example, a pdf or Microsoft Word document.

Use the method of the document filter object to set filter encoding. See the *online API reference* for further information.

Determining the document language

To determine the language of a document, use:

```
DKIKFLanguageIdentifier languageIdentifier =
                  new DKIKFLanguageIdentifier(ikfService);
DKIKFLanguageIdentificationResult[] languageResults =
                  languageIdentifier.analyze(document);
document.setLanguage(languageResults[0].getLanguage());
```

This returns an array listing all the objects containing language and confidence value results, with the highest confidence value listed first.

Use the methods of the language result objects to return confidence values and language.

```
string language = languageResults[0].getLanguage();
float  confidence = languageResults[0].getConfidence();
```

For further information on the Language Identification service and the default settings, see the *online API reference.*

Generating a document summary

To create a document summary, use:

```
DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);
DKIKFSummarizationResult summarizationResult = summarizer.analyze(document);
```

To return the summary, use:

```
string summary = summarizationResult.getSummary();
```

For further information on the Summarization service and the default settings, see the *online API reference*.

Extracting document information

To extract information from a document, use:

```
DKIKFInformationExtractor extractor = new DKIKFInformationExtractor(ikfService);
DKIKFInformationExtractionResult information = extractor.analyze(document);
```

Use the methods of the information extraction result objects to explore additional analytical functionality.

```
DKIKFFeature[] features = information.getFeatures();
```

For further information on the Information Extraction service and the default settings, see the *online API reference*.

Clustering

You can cluster documents, so that similar documents are grouped together by using:

```
DKIKFClusterer clusterer = new DKIKFClusterer();
cluster.analyze(doc1);
cluster.analyze(doc2);
cluster.analyze(doc3);
DKIKFClusterResult clusterResult = clusterer.cluster();
```

Use the methods of the cluster result object to return the cluster nodes, the number of clusters, and the total number of documents.

```
int clusterCount = clusterResult.getClusterCount();
DKIKFClusterNode[]nodes = clusterResult.getClusterNodes();
int documentCount = clusterResult.getDocumentCount()
```

For further information on the Clustering service and the default settings, see the *online API reference*.

Categorizing

To assign categories to documents, use:

```
DKIKFCategorizer categorizer = new DKIKFCategorizer(catalog);
DKIKFCategorizationResult[] categorizationResults = categorizer.analyze(document);
```

This returns an array listing all the categorization result objects containing categories and confidence values, with the highest confidence value listed first.

Use the methods of the categorizer result objects to return confidence values and the categories.

```
DKIKFCategory bestCategory = categorizationResults[0].getCategory();
```

Note

To use the categorization service, you must first create and train the catalog using the Information Structuring Tool (IST). The categorization service uses one catalog.

Also note that the returned category objects do not belong to the taxonomy object. Use the taxonomy methods to traverse parent and child categories.

For further information on the Categorization service and the default settings, see the *online API reference*.

Creating records and storing metadata in catalogs

The classes to create records and store metadata are in the **com.ibm.mm.sdk.common.infomining** package. Use these to:

- Create a new record in a catalog
- Retrieve a record from a catalog
- Return the categories for a record
- Update a record value
- Update record category assignment
- Delete a record

Note that the information mining service API can only manage catalogs created by the Information Structuring Tool (IST). Once created, records and metadata can then be added.

Creating a new record in a catalog

A record is created using a PID and a catalog schema:

```
DKIKFRecord record = DKIKFRecord.create("PID", schema);
record.setValue("IKF_TITLE", "This is the title");
record.setValue("IKF_SUMMARY", "This is the document summary");
record.setValue("IKF_CONTENT", "This is the document content");
DKIKFCategory[] categoriesParam = {bestCategory};
catalog.createRecord(record, categoriesParam);
```

In the example, the title, summary and content values of the record are set. The record is created using the values set above and can be assigned to one or more categories.

Retrieving a record from a catalog

To retrieve a record from a catalog using the PID, enter:

```
DKIKFRecord retrievedRecord = catalog.getRecord("PID");
```

Returning the categories for a record

To return the categories of a record, use:

```
DKIKFCategory[] recordCategories = catalog.getCategoriesForRecord("PID");
```

Updating a record value in a catalog

To update a record value, for example the title of a record, use:

```
record.setValue("IKF_TITLE", "This is the new title");
catalog.updateRecord(record);
```

Note

Three different update methods are available to:

- Update the record with record values only (see the above title example)
- Update the record with record values and assigned categories
- Update the record with assigned categories only (see the example below where a record in one category is added to another category as well)

See the Javadoc for further information.

Adding a record in a category to another category

To add a record to another category, use:

```
DKIKFCategory oldCategories = catalog.getCategoriesForRecord("PID");
DKIKFCategory newCategory = taxonomy.getCategory("Webdirectory/arts/movies");
List categoriesList = Arrays.asList(oldCategories);
categoriesList.add(newCategory);
catalog.updateRecord("PID",
   categoriesList.toArray(new DKIKFCategory[categoryList.size()]));
```

In the example, the record is added to a new Movies category. If you want to keep your original category assignments, you must include the original categories in the update record call, as shown in the above example. An update record call on a new category only will remove all original category assignments.

There are three methods to update a record, see “Updating a record value in a catalog” on page 379 for further information.

The record can also be added to the root category and to more than one category. See the *online API reference* for details.

Deleting a stored record

To delete a record, use:

```
catalog.deleteRecord("PID");
```

Searching for documents

The classes to search the records are in the **com.ibm.mm.sdk.common.infomining** package.

Finding records containing a specific word

For example, to find a record containing the word ″Bach″ in the category ″Music″, use:

```
String queryString = "(\"IKF_CONTENT\" contains \"Bach\") and
  (DKIKF_CATEGORY = \"Webdirectory/Music\")";
DKIKFSearchConfiguration searchConfiguration = new DKIKFSearchConfiguration();
searchConfiguration.setTaxonomy(taxonomy);
DKIKFSearchResult searchResult =
      catalog.searchRecords(queryString, searchConfiguration);
Iterator resultPIDs = searchResult.iterator();
```

To run the searchRecords method on the catalog object you need:

- A query string, and
- A search configuration object

The search configuration object specifies search properties, for example, the maximum number of search results. If you are using categories in the query string, the taxonomy object is also necessary.

The search results can be PID strings or records, and can be specified in the DKIKFSearchConfiguration call.

Running a server task

If you want to execute a defined sequence of tasks from a client application, you can bundle these calls in a server task, send it across to the server once, and then call this server task as often as is needed from the client. This keeps the level of network traffic as low as possible and improves performance significantly.

A server task is an object that implements the interface `com.ibm.mm.sdk.common.infomining.DKIKFServerTask`. The object is instantiated from the client application, set on the service object using the `setServerTask` method and then executed on the server using the `runServerTask` method.

In the example below, an object of class AnalysisTask, which implements the server task interface is set on the service and then executed:

```
...
ikfService.setServerTask(new AnalysisTask(secondCatalog.getName()));
HashMap documentMap = new HashMap();
documentMap.put("PID1", DKIKFTextDocument.create("content1"));
documentMap.put("PID2", DKIKFTextDocument.create("content2"));
documentMap.put("PID3", DKIKFTextDocument.create("content3"));
Map recordMap = (Map)ikfService.runServerTask(documentMap);

Iterator pids = recordMap.keySet().iterator();
while(pids.hasNext())
  {
  DKIKFRecord record = (DKIKFRecord)recordMap.get(pids.next());
  System.out.println(record.getPID());
  System.out.println(record.getValue("IKF_LANGUAGE"));
  System.out.println(record.getValue("IKF_SUMMARY"));
  }
...
```

A map of documents is passed to the server task for processing. The server task returns a map of the records for the specified documents containing the created metadata (document language and summary in this case).

The source for the sample server task is as follows:

```
public class AnalysisTask implements DKIKFServerTask {
private String catalogName;
private DKIKFSchema catalogSchema;

public AnalysisTask(String catalogName) {
  this.catalogName = catalogName;
}

public Serializable runServerTask(DKIKFService ikfService, Serializable argument)
                    throws DKIKFServerTaskException {
  try {
    //the schema is retrieved only once
    if(catalogSchema == null) {
     catalogSchema = ikfService.getLibrary().getCatalog(catalogName).getSchema();
    }

    //preparing the argument, tools, and the map to be returned
    Map documentMap = (Map)argument;
```

```
      DKIKFLanguageIdentifier languageIdentifier =
                          new DKIKFLanguageIdentifier(ikfService);
      DKIKFSummarizer summarizer = new DKIKFSummarizer(ikfService);
      HashMap recordMap = new HashMap();
      Iterator pids = documentMap.keySet().iterator();

      //creating a record for each pid
      while(pids.hasNext()) {
        String pid = (String)pids.next();
        DKIKFTextDocument document =
                (DKIKFTextDocument)documentMap.get(pid);
        DKIKFRecord record = DKIKFRecord.create(pid, catalogSchema);
        String language = languageIdentifier.analyze(document)[0].getLanguage();
        document.setLanguage(language);
        record.setValue("IKF_LANGUAGE", language);
        record.setValue("IKF_SUMMARY", summarizer.analyze(document).getSummary());
        recordMap.put(pid, record);
      }

      //returning the results
      return recordMap;
    }
    catch(Exception e) {
      throw new DKIKFServerTaskException(e);
     }
  }
}
```

The server task is instantiated with the name of the catalog that contains the schema required for record creation. The method `runServerTask` retrieves the schema only once. For each of the documents, it creates a record, runs the tools to analyze the document, and stores the created metadata in the record. Finally, all created records are returned to the caller (the client application).

Note

Only the metadata is returned in the above example code, no records are created in the catalog.

Example of an information mining application based on JSPs

The information mining Java Server Page (JSP) application searches within the information mining component for documents within categories. It displays the found documents in a category structure.

The JSP sample application is located in the following directory:

Windows NT® `\samples\jsp\clientapp\infomining`

AIX `/usr/ikf/samples/jsp`

Solaris `/opt/IBMcmb/samples/jsp/infomining/`

The directory contains these files:

advSearch.jsp The top-level file of the sample.

This part provides the advanced search-specific form-handling code and form-formatting instructions (HTML). It also acts as the controller for selecting views of the data. This file contains

initialization instructions specific to the advanced search, in particular the availability of categories that can be searched in.

catView.jsp This part provides the view-specific code and formatting instructions (HTML) for the category view of returned results. It contains loops for iterating through returned results, but it mostly contains formatting instructions.

classes.jsp This part provides the logic and bean-connection code. It contains only Java code. There is an implementation of a simple data structure class used for the viewing of returned results. There is also an implementation of an event handler to retrieve and manipulate returned results. This is where the core of the work is done after a results list has been returned from the advanced search.

logon.html The account and catalog input required to run the sample.

The account and catalog input comprises server name, user name, password and catalog name.

The source code is a sample of using the information mining beans, 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 Enterprise Information Portal and the information mining component installed. You also need a Web server that is capable of running JSPs. For details on how to install the JSPs, refer to the *EIP Planning and Installation Guide.*

For more details on JSP applications, go to:
`http://java.sun.com/products/jsp/index.html`

Document formats supported

The document format support in Enterprise Information Portal information mining is supplied by IntraNet Solutions, Inc. and is called Outside In Viewer Technology. The following formats are supported:

Word processing: Generic

ANSI Text (7 & 8 bit) All versions

ASCII Text (7 & 8 bit versions available) All versions

HTML Versions through 3.0 (some limitations)

IBM FFT All versions

IBM Revisable Form Text All versions

Microsoft Rich Text Format (RTF) All versions

Unicode Text All versions

Word processing: DOS

DEC WPS Plus (DX) Versions through 4.0

DEC WPS Plus (WPL) Versions through 4.1

DisplayWrite® 2 & 3 (TXT)	All versions
DisplayWrite 4 & 5	Versions through release 2.0
Enable	Versions 3.0, 4.0 and 4.5
First Choice	Versions through 3.0
Framework	Version 3.0
IBM Writing Assistant	Version 1.01
Lotus Manuscript	Versions through 2.0
MASS11	Versions through 8.0
Microsoft Word	Versions through 6.0
Microsoft Works	Versions through 2.0
MultiMate	Versions through 4.0
Navy DIF	All versions
Nota Bene	Version 3.0
Office Writer	Version 4.0 to 6.0
PC-File Letter	Versions through 5.0
PC-File+ Letter	Versions through 3.0
PFS:Write	Versions A, B, and C
Professional Write	Versions through 2.1
Q&A	Version 2.0
Samna Word	Versions through 4.0
SmartWare II	Version 1.02
Sprint	Versions 1.0
Total Word	Version 1.2
Volkswriter 3 & 4	Versions through 1.0
Wang PC (IWP)	Versions through 2.6
WordMARC	Versions through Composer Plus
WordPerfect	Versions through 6.1
WordStar	Version through 7.0
WordStar 2000	Version through 3.0
XyWrite	Version through III Plus

Word processing: International

JustSystems Ichitaro	Versions 5.0, 6.0, 8.0, 9.0 and 10.0

Word processing: Windows

AMI/AMI Professional	Versions through 3.1
Corel WordPerfect for Windows	Versions through 9.0
JustWrite	Versions through 3.0

Legacy	Versions through 1.1
Lotus WordPro (Win32 / Intel platforms)	SmartSuite® 96, 97 and Millennium
Lotus WordPro (Unix platforms - text only)	SmartSuite 97 and Millennium
Microsoft Windows Works	Versions through 4.0
Microsoft Windows Write	Versions through 3.0
Microsoft Word 97	Word 97
Microsoft Word 2000	Word 2000
Microsoft Word for Windows	Versions through 7.0
Microsoft WordPad	All versions
Novell Perfect Works	Version 2.0
Novell WordPerfect for Windows	Versions through 7.0
Professional Write Plus	Version 1.0
Q&A Write for Windows	Versions 3.0
WordStar for Windows	Version 1.0

Word processing: Macintosh

Microsoft Word	Versions 4.0 through 6.0
Microsoft Word 98	Word 98
WordPerfect	Versions 1.02 through 3.0
Microsoft Works	Versions through 2.0
MacWrite II	Version 1.1

Spreadsheets formats

VP Planner 3D	Version 1.0
Enable	Versions 3.0, 4.0 and 4.5
First Choice	Versions through 3.0
Framework	Version 3.0
Lotus 1-2-3® (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 for SmartSuite	SmartSuite 97and Millennium
Lotus 1-2-3 Charts (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 (OS/2®)	Versions through 2.0
Lotus 1-2-3 Charts (OS/2)	Version through 2.0
Lotus Symphony	Versions 1.0, 1.1 and 2.0
Microsoft Excel 97	Excel 97
Microsoft Excel 2000	Excel 2000
Microsoft Excel Macintosh	Version 3.0 through 4.0, 98

Microsoft Excel Windows	Versions 2.2 through 7.0
Microsoft Excel Charts	Versions 2.x through 7.0
Microsoft Multiplan	Version 4.0
Microsoft Windows Works	Versions through 4.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Mosaic Twin	Version 2.5
Novell Perfect Works	Version 2.0
QuattroPro for DOS	Versions through 5.0
QuattroPro for Windows	Versions through 9.0
PFS:Professional Plan	Version 1.0
SuperCalc 5	Version 4.0
SmartWare II	Version 1.02

Database formats

SmartWare II	Version 1.02
Access	Versions through 2.0
dBase	Versions through 5.0
DataEase	Version 4.x
dBXL	Version 1.3
Enable	Versions 3.0, 4.0 and 4.5
First Choice	Versions through 3.0
FoxBase	Version 2.1
Framework	Version 3.0
Microsoft Windows Works	Versions through 4.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Paradox (DOS)	Versions through 4.0
Paradox (Windows)	Versions through 1.0
Personal R:BASE	Version 1.0
R:BASE 5000	Versions through 3.1
R:BASE System V	Version 1.0
Q & A	Versions through 2.0
Reflex	Version 2.0

Standard graphic formats

PNG - Portable Network Graphics Internet Format	Version 1.0
Binary Group 3 Fax	All versions

BMP(including RLE, ICO, CUR & os/2 DIB)
Windows

CDR (if TIFF image is embedded in it)
Coral Draw versions 2.0 – 9.0

CGM - Computer Graphics Metafile
ANSI, CALS, NIST, Version 3.0

CMX - Corel Clip Art Format Versions 5 through 6

DCX (multi-page PCX) Microsoft Fax

DRW - Micrografx Designer Version 3.1

DRW - Micrografx Draw Versions through 4.0

DXF (Binary and ASCII) AutoCAD Drawing Interchange Format
Versions through 14

EMF Windows Enhanced Metafile

EPS Encapsulated PostScript If TIFF image is embedded in it

FMV - FrameMaker graphics Vector and raster format through Version 5.0

FPX - Kodak Flash Pix No specific format

GDF - IBM Graphics Data Format
Version 1.0

GEM - Graphics Environment Manager Metafile
Bitmap and Vector

GIF - Graphics Interchange Format
Compuserve

GP4 - Group 4 CALS Format Type I and Type II

HPGL - Hewlett Packard Graphics Language
Version 2.0

IMG - GEM Paint No specific version

JFIF (JPEG not in TIFF format)
All versions

JPEG - Joint Photographic Experts Group format
All versions

MET - OS/2 PM Metafile Version 3.0

PBM - Portable Bitmap No specific version

Kodak Photo CD Version 1.0

PCD - PCX Bitmap PC Paintbrush

Perfect Works (Draw) Novell version 2.0

PGM - Portable Graymap No specific version

PIC - Lotus 1-2-3 Picture File Format
No specific version

PICT1 & PICT2 (Raster) Macintosh Standard

PIF - IBM Picture Interchange Format
Version 1.0

PNTG	MacPaint
PPM - Portable Pixmap	No specific version
Progressive JPEG	No specific version
PSP - Paintshop Pro (Win32 only)	Versions 5.0, 5.0.1
RND - AutoShade Rendering File Format	Version 2.0
SDW Ami Draw Snapshot (Lotus)	All versions
SRS - Sun Raster File Format	No specific version
Targa	Truevision
TIFF	Versions through 6
TIFF CCITT Group 3 & 4	Fax Systems
VISO (Page Preview mode only for Version 4)Visio 4, 5, 2000	Visio 4, 5, 2000
WMF	Windows Metafile
WordPerfect Graphics [WPG and WPG2]	Versions through 2.0
XBM - X-Windows Bitmap	x10 compatible
XPM - X-Windows Pixmap	x10 compatible
XWD - X-Windows Dump	x10 compatible

High-end graphics formats

PSD - Adobe Photoshop File Format	Version 4.0
AI - Adobe Illustrator File Format	Versions through 7.0
CDR - Corel Draw	Versions through 8.0
DSF - Micrografx Designer	Windows 95, version 6.0
DWG - AutoCAD Native Drawing Format	Versions 12 through 14
IGES - Initial Graphics Exchange Specification	Version 5.1
PDF - Portable Document Format	Acrobat version 2.1, 3.0, 4.0, including Japanese PDF
PS - Postscript	Level 2

Presentation formats

Microsoft PowerPoint for Macintosh	Version 4.0, 98
Corel Presentations	Version 8.0 and 9.0
Novell Presentations	Versions 3.0 and 7.0

Harvard Graphics for DOS	Versions 2.x and 3.x
Harvard Graphics	Windows versions
Freelance 96	Freelance 96
Freelance for Windows 95	SmartSuite 97 and Millennium
Freelance for Windows	Version 1.0 and 2.0
Freelance for OS/2	Versions through 2.0
Microsoft PowerPoint for Windows	Versions through 7.0
Microsoft PowerPoint 97	PowerPoint 97
Microsoft PowerPoint 2000	PowerPoint 2000

Compressed and encoded formats

ZIP PKWARE	Versions through 2.0g
GZIP	No specific version
LZA Self Extracting Compress	No specific version
LZH Compress	No specific version
Microsoft Binder	Version 7.0, Binder 97
MIME (text mail)	No specific version
UUEncode	No specific version
UNIX Compress	No specific version
UNIX TAR	No specific version

Other

vCard Electronic Business Card	Versit Version 2.1
Executable (EXE, DLL)	No specific version
Executable for Windows NT	No specific version
MSG (text only)	Microsoft Outlook mail format
Microsoft Project (text only)	Project 98

Chapter 13. Working with Content Manager 8.1 in C++

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

You can use the ICM connector APIs to build and deploy custom applications that access a Content Manager content server. You can also use the APIs to integrate your existing applications into a Content Manager content server.

This section contains the following information:

- Understanding the Content Manager system
- Understanding Content Manager concepts
- Planning a Content Manager application
- Creating a Content Manager application
- Controlling access to information
- Processing transactions
- Searching for items

Understanding the Content Manager system

The main components of the Content Manager system include a library server, one or more resource managers, and a set of object-oriented application programming interfaces (APIs). To administer your Content Manager system, you are also provided with a Java-based system administration client.

The library server provides you with flexible data modeling capabilities, secure access to your system, efficient managing of content, and other features. The library server manages the relationships between items in the system and controls access to all of the system information, including the information stored in the resource managers configured in the system.

The resource manager is the component that stores the actual content of any binary object, like a scanned image, an office document, or video. You can integrate other resource managers, like Content Manager VideoCharger or other non-IBM products into your Content Manager system. With the resource manager you can complete the following tasks:

- Automatically move content from slow, costly media to high speed, less expensive media using System Managed Storage (SMS)
- Access the resource manager directly from a Web browser
- Retrieve all or part of an object
- Synchronize your data with the library server

Available in Java and C++, the APIs provide applications with access to the Content Manager system. Using the APIs, your applications can take advantage of all of the Content Manager functionality, such as data modeling, integrated parametric and text search, third-party data access and delivery, and so forth.

The diagram in Figure 54 illustrates how the system components fit together. Keep in mind that this is only one implementation of a Content Manager system. In another system configuration you might have four resource managers, for example.

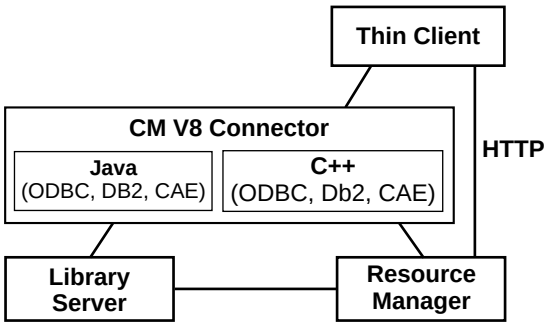

Figure 54. System configuration

Understanding Content Manager concepts

This section describes important Content Manager concepts. It is imperative that you understand the Content Manager concepts before you proceed to the programming tasks. The information described in this section includes:

- Items
- Attributes
- Item types
- Root and child components
- Objects
- Links and References
- Documents
- Folders
- Versioning
- Access control
- Data model
- Document management data model

Items

An item is the basic entity managed by the library server. Examples of items include a policy, claim, phone number, and so forth. An *item* is a generic term for an instance of an item type. If an object is a discrete piece of digital content, then an item is a representation of that object. The item is not the object, but it thoroughly identifies it and how to find it. In the system, items are used to represent objects, including documents and folders. To define business objects, like a document, you work with item definitions.

Items can be resource items or non-resource items. Resource items have associated data stored in the resource manager and in the library server. Non-resource items exist only in the library server (as meta data only).

When an application creates an item, Content Manager assigns the item several system-defined attributes and allows you to define your own attributes.

The system-defined attributes include a creation time stamp and an item identifier (item ID). The item ID is unique for every item. The itemID is stored by Content Manager and used to locate the item within the library server. When writing your application, you use the itemID to access all of the data associated with the item.

Attributes

An *attribute* is a unit of data that describes a certain characteristic or property (for example, name, address, age, and so forth) of an item, and which can be used to locate that item.

You can group attributes into attribute groups. For example, the address attribute can be made up of a group of attributes including street, city, state, and zip code.

You can also define attributes that have multiple values. Such attributes are called multi-valued attributes. For example, you can store multiple addresses, home address, work address, and so forth, for a policy owner.

Attribute groups

Attribute groups are a convenience grouping of one or more attributes. For example, the attribute group Address might include the attributes Street, City, State, and Zip.

Item types

An *item type* (index class in earlier Content Manager versions) is essentially a template for defining and later locating like items. An item type consists of a root component, zero or more child components, and a classification. An item type is the overall structure containing all the components and associated data. For example, in an insurance scenario a policy item type contains items like policy number, name, claim, and so forth.

In Content Manager, there are two kinds of item types: non-resource item types and resource item types. A non-resource item type represents entities that are not stored in a resource manager. A resource item type represents objects stored in a resource manager, like files in a file system, video clips in a video server, LOBs (large objects) in database tables, and so forth. Content Manager provides a base set of resource item types: LOB, text, image, stream, and video objects.

Root and child components

An item type is composed of components: a root component and any number of child components, which are optional.

A *root component* is the first or only level of a hierarchical item type. An item type consists of both system-defined and user-defined attributes. Internally, the most basic item type contains only one component.

A *child component* is an optional second or lower level of a hierarchical item type. Each child component is directly associated with the level above it.

Figure 55 on page 394 shows an example of a root and child component hierarchy.

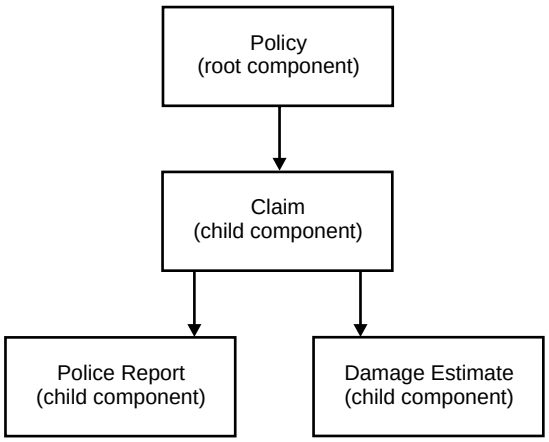

Figure 55. Components hierarchy

Objects

An *object* is any digital content that a user can store, retrieve, and manipulate as a single unit (for example, JPEG images, MP3 audio, AVI video, and a text block from a book). An object is always stored in a resource manager. Access to an object is controlled by the library server.

Links and references

A *link* is a one-to-many association between items. Links can only be used to associate items that are root components. You can define an unlimited number of links.

You can use a link to represent the foldering or container-containee relationship. If you choose to implement foldering using links, remember that the container does not *own* the containee, which means that the items in the container are not deleted when the container is deleted.

A *reference* is a one-to-one association between components of an item to other items. A reference is represented as a reference attribute in a component. A component can have any number of reference attributes that refer to other root components. A reference usually does not indicate ownership, but you can implement an ownership relationship if necessary.

When you add a reference to a component type, items of that component type can refer to another item. In terms of the DDO, the DDO has an attribute that is identified by the name of the reference. The attribute's value can be set to another DDO. The attribute value for the DDO is the DDO that is referred to by the reference.

References can be defined in both root and child component types. References also refer to a specific version of an item, whereas links refer to all versions. References are also modeled in the data definition and links are not.

Documents

A document is an object comprised of other objects containing content (scanned image, office document, video clip). Additionally, a document can be associated with objects like an annotation, a note, or other affiliated information.

You can use child components to associate objects with a document. You can use more than one object to construct a document. For example, each page in a document can be a separate object. An object number is stored in the child component, providing the order of the objects to an application. You can also use a child component to point to an object (a reference attribute) that contains additional information about the object such as size, the resource manager ID on which the object is stored, the collection name for the object, and so forth.

Folders

A *folder* is an item that contains any kind of item and can also contain other folders. In Content Manager Version 8 Release 1, the concept of folders is implemented by using link relationships between items. The document-folder relationship is conceptually a container-containee relationship in the Content Manager system. Items can contain other items to form a containment hierarchy called a folder hierarchy. For example, a policy item belongs to the policy item type, and potentially has many claims, making policy a folder that holds other items such as a photo, a social security number, and so forth. Folders are very flexible because any item can be a folder and can contain any number of other items.

Versioning

Versioning is the ability to store and maintain multiple versions of an item, including versions of the item's child components. You specify versioning when you define an item type. Every instance (item) of an item type that is enabled for versioning is also enabled for versioning. When an item is enabled for versioning, a new version of the item is created every time the item is updated and the new version of the item is stored into the datastore.

Versioning is handled by the Content Manager library server. Different versions of version-enabled objects are stored in the resource manager as separate objects. Important characteristics of versioning support include the following:

- Versioning involves a root component and its entire hierarchy.
- Item types can have one of three possible versioning policies: versioned-always, versioned-never (the default) and application-controlled versioning.
- All of the versions of an item in the CM system are searchable and retrievable.
- Any version of an item can be updated and deleted.
- When a version-enabled item is updated, the user has the option of applying the updates to the existing version or creating a new version based on the updates.
- Each version of an item has its own persistent identifier (PID). The PID has several parts of which two are relevant in the current context. The first relevant part is the ItemId which is the same across all different versions of the item. The other is the version number. Each version of the item has a different version number that can be retrieved and set as a string. Below is a sample that demonstrates how to work with version numbers.

```
DKPidICM * pid = (DKPidICM *)ddo->getPidObject();
   //Retrieving versioning information for an item (DDO) as a string
    DKString version = pid->getVersionNumber();
    ....
//Setting the version number for the PID associated with an item (DDO).
    pid->setVersionNumber((char *)version);
```

- If an update to an item exceeds the maximum number of allowed, the oldest saved version is dropped and a new version is created by the system.
- If a version-enabled item is re-indexed, all previous versions of the item are automatically deleted.

- Child components of an item inherit the version of their parent component.
- The version of a child component type cannot be changed, since it follows the versioning of its parent type.
- For an item with parts, if the item type has a version policy of versioned-never, the various parts in the item type also have a version policy of versioned-never.
- Part-level versioning rules can be obtained from the item type relation object that represents the types.

Access control

The Content Manager access control model is comprised of the following fundamental elements:

- Privileges and privilege sets
- Controlled entities
- Users and user groups
- Access control lists

The various access control elements work as follows. Each Content Manager user is granted a set of user privileges. These privileges define the a maximum operation a user can perform. A user's effective access rights will never exceed the user's defined privileges.

The access control model of Content Manager is applied to the controlled entity. A *controlled entity* is a unit of protected user data. In Content Manager, the controlled entity can be at the level of item, item-type, or at the level of the entire library. For example, you can bind an ACL to an item type to enforce access control at the item type level. Operations on controlled entities is regulated by one or more control rules, called access control lists (ACLs). Every controlled entity in Content Manager system must be bound to an ACL.

When a user initiates an operation on an item, the system checks the user's privilege and the ACL bound to the item to determine if the user has the right to do such an operation on the item. Logically, the right to access an item also requires the right to access the item type, where the item is defined. Figure 56 shows an example of how the system determines a user's access rights to an item based on privileges and ACLs.

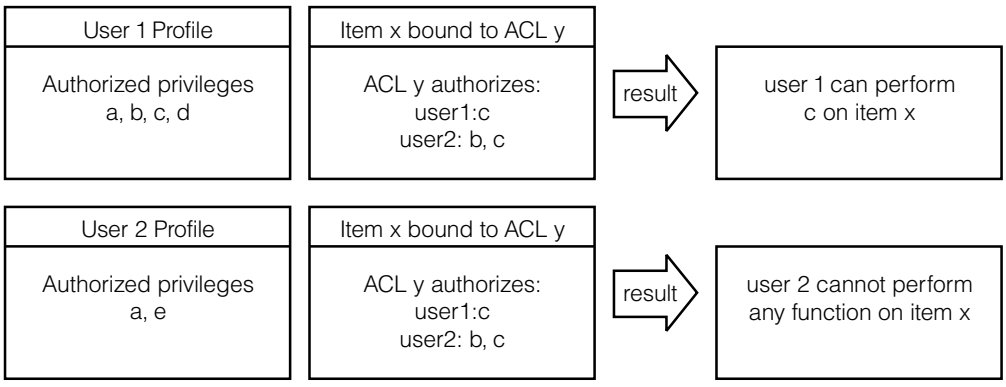

Figure 56. Access control diagram

Privileges and privilege sets

Privileges allow a user to perform a specific action on an item in the system, such as create or delete it. Every Content Manager user is granted a set of user

privileges. The privileges define the maximum operations a user can perform on information in the Content Manager system. A user's access rights do not exceed the defined user privileges for the user.

Content Manager provides a number of pre-defined privileges that you cannot change, called system-defined privileges. You can also define your own privileges, called user-defined privileges. You enforce user-defined privileges in your application using user exit routines.

Every privilege has a system-generated, unique code called a privilege definition code. The privilege definition codes 0 to 999 are reserved for system-defined privileges. You can use codes of 1000 and above for user-defined privileges.

The system-defined privileges are classified into two categories: system administration privileges, and data access privileges. You can use the system administration privileges to model user data, administer, and maintain the Content Manager system. You need system administration privileges to complete tasks such as configuring the system, managing the library server configuration, and managing item types. You can use the data access privileges to access and modify the system data, like items and item types.

A group of privileges assigned to a user is a *privilege set*. For example, one privilege set can contain the privileges create, update, and delete. Privilege sets allow for easier system administration. You must group privileges into a set before you can use them. There is no limitation on the number of privileges a set can contain.

The Content Manager pre-defined privilege sets include:

AllPrivSet; PrivSetCode: 1
: A user with this privilege set can perform all functions on all Content Manager entities. The privileges contained in this privilege set include: All system-defined and user-defined privileges.

NoPrivSet; PrivSetCode: 2
: Users with this privilege set cannot perform any functions on any Content Manager entities. This privilege set does not include any privileges.

SystemAdminPrivSet; PrivSetCode: 3
: Users with this privilege set can perform all Content Manager system administration and data modeling functions.

ItemAdminPrivSet; PrivSetCode: 4
: Users with this privilege set can perform all Content Manager data modeling and item access functions.

Table 37. Privilege Codes

Privilege name	Code
ICMLogon	1
SystemAdmin	40
SystemDefineItemType	45
ItemSQLSelect	121
ItemTypeQuery	122
ItemQuery	123
ItemAdd	124

Table 37. Privilege Codes (continued)

Privilege name	Code
ItemSetUserAttr	125
ItemSetSysAttr	126

Users and user groups

Most likely, you have a group of users that require the same type of access to the system. For example, all of the underwriters in an insurance company require search, retrieve, and update privileges to the claims item type. You can group the underwriters and any other users with common access needs into a user group. You cannot, however, put one user group into another user group.

A user group is solely a convenience grouping of individual users who perform similar tasks. A user group consists of zero or more users. You do not assign a user group a privilege set. Each user in a user group has a privilege set. A user group makes it easier to create access control lists for objects in your system. A user group cannot belong to other groups.

Access control lists

When a user creates an item in the Content Manager system, that user must define the access that other users will have to that item, and what operations they can perform on that item. The list of users that have access to the item and the operations that they can perform on the item is called an *access control list* (ACL). An ACL can contain one or more individual user IDs or user groups and their associated privileges. You can associate items, item types, and worklists with an ACL. Privilege sets define an individual's maximum ability to use the system, an ACL restricts that individual's access to an item. For example, if its ACL allows the photograph item to be deleted but John doesn't have the delete privilege in his privilege set, then John cannot delete the photograph.

A controlled entity is bound to a specific ACL through the ACL code. When associated with controlled entities, ACLs define the authorization of the bound entities and do not circumvent the user privileges. An ACL is enforced and user privileges are checked.

The users specified in access control rules can be individual users, user groups, or public. The interpretation is determined by the UserKind field of a rule. The types of rules, for illustration purposes, can be given the names ACL Rule for User, ACL Rule for Group, and ACL Rule for Public, respectively. By specifying public, the ACL Rule for Public authorizes all the users to perform operations specified in the ACL Privileges on the bound entity, provided the users pass their User Privileges check. The ACL privileges on the bound entity to Public can be configured in System level. The capability of opening a bound entity to public can be configured system-widely. The configuration parameter is named PubAccessEnabled (defined in table ICMSTSysControl). When disabled, all the ACL Rules for Public are ignored during the access control process.

Within the same ACL, a user can be specified in more than one type of rule. The precedence of the three types, from highest to lowest, is ACL Rule for Public, ACL Rule for User, and ACL Rule for Group. When applying ACL checks, if any higher-precedence rule type passes, the authorization is resolved and the process stops. If the check for ACL Rule for Public failed, the checking process will continue on the lower-precedence rule types.

If the check for ACL Rule for User failed, however, the checking stops; i.e., the ACL Rule for Group is not checked (there is no need to continue the check on the Group type because if a user does an individual user check (which means the user has a rule in the Access List table), the user will be excluded from the group type access based on the access control algorithm. The access control check for individual User type and Group type is not a sequential process. It is an either-or situation, even though there is no harm in doing a sequential check).

If the user has failed to pass an individual user type check (or the user does not have a rule in the Access List table), the checking process will continue to the group type. If the user belongs to one of the groups and the check of the privilege passes, the authorization is resolved and the process stops. Otherwise, access is denied and the process also stops. When a user is specified in more than one ACL Rule for a Group, the user is authorized by the union of all those rules' ACL Privileges. A user is never specified in more than one ACL Rule for User.

The CM system provides the following pre-configured ACLs: SuperUserACL, NoAccessACL and PublicReadACL.

SuperUserACL
: This ACL consists of a single rule that authorizes the CM pre-configured user ICMADMIN to perform all CM functions (AllPrivSet) on the bound entities.

NoAccessACL
: This ACL consists of a single rule that specifies, for all CM users (public), no actions (NoPrivSet) is allowed.

PublicReadACL
: This ACL consists of a single rule that specifies, for all CM users (ICMPUBLC), the read capability (ItemReadPrivSet) is allowed. This is the default value assigned to a user's DfltACLCode.

The data model

This section describes two possible implementations of the data model. The data model that you implement is dictated by the needs of your business. At this point you should have already analyzed your business requirements and determined which model best suits your needs. For more information about planning your data model see *Planning Your Content Manager System*.

One implementation of the data model is an item type with attributes, represented by a single component in the system. For example, an insurance company might implement the basic data model by defining the item type policy that contains the attributes policynumber, insuredfirstname, insuredlastname, and so forth.

Another implementation of the data model is an item type with a hierarchy of child components. For example, XYZ Insurance has implemented their business model by defining a policy item type that contains the attributes policynumber, insuredfirstname, insuredlastname, and claim. The policy item type has several claims associated with it and the claims have several reports associated with them. This implementation is represented as a hierarchy of components in the Content Manager system.

To see how to create a complete data model, see the insurance data model samples in the `samples` directory.

Planning a Content Manager application

This section helps you identify requirements for creating a Content Manager application and provides information about how Content Manager operates. The following topics are covered in this section:

- Determining the features of your application
- Handling errors

Determining the features of your application

The approach you take to develop your application varies based on the needs of your organization. To produce an effective application, all interested parties in your organization should contribute to the planning and design of the application. For additional help with planning, see *Planning Your Content Manager System*.

Before you can create your application, you should be able to answer all or most of the following questions:

- What types of documents does your organization use?
- What type of content is in your existing documents?
- How do you process documents?
- Can you automate your document process?
- How do you receive, display, store, and distribute documents?
- How often do you retrieve documents after they are stored?
- What is the volume of documents that your organization manages?
- What types of storage media do you want to use to store your large objects?
- Are there other applications your organization uses?

Use the answers to the questions above to help you determine which features to include in your application.

Handling errors

The most important exception to catch is the DKException class. Do not use exceptions for program logic, and do not rely on catching exceptions to detect if something exists in the datastore or for any reason other than for truly exceptional cases. Using exceptions in program logic decreases performance and can render tracing and log information useless for debugging and support.

Carefully review all of the exception information. There are numerous sub-classes of DKException and depending on the program, it may be best to handle each or some exceptions individually. Table 38 contains DKException information.

Table 38. DKException information

DKException	Description
Name	Exception Class Name. Contains sub-class name.
Message	A specific message explains the error. The message can contain a lot of information, sometimes encapsulating important variable states at the time the error was detected.
Message ID	A unique Message ID identifies this error type and matches it to a core message used above.

Table 38. DKException information (continued)

Error State	Might contain additional error information about the state of the OO API or library server error. If the library server detects an error, the following four pieces of information are packaged here: Return code Reason code Ext / SQL return code Ext / SQL reason code
Error Code	Might contain the library server return code.
Stack Trace	Important information indicating the failure point in the user program and exactly where the error was last detected or handled by the OOAPI.

The SConnectDisconnectICM.cpp sample in the `samples` directory demonstrates how to catch and print errors.

For information about logging and tracing, see *Messages and Codes*.

Working with the Content Manager samples

Content Manager provides a comprehensive set of code samples to help you complete key Content Manager tasks. The samples are located in the `cmbrooot\samples\cpp\icm` directory. Also included in the samples directory is the samples README, which contains information to help you make the best use of the samples.

Every sample name has the prefix S, which stands for samples. For example, SItemUpdateICM. Every sample is thoroughly documented and provides in-depth conceptual information and an explanation of each task step. Additional information contained in each sample includes:

- Detailed header information explaining the concepts shown in the sample
- A description of the sample file including prerequisite information and command line usage
- Fully commented code that you can easily cut, customize, and use in your applications

The Getting Started section in the samples README helps you to quickly learn how to complete the following general tasks:

- Data modeling
- Connecting to a server and handling errors
- Defining attributes and attribute groups
- Working with reference attributes
- Defining your data model
- Working with items
- Working with resource items
- Working with folders
- Working with links
- Defining the resource manager
- Defining an SMS collection
- Searching for items

The insurance scenario sample

Content Manager provides a code sample of one possible implementation in the real-world using an insurance company. The information used to create the insurance company sample is fabricated and created only to help explain key Content Manager features. The insurance scenario samples are also located in the samples directory, `cmbroot\samples\cpp\icm`. For a complete list of the samples that make up the insurance scenario, see the samples README (*README_SAMPLES_CPP_ICM.txt*).

Tools and wrappers

Content Manager also provides various tools and wrappers that you can use to create your applications. The tools are located in the samples directory, `cmbroot\samples\cpp\icm`. The file name for every tool has the prefix T. For example, TXMLAttrDefImportICM. Do not use the tools as samples. For information about how to use each tool and wrapper, read the header information in the tool or wrapper's header file.

Creating a Content Manager application

The APIs that implement Content Manager Version 8 Release 1 functionality are grouped into what is called the ICM connector. The ICM connector APIs have an ICM suffix, as in the example `DKDatastoreICM`.

This information in this section includes:
- Understanding the software components
- Representing items using DDOs
- Connecting to the Content Manager system
- Working with items

Understanding the software components

For conceptual purposes, you can categorize the OO APIs into the following groups of services:

- Data and document modeling
- Search and retrieve
- Data import and delivery
- System management
- Document routing

The data and document modeling module contains the APIs that enable you to map your business data model to the underlying Content Manager hierarchical data model. For example, an insurance company's data model includes *policies*, which in the Content Manager data model are essentially items. The data and document modeling module APIs provide interfaces to define items that represent *policies*.

The search and retrieve module processes requests about managed items like documents and folders. The search module APIs enable you to perform combined text and parametric searches for items contained in the Content Manager system. The search results are returned to the application in the form of search result sets.

The data import and delivery module provides the APIs that enable you to import data into your system and deliver that data through various media, like a network or the Web.

The system management module provides you with the interfaces to configure and maintain an efficient, secure Content Manager system. For example, you can incorporate the system management APIs into your application to allow you to adjust the system control settings, manage users, assign users privileges, allow access to the system, and so forth.

The document routing module APIs help you to route business objects, like documents, through a process, as defined by the needs of your business.

Representing items using DDOs

Before you can create an application, you must understand the DDO/XDO protocol concepts explained in the "Understanding dynamic data object concepts" on page 12. The information in this section is specific to Content Manager Version 8 Release 1.

A DDO is essentially a container of attributes. An attribute has a name, value, and several properties. One of the most important properties in the DDO is the attribute type. A DDO has a persistent ID (PID) to indicate the location where the object resides in persistent storage. A DDO has some methods to populate itself, and corresponding methods to retrieve an item's information. The DDO methods include add, retrieve, update and delete. You use these methods to move an item's data in and out of persistent storage, like Content Manager.

In memory, Content Manager items are represented as DDOs. Item attributes are represented as DDO attributes with a name, type, and a value. Links and references are represented as special types of attributes. The difference between a link attribute and a reference attribute, however, is that a reference attribute refers to another (single) DDO or XDO, and a link attribute refers to a collection (multiple) of DDOs or XDOs. XDOs are used to represent large objects (LOBs).

A reference to an item, either to an XDO or another DDO, has a name with the type property set to object reference, and value set to refer to the instance of the referenced object. Child components and links are also represented as DDO attributes with the type property set to a collection of data objects, and value set to a collection of DDOs. In the case of a child component, the attribute name is the name of the child component. The value is the collection of child components belonging to the root component. If the root item is deleted, all of the child components of the root item are also deleted.

Connecting to the Content Manager system

The first step in using the IBM Content Management (ICM) connector is to establish a connection with the associated datastore. To establish a connection to an ICM datastore, use the DKDatastoreICM.connect() method. To connect to the datastore you need four pieces of information: Database Name, User Name, and Password. You can also use a fourth parameter, connection options, but it is often not needed to connect to the database.

To access the Content Manager server, your application needs to create a datastore object, which acts as a common server. To create a datastore:

1. Create a datastore object.

   ```
   DKDatastoreICM * dsICM = new DKDatastoreICM();
   ```

2. Set up the connection parameters.

```
char * database = "icmnlsdb";
char * userName = "icmadmin";
char * password = "password";
```

3. Call the connect operation.

```
dsICM->connect(database, userName, password, "");
```

Depending on your system configuration, you might have several library servers and resource managers that you can connect to. To see a list of the names of the library servers that you can connect to, use DKDatastoreICM and call the listDataSourceNames() method, and then the listDataSources() method. The listDataSources() method lists the library servers that are currently available to connect to.

After you connect to a library server, use the DKRMConfiguration and call the listResourceMgrs()method, to get the list of resource managers associated with that library server.

Disconnecting from the Content Manager system

To disconnect from the datastore, use the disconnect() and destroy() methods.

```
dsICM->disconnect();
delete(dsICM);
```

Changing a password

You can allow users to change their password each time they begin a new library server session. To implement the change password option, use dkDatastore and call the changePassword() method.

Working with items

This section describes the processes involved in creating, updating, and deleting items.

For additional information about working with items see the SItemCreationICM sample located in the samples directory, cmbroot\samples\cpp\icm.

Creating an item type

Before you can create any items, you must create item types. You must define an item type for every item you create. For example, a claim item is of the item type policy owner. Item types consist of attributes, a default ACL, a version policy, and so forth. Data items are instances of item types.

When you create an item type, you can define a classification for the item type. An *item type classification* is a categorization within an item type that further identifies the items of that item type. All items of the same item type have the same item type classification. Content Manager supplies the following item type classifications: ICM document model, LOB, text, image, stream, and video stream. You can also define your own item type classifications. If you do not specify an item type classification when you create an item type, the item type classification defaults to item (DDO). The pre-defined item type classifications and the corresponding ID constant are listed in Table 39.

Table 39. Item type classifications

Classification	ID Constant	Number	Description
Item	DK_ICM_ITEMTYPE_CLASS_ITEM	0	A standard item (DDO).

Table 39. Item type classifications (continued)

Classification	ID Constant	Number	Description
Resource	DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM	1	A resource item that describes and contains data that is stored in the resource manager. Video, images, documents, and other data archived in the resource manager are examples of resource items.
Document model	DK_ICM_ITEMTYPE_CLASS_DOC_MODEL	2	An item that models documents using parts. A document is composed of any number of parts, which are contained in the attribute DK_CM_DKPARTS.
Part	DK_ICM_ITEMTYPE_CLASS_DOC_PART	3	Items (parts) in the document model classification.

Note: Constants are located in DKConstantICM

Example:

```
DKDatastoreICM* pDs;
DKDatastoreICM* pDs;
...
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*)pDs->datastoreDef();
//create new ItemType
DKItemTypeDefICM * bookItemType = new DKItemTypeDefICM(pDs);
bookItemType->setName("book");
bookItemType->setDescription("This is an example item type name.");
//Create new Attribute; add it to datastore and to the ItemType
DKAttrDefICM * attr = (DKAttrDefICM *)dsDefICM->createAttr();
  attr->setName("book_title");
  attr->setType(DK_CM_VARCHAR);
  attr->setSize(80);
  attr->setTextSearchable(TRUE);
  attr->setUnique(TRUE);
  attr->setNullable(FALSE);
//Persist the attribute to the datastore
attr->add();
//Add the newly created attribute to the item type.
bookItemType->addAttr(attr);
//Create new Attribute; add it to datastore and to ItemType
  attr = (DKAttrDefICM *)dsDefICM->createAttr();
  attr->setName("book_num_pages");
  attr->setType(DK_CM_INTEGER);
  attr->setTextSearchable(FALSE);
  attr->setUnique(FALSE);
  attr->setNullable(FALSE);
  attr->add();
bookItemType->addAttr(attr);
//Add the entity, bookItemType, to the datastore.
bookItemType->add();
```

Listing item types

The two following examples demonstrate two ways to get a list of item types.

Example 1:

```
long larraySize = 0;
    DKString * itemTypeName = dsICM->listEntityNames(larraySize);
    for (int i = 0; i < larraySize; i++) {
           cout <<(char*)itemTypeName[i] <<endl;
    }
    delete [] itemTypeName;
```

Example 2:

```
DKSequentialCollection * itemTypeColl = (DKSequentialCollection *)
   dsICM->listEntities();
    dkIterator * iter = itemTypeColl->createIterator();
    while (iter->more()) {
        DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)((void*)(*iter->next()));
        cout <<(char*)itemType->getName() <<endl;
        delete(itemType);
    }
    delete(iter);
    delete(itemTypeColl);
```

See the samples in the cmbroot\samples\cpp\icm directory for more information.

Creating attributes

To create attributes:

1. Create an attribute definition object.
2. Describe the object that you create by setting its name, description, type, size, and so forth. Examples of types of attributes that you can create include:
   - DK_CM_TIMESTAMP
   - DK_CM_DATE
   - DK_CM_CHAR
   - DK_CM_VARCHAR
   - DK_CM_SHORT
   - DK_CM_INTEGER
   - DK_CM_LONG
   - DK_CM_FLOAT
   - DK_CM_DECIMAL
   - DK_CM_TIME
   - DK_CM_BLOB
   - DK_CM_CLOB
3. Add the new definition to the persistent datastore.

Examples:

1. This example defines an attribute for the title of a book.

   ```
   DKDatastoreICM * dsICM;
   DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
          attr->setName("book_title");
          attr->setDescription("The title of the book.");
          attr->setType(DK_CM_VARCHAR);
          attr->setSize((long) 100);
          attr->add();
   ```

2. This example defines an attribute for the number of pages in a book.

   ```
   DKAttrDefICM * attr = new DKAttrDefICM(dsICM);
          attr->setName("book_num_pages");
          attr->setDescription("The number of pages in the book.");
   ```

```
attr->setType(DK_CM_INTEGER);
attr->setMin((long) 0);
attr->setMax((long) 100000);
attr->add();
```

Listing the attributes in a datastore

The following example demonstrates how to get a list of attributes in a datastore.

```
DKDatastoreICM * dsICM;
DKDatastoreDefICM*dsDefICM =(DKDatastoreDefICM *)dsICM->datastoreDef();
//Get a collection containingall Attribute Definitions.
DKSequentialCollection*attrDefColl =
        (DKSequentialCollection *)dsDefICM->listAttrs();
if (attrDefColl &&(attrDefColl->cardinality()>0)){
   //Create an iterator to iterate through the collection
   dkIterator*iter =attrDefColl->createIterator();
   while(iter->more()){
     //while there are still items in the list,continue
    dkAttrDef*attrDef =(dkAttrDef *)iter->next()->value();
    cout <<"-"<<attr->getName()<<":"<<attr->getDescription()<<endl;
    delete(attr);
  }
  delete(iter);
  delete(attrDefColl);
}
delete(dsDefICM);
```

Listing attribute names for an item type

The following example demonstrates how to get a list of attribute names for an item type.

```
//Get a collection containingall Attribute Definitions for the Item Type.
DKSequentialCollection*attrColl =(DKSequentialCollection*)
   dsICM->listEntityAttrs(itemTypeName);
//Accessing attribute each and printing the name and description.
cout <<"\nAttributes of Item Type '"<<itemTypeName <<"':
   ("<<attrColl->cardinality()<<')'<<
//Create an iterator to iterate through the collection
dkIterator* iter =attrColl->createIterator();
while(iter->more()) {
   //while there are still items in the list, continue
DKAttrDefICM* attr =(DKAttrDefICM*)iter->next()->value();
cout <<"-"<<attr->getName()<<":"<<attr->getDescription()<<endl;
delete(attr);
}
delete(iter);
delete(attrColl);
```

Creating an item

An item is an instance of an item type. When you create an item, you must assign it a property type. The types that you can assign to an item are document, folder, or item. You must specify the property type as the second parameter of the datastore's createDDO function. The value for the property type is stored in the DDO's property named, DK_CM_PROPERTY_ITEM_TYPE. Do not confuse this property type with the overall item type definition that describes the structure of the item. Table 40 on page 408 shows a list of available item property types.

Table 40. Item property types

Property type	Constant	Definition
Document	DK_CM_DOCUMENT	Item represents a document or stored data. The information contained in this item might form a document. This item can be considered a common document since it does not rigidly mean an implementation of a specific document model.
Folder	DK_CM_FOLDER	Item represents an object containing or referencing contents or objects. This item can be considered a common folder since it does not rigidly mean an implementation of a specific document model. In the ICM Connector, an empty DKFolder collection, which can hold contents is automatically created.
Item (default)	DK_CM_ITEM	Generic item. This item does not fit system defined or user defined semantic types.

Items are created as DKDDOs. Always use the DKDatastoreICM's createDDO() methods to create DKDDOs because the system uses the DKDatastoreICM's createDDO methods to automatically setup important information in the DKDDO structure.

The following example illustrates how to create several different types of items.

```
//createDDO(<Overall Item Type>,<Item Property /Semantic Type>);
DKDDO*ddoFolder =dsICM->createDDO("book",DK_CM_FOLDER);
DKDDO*ddoItem =dsICM->createDDO("book",DK_CM_ITEM);
DKDDO*ddoDocument =dsICM->createDDO("book",DK_CM_DOCUMENT);
```

When you create an item, you define the components that make up the item. For example, if you choose to create a hierarchical item, you must create child components.

1. Using the datastore's `createDDO` and `createChildDDO` methods, create an item DDO and set all of its attributes and other required information. This example uses the logged on, connected `DKDatastoreICM` object named `dsICM`.
2. Create a root component.
3. Create a child component.

   ```
   DKDDO myDDO = dsICM->createChildDDO(parentRootName,itemOrComponentTypeNameStr);
   ```

The example below shows how to link a child component to its parent:

```
short dataIdChildCollection = myDDO->dataId
  (DK_CM_NAMESPACE_CHILD,"myChildComponent");
DKAny anyChild =myDDO->getData(dataIdChildCollection);
DKChildCollection* myDDOChildCollection =
  (DKChildCollection*)((dkCollection*)(anyChild));
myDDOChildCollection->addElement((dkDataObjectBase*)myDDOChild);
```

When you define an item, you must also define its semantic type. *Semantic type* defines the usage or rules for an item. Content Manager comes with some pre-defined semantic types, but you can also define your own semantic types.

You can specify the semantic type as the second parameter of the datastore's createDDO function. The semantic type value is stored in the DDO's property DK_ICM_PROPERTY_SEMANTIC_TYPE, which can contain the same value as the

item property type. The ICM connector supports folder and document semantic types. You can also create your own semantic types. Table 41 lists the available semantic types.

Table 41. Pre-defined semantic types

Semantic type	Constant	Definition
Document	DK_ICM_SEMANTIC_TYPE _Document	Indicates that this item is a document
Folder	DK_ICM_SEMANTIC_TYPE _Folder	Indicates that this item is a folder
Annotation	DK_ICM_SEMANTIC_TYPE _Annotation	Indicates that this item or part is an annotation to the base part
Container	DK_ICM_SEMANTIC_TYPE _Container	Indicates that this item is a container, which can contains other items. The container-containee relationship is represented using links.
History	DK_ICM_SEMANTIC_TYPE _History	Indicates that this item or part is a history of the base part.
Note	DK_ICM_SEMANTIC_TYPE _Note	Indicates that this item or part is a note to the base part.
Base	DK_ICM_SEMANTIC_TYPE _Base	Indicates that this item or part is the base part that may have an annotation, note, or history associated with it.
User defined	User defined	A user defined semantic type interpreted by the application.

Note:

- In Content Manager Version 7, semantic type is called affiliated type.

If you create an item in an item type that has been defined as a resource item type, the correct sub-class XDO is returned. For more information about resources, please see the SItemTypeCreationICM.cpp sample in the samples directory.

Setting and retrieving item attribute values

When setting values for individual attributes, use the individual attribute definition name. In order to access attributes that belong to an attribute group, use this format: <Attribute Group Name>.<Attribute Name>.

```
//The code snippet below shows the value of the character attribute
// "book_title" for the item represented
//by the DDO ddoDocument is set to the value "Effective C++"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
                    DKString("book_title")),DKString("Effective C++"));
//The code snippet below shows the value of the integer attribute
//"book_num_pages" for the item represented
//by the DDO ddoDocument is set to the value "250"
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
                      DKString("book_num_pages")),(long)250);
//This code snippet shows how the value of the "book_title" attribute of the
//item represented by the DDO ddoDocument is retrieved into the "title"
//string variable.
DKString title =(DKString) ddoDocument->getData(ddoDocument->
   dataId(DK_CM_NAMESPACE_ATTR,DKString("book_title")));
```

Modifying an item's attributes

To modify an item's attributes, use the DDO's setData operation and set it to the required value by specifying that value in the expected data type (e.g. for character attributes pass in the data as a DKString).

```
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,"book_title"),
  DKString("More Effective C++"));
```

Updating an item

To update an item:

1. Retrieve an item or create an item and add it to the datastore.
2. Modify the item, its attributes, link collections, and so forth.
3. Call the DDO's update operation

   ```
   ddo->update();
   ```

Example:

```
//To create a new version of an item when updating it
    DKDDO * ddo = dsICM->createDDO("book",DK_CM_DOCUMENT);
    ddo->update(DK_CM_VERSION_NEW);

//To update the latest version of an item, use the following code:
ddo->update(DK_CM_VERSION_LATEST);

//To update with the appropriate options
    DKDatastoreICM * ds = new DKDatastoreICM();
    ds->connect(...);
    DKDDO * ddo = dsICM->createDDO(...);
    long options = DK_CM_VERSION_NEW + DK_CM_CHECKIN;
    ds->updateObject(ddo, options);
```

If an item is enabled for versioning, you can create a new version of the item instead of updating the current item, as shown in the following example:

```
ddo->update(DK_CM_VERSION_NEW);
```

To update the latest version of an item, use the format shown in the following example:

```
ddo->update(DK_CM_VERSION_LATEST);
```

You can also update a DDO by calling the updateObject method on DKDatastoreICM with the appropriate options, as shown in the example below:

```
int options = DK_CM_VERSION_NEW + DK_CM_CHECKIN;
ds->updateObject(ddo, options);
```

Retrieving an item

Below is an example that demonstrates how to retrieve the most current version of an item that is represented by a DDO.

```
ddo->retrieve(DK_CM_VERSION_LATEST);
```

Defining a resource item type

A resource item is an item with additional system defined attributes that define the location, type, size, and so forth, of the object that the item represents. The object is sometimes called a ″resource″ and can be a video file, an image, a word processor document, and so forth. For additional information, see the SItemTypeCreationICM sample (in the samples directory).

The steps below take you through the process of defining a resource item type.

1. Create new datastore object, and connect to the datastore.

   ```
   DKDatastoreICM* dsICM = new DKDatastoreICM();
   dsICM->connect("ICMNLSDB","icmadmin","password","");
   ```

2. Get the datastore definition object from the connected datastore.

   ```
   DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
   ```

3. Create a new item type definition.

```
DKItemTypeDefICM * itemType = new DKItemTypeDefICM(dsICM);
itemType->setName("SampleResource");
itemType->setDescription("Simple Resource Lob Item Type");
```

4. Add an attribute.

```
DKAttrDefICM * attr = (DKAttrDefICM*) dsDefICM->retrieveAttr("S_varchar");
itemType->addAttr(attr);
// Resource classification indicates that this class will contain
//data file
itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM);
```

5. Specify XDO Class and type of resource for this item type.

```
itemType->setXDOClassName(DK_ICM_XDO_LOB_CLASS_NAME);
itemType->setXDOClassID(DK_ICM_XDO_LOB_CLASS_ID);
itemType->add();
```

Creating a resource item

Creating a resource items is very similar to creating regular items. XDOs extend DDOs, anddepending on the type of resource item, the XDO can be extended further. Table 42 contains the resource item types and the class hierarchy used to create them. For more information, see the SResourceItemCreationICM sample (in the samples directory).

Table 42. Resource item type class hierarchy

Type	DDO	XDO	Extension
LOB	DKDDO ->	DKLobICM	
Text	DKDDO ->	DKLobICM ->	DKTextICM
Image	DKDDO ->	DKLobICM ->	DKImageICM
Stream	DKDDO ->	DKLobICM ->	DKStreamICM

The following steps take you through the process of creating a resource item.

1. Create the resource item. Note that it can be any semantic type, and that the DKDatastoreICM::createDDO call is also used to create a resource item in the same way that it is used to create a regular item. The type of resource returned from DKDatastoreICM.createDDO is cast to the correct sub-class (in this case DKLobICM), based on the XDO classification you used to define the item type on which this resource item is based.

```
DKLobICM*     lob = (DKLobICM*)
   dsICM->createDDO("SampleResource",DK_CM_DOCUMENT);
```

2. Set the MIME type for the resource. In this case, the resource is a MS Word document. The MIME type describes the type of content that is being stored.

```
lob->setMimeType("application/msword");
```

3. Add the data to the data store. In this case, a Word document. Note that the resource item content will be stored in a resource manager. Even though, it is possible to specify the resource manager for storing the item, the example below uses the default resource manager and SMS Collection for the item type on which this resource is based.

```
lob->add("SResourceItemICM_Document1.doc");
```

4. Cleanup the memory.

```
delete(lob);
```

Creating, updating, and deleting attribute groups

Attribute groups make it easier for you to add entire groups of attributes to items and sub-components. An attribute can belong to any number of attribute groups,

from zero to any number. An attribute can be added to multiple attribute groups. A data item (DDO) can contain multiple attribute groups. The same attribute name can appear in the DDO, but each attribute is completely separate, based on the namespace. When an attribute is added to an attribute group, it impacts only the component types that are created after the addition of the attribute to the attribute group. Pre-existing component types remain unchanged.

The following example demonstrates how to create an attribute group:

```
// Create a datastore definition object given the connected datasto
DKDatastoreDefICM* dsDefICM = (DKDatastoreDefICM*) dsICM->datastoreDef();
//Creating a new attribute group
DKAttrGroupDefICM* attrGroup = new DKAttrGroupDefICM(dsICM);
// Set a name, maximum 15 characters
 attrGroup->setName("Book_Details");
//Set a description for the new attribute group
attrGroup->setDescription("Detailed book information");
// Retrieve the definition of an attribute that will be
//added to this attribute group.
DKAttrDefICM* title = (DKAttrDefICM *) dsDefICM->retrieveAttr("book_title");
// Retrieve the definition of another attribute that will be added
//to this attribute group.
DKAttrDefICM* publisher = (DKAttrDefICM *) dsDefICM->retrieveAttr("publisher");
// Add the Attribute Definitions to the Attribute Group
attrGroup->addAttr(title);
attrGroup->addAttr(publisher);
// add it to the persistent datastore
attrGroup->add();

delete(attrGroup);
```

To update the name and description of an attribute group, call the update() method in DKAttrGroupDefICM and provide an array of new attribute IDs.

To delete an attribute group you also work with the DKAttrGroupDefICM class. When you delete the attribute group the primary attributes that used to make up the attribute group remain in library server. The following exceptions apply when you delete an attribute group:

- An attribute group cannot be deleted if it is associated with a component type and is persistent.
- An attribute can not be removed from an attribute group if the attribute group is associated with a component type and is persistent.
- You cannot add an attribute to an attribute group if the attribute group is associated with a component type and is persistent.

Searching for items

To find items that match a set of criterion, complete the following steps:

1. Connect to the datastore.
2. Build a query string
3. Process the correct query. Make sure that your query conforms to the query language. For more information see “Understanding the query language” on page 423.
4. Specify options as required
5. Save the query results in a DKResults object.

```
DKNVPair *options    = new DKNVPair[3];
// only one result will be returned
options[0].set(DK_CM_PARM_MAX_RESULTS, "1");// Retrieve only the attributes
options[1].set(DK_CM_PARM_RETRIEVE , (long)DK_CM_CONTENT_ATTRONLY);
// Last option has to be this value
```

```
options[2].set(DK_CM_PARM_END , NULL);
//Note if a query is expected to return more than one result,
//the DKDatastoreICM::execute method
//should be used. The execute method returns a dkResultSetCursor.
DKResults* results = (DKResults*)(dkCollection*)dsICM->evaluate
  (query, DK_CM_XQPE_QL_TYPE, options);
```

See the SSearchICM sample, in the cmbroot\samples\cpp\icm directory, for additional information and examples.

Retrieving items

To retrieve a document from a DKDatastoreICM that represents a Content Manager content server, you must know the document's item type name and item ID. You must also associate the DDO with a datastore and establish a connection.

1. After connecting to a datastoreICM object, search for the item using the appropriate item ID. For information about writing queries, see "Querying the Content Manager server" on page 424.
2. Save the query results in a DKResult object. In the code example below, queryStr is the search string in the correct query language format.

   ```
   DKResults* results = (DKResults*)(dkCollection*)
      dsICM->evaluate(queryStr, DK_CM_XQPE_QL_TYPE,NULL);
   ```
3. Create a DDO using the DKDatastoreICM's createDDO method. Do not use the DKDDO constructor. In the example below, the object dsICM is already connected to a Content Manager datastore. Also, the PID is a string named pidStr.

   ```
   DKDDO* ddo = dsICM->createDDO(pidStr);
   ```
4. Call the DDO's retrieve operation.

   ```
   ddo->retrieve();
   ```

If an item is enabled for versioning, you can retrieve a specific version of the item by using the retrieve method of the DKDDO class with the appropriate options. To retrieve the latest version of an item, you can either set the requested version number to 0 or use the constant DK_CM_VERSION_LATEST. By default (if no options are specified), the DKDDO::retrieve method retrieves the latest version of an item.

Example:

```
ddo->retrieve(DK_CM_VERSION_LATEST);
```

You can also retrieve a DDO by calling the retrieveObject method on the DKDatastoreICM object.

Example:

```
DKDatastoreICM * dsICM;
....
DKDDO*ddo =dsICM->createDDO("book",DK_CM_DOCUMENT);
```

To retrieve the latest version of an item including its attributes and children, use the following format:

```
DKDDO * ddo;
long options =DK_CM_CONTENT_ATTRONLY +DK_CM_CONTENT_CHILDREN +DK_CM_VERSION_LATEST;
dsICM->retrieveObject(ddo,options);
```

Deleting items

Use the delete method in the DDO to delete an item from the content server. Delete rules are set at design-time or data model definition and apply to references

and child component types (multivalue) attribute definitions. The pre-defined delete rules can prevent deletion of an item with children or references.

```
ddoDocument->del();
```

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

Checking in and checking out items

To prevent two users from making changes to the same item at the same time, you can require users to check out an item before updating it. Checking out an item grants exclusive update rights to the user who has the item checked out.

In order to update an item, it must be checked out. You can specify an option that checks out the item when it is retrieved. If an item is not checked out when it its retrieved, it must be explicitly checked out using the DKDatastoreICM.checkOut(ddo) method.

Checkout example:

```
//1.Retrieve the latest version of an item and check it out at
//the same time

ddoDocument->retrieve(DK_CM_CHECKOUT +DK_CM_VERSION_LATEST);
//Explicity check out an item
dsICM->checkOut(ddoDocument);
if (dsICM->isCheckedOut(ddoDocument)) cout <<"checked out";
```

Checkin example: A item that has been previously checked out must be checked in before it is updated in the datastore. The ″check in″ option can be specified implicitly in the update call or explicitly by calling the DKDatastoreICM.checkIn(ddo) method.

```
//Update an item and check it in at the same time
ddoDocument->update(DK_CM_CHECKIN);
//Explicitly check in an item
dsICM->checkIn(ddo);
if (!dsICM->isCheckedOut(ddo)) cout <<"checked in";
```

Setting and getting an item's versioning properties

The example below, demonstrates how to set an item's versioning properties and how to retrieve and item's versioning properties.

```
DKPidICM * pid = (DKPidICM *)ddo->getPidObject();
//Retrieving versioning information for an item (DDO) as a string
DKString version = pid->getVersionNumber();
....
//Setting the version number for the PID associated with an item (DDO).
pid->setVersionNumber((char *)version);
```

Working with versioning properties

This section provides examples to help you work with versioning properties.

Retrieving version control policy of an item type

An item has a version control policy, which contains the versioning property for the item. Following is the list of the three versioning properties available and the value used to represent each property in the version control policy.

- DK_ICM_VERSION_CONTROL_ALWAYS: Versioned-always
- DK_ICM_VERSION_CONTROL_NEVER: Versioning is not supported for this item type (default)

- DK_ICM_VERSION_CONTROL_BY_APPLICATION: The application determines the versioning scheme

```
short versionControlPolicy = 0;
DKItemTypeDefICM * item    = NULL;
...
versionControlPolicy = item->getVersionControl();
```

Setting version control for an item type

```
DKItemTypeDefICM * item = NULL;
short versionControl    = DK_ICM_VERSION_CONTROL_ALWAYS;
...
item->setVersionControl(versionControl);
```

Getting the maximum number of versions allowed for an item type

```
 short versionMax     = 0;
DKItemTypeDefICM * item = NULL:
....
versionMax = item->getVersionMax();
```

Setting the maximum number of versions allowed for an item type

In this example, only ten versions of an item that is based on this item type are maintained by the system.

```
short versionMax    = 10;
DKItemTypeDefICM * item = NULL;
 ....
item->setVersionMax(versionMax);
```

Setting the value of the versioning type for an item type

```
DKItemTypeDefICM * item = NULL;
short versionType = DK_ICM_ITEM_VERSIONING_OPTIMIZED;
...
item->setVersioningType(versioningType);
```

Working with folders

A folder is a fully supported DDO that represents an item. A folder has the full hierarchical data structure of the item type that it is created in. A folder is a DDO of semantic type folder and has an attribute, DK_CM_DKFOLDER, that contains a DKFolder, regardless of the item type's classification. The DKFolder object, a DKSequentialCollection can contain other DDOs.

The procedures below include code fragments for informational purposes only. Do not copy these code fragments and paste them directly into your application program, because they will not work in their current form. For a complete folders sample that you can run, see the SFolderICM.cpp sample in the `samples` directory.

Creating a folder

A folder is a DDO of semantic type ″folder″ that contains a DKFolder object. The DDO has an attribute called DK_CM_DKFOLDER that contains a DKFolder object, a DKSequentialCollection that can contain DDOs.

You can make an item a folder for any item type because there is no implicit functionality in an item type that prevents it from being used as a folder item.

Use the DKDatastoreICM.createDDO() method to create a DDO and specify the item property type or semantic type as DK_CM_FOLDER. See the example below:

```
// A folder item is created usingthe DKDatastoreICM.createDDO method
// with the DK_CM_FOLDER semantic type
DKDDO* ddoFolder = dsICM->createDDO("S_simple", DK_CM_FOLDER);
// call setData to populate this folder
```

```
ddoFolder->setData(....);
.....
// Persist the folder to the datastore
ddoFolder->add();
```

Adding contents to a folder

All contents placed in DKFolder must be persistent in the datastore before calling add() or update() on the folder DDO. This means that all folder contents must have already had the DKDDO.add() operation called.

The most efficient way to add items to a folder is to use the DKFolder's addElement() function. This allows any number of folder changes to be grouped into a single call to the library server.

Complete the following steps to add contents to a folder:

1. Create the items that will be added to the folder.

   ```
   DKDDO * ddoDocument = dsICM->createDDO("book", DK_CM_DOCUMENT);
   DKDDO * ddoFolder2  = dsICM->createDDO("book", DK_CM_FOLDER);
   DKDDO * ddoItem     = dsICM->createDDO("book", DK_CM_ITEM);
   ```

2. Persist the created items to the datastore.

   ```
   ddoDocument->add();
   ddoItem->add();
   ddoFolder2->add();
   ```

3. Retrieve the DKFolder attribute for the folder.

   ```
   DKFolder * dkFolder;
   short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKFOLDER);
   if (dataid!=0) dkFolder = (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
   ```

4. Check out the folder (A folder must be checked out before it is update)

   ```
   dsICM->checkOut(ddoFolder);
   ```

5. Add the created items to the folder.

   ```
   dkFolder->addElement(ddoDocument);
   dkFolder->addElement(ddoItem);
   dkFolder->addElement(ddoFolder2);  // Note, Folders can contain sub-folders.
   ```

6. Update the folder. This implictly also checks in the folder (unlocks it).

   ```
   ddoFolder->update();
   ```

7. Explicitly check in the folder.

   ```
   dsICM->checkIn(ddoFolder);
   ```

Removing a folder's contents

Complete the steps below to remove an item from a folder.

1. Create an item and add it to a folder. Note that the folder was created earlier.

   ```
    DKDDO * ddoItem   = dsICM->createDDO("book", DK_CM_ITEM);
   ddoItem->add();
   DKFolder * dkFolder;
   short dataid = ddoFolder->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKFOLDER);
   if (dataid!=0) dkFolder = (DKFolder*)(dkCollection*) ddoFolder->getData(dataid);
   dsICM->checkOut(ddoFolder);
   dkFolder->addElement(ddoItem);
   ddoFolder->update();
   ```

2. Explicitly check in the folder.

   ```
   dsICM->checkIn(ddoFolder);
   ```

3. Create an iterator for the folder.

   ```
   dkIterator* iter = dkFolder->createIterator();
   ```

4. Iterate through the contents of the folder until the item that is to be removed is found. Remove the item.

```
while(iter->more())
{
        DKDDO* ddo = (DKDDO*) iter->next()->value();
if ( ((DKPidICM*) ddo->getPidObject())->pidString() ==
     ((DKPidICM*) ddoItem->getPidObject())->pidString() )
       {
       //Found the element to  be removed. Remove it
      dkFolder->removeElementAt(*iter);
      // Now that we found the ddoItem, remove it.
       }
  }

 delete(iter);
```

Retrieving a folder's contents

To retrieve the contents from a folder, call the DKDDO::retrieve method on the DDO that corresponds to the folder. The options specified determine the data that is retrieved from the datastore. **Example:**

```
//To retrieve only the outbound links
ddoFolder->retrieve(DK_CM_CONTENT_LINKS_OUTBOUND);
// Retrieve the item Tree includes links & the folder contents.
ddoFolder->retrieve(DK_CM_CONTENT_ITEMTREE);
```

Obtaining all folders containing a specific DDO

Multiple folders can contain the same item (DDO). This allows you to store the same item in multiple, different places and associate the item with different applications or users. The example below demonstrates how to retrieve all the folders containing a specific item.

```
DKDDO* ddoDocument = ....;
//Create datastore object,connect to datastore,create DDO etc.
....
//Create a new datastore extension object from the connected datastore object
DKDatastoreExtICM* dsExtICM = (DKDatastoreExtICM*)
                              dsICM->getExtension(DK_CM_DATASTORE_EXT);
//Retrieve the PID for the created DDO
DKPidICM* pid = (DKPidICM*) ddoDocument->getPidObject();
//Retrieve a list of folders containing the DDO created earlier.
DKSequentialCollection *list = dsExtICM->getFoldersContainingDDO(ddoDocument);
//Create a iterator for the returned DDO collection
dkIterator* iter =list->createIterator();
while(iter->more()) {
DKDDO* ddo =(DKDDO*) iter->next()->value();
pid = (DKPidICM*) ddo->getPidObject();
cout << "-Item ID:" << pid->getItemId() << endl;
delete ddo;
}
delete iter;
```

Defining links between items

You can use links to associate a source item to a target item with an optional description item. You define links in DKLink objects, where you can specify the following types of link elements:

Table 43. Link data structure

DKLink	Definition
LinkTypeName	The type of link.
Source	The source item of a link.
Target	The target item of a link.
LinkItem	The description item. Optional.

Do not use the DKFolder link type name to define or delete links. You should also not use the DKFolder link type name in any way with a DKLinkCollection. For correct usage of folders, please refer to the SFolderICM.cpp sample in the `samples` directory.

A link itself does not belong to either the source or the target. A link just connects a source and target. For example, if source A is linked to target B, A is always the source and B is always the target, regardless of which DDO (A or B) is being referenced.

In memory, both the source and the target DDOs obtain copies of the same DKLink object reference. Since the DKLink object contains a reference to both the source and the target, a DDO containing a link also contains a link that refers to itself as well as to another DDO. For example, if Source A is linked to Target B, both A and B contain the same link, as shown in the example in Table 44.

Table 44. DKLink definition example

DKLink Defined for A to B	DDO A	DDO B
Source is **A**	Source is **A**	Source is **A**
Target is **B**	Target is **B**	Target is **B**

When you add or remove links, you only have to perform the operation on one of the two items, source or target. The link is automatically updated for the other item.

The procedures below include code fragments for informational purposes only. Do not copy these code fragments and paste them directly into your application program, because they will not work in their current form. For a complete links sample that you can run, see the SLinksICM.cpp sample in the `samples` directory.

Inbound and outbound links

When using links to reference a particular DDO, either the source or the target, you can consider the links to be *inbound* or *outbound*. If a DDO that is being referenced by other DDOs, links from itself **to** those DDOs, the links are considered to be outbound links. If a DDO being referenced by other DDOs links **from** those DDOs to itself, the links are considered to be inbound links. When a DDO is the source of the link, it is an outbound link. If the DDO is the target of a link, it is an inbound link. Inbound and outbound links allow a DDO to determine what DDOs link to it and break the links if necessary.

In the example in Table 44, the link from DDO A to DDO B is outbound link, while the same link to DDO B is the inbound link.

Link type names

Link relationships have names. Within a DDO, links are grouped into link collections. There are system defined link type names and you can also define your own. You can use any number of user-defined link type names or system-defined link type names. Table 45 lists the system defined link type names.

Table 45. System defined link type names

Link type name	Constant
Contains	DK_ICM_LINKTYPENAME_CONTAINS
DKFolder	DK_ICM_LINKTYPENAME_DKFOLDER

The DKFolder link type name is provided and used by the system to manage folders. The DKFolder object is a simplified interface to an outbound link collection to make folders easy to work with. However, you must specify the DKFolder link type name in order to search folders using links.

See the SLinksICM sample, in the `cmbroot/samples/cm/icm` directory, for additional information about links.

Retrieving linked items

When you retrieve links, you have the option of retrieving only inbound, only outbound, or both types. In order to retrieve inbound and outbound, you must set a retrieve option. You can set the following options to retrieve links:

- DK_CM_CONTENT_LINKS_OUTBOUND
- DK_CM_CONTENT_LINKS_INBOUND
- DK_CM_CONTENT_LINKS_OUTBOUND + DK_CM_CONTENT_LINKS_INBOUND
- DK_CM_CONTENT_ITEMTREE

Keep in mind that before you make a call to the DDO `add()` or `update()` methods, all the items that are associated with links must already be persistent.

Working with the access control APIs

If you have access to either the Content Manager system administration client or to the Content Manager APIs for writing your own administration program, you can use the access control functions to control access to the information in your Content Manager system. The various access control APIs allow you to control access to the entire system, to a closely related set of operations on the system, or to an individual item.

Some of the tasks that you can complete using the access control APIs include:

- Create privileges
- Associate a list of actions with information in the system
- Give permission to users to perform actions on the information in the library server

Creating a privilege

A privilege is the ability to use the Content Manager system. Content Manager Privileges are used to grant access to individual users and to define access control lists (ACL).

A privilege is represented by the class DKPrivilegeICM. Follow the steps below to create a new privilege object, make it persistent, and retrieve a privilege. Note that a connection to the datastore must exist before a privilege can be created.

1. Create the datastore object as well as all the related administration management objects.

```
//Create the datastore object
DKDatastoreICM * ds = new DKDatastoreICM();
//Connect to the underlying datastore
ds->connect("ICMNLSDB", "icmdmin", "password", "");
//Retrieve the datastore definition object (used to access
//and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process datastore
//administration functions.
```

```
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
    dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
                  dsAdmin->authorizationMgmt();
```

2. Create a new privilege object and set its properties

```
DKPrivilegeICM * priv = new DKPrivilegeICM(ds);
//Set the name of the privilege object
priv->setName("UserPriv");
//Set the privilege description
priv->setDescription("This is user-defined privilege");
```

3. Add the privilege to the datastore via the authorization management object.

```
aclMgmt->add(priv);
```

Creating a privilege set

A Content Manager privilege set is a named group of privileges. Privilege sets make system administration easier. You must group, into sets, Content Manager privileges before you can grant them to users or use them to define ACLs.

A privilege set is represented by the class DKPrivilegeSetICM. The following steps demonstrate how to create a privilege set and add a some privileges to it.

1. 1. Create the datastore object as well as all the related administration management objects.

```
//Create the datastore object
DKDatastoreICM * ds = new DKDatastoreICM();
//Connect to the underlying datastore
ds->connect("ICMNLSDB", "icmdmin", "password", "");
//Retrieve the datastore definition object
//(used to access and manipulate CM meta-data)
dkDatastoreDef * dsDef = (dkDatastoreDef *) ds->datastoreDef();
//Create the class used to represent and process datastore administration
// functions.
DKDatastoreAdminICM * dsAdmin = (DKDatastoreAdminICM *)
     dsDef->datastoreAdmin();
//Retrieve the class used to represent and manage the authorization
//related functionality of the ICM datastore
DKAuthorizationMgmtICM * aclMgmt = (DKAuthorizationMgmtICM *)
             dsAdmin->authorizationMgmt();
```

2. Create three privileges and set their properties.

```
DKPrivilegeICM * priv_1 = new DKPrivilegeICM(ds);
priv_1->setName("ItemCheckOut");
DKPrivilegeICM * priv_2 = new DKPrivilegeICM(ds);
priv_2->setName("ItemQuery");
DKPrivilegeICM * priv_3 = new DKPrivilegeICM(ds);
priv_3->setName("ItemAdd");
```

3. Create a new privilege set and set its properties.

```
DKPrivilegeSetICM * privSet1 = new DKPrivilegeSetICM(ds);
privSet1->setName("UserPrivSet");
privSet1->setDescription("This is a user-defined priv set");
```

4. Add the created privileges to the new privilege set.

```
privSet1->addPrivilege(priv_1);
privSet1->addPrivilege(priv_2);
privSet1->addPrivilege(priv_3);
```

5. Add the privilege set to the datastore using the authorization management object.

```
aclMgmt->add(privSet1);
```

Displaying privilege set properties

The following examples shows how to display privilege set properties.

```
//Retrieve a privilege set using its name
DKPrivilegeSetICM * sPrivSet = (DKPrivilegeSetICM *)
   aclMgmt->retrievePrivilegeSet("UserPrivSet");
//Display privilege set properties
cout<<"privilege set name = "<< (char *)sPrivSet->getName() << endl;
cout<<"privilege set description = "<< (char *)sPrivSet->getDescription() << endl;
//Retrieve the list of privileges that are a part of this privilege set
dkCollection * coll = sPrivSet->listPrivileges();
dkIterator * iter = coll->createIterator();
while (iter->more())
{
         DKPrivilegeICM* _priv = (DKPrivilegeICM *)(void *)(*iter->next());
         cout<<"  privilege name = "<< (char *)_priv->getName() << endl;
}
delete(iter);
```

Defining an ACL

The Content Manager access control model is applied at the level of the controlled entity. A controlled entity is a unit of protected user data. A controlled entity can be an individual item, item-type, or the entire library. Operations on controlled entities are regulated by one or more control rules. The access control list (ACL) is the container for these control rules. The DKAccessControlListICM class represents an ACL. Every controlled entity in a Content Manager system must be bound to an ACL.

Complete the following steps to define an ACL.

1. Define new DKACLData objects. Each DKACLData object is used to hold ACL related data.

   ```
   DKACLData * aclData1 = new DKACLData();
   // set the name of the user group to be associated with this ACL
   aclData1->setUserGroupName("ICMADMIN");
   // Set  the ACL patron type. Can be one of 3 possible
   //values:DK_CM_USER_KIND_USER or DK_CM_USER_KIND_GROUP or
   //DK_CM_USER_KIND_PUBLIC.
   aclData1->setPatronType(DK_CM_USER_KIND_USER);
   // Set the privilege set associated with the ACL.
   DKPrivilegeSetICM * privSet_1 = (DKPrivilegeSetICM *)
      aclMgmt->retrievePrivilegeSet("UserPrivSet");
   aclData1->setPrivilegeSet(privSet_1);
   ```

2. Create another DKACLdata object.

   ```
   DKACLData * aclData2 = new DKACLData();
   aclData2->setUserGroupName("ICMPUBLIC");
   aclData2->setPatronType(DK_CM_USER_KIND_GROUP);
   DKPrivilegeSetICM * privSet_2 = (DKPrivilegeSetICM *)
     aclMgmt->retrievePrivilegeSet("PublicPrivSet");
   aclData2->setPrivilegeSet(privSet_2);
   ```

3. Create a new ACL. Set property values for this new ACL.

   ```
   DKAccessControlListICM * acl1 = new DKAccessControlListICM(ds);
   //Assign a new name to the newly-created ACL.
   acl1->setName("UserACL");
   // Assign a description to the newly-created ACL.
   acl1->setDescription("This is a user-defined ACL");
   ```

4. Add the ACL, created in step three, data objects to the ACL.

   ```
   acl1->addACLData(aclData1);
   acl1->addACLData(aclData2);
   ```

5. Add the ACL, created in step three, to the authorization manager.

```
aclMgmt->add(acl1);
```

Retrieving and displaying ACL information

To retrieve and display ACL information, complete the following steps.

1. Retrieve the ACL from the authorization management object using the ACL's name.

```
DKAccessControlListICM * acl_1 = (DKAccessControlListICM *) aclMgmt->
                        retrieveAccessControlList("UserACL");
cout<<"ACL name = "<< (char *)acl_1->getName() << endl;
cout<<"    desc = "<< (char *)acl_1->getDescription() << endl;
```

2. Retrieve the ACL data associated with the ACL.

```
dkCollection * coll = acl_1->listACLData();
dkIterator * iter = coll->createIterator();
char szpatronType[12] = {0x00};
while (iter->more())
{
       DKACLData * aclData = (DKACLData *)(void *)(*iter->next());
       DKPrivilegeSetICM * _privSet = (DKPrivilegeSetICM *)aclData->
                                      getPrivilegeSet();
       cout<<"privSet name = "<< (char *)_privSet->getName() << endl;
       cout<<"privSet desc = "<< (char *)_privSet->getDescription() << endl;
       DKString usrGrpName = aclData->getUserGroupName();
       cout<<"    UserGroupName = "<< (char *)usrGrpName << endl;
       short patronType = aclData->getPatronType();
       sprintf(szpatronType, "%d", patronType);
       cout<<"    Patron type   = "<< szpatronType << endl;
}
delete(iter);
```

Assigning an ACL to an item type

Once an ACL is created, it needs to be associated with a specific item or item type in order to regulate access to that specific controlled entity. The following steps show an ACL is associated with an item type. Note that if the ACL binding is done at the item type level, this ACL is enforced so that all items in the item type are bound to the ACL. At the item level, this code is used as the default ACL.

1. Set up a new item type.

```
DKItemTypeDefICM * itemType = new DKItemTypeDefICM(dsICM);
// Assign a name to this item type.
itemType->setName("TextResource1");
// Assign a description to this item type.
itemType->setDescription("CMv8.1 Text Resource Item Type.");
```

2. Assign an ACL code to the item type, and set the item type's ACL flag.

```
itemType->setItemTypeACLCode((long) 1);
// By setting the item type's ACL flag to TRUE (1),
//we confirm that ACL binding is at the item type level.
//By setting the item type's ACL flag to FALSE(0),
// we would be saying that the ACL binding is not at the item type level.
itemType->setItemLevelACLFlag((short)1);
itemType->add();
// make the item type persistent.
```

Assigning an ACL to an item

A Content Manager ACL is associated with a controlled entity. As described previously, a controlled entity can be both an item type and an item. Complete the following steps to assign an ACL to an item.

1. Create a new item (DDO) based on an existing item type.

```
DKDDO * ddoItem = dsICM->createDDO("book", DK_CM_ITEM);
```

2. Create a new ACL (Access Control List) and set the ACL's properties.
   ```
   DKAccessControlListICM * acl1 = new DKAccessControlListICM(dsICM);
   ```
3. Assign a name to the ACL.
   ```
   acl1->setName("MyACL");
   ```
4. Add a new property to the DDO. This property will be called DK_ICM_PROPERTY_ACL.
   ```
   ushort propId = ddoItem->addProperty(DK_ICM_PROPERTY_ACL);
   ```
5. Set the ACL, created above, as the value for this property. Note that a user can choose to retrieve an existing ACL and use it as the ACL this item.
   ```
   ddoItem->setProperty(propId, "MyACL");
   ```
6. Persist the DDO into the data store.
   ```
   ddoItem->add();
   ```

Understanding the query language

A powerful XML-based query language is now available in Content Manager 8.1. When an application searches for items stored in the Content Manager system, the underlying engine performs the processing of the search based on a query. To efficiently traverse Content Manager's hierarchical data model, you use the Content Manager query language. The query language provides the following benefits:

- Supports the full data model
- Enables searches within component type view hierarchies, across linked items, and references
- Combines parametric and text search
- Provides SORTBY capabilities
- Enforces Content Manager access control
- Conforms to XQuery Path Expressions (XQPE), a subset of W3C XML Query working draft
- Executes high performance searches

The Content Manager query language is an XML-based query language that, unlike proprietary languages, conforms to XQuery Path Expressions (XQPE), a subset of W3C XML Query working draft. The query language searches hierarchical item types and locates items quickly and easily. Before you begin to write queries, you must understand the query language concepts, syntax, and grammar.

All queries are done on component type views. Therefore, the names that you use for root components or child components in your query strings can be the names of either base component type views that are created for you by the system (for example, Journal, Journal_Article) or the names of user-defined component type views (for example, My_Journal, My_Book_Section).

When you submit a query for a document, folder, object, and so forth, your request is directed to the Content Manager query engine, which processes the query and translates it to the appropriate SQL. The generated SQL goes through the library server where access permission is verified by applying the query to the system ACLs. After the query passes through the ACLs, it is applied to database tables to locate the items. The search results are returned to the application in the form of DDOs and XDOs that are accessed through either a collection, a result set cursor, or a callback object. By interfacing the DDO and XDO collections with the Content

Manager persistent storage, your application can create, read, update, and delete documents into and from the Content Manager datastore.

Querying the Content Manager server

To query the Content Manager library server:

1. Query its datastore by creating a query string to represent your search conditions.
2. Call the evaluate, execute, or executeWithCallback method with your query string and query options.
3. Receive the results through a DKResults object, a dkResultSetCursor, or a dkCallback object.

The query APIs perform the query processing tasks, like preparing and executing a query, monitoring the status of a query execution, and storing the results.

Logically, a query string can contain one of three types of queries: parametric, text, and combined. A parametric query uses conditions like equality and comparison. A text query uses text search functions to make the search more powerful. A combined query is composed of both text and parametric conditions.

To run a query, you can use one of the following methods: evaluate, execute, or executeWithCallback. The execute method returns a dkResultSetCursor object, which has the following characteristics:

- The dkResultSetCursor works like a datastore cursor.
- You can use it for large result sets because the DDOs it returns are retrieved in blocks as the user requests them.
- You can use dkResultSetCursor to rerun a query, by calling the close and open methods.
- You can use the dkResultSetCursor to delete and update the current position of the result set cursor.

Applying the query language to the Content Manager data model

To help you understand the query language, you can conceptually view the library server as an XML document. The XML document analogy is used only for the purpose of explaining the query language. Therefore, it is very important to remember that the XML representation of the library server is only a virtual representation and items in a library server are not XML elements, nor do you get XML elements when you perform a query. In the XML representation of the library server, Content Manager data model elements are represented as follows:

Items In general, each CM Item is represented by nested XML elements, with the top level XML element representing the root component and the nested XML elements representing the descendent components. The nesting of XML elements thus represents the component hierarchy.

Root components
A root component is represented by the first level of an XML element. A root component has the following XML attributes: `ID ITEMID`, `STRING COMPONENTID`, `INTEGER VERSIONID`, `INTEGER SEMANTICTYPE`, and any other user-defined attributes of the component. In the library server, the ITEMID is unique.

Child components
A child component is represented by a nested XML element and has the

following attributes: STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID, and any other user-defined attributes of the component.

Note that COMPONENTID alone is only unique within a Content Manager component. TheITEMID and the VERSIONID are exactly the same as the child's root component ITEMID and VERSIONID.

User-defined attributes

Each user-defined attribute is represented by a nested XML attribute within the XML element representing the containing component.

Links Although the inbound and outbound links are not a part of an Item itself in the CM data model (i.e. they are stored separately in the Links table), for the purpose of querying it is very convenient to conceptually think of them as being a part of the XML element representing the Item. This relieves applications from writing joins explicitly in the queries. Links represent a one-to-many relationship between items. Note that this relationship is only between the root component (a link cannot originate or end in a child component). The links originating at an Item are represented by <OUTBOUNDLINK> XML elements with attributes: IDREF LINKITEMREF, IDREF TARGETITEMREF and STRING LINKTYPE. The LINKITEMREF is a reference to an Item that contains meta-data for the link. The TARGETITEMREF is a reference to the Item pointed to by the link. The LINKTYPE is the type of the link. Similarly links pointing to an Item are represented by <INBOUNDLINK> XML elements with attributes: IDREF LINKITEMREF, IDREF SOURCEITEMREF and STRING LINKTYPE. The SOURCEITEMREF is a reference to the Item where the link originates.

References (reference attributes)

Reference attributes are represented by XML attributes of type IDREF. A reference represents a one-to-one relationship between an item or a component and another item. Therefore, the target of a reference can only be a root component, not child. A reference attribute, however, can originate in either root or child components.

Reference attributes can be either system-defined (SYSREFERENCEATTRS) or user-defined (PublicationRef in sample queries below). References can be traversed in both directions.

Reverse traversal of references is performed in a way similar to reverse traversal of links, as described above. You can conceptually think of the item that is being referenced as having an XML element called REFERENCEDBY that contains an XML attribute called REFERENCER (of type IDREF), which points to the component that references this item. This is similar to the INBOUNDLINK element with the SOURCEITEMREF attribute for reverse traversal of links.

Unlike links, references remember the versions of the components they connect. References also support delete semantics (restrict, cascade, set null, or no action).

Documents

The query language offers a virtual XML element called ICMPARTS that allows access to all the ICM parts item types contained in an item type with the classification of document.

Understanding parametric search

Items are often retrieved by initiating a search on selected attributes. A single query can examine both system-defined and user-defined attributes of the items in the datastore. Simple search conditions consist of an attribute name, an operator, and a value that are combined into a clause. Content Manager provides you with many comparison operators to complete parametric searches. The operators include:

- "="
- "< "
- "<="
- ">"
- ">="
- "!="
- "LIKE"
- "NOT LIKE"
- "BETWEEN"
- "NOT BETWEEN"
- "IS NULL"
- "IS NOT NULL"

You can specify complex search conditions by combining simple search conditions into a clause using the Boolean operators AND, OR, and NOT. Refer to query examples for more details.

Understanding text search

Using the DB2 Universal Database Text Information Extender (TIE), Content Manager provides two types of text search: text search of attributes that contain text in components and text search of objects. The main difference between the two types of text search is how the content is stored. When you define an attribute is to be text searchable, you are indicating that one can search text contained in the column of that attribute. To make an attribute (column) text searchable, TIE creates a text index. The text index holds information about the text to be searched. This information is used to perform text search efficiently. For example, Fred, a system administrator, creates an item type called `Journal` that has a child component type view `Journal_Article`, which he wants to enable for text search. One of the attributes for `Journal_Article` is `Title`, which Fred enables for text search. When Lily, an underwriter, searches for `Title` that contains the word "Java", the system searches the `Title` text index for any hits on "Java".

For information about TIE, see the *DB2 Universal Database Text Information Extender (TIE)* documentation.

Searching for object contents

Searching for contents of objects works a little differently. Instead of indexing a column directly, the system uses a reference to the object's location on a resource manager. TIE uses the reference to fetch the content when it creates a text index. An end user performing a search does not notice any difference when searching for objects stored in a resource manager. A system administrator, however, has to set up a text resource item type view in order for the search mechanism to locate

the content in the resource manager. The text search is performed on the resource item type's attribute ″TIEREF″, which refers to the contents stored on the resource manager for text search purposes.

Searching for documents

You can perform text search on the contents of document parts. A virtual component type view ″ICMPARTS″ is supported in query as a child of every document in the system. The ″TIEREF″ attribute under the ″ICMPARTS″ component type view refers to the contents of all the text-searchable parts of that document for text search purposes. Refer to query examples for specific usage of this functionality.

Making user-defined attributes text searchable

You can make your user-defined attributes text searchable by using the DKAttrDefICM and DKItemTypeDefICM APIs. Default properties of the created text index can be modified by using the DKTextIndexDefICM class. For more information on the APIs, see the online API reference.

Understanding text search syntax

You can perform text search queries by using either basic or advanced text search syntax.

Basic text search

Since the majority of text searches are done by simply listing a few words one after the other, basic (simplified) text search syntax was designed specifically to make this most common case easy for users. The syntax also allows for use of ″+″ and ″-″, as well as for use of quoted phrases. Simplified text search is done by using ″contains-text-basic″ and ″score-basic″ functions. The ″contains-text-basic″ function is used to search within attributes or within content of resources or documents. The ″score-basic″ function uses the same syntax as the "contains-text-basic" function and is used for sorting results based on the rank of the text search results. Don't forget to equate the ″contains-text-basic″ function to 1 to check if it is true, and to equate it to 0 to check if it is false. Refer to query examples for how these functions are used.

Additional information about basic text search syntax includes the following:

- Performs case-insensitive text search (just like advanced syntax by default). See the TIE documentation for case-sensitive search options.
- Terms within quotes are assumed to be a phrase
- Use of + (plus) - (minus)
  - + (plus) = document must include this word
  - - (minus) = document must not include this word
  - When a + or - is not specified, the query engine uses an algorithm to match the words to the text.
- Boolean operators (AND, OR, NOT) are not valid and are ignored
- Parentheses in the basic syntax are not supported
- Valid wildcards
  - ? (question mark) = represents a single character
  - * (asterisk) = represents any number of arbitrary characters

Advanced text search

Advanced text search syntax is used to allow the user to specify more complex conditions for text search. The text search uses the TIE text search syntax, and allows such powerful features as proximity search and fuzzy search. Advanced text search syntax uses ″contains-text″ and ″score″ functions similar to the way the ″contains-text-basic″ and ″score-basic″ functions are used for basic text search. The strings that are supplied to the advanced functions should be in TIE syntax, except as follows: change double quotes to single quotes, and vice-versa. For example, CONTAINS (description,' ″IBM″ ')=1 condition in TIE would become contains-text(@description, ″ 'IBM' ″)=1 in CM query language. This needs to be done to support simplicity of writing queries with minimal use of escape characters. Don't forget to equate the ″contains-text″ function to 1 to check if it is true, and to equate it to 0 to check if it is false. Refer to query examples for more details on advanced text search.

Creating combined parametric and text search

Searches can be performed based on virtually any piece of an item or component, text within an item or component, or text within resource content. Search can be one of the following types:

Parametric search
: Searching is based on item and component properties, attributes, references, links, folder contents, and so forth.

Text Search
: Searching within text and based on text contained in anything marked as textSearchable (true) using a text search engine such as DB2 Text Information Extender (TIE).

Combined Search
: Searching using both parametric and text search.

The following example demonstrates how to create a combined parametric and text search.

1. Specify the search options. Note that the options array always has to have an end element. So, for example, if you want to specify two options, the options array must have three elements.

```
DKNVPair * parms = new DKNVPair[3];
DKNVPair * pparm = NULL;
DKString strMax  = "5";
DKAny * anyNull  = new DKAny();
//Allow a maximum of 5 items to be returned from the search
pparm = new DKNVPair(DK_CM_PARM_MAX_RESULTS, strMax);
parms[0] = *pparm;
delete pparm;
//Specify what content is to be retrieved
pparm = new DKNVPair((long)DK_CM_PARM_RETRIEVE, DK_CM_CONTENT_ATTRONLY +
                     DK_CM_CONTENT_LINKS_OUTBOUND);
parms[1] = *pparm;
delete pparm;
pparm = new DKNVPair(DK_CM_PARM_END, *anyNull);
parms[2] = *pparm;
delete pparm;
```

2. Execute the search. There are three ways to execute a search:

   evaluate
   : Returns all the results as a collection; good for small sets.

execute
Returns a result set cursor, which the caller uses to iterate over the results.

executeWithCallback
Creates a thread that iterates over the result set and calls the callback object for each block of results. Caller uses the callback object to get the results

In the example below, only five results are desired, so the DKDatastoreICM.evaluate method is used.

```
DKResults * resultsCollection = (DKResults *)(dkCollection *)
        dsICM->evaluate(queryString,DK_CM_XQPE_QL_TYPE, parms);
```

3. Display the results of the search.

```
// Create an iterator to go through Results collection.
dkIterator* iter = resultsCollection->createIterator();

cout << "Results:" << endl;
cout << "     - Total:  " << results->cardinality() << endl;

while(iter->more())
{
//Each element in the returned array is an item (DDO)
DKDDO* ddo = (DKDDO*) iter->next()->value();
           cout << "     - Item ID:  " << ((DKPidICM*)ddo->getPidObject())
           ->getItemId() << "  (" << ((DKPidICM*)ddo->getPidObject())
           ->getObjectType() << ")" << endl;

}
```

4. Clean up.

```
delete(iter);
delete[] parms;
....
```

Query Examples

To help you better understand the query language and to get you started with writing queries, this section provides you with the following information:

- A sample data model
- An XML document representation of the data model
- Sample queries
- Query language grammar

The sample queries in the following sections are based on the data model in Figure 57 on page 430. Refer to the data model illustration when you review the sample queries.

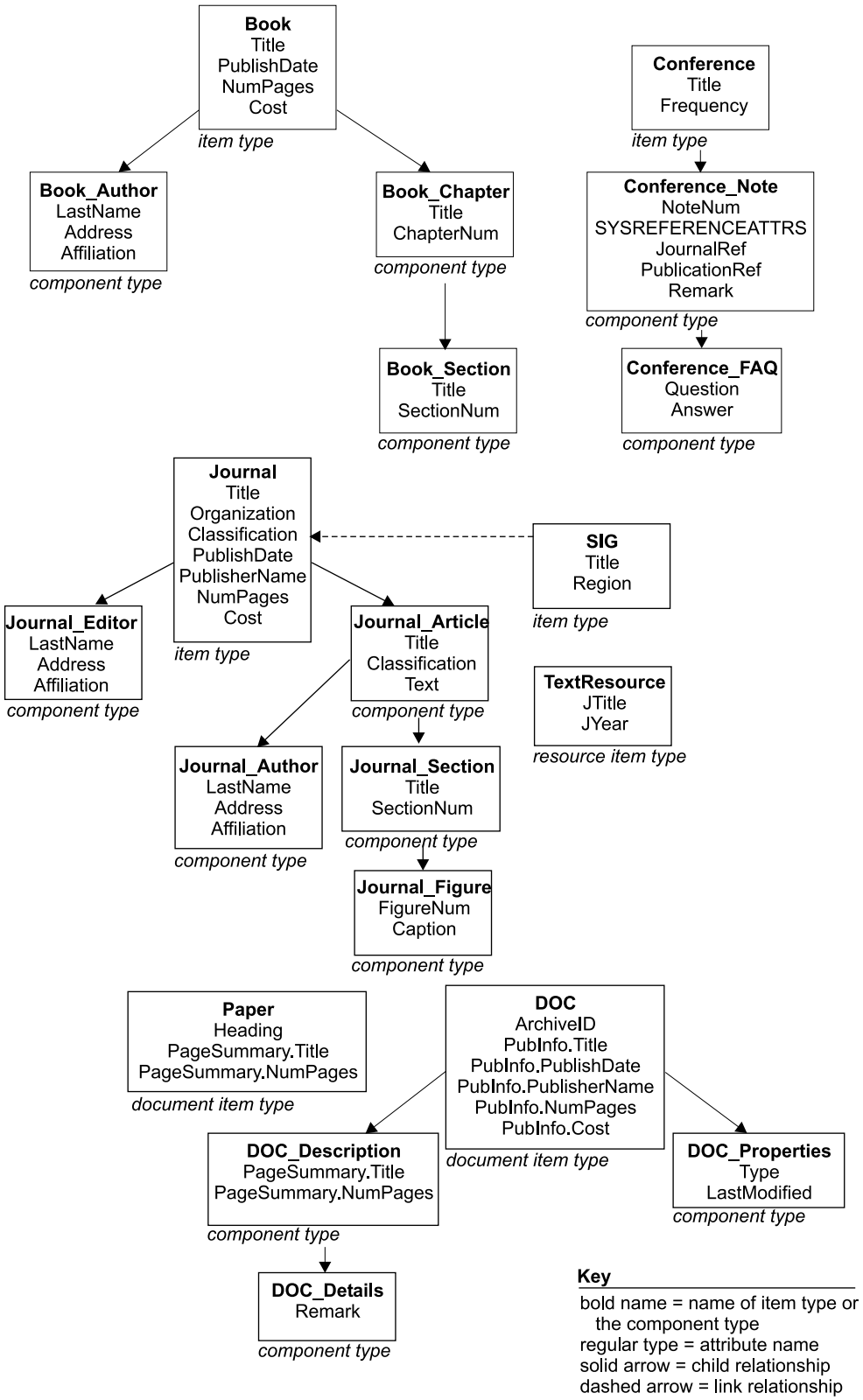

Figure 57. Query examples data model

The XML document below is a representation of the data model.

XML Representation of the query examples data model:

```
<Journal (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                 INTEGER SEMANTICTYPE, Title, Organization, Classification,
                 PublishDate, PublisherName, NumPages, Cost)>
    <Journal_Editor (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                     LastName, Address, Affiliation)>
    </Journal_Editor>
    ... (repeating <Journal_Editor>)

    <Journal_Article (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                      Title, Classification, Text)>
        <Journal_Section (STRING ITEMID, STRING COMPONENTID,
                          INTEGER VERSIONID, Title, SectionNum)>
            <Journal_Figure (STRING ITEMID, STRING COMPONENTID,
                             INTEGER VERSIONID, FigureNum, Caption)>
            </Journal_Figure>
            ...(repeating <Journal_Figure>)
        </Journal_Section>
        ... (repeating <Journal_Section>)

        <Journal_Author (STRING ITEMID, STRING COMPONENTID,
                         INTEGER VERSIONID, LastName, Address, Affiliation)>
        </Journal_Author>
        ... (repeating <Journal_Author>)
    </Journal_Article>
    ... (repeating <Journal_Article>)

    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                   STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                  STRING  LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</Journal>
...(repeating <Journal>)

<Book (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID, INTEGER
       SEMANTICTYPE,Title, PublishDate, NumPages, Cost)>
    <Book_Author (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                  LastName, Address, Affiliation)>
    </Book_Author>
    ... (repeating <Book_Author>)

    <Book_Chapter (STRING ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
                   Title, ChapterNum)>
        <Book_Section (STRING ITEMID, STRING COMPONENTID,
                       INTEGER VERSIONID, Title, SectionNum)>
        </Book_Section>
        ... (repeating <Book_Section>)
    </Book_Chapter>
    ... (repeating <Book_Chapter>)

    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                   STRING LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                  STRING  LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)
```

```
    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</Book>
... (repeating <Book>)

<SIG (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
      INTEGER SEMANTICTYPE, Title, Region)>
    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                   STRING  LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                  STRING  LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</SIG>
... (repeating <SIG>)

<TextResource (ID ITEMID, STRING COMPONENTID, INTEGER VERSIONID,
               INTEGER SEMANTICTYPE, JTitle, JYear)>
    <OUTBOUNDLINK (IDREF LINKITEMREF, IDREF TARGETITEMREF,
                  STRING  LINKTYPE) >
    </OUTBOUNDLINK>
    ... (repeating <OUTBOUNDLINK>)

    <INBOUNDLINK (IDREF LINKITEMREF, IDREF SOURCEITEMREF,
                  STRING LINKTYPE)>
    </INBOUNDLINK>
    ... (repeating <INBOUNDLINK>)

    <REFERENCEDBY (IDREF REFERENCER)>
    </REFERENCEDBY>
    ... (repeating <REFERENCEDBY>)
</TextResource>
... (repeating <TextResource>)
```

Query examples

The example queries provided in this section are based on the sample data model, Figure 57 on page 430, and the sample XML document, 430. Here are some hints to help you understand the query examples:

- Follow the query string as you would follow a directory structure
- "/" single slash indicates a direct child relationship
- "//" double slash indicates either a child relationship or a descendant relationship
- "." (DOT) represents the current component in the hierarchy
- ".." (DOT-DOT) represents the parent of the current component
- "@" (AT sign) denotes an attribute
- "[ ]" (square brackets) denote a conditional statement or a list
- "=>" (DEREFERENCE operator) represents linking or referencing action
- The result of the query must be a component (for example, an attribute cannot be the last thing in the path)

Example 1: access to components

This query finds all journals.

```
/Journal
```

Explanation: The "/" starts at the implicit root of the XML document, which in this case is the entire library server. Each item type is an element under this root. If LS.xml is the XML document that contains the entire model as described above, then the explicit document root is document (LS.xml).

Example 2: access to attributes

This query finds all journal articles with a total of 50 pages in them.

```
/Journal[@NumPages=50]
```

Explanation: The predicate @NumPages = 50 evaluates to true for all journals that have the Content Manager attribute "NumPages"set to 50.

Example 3: multiple item types

This query finds all books or journals that have "Williams" as one of the authors and have a section title beginning with "XML".

```
(/Book | /Journal)
[(.//Journal_Author/@LastName = "Williams"
OR .//Book_Author/@LastName = "Williams")
AND (.//Book_Section/@Title LIKE "XML%"
OR .//Journal_Section/@Title LIKE "XML%")]
```

OR

```
(/Book[.//Book_Author/@LastName = "Williams"
AND .//Book_Section/@Title LIKE "XML%"])
| (/Journal[.//Journal_Author/@LastName = "Williams"
 AND .//Journal_Section/@Title LIKE "XML%"])
```

Explanation: The above two queries produce the same result. ".//Journal_Author" means that a component Journal_Author should be found either directly under the current component in the path (which in the first case is either a Book or a Journal) or somewhere deeper in the hierarchy. Note that the LIKE operator is used in conjunction with a wildcard character, in this case" %".

Example 4: arithmetic operations in conditions

This query finds all journals with the number of pages between 45 and 200.

```
/Journal[@NumPages BETWEEN 49-4 AND 2*100]
```

Explanation: Note that you can perform arithmetic operations to calculate the resulting values to be used with the BETWEEN operator.

Example 5: traversal of links in the forward direction

This query finds all articles in journals edited by "Williams" that are contained in SIGs with title "SIGMOD".

```
/SIG[@Title = "SIGMOD"]/OUTBOUNDLINK
 [@LINKTYPE = "contains"]/@TARGETITEMREF =>
 Journal[Journal_Editor/@LastName = "Williams"]
 /Journal_Article
```

Explanation: This is an example of following links in the forward direction. The virtual XML component OUTBOUNDLINK and its attribute TARGETITEMREF are used to traverse to all Journals and then finally the underlying Journal_Articles. The last component in the path is what is

returned as the result of the query. The result can be constrained by traversing only specific link types (“contains” in this example) to a specific type of items (Journal in this example). Since the conceptual XML representation of the library server looks at inbound and outbound links as being parts of items, the dereferencing operator can be used to relieve applications from writing explicit joins.

Example 6: traversal of links in the backward direction

This query finds all items of any type that have journals which cost less than five dollars with articles by author“ Nelson”.

```
/Journal[@Cost < 5
AND .//Journal_Author/@LastName = "Nelson"]
/INBOUNDLINK[@LINKTYPE = "contains"]
/@SOURCEITEMREF => *
```

Explanation: This is an example of following links in the backward direction. The wildcard“*”, following the dereference operator “=>” ensures that items of ANY type are returned as the result.

Example 7: text search (contains-text and score functions)

This query finds journal articles with author“ Richardt” that contain the text “Java” and the text “XML”. The results are ordered by the text search score.

```
//Journal_Article[Journal_Author/@LastName = "Richardt"
AND contains-text(@Text, " 'Java' & 'XML' ")=1]
SORTBY(score(@Text, " 'Java' & 'XML' "))
```

Explanation: This is an example of performing text search with the contains-text function. For the syntax supported by this function, see the documentation for DB2 Text Information Extender (TIE). Note that the contains-text function should be equated with 1 to be true and 0 to be false. The score function uses the ranking information returned by TIE, which is used in this case to sort the resulting journal articles through SORTBY.

Example 8: text search (contains-text function and attribute sorting)

This query finds all journals that have either the word “Design” or the word“ Index” in their title and sorts the results in descending order by their title.

```
/Journal
[Journal_Article[contains-text(@Title, " 'Design' | 'Index'  ")=1]]
SORTBY (@Title DESCENDING)
```

Explanation: This is another example of performing text search using the contains-text function. The sorting in this case uses the DESCENDING operator on the “Title ”attribute. The default for the SORTBY is ASCENDING.

Example 9: text search (contains-tex-basic and score-basic functions)

This query finds all journal articles that contain the text “Java” and the text “JDK 1.3” but not the text “XML” using the simplified (basic) text search syntax and sort the results by the text search score.

```
//Journal_Article
[contains-text-basic(@Title, " +Java -XML +'JDK 1.3'")=1]
SORTBY (score-basic(@Title, " +Java -XML +'JDK 1.3' "))
```

Explanation: This is an example of performing text search using the simplified text search syntax. Use a“ +” to indicate the words or phrases that should be present in the attribute“ Title”, and, similarly, use a “-” to

exclude other words or phrases. The score-basic function works similarly to the score function in the previous example, but uses a simplified syntax.

Example 10: text search on resource items

This query finds text resources in a text resource item type “TextResource” that contain the text “Java” and the text “XML”.

```
/TextResource[contains-text(@TIEREF, " 'Java' & 'XML'
")=1]
```

Explanation: This is an example of performing text search inside of the resources in the resource manager. Note that the “TIEREF” attribute is used as a representation of the resource that is represented by the item of type “TextResource”. TIE syntax is used as usual in this case inside the contains-text function. For the syntax supported by this function, see the documentation for DB2 Text Information Extender (TIE).

Example 11: traversal of references in the forward direction

This query finds all the frequently asked questions for conferences, for which the conference notes refer to books with titles mentioning EIP.

```
 /Conference/Conference_Note [@PublicationRef =>
Book[@Title LIKE "%EIP%"]]
/Conference_FAQ
```

Example 12: traversal of references in the forward direction

This query finds all chapters of books referenced in the notes of conferences related to Internet.

```
/Conference[@Title LIKE "%Internet%"]
/Conference_Note/@PublicationRef => */Book_Chapter
```

Example 13: traversal of references in the reverse direction

This query finds all the components that have references pointing to any books.

```
/Book/REFERENCEDBY/@REFERENCER => *
```

Example 14: traversal of references in the reverse direction

This query finds all the frequently asked questions under conference notes that refer to books about XML.

```
/Book[@Title LIKE "XML"]/REFERENCEDBY/@REFERENCER =>
Conference_Note/Conference_FAQ
```

Explanation: Note that since the reference attributes originate inside of the Conference_Note component, this is the component that must appear as the first component after the dereference operator. This query produces an empty result set if, for example, Conference follows the“ =>” operator.

Example 15: traversal of references in the reverse direction

This query finds all the components that contain XML in their remarks and that have references pointing to books.

```
/Book/REFERENCEDBY/@REFERENCER =>
*[@Remark LIKE "%XML%"]
```

Example 16: latest version function

This query finds all the journals of the latest version. By default, all versions of the indicated component type view that match the query are returned. VERSIONID is a system-defined attribute that is contained in every component type.

```
 /Journal[@VERSIONID = latest-version (.)]
```

Example 17: latest version function on the target of the dereference

This query finds all the books of the latest version that are referenced in the notes of any conferences.

```
/Conference/Conference_Note/@SYSREFERENCEATTRS => Book
[@VERSIONID = latest-version(.)]
```

Example 18: latest version function on wildcard components

This query finds all the components of the latest version that have references pointing to any books.

```
/Book/REFERENCEDBY/@REFERENCER => *
[@VERSIONID = latest-version(.)]
```

Example 19: system-defined attributes

This query finds all the root components with a specific item ID.

```
/*[@ITEMID = "A1001001A01J09B00241C95000"]
```

Example 20: text search on document model

This query finds all documents that contain the word “XML” in any one of its parts.

```
/Doc[contains-text(.//ICMPARTS/@TIEREF, " 'XML' ")=1]
```

Explanation: The query language offers a virtual component “ICMPARTS” that allows access to all the ICM Parts item types contained under a specific item type of Document classification.

Example 21: document model (access to ICM Parts)

This query finds all the parts of the document with the storage ID of 555.

```
/Doc[@ArchiveID = 555]/ICMPARTS/@SYSREFERENCEATTRS => *
```

Example 22: document model (access to ICM Parts)

This query finds all the parts in all of the documents in the system.

```
//ICMPARTS/@SYSREFERENCEATTRS => *
```

Explanation: Because both the Doc and Paper item types have been defined as being Documents in the system, the ICM Parts from both of them are returned in the result.

Example 23: existence of attributes

This query finds all root components that have a title.

```
/*[@Title]
```

Explanation: To eliminate the restriction that only root components should be returned, the query can be rewritten to start with a double-slash

```
//*[@Title]
```

Example 24: list of both literals and expressions

This query finds all journals that have a title that is equal to either its article’s title, its section’s title, or ″IBM Systems Journal″.

```
/Journal[@Title = [Journal_Article/@Title,
.//Journal_Section/@Title,"IBM Systems Journal"]]
```

Example 25: list of numeric literals

This query finds all books that cost either $10, or $20, or $30.

```
/Book[@Cost = [10, 20, 30]]
```

Example 26: list of a result of query

This query finds all journals or all books with the title “Star Wars”.

```
[/Journal, /Book[@Title = "Star Wars"]]
```

Example 27: attribute groups

This query finds all details on documents in which the description is at least 20 pages long.

```
/Doc[Doc_Description/@PageSummary.NumPages >= 20]//Doc_Details
```

Explanation: Note that if an attribute (for example, "NumPages") is contained in an attribute group (for example, "PageSummary"), then you must refer to that attribute as GroupName.AttrName (for example, PageSummary.NumPages). The attribute "@NumPages" would not be found under Doc_Description.

Intermediate results obtained by INTERSECT/EXCEPT cannot be combined with arithmetic(unary/binary) or comparison operators. They can be combined by set operators (UNION/INTERSECT/EXCEPT) or appear by themselves.

Examples of valid usage of UNION/INTERSECT/EXCEPT:

1.
```
 (/Journal/Journal_Article[@Title = "Content Management"]
EXCEPT
//Journal_Article[@Classification = "Security"])/Journal_Section
```

   This query is valid because the result of the EXCEPT is the result of the entire query - it is not combined using any operators.

2.
```
/Journal[(Journal_Editor/@LastName
UNION .//Journal_Author/@LastName) = "Davis"]
```

   This query is valid because there is no restriction on UNION operator.

3.
```
/Journal[Journal_Article[Journal_Section/@Title INTERSECT
.//Journal_Figure/@Caption]/@Title = "Content Management"]
```

   This query is valid because the result of INTERSECT is not combined using any operator.

4.
```
/Journal[@Title = "VLDB"]
UNION /Journal[@Cost = 20]
INTERSECT /Journal[@Organization = "ACM"]
```

   This query is valid because the result of INTERSECT operator is combined using a set operator (UNION).

Examples of invalid usage of INTERSECT/EXCEPT:

1.
```
/Journal[(Journal_Editor/@LastName
INTERSECT .//Journal_Author/@LastName) = "Davis"]
```

   This query is invalid because the result of INTERSECT operator is combined using a comparison operator (=).

2.
```
/Journal[(.//Journal_Section/@SectionNum
EXCEPT .//Journal_Figure/@FigureNum) + 5 = 10]
```

   This query is invalid because the result of EXCEPT is combined using an arithmetic operator (+).

Escape sequences in the query language

To support advanced features of the query language (like the wildcards ″%″ or ″_″ inside of text strings), escape sequences are used to differentiate between the cases when wildcards are treated as regular characters versus when they are given the special meaning of wildcard characters. For the user, it is important to know which characters are used as wildcards because wildcard characters, when intended to be

treated as regular characters, must be preceded by an escape character. Escape sequences are also used to handle single and double quotes.

You need to add escape sequences when the strings used in queries contain either special characters (double-quote, apostrophe) or wildcard characters (percent sign, underscore, star, question mark) or a default escape character (a backlash). This handling is the simplest for strings used in comparison conditions and becomes a bit more involved for the LIKE operator and text search functions. Proper handling of special characters will ensure successful execution of queries and correctness of query results.

Important: Use wildcard characters sparingly as using them in your queries can increase the size of your result list significantly, which can decrease performance and return unexpected search results.

Using escape sequences with comparison operators (″=″, ″!=″, ″>″, ″<″, ″BETWEEN″ and others)

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[@Title = "Analysis of ""The Time Machine"" by H.
G. Wells himself"]
```

Since the article's title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark (apostrophe) ’

You do not need to escape in this case.

Example:

```
/Book[@Title != ″Uncle Tom's Cabin″]
```

Using escape sequences with the LIKE operator

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[@Title LIKE "Analysis of ""The Time Machine"" %"]
```

Since the article's title contains the name of the book in double quotes,″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark (apostrophe) ’

You do not need to escape in this case.

Example:

```
 /Book[@Title LIKE "Uncle Tom's Cabin"]
```

Wildcards (″%″, ″_″)

The percent sign ″%″ is a wildcard character used to represent any number of arbitrary characters in a string used with the LIKE operator. The underscore ″_″ is a wildcard character used to represent a single arbitrary character. If you want these wildcard characters to be treated as regular characters, you need to do the following:

1. Precede the wildcard character with an escape character

2. Add an ESCAPE clause with the escape character after the LIKE phrase.

Example A:

```
/Book[@Title LIKE "Plato%s%S_mposium"]
```

This example shows how wildcards ″%″ and ″_″ are used to find a book whose title's spelling is uncertain.

Example B:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ in query"
ESCAPE "!"]
```

Since the search string in this example contains the underscore ″_″ as a regular character (not a wildcard), you can escape the underscore with an exclamation point character ″!″. Any single character can be used as an escape character.

Example C:

```
//Journal_Article[@Title LIKE "_sage of underscore \_ in%" ESCAPE
"\"]
```

In this query, wildcard characters are used as both regular characters (″_″ escaped by ″\″) and as wildcards (″_″) to catch both uppercase and lowercase versions of the word ″Usage″, as well as ″%″ to catch multiple endings of the string.

Example D:

```
//Journal_Article[@Title LIKE "Usage of underscore !_ on Yahoo!!"
ESCAPE "!"]
```

You can also use an escape character as a regular character. To do so, precede the escape character with itself, as in the example to search for ″Yahoo!″ below.

Using escape sequences with advanced text search (″contains″ and ″score″ functions)

Double quotation mark ″
Precede your double quote with another double quote.

Example:

```
//Journal_Article[contains-text (@Title, " 'Analysis of ""The Time
Machine"" %' ")=1]
```

Since the article's title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped.

Single quotation mark(apostrophe) ’
Precede the apostrophe with another apostrophe. A single apostrophe is not allowed in advanced text search because a set of apostrophes is used to

enclose a term or a phrase. If an apostrophe appears inside a term, then the apostrophe needs to be escaped to differentiate it from the apostrophe that ends the term or the phrase.

Example A:

```
/Book[contains-text (@Title, " 'Uncle Tom''s Cabin' ")=1] SORTBY
(score (@Title, " 'Uncle Tom''s Cabin' "))
```

Note that Tom''s has two apostrophes.

Example B:

```
/Book[contains-text (@Title, " ('Greek' & 'Plato''s Symposium') &
NOT ' Socrates' ")=1] SORTBY (score (@Title, " ('Greek' & 'Plato''s
Symposium') & NOT ' Socrates' "))
```

Note that Plato''s has two apostrophes.

Wildcards (″%″, ″_″)

Just as the LIKE operator, advanced syntax uses ″%″ and ″_″ as wildcards. The percent sign ″%″ is a wildcard character used to represent any number of arbitrary characters. The underscore ″_″ is a wildcard character used to represent a single arbitrary character. If you want a wildcard character to be treated as a regular character, you need to do the following:

1. Precede the wildcard character with an escape character
2. Add an ESCAPE clause after EACH term where you use the escape character

Example A:

```
/Book[contains-text (@Title, " 'Usage of underscore !_ in query'
ESCAPE '!' ")=1] SORTBY (score (@Title, " 'Usage of underscore !_ in
query' ESCAPE '!' "))
```

In this example, an exclamation mark ″!″ is used as an escape character before the underscore.

Example B:

```
/Book[contains-text (@Title, " 'Usage of underscore !_ in query'
ESCAPE '!' | 'Yahoo! For Dummies' | 'Usage of underscore !_ on
Yahoo!!' ESCAPE '!' | 'War and Peace' ")=1]
```

Note that an ESCAPE clause must be added after every term in your text search string where you escape wildcards, even if the escape character is the same in all the terms.

Using escape sequences with basic text search (″contains-text-basic″ and ″score-basic″ functions)

Double quotation mark ″

Precede your double quote with another double quote.

Example:

```
//Journal_Article[contains-text-basic (@Title, "Analysis of '""The
Time Machine""' ")=1]
```

Since the article's title contains the name of the book in double quotes, ″The Time Machine″, these internal double quotes need to be escaped. The book title is inclosed in apostrophes to keep it as a phrase.

Single quotation mark (apostrophe) ’

Precede the apostrophe with another apostrophe. Basic text search syntax allows terms enclosed within single quotes, so that a term can contain a space. The doubling of the apostrophe is therefore necessary to differentiate the case of an apostrophe occurring within a term from the case of an apostrophe starting a new term.

Example A:

```
/Book[contains-text-basic (@Title, "Uncle Tom''s Cabin")=1]SORTBY
(score-basic (@Title, "Uncle Tom''s Cabin"))
```

Note that Tom’’s has two apostrophes.

Example B:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato''s Symposium'
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek +'Plato''s
Symposium' -Socrates "))
```

Note that Plato’’s has two apostrophes and ’Plato’s Symposium’ is enclosed in single quotes since it is a phrase.

Wildcards (″*″, ″?″ and ″\″)

Precede ″*″, ″?″, and ″\″ characters with a backslash ″\″ if these characters are not to be treated as wildcards. Star ″*″ is a wildcard character used to represent any number of arbitrary characters in basic text search for the functions contains-text-basic and score-basic. The question mark ″?″ is a wildcard character used to represent a single arbitrary character. For basic text search, the query language provides an escape character backslash ″\″ to be used when the term to be searched contains the wildcard character in it and you want to treat that wildcard character as a regular character.

Example A:

```
/Book[contains-text-basic (@Title, " +Greek +'Plato*s*S?mposium'
-Socrates ")=1] SORTBY (score-basic (@Title, " +Greek
+'Plato*s*S?mposium' -Socrates "))
```

This example shows how to use basic text search when the spelling of a term is not certain. The ″*″ and ″?″ characters are meant to be wildcards in this case, so they are not escaped.

Example B:

```
/Book[contains-text-basic (@Title, "Why forgive\?")=1] SORTBY
(score-basic (@Title, "Why forgive\?"))
```

In this example, the title contains the question mark ″?″ as a normal character, so this character can be escaped with a backslash.

Example C:

```
//Journal_Section[contains-text-basic (@Title,
"C:\\OurWork\\IsNeverDone")=1] SORTBY (score-basic (@Title,
"C:\\OurWork\\IsNeverDone"))
```

Each backslash that naturally occurs in the search term`"C:\OurWork\IsNeverDone"` must be escaped with another backslash.

Using escape sequences in c++

Precede special characters (for example, double quotes and backslash) with a backslash.

Example:

Query:

```
/Book[contains-text-basic (@Title, ″Why forgive\?″)=1]
```

C++:

```
DKString query (″/Book[contains-text-basic (@Title, \″Why forgive\\?\″)=1]″);
```

Note how the internal double quotes and the backslash before the question mark are preceded by a backslash. This handling is inherent to Java and C++ programming languages. For more information, refer to the specifications for these languages.

The query language grammar

The query language formal grammar is as follows:

```
(* keywords *)
AND = (″a″ | ″A″), (″n″ | ″N″), (″d″ | ″D″) ;
ASCENDING = (″a″ | ″A″), (″s″ | ″S″), (″c″ | ″C″), (″e″ | ″E″), (″n″ | ″N″), (″d″ | ″D″), (″i″ | ″I″), (″n″ | ″N″), (″g″ | ″G″) ;
BETWEEN = (″b″ | ″B″), (″e″ | ″E″), (″t″ | ″T″), (″w″ | ″W″), (″e″ | ″E″), (″e″ | ″E″), (″n″ | ″N″) ;
DESCENDING = (″d″ | ″D″), (″e″ | ″E″), (″s″ | ″S″), (″c″ | ″C″), (″e″ | ″E″), (″n″ | ″N″), (″d″ | ″D″), (″i″ | ″I″), (″n″ | ″N″), (″g″ | ″G″) ;
DIV = (″d″ | ″D″), (″i″ | ″I″), (″v″ | ″V″) ;
EXCEPT = (″e″ | ″E″), (″x″ | ″X″), (″c″ | ″C″), (″e″ | ″E″), (″p″ | ″P″), (″t″ | ″T″) ;
INTERSECT = (″i″ | ″I″), (″n″ | ″N″), (″t″ | ″T″), (″e″ | ″E″), (″r″ | ″R″), (″s″ | ″S″), (″e″ | ″E″), (″c″ | ″C″), (″t″ | ″T″) ;
 LIKE = (″l″ | ″L″), (″i″ | ″I″), (″k″ | ″K″), (″e″ | ″E″) ;
 MOD = (″m″ | ″M″), (″o″ | ″O″), (″d″ | ″D″) ;
NOT = (″n″ | ″N″), (″o″ | ″O″), (″t″ | ″T″) ;
OR = (″o″ | ″O″), (″r″ | ″R″) ;
SORTBY = (″s″ | ″S″), (″o″ | ″O″), (″r″ | ″R″), (″t″ | ″T″), (″b″ | ″B″), (″y″ | ″Y″) ;
UNION = (″u″ | ″U″), (″n″ | ″N″), (″i″ | ″I″), (″o″ | ″O″), (″n″ | ″N″) ;
IS = (″i″ | ″I″), (″s″ | ″S″);
NULL = (″n″ | ″N″), (″u″ | ″U″), (″l″ | ″L″), (″l″ | ″L″);
ESCAPE_KEYWORD = (″e″ | ″E″), (″s″ | ″S″), (″c″ | ″C″), (″a″ | ″A″), (″p″ | ″P″), (″e″ | ″E″);
KEYWORD = ( AND | ASCENDING | BETWEEN | DESCENDING | DIV | EXCEPT | INTERSECT | LIKE | MOD | NOT | OR | SORTBY | UNION | IS | NULL) ;
(* literals *)
```

DIGIT = (″0″ | ″1″ | ″2″ | ″3″ | ″4″ | ″5″ | ″6″ | ″7″ | ″8″ | ″9″) ;

NONZERO_DIGIT = (″1″ | ″2″ | ″3″ | ″4″ | ″5″ | ″6″ | ″7″ | ″8″ | ″9″) ;

Exponent = (e | E), [″+″ | ″-″], DIGIT, { DIGIT }

INTEGER_LITERAL = ″0″ | NONZERO_DIGIT, {DIGIT} ;

FLOAT_LITERAL = DIGIT, { DIGIT }, ″.″, { DIGIT }, [Exponent] | [″.″], DIGIT, { DIGIT }, [Exponent] ;

(* UNICODE_CHARACTER is the set of all unicode characters and escape sequences. It's definition is not included in this document *) (* String literals are delimited by double quotes and can contain any character except double quote. To include a double quote as the part of the string literal, specify two consecutive double quotes i.e. a double quote is escaped by another double quote. These will be treated as one double quote character *)

STRING_LITERAL = '″', { (UNICODE_CHARACTER - '″') | ('″', '″') } , '″' ;

(* Escape sequence is a single character delimited by double quotes. To specify a double quote itself as the escape character, specify two consecutive double quotes i.e. a double quote is escaped by another double quote. These will be treated as one double quote character. For the complete explanation of the legal values for ESCAPE_CHARACTER, please see the DB2 SQL Reference section on the LIKE Predicate. *)

ESCAPE_LITERAL = '″', ((ESCAPE_CHARACTER - '″') | ('″', '″')), '″';

LETTER = (″a″ | ″b″ | ″c″ | ″d″ | ″e″ | ″f″ | ″g″ | ″h″ | ″i″ | ″j″ | ″k″ | ″l″ | ″m″ | ″n″ | ″o″ | ″p″ | ″q″ | ″r″ | ″s″ | ″t″ | ″u″ | ″v″ | ″w″ | ″x″ | ″y″ | ″z″ | ″A″ | ″B″ | ″C″ | ″D″ | ″E″ | ″F″ | ″G″ | ″H″ | ″I″ | ″J″ | ″K″ | ″L″ | ″M″ | ″N″ | ″O″ | ″P″ | ″Q″ | ″R″ | ″S″ | ″T″ | ″U″ | ″V″ | ″W″ | ″X″ | ″Y″ | ″Z″ | ″_″ | ″$″) ;

(* An IDENTIFIER begins with a letter (a-z, A-Z) or an underscore or a dollar character, followed by zero or more letters, underscores, dollar characters or digits (0-9). A keyword can be an IDENTIFIER only if it is enclosed within single quotes *)

IDENTIFIER = (LETTER, { LETTER | DIGIT }) - KEYWORD | ″'″, LETTER, { LETTER | DIGIT }, ″'″;

ExpressionWithOptionalSortBy = LogicalOrSetExpression, SORTBY, ″(″, SortSpecList, ″)″ | Expression ;

Expression = LogicalOrSetExpression ;

SortSpecList = SortSpec, { ″,″, SortSpec } ;

SortSpec = Expression, [ASCENDING | DESCENDING] ;

LogicalOrSetExpression = LogicalOrSetTerm | LogicalOrSetExpression, (OR | UNION | ″|″ | EXCEPT), LogicalOrSetTerm ;

LogicalOrSetTerm = LogicalOrSetPrimitive | LogicalOrSetTerm, (AND | INTERSECT), LogicalOrSetPrimitive ;

LogicalOrSetPrimitive = [NOT], SequencedValue ;

SequencedValue = ValueExpression ;

ValueExpression = Comparison ;

Comparison = ArithmeticExpression | Comparison, CompareOperator, ArithmeticExpression, ESCAPE_KEYWORD, ESCAPE_LITERAL | Comparison, CompareOperator, ArithmeticExpression | Comparison, [NOT], BETWEEN, ArithmeticExpression, AND, ArithmeticExpression | Comparison, IS, [NOT], NULL ;

ArithmeticExpression = ArithmeticTerm | ArithmeticExpression, (″+″ | ″-″), ArithmeticTerm ;

```
ArithmeticTerm = ArithmeticFactor | ArithmeticTerm, ("*" | DIV | MOD),
ArithmeticFactor ;
ArithmeticFactor = ArithmeticPrimitive | ("+" | "-"), ArithmeticFactor ;
ArithmeticPrimitive = BasicExpression, OptionalPredicateList | PathExpression ;
PathExpression = Path | ("/" | "//"), Path | BasicExpression,
OptionalPredicateList, ("/" | "//"), Path ;
 Path = Step | Path, ("/" | "//"), Step ;
 Step = NodeGenerator, OptionalPredicateList ;
NodeGenerator = NameTest | "@", NameTest | "@", NameTest, "=>", NameTest
| ".." ;
 OptionalPredicateList = {Predicate} ;
Predicate ::= [", Expression, "]" ;
BasicExpression = Literal | FunctionName, "(", OptionalExpressionList, ")" | "("
Expression ")" | ListConstructor | "." ;
FunctionName = QName ;
Literal = STRING_LITERAL | INTEGER_LITERAL | FLOAT_LITERAL ;
OptionalExpressionList = [ ExpressionList ] ;
ExpressionList = Expression, {",", Expression } ;
ListConstructor = "[", [ListContent], "]" ;
ListContent = Expression, {",", Expression } ;
NameTest = QName | "*" ;
 QName = LocalPart ;
LocalPart = IDENTIFIER;
CompareOperator = "=" | "< " | "<=" | ">" | ">=" | "!=" | [NOT] LIKE;
```

Working with the resource manager

Within Content Manager, every managed entity is called an item. Items come in two types, the type that represent pure logical entities, such as documents or folders, or entities that represent physical data objects, such as the text data of a word processing document, the scanned image of a claim or the video clip of a automobile accident. Objects have a special state and behavior needed to handle the physical data associated to a logical document.

Resource objects also represent things like files in a file system, video clips in a video server, and BLOBs. At run time, resource objects are used to access the physical data they point to. For that reason, resource objects in Content Manager have a type. That is, they have a specific state and behavior. The library server and the resource manager share a schema to store the state of an object. The base object types provided by Content Manager are: generic BLOBs or CLOBs, Text, Image, andVideo content objects. You can also create sub-classes of the pre-defined types. A resource object can also have user-defined attributes, which are used for search and retrieval.

From the Content Manager system perspective, each object is represented by a unique logical identifier, its Uniform Resource Identifier (URI). The library server manages the URI name space. On request, the library server maps URIs onto Uniform Resource Locators (URL). URLs are used to gain access to the physical data. URLs do not point directly to a storage area managed by the resource manager. Instead, the resource manager uses a local name space to convert logical object names to physical file names. Object URIs are created by the specific

resource manager. The library server or the end-user can suggest an object URI (its name), but the decision is made by the resource manager.

You can access an object using the Content Manager resource manager APIs (store, retrieve, update, delete, and so forth). In some cases, you can use APIs that are native to the object (stream, multicast, and stage) or filesystem.

For information about how to work with resource manager objects, see the SResourceItemCreationICM sample in the `samples` directory, `cmbroot/samples/cpp/icm`.

Managing documents in Content Manager

Content Manager implements a flexible document management data model that you can use for managing business objects. The basic elements of the data model includes folders, documents, and objects.

As mentioned earlier, documents, folders, and other objects are all represented by items in the Content Manager system. One difference between a folder and a document, however, is the semantic type attribute. A document is comprised of attributes, or metadata, that describe the document, including single valued attributes (document name, date, subject), multi-valued attributes (keywords), and collections of multi-valued attributes (address, consisting of street, city, state, and zip).

The document management data model uses a child component to associate objects (resource items) with the document. This model supports more than one part to construct a document. For example, each page could be a separate part. In order for an application to determine the order of the parts that make up a document, a part number is stored in the child component. The child component contains a pointer to the object (a reference attribute) which contains other information about the part such as MIME type, size, the resource manager ID which contains the part, the collection name on that resource manager and so forth. Every object can have different attributes. For example, an annotation might have X and Y coordinates, while a note log might have the CCSID of the text of the note.

To help you understand the document management data model, consider the following scenario of a user who imports a document using a client application:

- A window is displayed to the user.
- The user enters (or selects) the name of the file to import into the system. For example, a police report.
- The user selects the type of document (memo, claim, design).
- A new window opens, where the user can enter attributes that describe the document. The date, a claim number, and insurance policy number, for example.
- The user defines a child component and enters some values for the attributes. The police report is a child component of a claim, for example.
- The user finishes entering the document descriptions and completes the task. The police report is created.

Using either the APIs or the JavaBeans, the client application then connects to the library server and the resource manager. The system creates two items (a non-resource item and a resource item) to store the document. Two items are created because the police report contains a photograph, which is stored in the resource manager. The document is created with a single API call. The object is

then stored in the resource manager and the resource manager returns the timestamp, and other metadata for the object. The resource manager creates a reference attribute for the object, and inserts the reference attribute into the child component of the document. A final call to the library server is made to store the child component and to update the attributes. The entire process is bound by a transaction so that any API failure does not result in a partially created document.

After you create a document, you can update it. You can perform two types of updates: change the metadata or change the content. The library server automatically creates a new item record with the next version number (if the item is version-enabled) and copies all of the child components associated with the item.

Working with the document model

This section helps you complete the main tasks associated with the document management data model:

- Creating a document item type
- Create a document
- Updating a document
- Retrieving and deleting a document
- Versioning of parts in the document management data model

Creating a document item type

```
//Retrieve the datastore definition object
DKDatastoreDefICM * dsDefICM =  (DKDatastoreDefICM *)dsICM->datastoreDef();
//Retrieve the datastore administration object
DKDatastoreAdminICM * pdsAdmin = (DKDatastoreAdminICM *)dsDefICM->datastoreAdmin();
DKItemTypeDefICM *itemType = NULL;
DKItemTypeRelationDefICM *itemTypeRel = NULL;
DKAttrDefICM* attr = NULL;
//Retrieve an attribute for the document item type.
//If the attribute does not exist, create it.
dkAttrDef *pAttr = dsDefICM->retrieveAttr("docTitle1");
if(pAttr == NULL)
{
     attr = new DKAttrDefICM(dsICM);
     attr->setName("docTitle1"); //attribute name column name
     attr->setType(DK_CM_CHAR);
     attr->setSize(100);
     attr->setNullable(false);
     attr->setUnique(false);
     attr->add();
 pAttr = attr;
}


itemType = new DKItemTypeDefICM(dsICM);
itemType->setName("DocModelTest");
itemType->setDescription("This is a test Item Type");
itemType->setClassification(DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);
itemType->setAutoLinkEnable(false);

itemType->setVersionControl((short)DK_ICM_VERSION_CONTROL_ALWAYS);
itemType->setVersioningType(DK_ICM_ITEM_VERSIONING_FULL);

itemType->addAttr(pAttr);


//create relation between the document we just created and part1
//To do that, first retrieve EntityDef for each Part
```

```
DKItemTypeDefICM *itemTypePart1 = (DKItemTypeDefICM *)
   dsDefICM->retrieveEntity("ICMBASE");
//int part1ITypeid = itemTypePart1->getItemTypeId();
int part1ITypeid = itemTypePart1->getIntId();

//for each part, create a ItemType Relation and set values
DKItemTypeRelationDefICM *itemTypeRelPart1= new DKItemTypeRelationDefICM(dsICM);
itemTypeRelPart1->setTargetItemTypeID(part1ITypeid);

//sets the default resource manager
itemTypeRelPart1->setDefaultRMCode((short)1);

//sets the default ACL code
itemTypeRelPart1->setDefaultACLCode(1);
//Set the default collection where item resources pertaining
//to this item type are to be stored
itemTypeRelPart1->setDefaultCollCode((short)1);

//Set the default prefetch collection where item resources
//pertaining to this item type are to be stored
itemTypeRelPart1->setDefaultPrefetchCollCode((short)1);

itemTypeRelPart1->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);
itemTypeRelPart1->setSourceItemTypeID(itemType->getIntId());


// add the ItemType relation to the Document (Source)
itemType->addItemTypeRelation(itemTypeRelPart1);

//create relation between the document we just created and part2
//To do that, first retrieve EntityDef for each Part
DKItemTypeDefICM *itemTypePart2 = (DKItemTypeDefICM *)
   dsDefICM->retrieveEntity("ICMANNOTATION");
int part2ITypeid = itemTypePart2->getIntId();

//for each part, create a ItemType Relation and set values
DKItemTypeRelationDefICM * itemTypeRelPart2= new
  DKItemTypeRelationDefICM(dsICM);
itemTypeRelPart2->setTargetItemTypeID(part2ITypeid);
itemTypeRelPart2->setDefaultRMCode((short)1);
itemTypeRelPart2->setDefaultACLCode(1);
itemTypeRelPart2->setDefaultCollCode((short)1);
itemTypeRelPart2->setDefaultPrefetchCollCode((short)1);

itemTypeRelPart2->setVersionControl((short)DK_ICM_VERSION_CONTROL_NEVER);

itemTypeRelPart2->setSourceItemTypeID(itemType->getIntId());

// add the ItemType relation to the Document (Source)
itemType->addItemTypeRelation(itemTypeRelPart2);

// Update the definition of the item type in the Library Server
itemType->add();
```

Creating a document

The following example shows how to create a document based on the document model. **Example:**

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Create a new DDO of type DocModelTest and semantic type DK_CM_DOCUMENT
DKDDO* ddoDocument = dsICM->createDDO("DocModelTest",DK_CM_DOCUMENT);
ddoDocument->setData(ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,
                     DKString("docTitle1")),DKString("this is a string value"));
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
      pdsDef->retrieveEntity("DocModelTest");
// Retrieves the collection of DKItemTypeRelationDefICM object for the given source
```

```
// item type from the persistent store
dkCollection* pRelationColl = itemType->retrieveItemTypeRelations();
dkIterator* pIter = pRelationColl->createIterator();
int noOfPartsToCreate = pRelationColl->cardinality();
DKParts* pPartColl= NULL;
// Create the parts collection for the object if it does not exist.
short dataId = ddoDocument->dataId(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
if (dataId ==0) {
dataId = ddoDocument->addData(DK_CM_NAMESPACE_ATTR,DK_CM_DKPARTS);
pPartColl = new DKParts();
ddoDocument->setData(dataId,pPartColl);
}
else {
pPartColl =(DKParts*) (ddoDocument->getData(dataId).value());
if (pPartColl ==NULL) {
pPartColl = new DKParts();
ddoDocument->setData(dataId,pPartColl);
}
}
int i=0;
DKItemTypeRelationDefICM* itemTypeRelPart =NULL;
DKItemTypeDefICM* pEnt =NULL;
// CV v8 BLOB
DKLobICM* pPart =NULL;
DKString str = "This is to test the document model with two parts";
while(pIter->more()) {
i=i+1;
itemTypeRelPart=(DKItemTypeRelationDefICM*) pIter->next()->value();
pEnt =(DKItemTypeDefICM*)
     ((DKDatastoreDefICM*) pdsDef)->retrieveEntity(
               (long)itemTypeRelPart->getTargetItemTypeID());
pPart =(DKLobICM*) dsICM->createDDO(pEnt->getName(), DK_CM_RESOURCE);
pPart->setPartNumber(i);
pPart->setContent(str);

DKAny any = (dkDataObjectBase*) pPart;
pPartColl->addElement(any);
}
//Add the DDO to the datastore
ddoDocument->add();
```

Updating a document

```
DKDatastoreDefICM* pdsDef = (DKDatastoreDefICM*) dsICM->datastoreDef();
// Create a Document DDO from a PID string
DKString pidString = ....;
//...
DKDDO* ddoDocument = dsICM->createDDO(pidString);
DKPidICM* pPID = (DKPidICM*) ddoDocument->getPidObject();
DKString* pstrItemType = &pPID->getObjectType();
// Retrieves the definition for the item type "DocModelTest" from the
// persistent datastore. The definition is returned as a the *dkEntityDef
// object that is in turn typecast to a DKItemTypeDefICM object
DKItemTypeDefICM* itemType = (DKItemTypeDefICM*)
          pdsDef->retrieveEntity(*pstrItemType);
ddoDocument->setData(ddoDocument->dataId(
                    DK_CM_NAMESPACE_ATTR,DKString("docTitle1")),
                    DKString("this is a new string value"));

DKString updateString = "This is updated part";
DKSequentialIterator* pSeqIter = NULL;
DKLobICM* pPart = NULL;
short dataId = ddoDocument->dataId (DK_CM_NAMESPACE_ATTR, DK_CM_DKPARTS);
DKParts* pParts = (DKParts*) &ddoDocument->getData(dataId);
if (pParts != NULL) {
   pSeqIter = (DKSequentialIterator*) pParts->createIterator();
}
```

```
else return; // quit
pPart = (DKLobICM *) pSeqIter->next();
// Update the existing part with the new content
pPart->setContent(updateString);
// Add a new part to the Document
DKLobICM* pPart1 = (DKLobICM*) dsICM->createDDO ("ICMNOTELOG", DK_CM_RESOURCE);
pPart1->setPartNumber(3);
DKString pTempData = "This is to test the document model with two parts";
pPart1->setContent(pTempData);
DKAny any = (dkDataObjectBase*)(DKDDO*) pPart1;
pParts->addElement(any);
//Update the DDO information in the datastore.
ddoDocument->update();
delete pSeqIter;
```

Retrieving and deleting a document

To retrieve a document, call ddo->retrieve(option). If you set option to DK_ICM_CONTENT_YES, the parts TOC list as well as the parts is retrieved. Otherwise, only the TOC list of the parts is retrieved.

To delete a document, call ddo->del(). The document and its attached parts are deleted

Versioning of parts in the document management data model

Versioning properties of document parts are dependent on the versioning properties of the document. Versioning characteristics of document parts include the following:

- Like regular documents, parts can have one of three versioning models: versioned-always, versioned-never (default) and application-controlled versioning.
- If an item type has a version policy of versioned-never, its parts also have a versioning policy of versioned-never.
- If an item type T has a version-policy of versioned-always and an item I of item type T and you modify (by adding/deleting or updating a part) either its attributes or its parts collection , a new version of item I is created.
- Parts of documents, unlike documents themselves, do not have a maximum number of versions.
- You can obtain part-level versioning rules from the item type relation object for the part of interest (base, note, annotation etc.).

The example below shows how to get the versioning rules for a base part item type.

```
DKString itemTypeName="book";  //example item type
//Specify the part id for which we need versioning information
long partId = DK_ICM_PART_BASE;
DKItemTypeDefICM  * itemType;
//Retrieve the entity corresponding to the "book" item type
itemType = (DKItemTypeDefICM * )dsICM->datastoreDef()->retrieveEntity(itemTypeName);
//Retrieve the relation object for the specified part id
DKItemTypeRelationDefICM * itemTypeRelation =
  (DKItemTypeRelationDefICM*)itemType->retrieveItemTypeRelation(partId);
//Retrieve the version control policy for the specified part
int versionControlPolicy = itemTypeRelation->getVersionControl();
```

Working with transactions

Transactions allow Content Manager to maintain consistency between the library server and any adjoining resource manager. A transaction is a user-determined, recoverable, unit of work, that consists of one or more consecutive API calls made through a single connection to the library server. The sequence of consecutive DKDatastoreICM method calls are made either directly or indirectly, through the DDOs and XDOs.

The scope of a transaction and the amount of work within that transaction is by default the work performed by a single API method (implicit transaction). This type of transaction is recommended and is the best performing scope of a transaction. You can, however, change the scope of a unit of work, making it larger to include multiple method calls (explicit transaction), but using this type of transaction can introduce some performance overhead.

When a transaction ends, the entire transaction is either committed or rolled back. If it is committed, all of the Content Manager server changes made by API calls within the transaction are permanent. If a transaction is rolled back or fails, all the changes made within the transaction are reversed during rollback processing.

The commit and rollback of a transaction is done automatically in the case of implicit transactions. In the case where explicit transactions are in use, the transaction commit is controlled by the application, whereas a transaction rollback can be initiated by an application or automatically by the Content Manager system. The Content Manager system initiates a rollback when a severe error occurs or when it is necessary to resolve a deadlock between the library server and the database.

Within a transaction, uncommitted resource manager changes are not visible to the application that made the changes until the transaction is committed. For example, you make changes to a resource manager item and you store it. If you retrieve that item before the transaction is committed, the item will not reflect the changes that you just made. You will not see the updated item until the transaction is committed.

Concurrent or overlapping transactions through a single library server connection are not supported. To maintain concurrent transactions, you must make multiple connections between the library server and the database, or initiate multiple client processes or threads if you are working with a client application. Applications like IBM WebSphere ® Application Server handle processes, connections, and sessions.

The execute() and executeWithCallback() methods in DKDatastoreICM automatically create an additional connection to the database when invoked. The new database connection is then used to execute the query. Since queries use a separate database connection, they also have a separate transaction scope from other datastore operations. The connection to the database is closed (or returned to the pool, if pooling is enabled) when the DKResultSetCursor is closed.

Things to consider when designing transactions in your application

If a client node or library server fails before the transaction is committed, the database recovery function rolls back the transaction on the library server immediately. The resource manager changes made during the failure are undone immediately if the client node and resource manager are both active. If the client

node itself failed, you should put the resource manager through a cycle of the Asynchronous Recovery Utility in order to restore consistency between the RM and the LS. Before the utility runs, the servers still have data integrity. What is affected are operations on the in-progress items that had the failure, which will be rejected until the RM is recovered. Failure during in-progress update of an object prevents another update of that same object, until the first failure is reconciled.

If the resource manager fails, you should run the asynchronous recovery utility to remove inconsistencies. On S/390, the resource manager has native transaction capabilities, such as Object Access Method (OAM), which are used to recover more expediently.

Caution when using explicit transactions

For explicit transactions, where you control the transaction scope using DKDatastoreICM.startTransaction() and DKDatastoreICM.commit(), use caution when developing an application where you work with Content Manager Documents with parts and when performing DKLobICM create, retrieve, update, and delete (CRUD) operations. When performing these operations, you should perform CRUD operations as close as possible to the end of the transaction. You should also keep a transaction as short as possible, since a long transaction increases the potential for database locking problems.

Locking problems are most apparent when updating an item, and the application chooses to not commit the transaction immediately. As long as the transaction is not committed, the item that is being updated, is still visible to other applications. When another user attempts to access or view the item, that user is locked out until the update transaction is committed. The same problem (database locking) occurs when creating new items in a folder. If the folder is visible to another user, and that user attempts to retrieve the new item, the user is locked out until the transaction is committed. The amount if time prior to the transaction commit is the amount of time the user is locked out.

The best approach to avoiding database locking is to commit transactions often and to avoid long running transactions. If you must perform CRUD operations within a transaction, it is recommended that you perform these operations when it is understood that no one else will access the items being updated.

Using check-in and check-out in transactions

Content Manager supports check-out and check-in operations on items. The check-out operation is called to acquire a persistent write lock for items. When an item is checked out by a user, other users can not update it although they can still retrieve and view it. You need to call the check-out operation prior to updating or re-indexing an item, regardless of the transaction mode (implicit or explicit) that you use. When you are done with the item, call the check-in operation to release the persistent lock and make the item available for other users to update. After you create an item, you have the option to keep it in checked out state to prevent other users from changing it until you are completely done with the work. If you check-out (or check-in) an item using an explicit transaction, the checkout is undone if the transaction is rolled back. If you check-out an item using an implicit transaction, the checkout is committed. It is the application's responsibility to check the item back in, using checkin options or methods.

Processing transactions

The transaction scope can be controlled by a client API call, but it must be designed carefully. To group a set of API calls into a transaction, you must build it explicitly by completing the following steps:

1. Call the `startTransaction()` method of the `DKDatastoreICM` class. You work with the `DKDatastoreICM` methods to complete all the transaction steps.
2. Call all of the APIs that you want to include in the transaction in the order that you want them called.
3. Call the `commit` or `rollback` methods to end the transaction.

All of the API calls made between the startTransaction() and either commit() or rollback(), are treated as one transaction.

All APIs can be included in transactions, unless specifically noted. See the Online API Reference for details. Some administrative APIs cannot be included in explicit transactions. For example, the method to define or update item types.

Below is the list of class methods involved in Content Manager transactions in relation to item creation and update.:

DKDatastoreICM.startTransaction()
Starts an explicit transaction.

DKDatastoreICM.commit()
Commits transaction changes to the database.

DKDatastoreICM.rollback()
Rolls back or removes transaction changes from the database.

DKDatastoreICM.checkOut()
Acquires a persistent write lock on an item.

DKDatastoreICM.checkIn()
Releases a previously acquired persistent write lock.

DKDatastoreICM.add()
Creates a new item in the database.

DKDatastoreICM.updateObject
Updates an item. The item must be checked out prior to calling this method.

DKDatastoreICM.retrieveObject
Retrieves an item from the database.

DKDatastoreICM.deleteObject
Deletes an item from the database.

DKDatastoreICM.moveObject
Re-index an item. Moves an item from one item type to another item type. The item must be checked out prior to calling this method.

Routing a document through a process

Content Manager provides an integrated document routing service to help you route documents through a business process. The document routing APIs enable you to build new applications using document routing, or add document routing functionality into your existing applications. Document routing provides you with the following features:

- Synchronization of all items in a document routing process because document routing functions are included in Content Manager transactions.
- Presentation of only the work that the user has access to.
- Single audit trail that includes records for document creation, modification, and routing.
- Optional automatic routing of items. When an item is defined, it can be automatically added to an appropriate process.

Understanding document routing concepts

Before you can work with the document routing APIs, you must understand the document routing concepts and the data model. This information provides you with conceptual and data model information:

Item In the realm of document routing, folders and documents are items.

Work node

A step within a process where items wait for actions to be taken by end users or applications, or a process through which items move automatically. Work lists and collection points are the types of work nodes available. Every work node has a corresponding work list. A work node is an item with a root component and a child component, to implement work packages, that contains the following attributes:

- ID (an entry in the NLS keyword table containing name and description)
- Type (work list, collection point, decision point, split, join)
- Notify time or 0 (in hours)
- Overload limit
- Exits to be run on the server (DLL and function name)
  - Overload exit
  - entering work node exit
  - Leaving work node exit
  - Notify exit

The administrator defines the maximum time that a work package should be at a given work node. When that time is reached, the `notify` flag is set and the Notify exit is called.

Process

A series of steps, defined by an administrator, through which an item is routed. A *process* consists of a root component and a child component which define the route. The root component uses a system defined attribute for *process ID*. The *process ID* maps to a name and description in the NLS Keyword table.

The child component has the following attributes:

- **From** work node (null for the *start* node)
- **To** work Node (null for the“ end” node)
- Menu entry (ID in the NLS keyword table) for routing between **From** and **To**

Work list

One or more work nodes from which a user obtains a list of work packages or the next work package based on priority, or state (suspend or notify). Items are ordered first by priority (or date or another key) and second by work node within the list. Each work list has an ACL. A work

node may be in more than one work list. Using a work list allows an administrator, or an application, to dynamically change work assignments without contacting end users. A work list is an item which defines the selection criteria for a list of work and contains the following attributes:

- ID (an entry in the NLS keyword table containing name and description)
- List of work nodes (Atomics)
- Selection order (priority, last work date ascending/descending)
- Selection filter (suspended 0/1, notify 0/1)
- Number return (0=all, 1=system assigned, >1 for a limit)
- SQL statements that implement both the *count* and *retrieve* functions

You can define a work list to span all work lists (or a selected subset) to show work packages in *notify* or *suspend* state. A user authorized to the work list can see the count of waiting work packages, and can open the list and make decisions about each one, such as increasing the priority, forcing early resume, forcing exit from a collection point, changing user assignment, and so forth.

Work package

A set of information such as priority, state, resume time, process, and `ItemID` being routed. End users are not aware of the existence of a work package. A work package is used to associate the item being routed with a work node. A work package is a child component of a work node which associates a document or folder being routed with a work node within a process, and contains work specific information such as priority. Work package attributes include:

- Priority
- Reference attribute to item being routed
- Item ID of the process
- User ID of the process
- User ID of the user who last moved the work package
- Date/time when the work package was last moved
- Suspend state (0/1)
- Notify time (UNIX time in the future to switch to notify state, 0 if not used)
- Resume time (UNIX time, 0 if not suspended for time)
- Resume list
  - Child component with one row per required item type
  - Contain item type (or 0 for *any*) and number of that type required
  - Setting number required to 0 allows continuation when any item of any required type arrives

Collection point

A collection point is created by the administrator and is associated with a work node. A collection point allows automatic suspension of a folder when it arrives at a work node, unless all the required items have been added to the folder. When the required items arrive in the folder, the work package advances to the next step in the process. End users do not access work packages at a collection point, other than for exception processing such as “forcing” continuation even though required documents have not been received.

A collection resume list is defined in a collection point by the administrator. When the collection resume list is satisfied, the work package on the collection point advances automatically to the next work node. If the collection point has more than one branch, the first branch is selected. The suspend flag is not set by using the collection resume list. The notify flag can be set in a collection point, if the notification time is defined by the administrator. Just like a regular work node, a user can apply continue, terminate, suspend, and resume operations on a collection point.

The notify flag is based on the scheduled time in the DOCROUTINGUPDATE field in the system control table. When the scheduled time is reached, the system updates the field by adding the time that is indicated in the DOCROUTINGFREQ field. The default value for the DOCROUTINGFREQ field is 10 minutes.

Understanding the document routing process

Document routing consists of processes, work nodes, collection points, work lists, and work packages. The system administrator creates the work nodes, collection points, processes, and work lists through the system administration client. A process consists of work nodes and collection points. Each work node and collection point in the process is a separate step in the process. You can create a process that branches out in several directions. The user determines which branch the work node goes to next. The user can choose from a list of possible selections that the system administrator defines. You can define a server exit when you define a work node. You can define server exits for entering a work node, leaving a work node, and to notify the user when the overload limit is reached. When a process is started, a work package is created. The work package is the routing element and contains the attributes of the work. The attributes of the work package consist of the item PID, priority, owner, and so forth.

Collection points are work nodes with additional function. A work package at a collection point node continues to the next work node in the process once the specified number of items of a specified item type exist in the specified folder. Work lists define the work packages assigned to a user. You can have one or more work lists. Each work list can include one or more work nodes. You can specify the order of the work packages in the work list by priority, time, and owner. You can also define the the order of work nodes in the work list.

When you retrieve work lists, you can filter the results to include or exclude suspended work. Suspended work are work packages that are in *notify* state. *Notify* state is when work packages have been at the node for longer than the time specified by the administrator. Remember that a work node can be in more than one work list. The number of packages returned in a work list is defined by the system administrator.

Those basic operations you can perform using document routing include:

- Start a process
- End a process
- Continue a process
- Suspend a process
- Resume a process
- Get work from a work list
- Get the next item from a work list
- Define, update, and delete a process

- Define, update, and delete a work node
- Define, update, and delete a work list

Setting up a document routing process

There are nine APIs that you can use to implement document routing functionality into your application. You can find the details about these APIs and methods in the *online API reference*. The nine document routing APIs include:

DKDocRoutingServiceICM
This class provides the methods for managing a process: start, terminate, continue, suspend, and resume.

DKDocRoutingServiceMgmtICM
This class provides the methods to manage the helper classes: DKProcessICM, DKWorkNodeICM, and DKWorkListICM. You can access the DKDocRoutingServiceMgmtICM object from the DKDocRoutingServiceICM object.

DKProcessICM
This class represents a process in the library server.

DKWorkNodeICM
This class represents a work node in the library server.

DKWorkListICM
This class represents a work list in the library server.

DKRouteListEntryICM
This class defines the route that a process can take (from, to). A process object (that is, DKProcessICM) contains a collection of route entry objects (that is, DKRouteListEntryICM).

DKCollectionResumeListEntryICM
This class represents an entry in the resume list for the work nodes. A work node can contain a collection of DKCollectionResumeListEntryICM objects.

DKWorkPackageICM
This class represents a work package in the library server. When a process is started, a work package is created.

DKResumeListEntryICM
This class represents a resume list. A work package can contain a collection of resume lists.

Creating document routing objects

The example below shows you how to create document routing objects.

```
char * database = "icmnlsdb";
char * userName ="icmadmin";
char * password ="password";

// Create new datastore object.
DKDatastoreICM* dsICM = new DKDatastoreICM();
//Connect to datastore
dsICM->connect(database,userName,password,"");
//The DKDocRoutingServiceMgmtICM object is a helper class that
//provides methods to manage DKProcessICM, DKWorkNodeICM,
//and DKWorkListICM and the meta-data required to define these objects
DKDocRoutingServiceMgmtICM* routingMgmt = new
   DKDocRoutingServiceMgmtICM(dsICM);

//The DKDocRoutingServiceICM class provides the core routing services
```

```
// such as starting, terminating, continuing, suspending, and resuming
//a process.
DKDocRoutingServiceICM* routingService = new
   DKDocRoutingServiceICM(dsICM);
```

Defining a new regular work node

A work node is a step in a document routing process definition. A user appliation can exit a work node at any time. An application is responsible for validating its own exit criteria.

```
// Create new Work Node Object.
 DKWorkNodeICM* workNode1 = new DKWorkNodeICM();
// Choose a Name that is 15 characters or less length
 workNode1->setName("ValidateCreditCard");
// Choose a Description for more information than the name.
workNode1->setDescription("Buyer's credit card is validated
    with the credit card agency");

// Sets the value of the maximum time that an item can spend at
//this work node (in minutes).
workNode1->setTimeLimit(100);

// Specify max number of work packages that can be at this node
// at any given time
workNode1->setOverloadLimit(200);

// Set the type to be a regular work node
workNode1->setType(0);

//Add the new work node definition to the document routing
//management object
routingMgmt->add(workNode1);

// Free memory. This object will no longer be needed.
delete(workNode1);
```

Defining a new collection point

A collection point is a work node with system-defined exit criteria, specifically applicable to routing folders. A set of requirements can be specified that must be met before the process can resume or advance past this point.

```
// Create a new Work Node Object.This will be the collection point
DKWorkNodeICM* collectionPoint = new DKWorkNodeICM();
// Choose a Name with 15 characters or less.
collectionPoint->setName("GatherOrderDetails");
// Choose a Description for more information than the name.
collectionPoint->setDescription("Gather all the information related to the
  order, shipping mechanism and shipping address");
// Sets the value of the maximum time that an item can spend at this
//work node (in minutes).
 collectionPoint->setTimeLimit(100);
// Specify max number of work packages that can be at this node.
collectionPoint->setOverloadLimit(200);

// Set the type of node to be a collection point.
collectionPoint->setType(1);
// Create the "Resume" List, which is the list of document types
//that the process must  wait for before moving on to the next node.
//A list will be created to  hold "resume entries" which are descriptions
//of requirements that must be met before the process may move on.

// Create a List / Collection to hold all Resume Entries.
dkCollection* resumeList = new DKSequentialCollection();

// Create as many requirements, or "Resume List Entries", which specify
//what Item Types it must wait for. The process cannot pass this
//collectionpoint unless the specified number of Item (DDO) of the
```

```
//specified Item Type reaches this collection point.
 DKCollectionResumeListEntryICM* resumeRequirement = new
    DKCollectionResumeListEntryICM();
// Set the Item Type Name Folder Item that is being routed.
resumeRequirement->setFolderItemTypeName("book");

// Make the collection wait for an Item of the specified Item Type to
//be added to the folder before proceeding.
resumeRequirement->setRequiredItemTypeName("AnItemType");

//Specify the number of Items of the specified Item Type that
//it must wait for.
resumeRequirement->setQuantityNeeded(1);

// Add the requirement (Entry) to the List of Requirements (Resume List).
resumeList->addElement(resumeRequirement);
resumeRequirement = new DKCollectionResumeListEntryICM();
resumeRequirement->setFolderItemTypeName("book");
resumeRequirement->setRequiredItemTypeName("AnotherItemType");
resumeRequirement->setQuantityNeeded(1);
resumeList->addElement(resumeRequirement);

// When all requirements (resume list entries) have been added to
//the list of requirements (resume list), set the resume list in the
//collection point.
collectionPoint->setCollectionResumeList(resumeList);
// Add the new collection point definition to the document routing
// managment object
routingMgmt->add(collectionPoint);

//Free the memory associated with this collection point
delete(collectionPoint);
```

Defining a work list

A work list consists of one or more work nodes from which a user obtains a list of work packages or the ″next″ work package. A work node can be in more than one work list. Using a work list allows an administrator or a user application to dynamically change work assignments without contacting end users.

```
// Create a new work list.

 DKWorkListICM* workList = new DKWorkListICM();
//Choose a name of 15 characters or less.
workList->setName("ValidateWorkList");

workList->setDescription("Work List Covering the credit card
   validation work node.");
//Specify that work packages returned by the work list will be
//sorted by time
workList->setSelectionOrder(DK_ICM_DR_SELECTION_ORDER_TIME);

//Specify that the work packages returned will be the one that are not
//in the suspend state
workList->setSelectionFilterOnSuspend(DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that work packages returned are not the ones in the notify state
workList->setSelectionFilterOnNotify(DK_ICM_DR_SELECTION_FILTER_NO);

//Specify that at most 100 work packages should be listed in this work list
workList->setMaxResult(100);

DKString* wnNames = new DKString[1];
wnNames[0] = "ValidateCreditCard";
//Add the work node to the work list
workList->setWorkNodeNames(wnNames,1);

//Add the new work list definition to the document routing management
```

```
//object
routingMgmt->add(workList);

//Free the memory associated with this work list
delete(workList);
```

Defining a new process and associated route

The example below shows you how to define a new process and associated route.

```
//A document routing process is the defined routes that a work
//package being routed will follow. Multiple routing processes may
//re-use the same nodes and multiple routes between nodes  may be used.

// Create a new Process Definition
DKProcessICM* process = new DKProcessICM();
process->setName("Buy_Book");
process->setDescription("Purchase a book online");

// Define all possible Routes.

// Create a list of all possible routes between nodes.
dkCollection* routes = new DKSequentialCollection();

// Connect the Work Nodes using Route List Entries.  A simple route
// between two work nodes is specified by associating a 'From' work node
// and a 'To' work node.

// A Route List Entry simply connects two nodes with an implied direction.
 // Multiple routes may exist between nodes.  A specific route may be selected
 // by a user-defined "selection" keyword.  Examples might be "Continue", "Go",
 // "Accept", "Reject", "Complete", etc.

// Create a new connection between two nodes.
DKRouteListEntryICM* nodeRoute = new DKRouteListEntryICM();

// Every process must start with the start node.
nodeRoute->setFrom(DK_ICM_DR_START_NODE);
nodeRoute->setTo("ValidateCreditCard");
// Choose any user-defined name for an action that will make the transition
// from the 1st node to the 2nd
nodeRoute->setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("ValidateCreditCard");
nodeRoute->setTo("GatherShippingDetails");

// Choose any user-defined name for an action that will make the transition
// take place.
nodeRoute->setSelection("Continue");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);

nodeRoute = new DKRouteListEntryICM();
nodeRoute->setFrom("GatherOrderDetails");
nodeRoute->setTo(DK_ICM_DR_END_NODE);

// Choose any user-defined name for an action that will make the transition
// take place.
nodeRoute->setSelection("Complete");

// Add the individual route to the collection of all possible routes.
routes->addElement(nodeRoute);
```

```
// Set the route in the process.
process->setRoute(routes);

// Add the process to the routing Management.
 routingMgmt->add(process);

 delete(process);
```

Starting a document routing process

The following example shows how to start a document routing service:

Example:

```
//First create a document or folder that will be routed.
//An item type of name "book"must be pre-defined before a DDO of
// that name can be created.
DKDDO* ddoFolder = dsICM->createDDO("book", DK_CM_FOLDER);

// Save the created folder to the persistent datastore.
ddoFolder->add();

//Set the priority for this document routing process
int Priority = 1;

//Create the core document routing service object.
DKDocRoutingServiceICM* routingService = new DKDocRoutingServiceICM(dsICM);

//Start a process with the name "Buy_Book" (which must be pre-defined.
//The PID string of the work package that will be routed is returned by this call.
DKString workPackagePidStr = routingService->startProcess("Buy_Book", ((DKPidICM*)
   ddoFolder->getPidObject())->pidString(), Priority, "icmadmin");
```

Terminating a process

A process can be explicitly terminated before it reaches the end node. When a process is explicitly terminated, the work package being routed will be removed from the system.

Example:

```
//Terminating the process requires the PID string of the work
//package being routed by this process instance
routingService->terminateProcess(workPackagePidStr);
```

Continuing a process

The continueProcess method routes the item referenced by the item persistent identifier (PID) in the specified work package from the current work node to the next work node that is determined by the selection name specified in the method call. In the code snippet below, the name of the selection that will cause the transition from the current work node to the next work node is ″Continue″. The work package PID string of the current work package is specified in the method call.

```
char * userName = "icmadmin";

//This method call returns the PID string of the new work package
workPackagePidStr = routingService->continueProcess(workPackagePidStr,
   "Continue", userName);
```

Suspending a process

An instance of a document routing process can be suspended for a period of time (in minutes) or pending a set of requirements that must be met before it is resumed. This does not relate to processes and threads in a programming environment. The thread or process in the C++ runtime environment will not be stopped.

Example:

```
dkCollection * requirements = new DKSequentialCollection();
//If no requirements are to be provided and the process is only to be
//suspended for a fixed period of time, the user can also pass in a
//NULL collection to this method.
//dkCollection * requirements = NULL;

//Process will be suspended for 2 minutes.
routingService->suspendProcess(workPackagePidStr, 2, requirements);
delete(requirements);
```

Resuming a process

A suspended process (process in the suspended state) can be resumed explicitly (or implictly after expiration of a specified period of time or after some defined requirements have been met). This takes the process out of the suspended state, resuming it to normal operation. The resumeProcess method will reset the suspend flag of the specified work package to false before the suspension reaches the specified duration and the resume list is satisfied. No routing or checkout of the associated work item will be performed.

Example:

```
routingService->resumeProcess(workPackagePidStr);
```

Listing work package persistent identifier strings in a work list

A work list contains all a list of work packages. The following code sample shows howto list the PID strings for all the work packages in the specified work list.

```
long arraySize = -1;
// The userName specified here must be the process owner
char * userName = "icmadmin";
//At the end of this method call, the "arraySize" variable will contain the count
//of the work packages in the work list
DKString* workPackagePIDs = routingService->listWorkPackagePidStrings
   ("ValidateWorkList",userName,arraySize);

// Print Work Package PIDs
cout << "Work Packages in Work List:  (" << arraySize << ")" << endl;
for(int i=0; i< arraySize; i++)
{
    cout << "     - PID:  " << workPackagePIDs[i] << endl;
}
delete[] workPackagePIDs;
```

Retrieving work package information

When an instance of a document routing process is in progress, a work package is the vehicle through which an item (instance of an item type) moves through the routing process. A work package contains all the necessary information about the process and the item that it is transporting. The work package is the object that an application will use and manipulate as required.

```
//The retrieveWorkPackage method will return the DKWorkPackageICM
// object referenced by the specified work package PID (wpPidString)
DKWorkPackageICM* workpackage = NULL;
//Use an established document routing service

workpackage= routingService->retrieveWorkPackage((const char*)wpPidString);
cout << "Work Package pid = "" << workpackage->getPidString()
    << "\"" <<  endl;
cout << " Item PID String = \"" << workpackage->getItemPidString()
    << "\"" << endl;
cout << "Priority = \"" << workpackage->getPriority() << "\"" << endl;
cout << "Process Name = \"" << workpackage->getProcessName() << "\"" << endl;
cout << "Work Node Name = \"" << workpackage->getWorkNodeName()
     << "\"" << endl;
```

Listing document routing processes

The listProcessNames method lists all process names in the library server, and the listProcesses method returns a collection of DKProcessICM objects each representing a process in the library server.

```
// Obtain the document routing management object.
DKDocRoutingServiceMgmtICM* routingMgmt = new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the list of all running document routing processes
dkCollection* processes = routingMgmt->listProcesses();
if (processes && (processes->cardinality()>0))
{
   cout << "Running Processes: (" << processes->cardinality() << ")" << endl;
   dkIterator* iter = processes->createIterator();
   while(iter->more())
   {
       DKProcessICM* proc = (DKProcessICM*) iter->next()->value();
       cout << " - " << proc->getName() << ": " << proc->getDescription() << endl;
       delete(proc);
   }
   delete(iter);
   delete(processes);
}
delete(routingMgmt);
```

Getting the next work package from a work list

The getNextWorkPackage() method returns a DKWorkPackageICM object from the specified work list for the given owner. The behavior of the getNextWorkPackage() method depends on the work list setting in the system administration. Table 46 shows the various iterations for the work list setting.

Table 46. Work list setting in system administration

Setting	Owner field is empty	Owner = logged on user ID	owner = a user ID other than the logged on user ID
Filter on owner is not selected	All work packages in the work list are used	Work packages of logged on user ID in the work are used	Work packages of the specified user ID in the work list will be used, if the ICM_PRIV_ITEM _GET_ASGN_WORK privilege is set
Filter on owner is selected	Work packages of the logged on user ID in the work list are used	Work packages of the logged on user ID in the work list are used	work packages of the specified user ID in the work list will be used to perform the API, if the ICM_PRIV_ITEM _GET_ASGN_WORK privilege is set

Getting the count of work packages in a work list

The getCount method returns the number of work packages in the specified work list for the given owner. Below is how the system administration settings affect the counts returned:

Table 47. System administration settings for work package count

Setting	Owner field is empty	Owner = logged on user ID	owner = a user ID other than the logged on user ID
Filter on owner is not selected	Returns the number of all work packages in the work list	Returns the number of work packages of the logged on user ID in the work list	Returns the count of work packages of the specified user ID in the work list if the ICM_PRIV_ITEM _GET_ASGN_WORK privilege is set
Filter on owner is selected	Returns the number of work packages of the logged on user ID in the work list	Returns the number of work packages of the logged on user ID in the work list	Returns the count of work packages of the specified user ID in the work list if the ICM_PRIV_ITEM_ GET_ASGN_WORK privilege is set

A work list contains one or more work packages at any given point. The sample code below shows how to retrieve the work package count for the specified work list.

Example:

```
int count = 0;
// Use an established document routing service dr.
// Returns the count of work packages for the "ValidateWorkList" work list
char * userName= "icmadmin";
count =  routingService->getCount("ValidateWorkList",userName);
```

Listing work nodes

The listWorkNodeNames method lists all work node names in the library server, and the listWorkNodes method returns a collection of DKWorkNodeICM objects representing a work node in the library server.

```
// Obtain the document routing  management object.
DKDocRoutingServiceMgmtICM* routingMgmt = new DKDocRoutingServiceMgmtICM(dsICM);

// Obtain the collection containing all the  work nodes in the system.

dkCollection* workNodes = routingMgmt->listWorkNodes();
if (workNodes &&  ( workNodes->cardinality()>0) )
{
    cout << "Work Nodes in System: (" << workNodes->cardinality() << ")" << endl;
    dkIterator* iter = workNodes->createIterator();
    while(iter->more())
    {
        DKWorkNodeICM* workNode = (DKWorkNodeICM*) iter->next()->value();
        if(workNode->getType()==0)
        {
              cout << " Normal Node - " << workNode->getName() << ": " <<
              workNode->getDescription() << endl;
        }
        else
        {
             cout << " Collection Pt - " << workNode->getName() << ": " <<
             workNode->getDescription() << endl;
        }
        delete(workNode);
  }
```

```
  delete(iter);
  delete(workNodes);
}
delete(routingMgmt);
```

Ad hoc routing

Below is an ad hoc routing example procedure. In the example, the system administration client is used to set up the work nodes, processes, and work lists.

1. Create two work nodes, N1 and N2 for example.
2. Create two one-node processes, P1 and P2 such that P1 has one work node, N1 and P2 has one work node, N2.

   P1 looks like this:

From:	Action:	To:
START	Continue	N1
N1	Continue	END

   P2 looks like this:

From:	Action:	To:
START	Continue	N2
N2	Continue	END

3. Create two work lists, WL1 and WL2 such that WL1 has one work node, N1 and WL2 has one work node, N2.

At run time, complete the following steps to implement ad-hoc routing:

1. Start process P1 with a document PID (Example, ABC). A work package, WP1, is created. The work list WL1 displays the work package WP1 at work node N1.
2. To move the document ABC from process P1 to process P2, terminate work package WP1 and start process P2 with the same document (ABC). Work package WP2 is created.

The work list WL2 shows the work package WP2 at work node N2.

Document routing example queries

This sections contains example queries. For more information about writing queries, see “Understanding the query language” on page 423.

Example 1

Finds car documents, and returns only the associated workpackages that are active.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@SUSPENDFLAG =
0]
```

Example 2

Finds car documents and returns the associated workpackages that are in the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]
```

Example 3

Finds car documents where the name is ″Honda″, and returns the associated workpackages that are in the ″AccidentInvesigation″ process.

```
/Car[@Name = "Honda" AND @VERSIONID = latest-version(.) AND
@SEMANTICTYPE = 1]/REFERENCEDBY/@REFERENCER =>
WORKPACKAGE[@PROCESSITEMID = /ROUTINGPROCESS[@PROCESSNAME =
"AccidentInvestigation"]/@ITEMID]
```

Example 4

Finds car documents and returns the associated workpackages that are in the ″UnderReview″ step of the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]/@ITEMID
AND ../@WORKNODENAME =  "UnderReview"]
```

Example 5

Finds car documents and returns only the suspended workpackages that are in the ″UnderReview″ step of the ″AccidentInvestigation″ process.

```
/Car[@VERSIONID = latest-version(.) AND @SEMANTICTYPE =
1]/REFERENCEDBY/@REFERENCER => WORKPACKAGE[@PROCESSITEMID =
/ROUTINGPROCESS[@PROCESSNAME = "AccidentInvestigation"]
/@ITEMID AND ../@WORKNODENAME = "UnderReview" AND
@SUSPENDFLAG = 1]
```

Granting privileges for document routing

In order for a user to perform document routing operations, the user must have the appropriate privileges. The privileges associated with document routing are listed in the following table. The general privileges for items are applicable to processes, work nodes, and work lists.

Table 48. Document routing privileges

Privilege	Description	Related API
ICM_PRIV_ITEM _UPDATE_WORK	Used to see if the user is authorized to do the following for a work package: set the priority set the owner set the resume list set the duration for suspension	suspendProcess resumeProcess setWorkPackagePriority setWorkPackageOwner
ICM_PRIV_ITEM _ROUTE_START	Used to see if the user is authorized to start a process.	startProcess
ICM_PRIV_ITEM _ROUTE_END	Used to see if the user is authorized to terminate a process.	terminateProcess
ICM_PRIV_ITEM _GET_WORKLIST	Used to see if the user is authorized to get the count or work packages from a work list.	getCount listWorkPackagePidStrings
ICM_PRIV_ITEM _GET_WORK	Used to see if the user is authorized to get a work package.	getNextWorkPackagePidString getNextWorkPackage checkOutItemInWorkPackage retrieveWorkPackage
ICM_PRIV_ITEM _GET_ASGN_WORK	Used to see if the user is authorized to get a work package that is owned by a different a different user.	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings

Table 48. Document routing privileges (continued)

Privilege	Description	Related API
ICM_PRIV_ITEM _ROUTE	Used to see if the user is authorized to route a work package.	continueProcess

Working with access control lists for document routing

When an ACL is defined for a document routing entity such as a process, work node, and work list, the operations allowed on the entity are impacted. The effect of ACLs on the CM document routing entity and its associated privileges is listed in the following table.

Table 49. Access control lists and document routing

Privilege	Description	Related API
Process	startProcess	ICM_PRIV_ITEM_ROUTE_START
Work node	continueProcess suspendProcess resumeProcessterminateProcess setWorkPackagePriority setWorkPackageOwner	ICM_PRIV_ITEM_ROUTE ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_ROUTE_END ICM_PRIV_ITEM_UPDATE_WORK ICM_PRIV_ITEM_UPDATE_WORK
Work list	getNextWorkPackagePidString getNextWorkPackage getCount listWorkPackagePidStrings checkOutItemInWorkPackage	ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORK ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORKLIST ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORKLIST ICM_PRIV_ITEM_GET_ASGN_WORK ICM_PRIV_ITEM_GET_WORK

Document routing constants

You define document routing constants in DKConstantICM. Following is the list of document routing constants:

```
public final static int DK_ICM_DR_SELECTION_FILTER_NO = 0;
public final static int DK_ICM_DR_SELECTION_FILTER_YES = 1;
public final static int DK_ICM_DR_SELECTION_FILTER_EITHER = 2;
public final static int DK_ICM_DR_SELECTION_ORDER_PRIORITY = 0;
public final static int DK_ICM_DR_SELECTION_ORDER_TIME = 1;
public final static int DK_ICM_DR_MAX_RESULT_ALL = 0;
public final static String DK_ICM_DR_START_NODE = ″START″;
public final static String DK_ICM_DR_END_NODE = ″END″;
```

Chapter 14. Working with other content servers in C++

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 50.

Table 50. Server type and class name terminology

Server type	Class name
Content Manager Version 8.1	DKDatastoreICM
Content Manager for AS/400 (VisualInfo for AS/400)	DKDatastoreV4
ImagePlus for OS/390	DKDatastoreIP
Domino.Doc	DKDatastoreDD
Extended Search	DKDatastoreDES
OnDemand	DKDatastoreOD
relational database	DKDatastoreDB2, DKDatastoreODBC (for C++ on Windows), DKDatastoreDJ
earlier Content Manager	DKDatastoreDL

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` function, so it can be considered a subclass of query manager.

dkDatastoreDef
Defines functions 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 functions to:

- Create and delete an entity.
- Access entity information.
- Create and delete attributes.

In this class, all functions 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 functions to overwrite the exceptions, for example. Examples of concrete classes for the dkEntityDef interface are:

- DKIndexClassDefDL
- DKEntityDefIP

dkAttrDef

Defines functions to access attribute information and to create and delete attributes. Examples of concrete classes for dkAttrDef are:

- DKAttributeDefDL
- DKAttrDefV4

dkServerDef

Defines functions 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` functions, 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 538.

Working with earlier 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 earlier 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 earlier Content Manager content

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 274 and “Getting the DDKDO and attribute properties” on page 274 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 273.

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” on page 470, “Retrieving a document or folder” on page 474, and “Creating and using the DKPARTS attribute” on page 288.

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 474, and "Creating and using the DKFOLDER attribute" on page 290.

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 with dsDL
DKDDO* ddo = new DKDDO(&dsDL,pid);

// step 2.a: add attributes according to index-class GRANDPA
DKAny any;
DKBoolean yes = TRUE;
DKBoolean no = FALSE;
// add a new attribute named "Title"
unsigned short data_id = cddo->addData("Title");
// add type property VSTRING
any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
// add nullable property: non-nullable
any = no;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// add a new attribute named "Subject"
data_id = cddo->addData("Subject");

any = DK_VSTRING;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_TYPE, any);
any = yes;
cddo->addDataProperty(data_id, DK_CM_PROPERTY_NULLABLE , any);

// add some more attributes as necessary
//  ...

// step 2.b: add DKPARTS attribute
// create a new XDO blob
DKParts* parts = new DKParts;
```

```
DKBlobDL* blob = new DKBlobDL(&dsDL);

DKPidXDODL pidXDO;                          // create PID for this XDO object

pidXDO.setPartId(5);                        // set part number to 5
blob->setPid(&pidXDO);                     // set the PID for the XDO blob
blob->setContentClass(DK_CC_GIF);          // set content class type GIF
blob->setRepType(DK_REP_NULL);             // set rep type for the part
blob->setContentFromClientFile("choice.gif"); // set the blob's content

DKAny any = (dkDataObjectBase*) blob;
parts->addElement(any);               // add the blob to the parts collection

...                                          // create and add some more blobs
...                                          // to the collection as necessary

// 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 function 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 function, 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 function. Updating a folder DDO requires similar steps: you update the attribute values, or add or remove elements from DKFolder, then call the update function.

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 function in DKParts actually deletes the part from the content server.

Differences between update, add, and remove on a document DDO: DKParts addMember and removeMember functions provide conveniences for adding and removing a part in the collection and the content server. Compared to the update function in a document DDO, the addMember and removeMember are faster.

The update function 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 functions, 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` function 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 functions provide conveniences for adding and removing a document or folder the collection and the content server. Compared with the `update` function in a folder DDO, `add` and `remove` functions are faster.

The `update` function on a DDO updates all of the attributes in the DDO, including DKFolder and all of its members, whereas add and remove member functions 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` function 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 470 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 475 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 288.

```
// 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 290.

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 functions. You can query this content server. Refer to “Loading data to be indexed by Text Search Engine” on page 481 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)"
```

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. DKDastastoreTS does not have a property item type as a DKDatastoreDL object does. The format of its ID is also different. A

DDO resulting from a text query corresponds to a text part inside an item. It 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 functions for connecting and a function 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 function 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/dl` directory.

The following example shows the second connection function using the text search server with the hostname `apollo`, port number `7502`, and TCP/IP communication type `DK_CTYP_TCPIP`:

```
dsTS.connect("apollo","7502",DK_CTYP_TCPIP);
```

The following example shows the first connection function using the text search server hostname `apollo`, port number `7502`, communication type `T` (TCP/IP):

```
dsTS.connect("apollo","","", "PORT=7502;COMMTYPE=T");
```

The following example shows the first connection function using the text search server name TM, library server LIBSRVRN, user ID FRNADMIN, and password PASSWORD:

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

You can use the last parameter LIBACCESS, also called the connect string, to pass a sequence of parameters.

Tip: To prevent the Text Search Engine connection from timing out, connect to Text Search Engine, run your queries, and immediately disconnect. Do not leave the connection open.

Getting and setting text search options

Text search provides some options that you can set or get using its functions. See the online API 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.

Listing servers

The DKDatastoreTS object provides a function to list the text search servers that it can connect to. The following example shows how to retrieve the list of servers.

```
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 functions for listing the schema. For text search, these are text search indexes. The following example shows how to retrieve the index list.

The list of indexes is returned in a DKSequentialCollection object of DKIndexTS objects. After you get a DKIndexTS object, you can retrieve information about the index, such as its name and library ID, which you can use to form a query.

```
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 295 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 client 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, user ID, password
    dsDL.connect("LIBSRVN","FRNADMIN","PASSWORD");

    cout << "datastore connected" << endl;

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

    //--- 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 276.

After the data is loaded into Content Manager, use the `wakeUpService` function 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/dl 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("SAMPCORPMOD");
docSection->setName("SAMPCORPTITLE");
docSection->setTag("TITLE");
docModel->addDocSection(docSection);
docSection2->setName("SAMPCORPBODY");
docSection2->setTag("BODY");
docModel->addDocSection(docSection2);

dsTS.connect("TMMUF","","","");

dsDef = (DKDatastoreDefTS*)dsTS.datastoreDef();
dsAdmin = (DKDatastoreAdminTS*)dsDef->datastoreAdmin();

// create doc model
dsAdmin->createDocModel("",docModel);

// delete document model & sections
delete docModel;

dsTS.disconnect();
```

The complete sample application from which this example was taken (TCreateDocModelTS.cpp) and (TCreateStructDocIndexTS.cpp) are available in the Cmbroot/Samples/cpp/dl directory.

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/dl 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/dl 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/dl 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 58 on page 487 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.

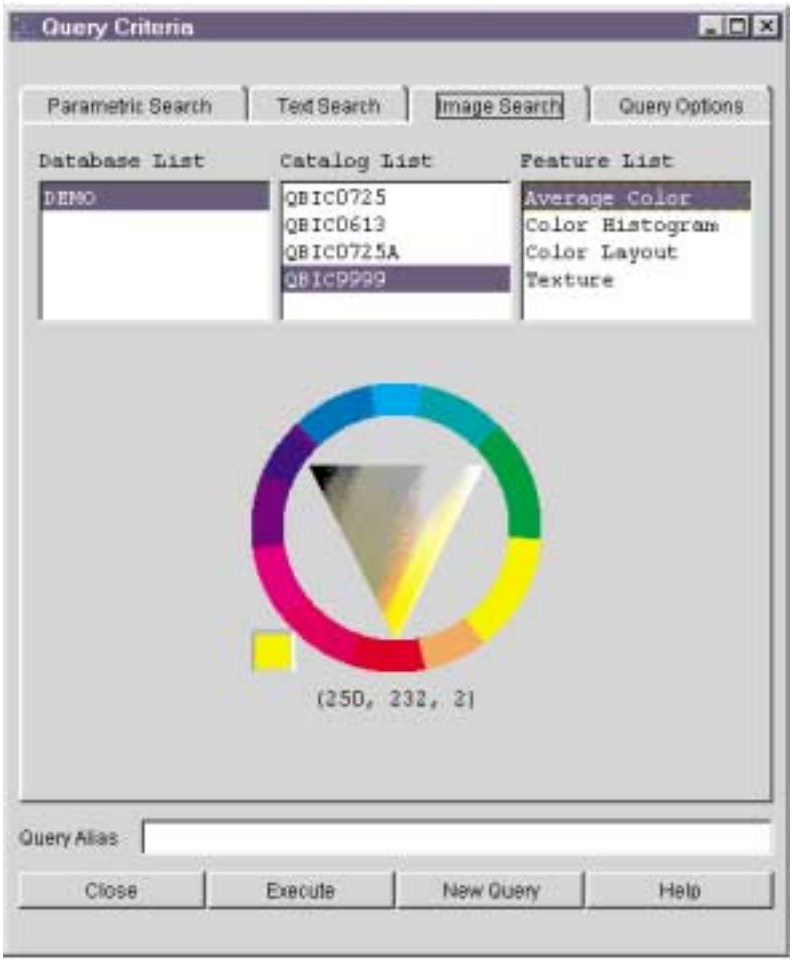

Figure 58. Image search sample client application

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 59 on page 488 shows an example of an image search server.

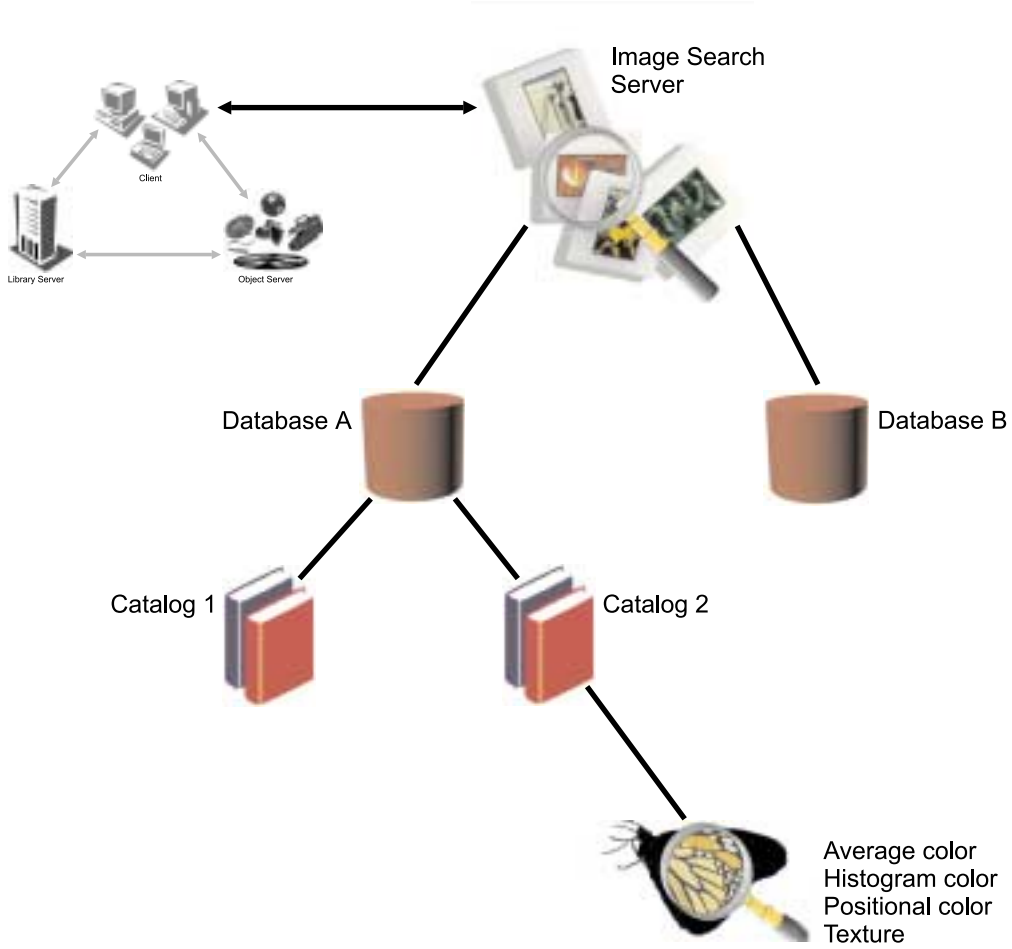

Figure 59. An image search server in an earlier Content Manager system. The image search server communicates with the other Content Manager components through the clients.

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 60 on page 490 shows an example of the image search client search and retrieve process.

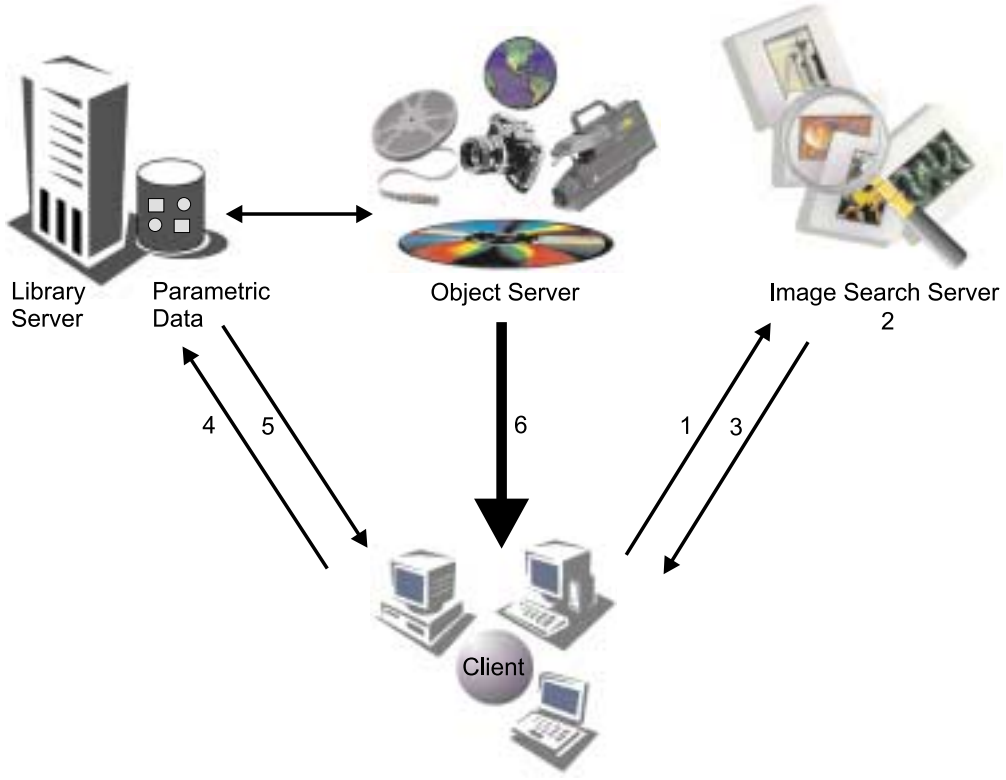

Figure 60. How image search clients search and retrieve images

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.

An image query is a character string that specifies the search criteria. The search criteria consist of:

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 51 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 `and max_results` to your query. `max_results` consists of the keyword `max`, an equal sign (=), and an integer greater than 0. Table 51 describes Image search query feature names and values.

Table 51. Image search query valid feature names and values

Feature name	Values
QbColorFeatureClass or QbColor	**color = < rgbValue , rgbValue , rgbValue >** where rgbValue is an integer from 0 to 255. **file = < fileLocation , " fileName " >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.
QbColorHistogramFeatureClass or QbHistogram	**histogram = < histList >** where histList consists of one or more `histClause` separated by a comma (,). A `histClause` is specified as `( histValue, rgbValue , rgbValue , rgbValue )`, where `histValue` is an integer from 1 to 100 (a percentage value), and `rgbValue` is an integer from 0 to 255. **file = < fileLocation , " fileName " >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

Table 51. Image search query valid feature names and values (continued)

Feature name	Values
QbDrawFeatureClass or QbDraw	**description = < ″ descString ″ >** where descString is a special encoded string describing a picker file. Format of the description string: 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. **file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.
QbTextureFeatureClass or QbTexture	**file = < fileLocation , ″ fileName ″ >** where fileLocation is either `server` or `client`, fileName is the complete file path specified in the format appropriate for the system on which the file resides.

Query examples:

1. Search for average color red:

   ```
   QbColorFeatureClass color=<255,0,0>
   ```

2. Search using a histogram of three colors, 10% red, 50% blue, and 40% green:

   ```
   QbColorHistogramFeatureClass histogram=
   <(10, 255, 0, 0) (50, 0, 255, 0), (40, 0, 0, 255)>
   ```

3. Search using an average color and a texture value. The texture value is provided by an image in a file on the client. The weight of the texture is twice that of the average color:

   ```
   QbColorFeatureClass color=
   <50, 50, 50> and QbTextureFeatureClass file=<client, "\patterns\pattern1.gif">
    weight=2.0
   ```

4. Search for the described color layout:

   ```
   QbDrawFeatureClass description=<"D100,50:R0,0,50,50,255,0,0">
   ```

5. Search for average color red and limiting the returned matches to five:

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

Establishing a connection in QBIC

Image search provides functions for connecting and disconnecting to the content server. The following example shows how to connect to an image search server named QBICSRV using the user ID QBICUSER and the password PASSWORD.

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
...                       // do some work
dsQBIC.disconnect();
```

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

The image search connection allows an application to connect to an image search server.

After connecting, your program can use functions that access the image search server, except for the functions that are not related to image search catalogs, such as listDatabases. An openCatalog function is required to open a catalog for processing. A closeCatalog function is called after processing is done. The following example shows how to connect, open a catalog, close the catalog, and disconnect.

```
DKDatastoreQBIC dsQBIC = new DKDatastoreQBIC();
dsQBIC.connect("QBICSRV", "QBICUSER", "PASSWORD", "");
dsQBIC.openCatalog("DEMO", "QBIC0725");
...                          // do some work
dsQBIC.closeCatalog();
dsQBIC.disconnect();
```

Listing image search servers

The image search server provides a function for listing the image search servers that it can connect to. The following example shows how to retrieve in a DKSequentialCollection object the list of servers that contain DKServerInfoQBIC objects. After you get a DKServerInfoQBIC object, you can retrieve the server name, the host name, and the port number.

```
DKDatastoreQBIC dsQBIC;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKServerDefQBIC *pSV = 0;
DKString strServerName;
DKAny a;
long i = 0;
cout << "list servers" << endl;
a = dsQBIC.listDataSources();
pCol = (DKSequentialCollection*)((dkCollection*)a);
pIter = pCol->createIterator();
while (pIter->more() == TRUE)
 {
   i++;
   pSV = (DKServerDefQBIC*)((void*)(*pIter->next()));
   strServerName = pSV->getName();
   cout << "Server Name [" << i << "] - " << strServerName << endl;
   delete pSV;
  }
delete pIter;
delete pCol;
```

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

Listing image search databases, catalogs, and features

DKDatastoreQBIC provides a function for listing all of the image search databases, catalogs, and features on an image search server. The list is returned in a DKSequentialCollection object that contains DKIndexQBIC objects. After you get a DKIndexQBIC object, you can retrieve the database, catalog, and feature name. The following example shows how to retrieve the list of databases, catalogs, and features.

```
DKDatastoreQBIC dsQBIC;
DKSequentialCollection *pCol = 0;
dkIterator *pIter = 0;
DKSequentialCollection *pCol2 = 0;
dkIterator *pIter2 = 0;
DKSequentialCollection *pCol3 = 0;
dkIterator *pIter3 = 0;
DKDatabaseDefQBIC *pEntDB = 0;
DKCatalogDefQBIC *pEntCat = 0;
DKString strCatName;
DKString strDBName;
DKString strFeatName;
DKFeatureDefQBIC *pAttr = 0;
DKAny a;
DKAny *pA = 0;
long i = 0;
long j = 0;
long k = 0;
cout << "connecting to datastore" << endl;
dsQBIC.connect("QBICSRV","USERID","PW");
cout << "list databases " << endl;
pCol = (DKSequentialCollection*)((dkCollection*)dsQBIC.listEntities());
pIter = pCol->createIterator();
i = 0;
while (pIter->more() == TRUE)
 {
   i++;
   pEntDB = (DKDatabaseDefQBIC*)((void*)(*pIter->next()));
   strDBName = pEntDB->getName();
   cout << "database name [" << i << "] - " << strDBName << endl;
   cout << "  list catalogs for DB " << strDBName << endl;
   pCol2 = (DKSequentialCollection*)((dkCollection*)pEntDB->listSubEntities());
   pIter2 = pCol2->createIterator();
   j = 0;
   while (pIter2->more() == TRUE)
    {
      j++;
      pA = pIter2->next();
      pEntCat = (DKCatalogDefQBIC*) pA->value();
      strCatName = pEntCat->getName();
      cout << "catalog name [" << j << "] - " << strCatName << endl;
      pCol3 = (DKSequentialCollection*)((dkCollection*)pEntCat->listAttrs());
        pIter3 = pCol3->createIterator();
        k = 0;
        while (pIter3->more() == TRUE)
        {
         k++;
         pA = pIter3->next();
         pAttr = (DKFeatureDefQBIC*) pA->value();
         cout << "    Attribute name [" << k << "] - "
              << pAttr->getName() << endl;
         cout << "      datastoreName " << pAttr->datastoreName()
              << endl;
         cout << "      datastoreType " << pAttr->datastoreType()
              << endl;
         cout << "      attributeOf   " << pAttr->getEntityName()
              << endl;
         delete pAttr;
    }
     delete pIter3;
     delete pCol3;
     delete pEntCat;
    }
  cout << "  " << j << " features listed for catalog: "
       << strCatName << endl;
  delete pIter2;
  delete pCol2;
```

```
  delete pEntDB;
 }
delete pIter;
delete pCol;
cout << i << " databases listed" << endl;
dsQBIC.disconnect();
```

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

Representing image search information with a DDO

A DDO associated with DKDatastoreQBIC contains specific information for representing image search results. A DDO resulting from an image query corresponds to an image part inside an item; it has the following set of standard attributes:

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/dl directory.

Running an image query from the content server

DKDatastoreQBIC provides a function 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/dl directory.

Evaluating an image query from the datastore

DKDatastoreQBIC provides a function to evaluate a query. The following example shows how to evaluate an image query from the content server. Results are returned in a DKResults collection.

```
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 XDOfor 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 function 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_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
// to signal the end of parameter list
par[2].setName(DK_PARM_END);

// execute the combined query
cq->execute(par);

// get the results
DKAny any = cq->result();
DKResults* res = (DKResults*) any.value();
if (res != NULL) {
    // process the results
    ...
}
```

The last `if` statement is necessary to ensure that the DKResults object is not null.

Using a scope

If you have a DKResults object that you want to use as the scope, 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_PARM_QUERY, pq);
par[1].set(DK_TEXT_QUERY, tq);
par[2].set(DK_SCOPE_DL, scope);
par[3].set(DK_SCOPE_TS, tscope);
par[4].setName(DK_PARM_END);
// execute the combined query
cq->execute(par);
...
```

The results of one combined query can also be used as a scope for another combined query, and sometimes you can query the results.

Ranking

If the combined query contains at least one text query, then the resulting DDO has the attribute DKRANK. This attribute is not stored, but is computed each time by the Text Search Engine. The value of the rank corresponds to the highest rank of the part in the document that satisfies the text query conditions.

Tips

If you have several parametric queries and scopes, it is more efficient to run one complete query. This is also true for text queries.

The query option "MAX_RESULTS=nn" limits the number of returned results. Usually, this option is more applicable to text queries, because the result is sorted in descending order by rank. If this option is set to 10, for example, it means that the caller only wants the 10 highest matching results.

The meaning of the query option "MAX_RESULTS=nn" is different for parametric queries. Because there is no notion of rank, the caller gets the first 10 results. The results are intersected with the result from the text query. Therefore, when combining parametric and text queries, it is advisable not to specify the query option "MAX_RESULTS=nn" for the parametric query.

Understanding the 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 functions.

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/dl directory.

Creating a workflow

Use DKWorkflowServiceDL to create a workflow. To do this, you typically complete the following six steps:

1. Create an instance of DKWorkFlowDL
2. Set the workflow name (″GOLF″)
3. Set the workbasket sequence (″NULL″) to indicate that this workflow contains no workbaskets
4. Set the privilege (″All Privileges″)
5. Set the disposition (DK_WF_SAVE_HISTORY)
6. Call the add function add ()

The example follows the six steps to create a workflow.

```
DKDatastoreDL dsDL;
DKAny input_option = DK_SS_CONFIG;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.setOption(DK_DL_OPT_ACCESS, input_option);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkFlowDL * newwf = new DKWorkFlowDL(&wfDL);
newwf->setName("GOLF");
newwf->setWorkBasketSequence((dkCollection *)NULL);
newwf->setAccessList("All Privileges");
newwf->setHistoryDisposition(DK_WF_SAVE_HISTORY);
newwf->add();
...                                  // do some work
dsDL.disconnect();
```

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

Important: If you connect to the 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 function for listing the workflows in the system as shown in the following example. The list is returned in a sequential collection of DKWorkFlowDL objects.

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1 = (DKSequentialCollection *)wfDL.listWorkFlows();
if (wfList1 != NULL)
   {
      dkIterator * pIter1 = wfList1->createIterator();
      DKWorkFlowDL * pwf1;
      while (pIter->more())
      {
         pwf1 = (DKWorkFlowDL *)((void*)(*pIter1->next()));
```

```
         pwf1->retrieve();
         ...                          // do some work
         delete pwf1;
      }
   }
dsDL.disconnect();
```

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

Creating a workbasket

Use DKWorkflowServiceDL to create a workbasket. To do this, you typically complete the following steps:

1. Create an instance of DKWorkBasketDL
2. Set the workbasket name (Hot Items)
3. Set the privilege (All Privileges)
4. Call the add function

The following example follows these steps to create a workbasket. 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 function for listing the workbaskets in the system as shown in the following example. The list is returned in a sequential collection of DKWorkBasketDL objects.

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKSequentialCollection * wfList1=(DKSequentialCollection *)wfDL.listWorkBaskets();
if (wbList1 != NULL)
   {
      dkIterator * pIter1 = wbList1->createIterator();
      DKWorkBasketDL * pwb1;
      while (pIter->more())
      {
         pwb1 = (DKWorkBasketDL *)((void*)(*pIter1->next()));
         pwb1->retrieve();
         ...                          // do some work
         delete pwb1;
      }
   }
dsDL.disconnect();
```

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

Listing items in a workflow

DKWorkflowServiceDL provides a function for listing the item IDs of the items in a workflow as shown in the following example. The list is returned in a sequential collection of DKString objects.

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
DKWorkFlowDL * wf = new DKWorkFlowDL(&wfDL, (char *)itemIDWF);
wf->retrieve;
DKSequentialCollection * pColDoc1 = (DKSequentialCollection *)wf->listItemIDs();
if (pColDoc1 != NULL)
   {
      dkIterator* pIterDoc1 = pColDoc1->createIterator();
      DKString DocID1;
      while (pIterDoc1->more() == TRUE)
      {
         DocID1 = (DKString)(*pIterDoc1->next()));
         ...                        // do some work
      }
   }
dsDL.disconnect();
```

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

Executing a workflow

DKWorkflowServiceDL provides functions for executing a workflow. The following example demonstrates how to start an item in a workflow, how to route an item to a workbasket, and how to complete an item in a workflow. To use this sample you must modify it to:

- Use a valid item ID instead of EP8L8OR9MHH##QES
- Use a valid workflow ID instead of HI7MOPALUPFQ1U47
- Use a valid workbasket ID instead of E3PP1UZOZUFQ1U3M

```
DKDatastoreDL dsDL;
DKWorkFlowServiceDL wfDL(&dsDL);
DKString itemID   = DKString("EP8L8OR9MHH##QES");
DKString itemIDWF = DKString("HI7MOPALUPFQ1U47");
DKString itemIDWB = DKString("E3PP1UZOZUFQ1U3M");
dsDL.connect("LIBSRVRN", "FRNADMIN", "PASSWORD");
wfDL.startWorkFlowItem(itemID,                  // itemID
                       itemIDWF,                // itemIDWB
                       NULL,                    // default(the first workbasket)
                       TRUE,                    // overload
                       DK_WIP_DEFAULT_PRIORITY  // initial_priority
                      );
...                                         // do some work
wfDL.routeWipItem(itemID,                   // itemID
                  itemIDWF,                 // itemIDWB
                  TRUE,                     // overload
                  DK_NO_PRIORITY_CHANGE     // initial_priority
                      );
...                                         // do some work
wfDL.completeWorkFlowItem(itemID);
dsDL.disconnect();
```

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

Using rights management in earlier Content Manager

You protect your documents by using rights management. It explains marking techniques that you can use to protect your property and describes how to use the generic marking C APIs provided with earlier Content Manager. Reference information for using these APIs is in the *C Application Programming Reference*.

Protecting your intellectual property

Multimedia objects stored in digital form are your intellectual property. Protection of these objects can be critical to your business objectives.. You can use the marking technology provided with earlier Content Manager to deter unauthorized use of your intellectual property by marking your multimedia digital objects for protection or by fingerprinting your objects for identification.

You can apply a mark to your valuable objects to:

- Identify the source, to deter unauthorized copying or reusing. This is known as a watermark and is typically visible.
- Identify the recipient of the content, to deter unauthorized copying or reusing. This is known as a fingerprint and is typically invisible.
- Provide a contact for obtaining additional information.
- Give information, such as time and date, for use in a Àvalue-addedÀ chain of distribution.

You can mark your digital objects before the object is delivered to your customers. Both watermarks and fingerprints can be applied before delivery. However, applying a fingerprint before delivery implies that the recipient is known and might require the mark to be applied dynamically in the delivery process. Applying the mark from your own controlled environment before delivery increases the security, because there is less risk of tampering.

You can apply marks at several stages in the management and delivery process. Your situation will influence what is appropriate. Marks can be applied at the following points in your process:

- Before the object is stored

  If a common mark is to be used for the object (for example, a visible watermark to identify the owner), you can apply the mark before or while storing the object. You can store both the original, unmarked object and the marked object in your earlier Content Manager system. Or you can store only the marked object and keep the unmarked object in a separate repository.
- After the object is stored

  If you want to mark objects that are stored in your system, you can retrieve the object, mark it, and either replace the unmarked object with the marked version or store the marked object as a new item.
- When the object is retrieved

  If the mark to be applied varies based on the recipient, you can apply the mark dynamically after the object is retrieved. The marked object can then be delivered instead of the original object

If you have many legacy objects in your system that are not marked and you do not want to take the time or use the resources to go back and mark each object, you can mark objects when they are retrieved.

Using the generic marking APIs

Earlier Content Manager provides a set of generic marking C APIs that you can use with different plug-in algorithms to mark an image. Content Manager also provides the following plug-in algorithms:

- Visible random brightness alteration (VRBA) algorithm
- Data hiding algorithm

Use the VRBA algorithm to apply a visible watermark to an image. Use the data hiding algorithm to apply an invisible watermark to an image. The following figure shows the relationship of the generic marking APIs, the plug-in algorithms, and your application.

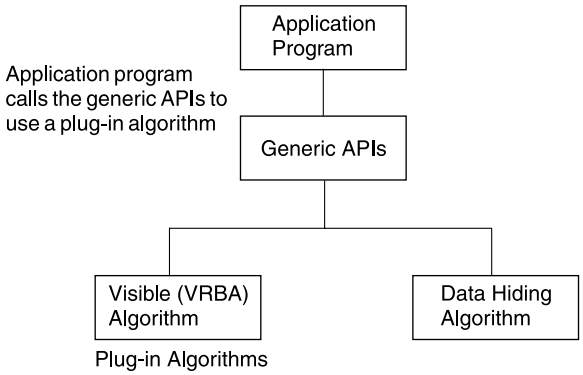

Figure 61. Rights management C APIs

The generic marking APIs provide the ability to mark an image using a specific marking algorithm. The following table shows the generic marking APIs provided with earlier Content Manager. You identify the algorithm that you want to use by passing its name to the frnpwigo function.

C API name	**description**
frnpwigi	Initialize the generic marking environment
frnpwigt	Terminate the generic marking environment
frnpwigo	Open a marking handle to the environment for a specific image
frnpwigr	Read marked data to a buffer from a marking handle
frnpwigc	Close the handle to a marking environment for a specific image
frnpwihe	Return the error code stored in a marking handle
frnpwige	Extract the generic API part of the error code (higher two bytes)
frnpwipe	Extract the plugin part of the error code (lower two bytes)

Use the generic marking C APIs as described below to set up a marking environment and mark an image. Figure 6 on page 87 shows the standard flow.

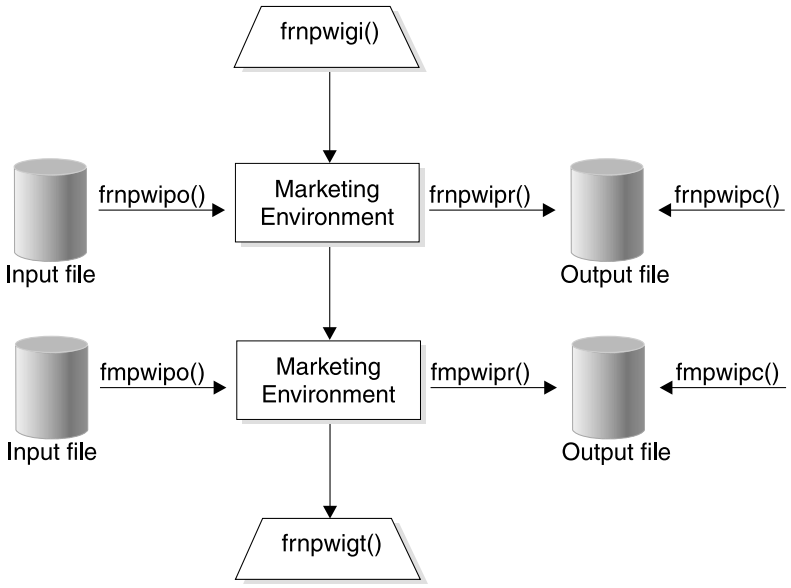

Figure 62. Marking C API calling sequence

1. **frnpwigi**

   Use the frnpwigi function to initialize the generic marking environment. The frnpwigi function finds, loads, and registers all the marking plug-in algorithms that exist in the directory for plug-in algorithms.

2. For each image to be marked:

   a. **frnpwigo**

      Use the frnpwigo function to set up an environment for marking a specific image using a specific marking algorithm. The frnpwigo function returns a marking handle. Parameter information:

      - algName: A pointer to the name of the marking algorithm.
      - parameters: A string of keyword-value pairs that are the parameters for the marking algorithm.

        The keyword-value pairs are separated by a single blank character and the string is terminated by a null character. Values that contain blank characters should be enclosed in double quotes. For example:

        ```
        key=123 intensity=10 id="J. Smith"
        ```

      - objects: A pointer to a structure in which a sequence of marking-related objects are stored. Each marking algorithm defines its own sequence of objects.

   b. **frnpwigr**

      Use the frnpwigr function to store the marked image in a buffer. Parameter information:

      - hdl: The handle for the marking environment that was returned by the frnpwigo function.
      - buffer: A pointer to the buffer that contains the marked image.
      - sizeOfTheBuffer: The size of the buffer.

   c. **frnpwigc**

      Use the frnpwigc function to close the marking environment for a specific image. Parameter information:

      - hdl: The handle for the marking environment that was returned by the frnpwigo function.

3. **frnpwigt**

   Use the frnpwigt function to terminate the generic marking environment after all images have been marked. The frnpwigt function frees the resources allocated by the frnpwigi function.

Applying a visible watermark

You can use the generic marking APIs with the IBM VRBA algorithm to apply a visible watermark to your image. You can scale the watermark, position it on the image, and vary the intensity of the watermark and of the noise field superimposed on the watermark. You can allow the scaling and position values to vary based on the output of a pseudo-random number generator. By specifying the seed for the pseudo-random number generator, you can create an exact duplicate of a watermarked image.

An example of using the VRBA algorithm is provided with earlier Content Manager.

You can also use the earlier Content Manager watermarking APIs, WMWMark andWMUseCt, to add a visible watermark to an image.

Creating a mask file for a visible watermark

You can use any drawing or image creation tool to create a mask file for a visible watermark. Create the mask file using the following guidelines:

- Make a monochromatic image, 8 bits per pixel.
- Use 3 levels:
  white, 0xFF
  gray, 0x80
  black, 0x00

  The gray areas of the mask are transparent to the image. The white areas of the mask will brighten the image. The black areas of the mask will darken the image. Other values can be used, but they are not recommended.
- Set the background of the mask to medium gray, 0x80.
- Make the mask bold. Fine lines and small characters are hard to distinguish in a marked image.
- Make the mask simple. An elegant mask becomes a distraction to the image.

Applying an invisible mark

You can use the generic marking APIs with one of the following data hiding algorithms to apply an invisible watermark to your image.

IBM DHLE
: Hides the watermark in the least significant bit of each pixel at each color plane. Use this algorithm for embedding a large amount of data in images that will not be compressed.

IBM DHJE
: Hides the watermark in selected groups of pixels. Use this algorithm for embedding data in JPEG images.

IBM DHGE
: Hides the watermark in selected groups of pixels. Use this algorithm for embedding data in GIF images.

An example of using the data hiding algorithm is provided with earlier Content Manager.

Sample programs

Sample programs are provided with earlier Content Manager to show how the generic marking C APIs can be used in an application. These programs can be found in the %FRNROOT%\samples\wm directory.

The following samples are provided:

vrbasample.c
An example of using the VRBA algorithm with the generic marking APIs to embed a visible watermark.

dhembsample.c
An example of using the data hiding algorithm with the generic marking APIs to embed an invisible watermark.

dhretsample.c
An example of using the data hiding algorithm to retrieve an invisible watermark.

In addition, a sample of a Java ® interface for the generic marking APIs is available in the %FRNROOT%\samples\wm\jwic directory. A sample Java program, WmDemo.cmd, is available in the %FRNROOT%\samples\wm\gui directory. These samples use JNI and JDK 1.1.3. You can use these samples as a reference when writing a Java application.

Working with Content Manager ImagePlus for OS/390

The EIP classes and APIs for Content Manager ImagePlus for OS/390 include the ability to perform the following operations:

- 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 functions to list all of the attributes associated with each entity.

```
// Method 1:
cout << "---List attributes using listAttrNames and getAttr functions---" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
  docDef = (DKEntityDefIP*)(pIter->next()->value());
  cout << " Document type     : " << docDef->getType() << endl;
  cout << " Document type name: " << docDef->getName() << endl;

  long tmpCount;
  DKString* attrNames;

// Upon return, tmpCount contains the number of elements in the list.
  attrNames = docDef->listAttrNames(tmpCount);
  for (int i=0; i<tmpcoun; i++)
  {
    cout << "   Attr name before lookup " << attrNames[i] << endl;
    attrDef = (DKAttrDefIP*)(docDef->getAttr(attrNames[i]));
    cout << "   Attr name [" << i << "] : " << attrDef->getName() << endl;
    cout << "   Attr id       : " << attrDef->getId() << endl;
    cout << "   Entity name   : " << attrDef->getEntityName() << endl;
    cout << "   Datastore name: " << attrDef->datastoreName() << endl;
    cout << "   Attr type     : " << attrDef->getType() << endl;
    cout << "   Attr restrict : " << attrDef->getStringType() << endl;
    cout << "   Attr min val  : " << attrDef->getMin() << endl;
    cout << "   Attr max val  : " << attrDef->getMax() << endl;
    cout << "   Attr display  : " << attrDef->getSize() << endl;
    cout << "   Attr precision: " << attrDef->getPrecision() << endl;
    cout << "   Attr scale    : " << attrDef->getScale() << endl;
    cout << "   Attr update   ? " << attrDef->isUpdatable() << endl;
    cout << "   Attr nullable ? " << attrDef->isNullable() << endl;
    cout << "   Attr queryable? " << attrDef->isQueryable() << endl;
    cout << "" << endl;
    delete attrDef;
   }  // end for

 delete [] attrNames;

}  // end while
delete pIter;
```

The following example uses the listEntityAttrs function to list all the attributes associated with each entity.

```
//  Method 2:
cout << "---List attributes using listEntityAttrs function---" << endl;

pIter = pCol->createIterator();
while (pIter->more())
{
  docDef = (DKEntityDefIP*)(pIter->next()->value());  // iterator returns DKAny*
  cout << " Document type     : " << docDef->getType() << endl;
  cout << " Document type name: " << docDef->getName() << endl;
  DKSequentialCollection* pAttrCol = (DKSequentialCollection*)
                                     (dsIP.listEntityAttrs(docDef->getName()));
```

```
  if ( pAttrCol == 0 )
  {
    cout << "collection of entity attrs is null for entity "
         << docDef->getName()
         << endl;
   }
  else
  {
    int i=0;
    dkIterator* pAttrIter = pAttrCol->createIterator();
    while (pAttrIter->more())
    {
      i++;
      // ----- The iterator returns a pointer to DKAny
      attrDef = (DKAttrDefIP*)(pAttrIter->next()->value());
      cout << "   Attr name [" << i << "] : " << attrDef->getName() << endl;
      cout << "   Attr id       : " << attrDef->getId() << endl;
      cout << "   Entity name   : " << attrDef->getEntityName() << endl;
      cout << "   Datastore name: " << attrDef->datastoreName() << endl;
      cout << "   Attr type     : " << attrDef->getType() << endl;
      cout << "   Attr restrict : " << attrDef->getStringType() << endl;
      cout << "   Attr min val  : " << attrDef->getMin() << endl;
      cout << "   Attr max val  : " << attrDef->getMax() << endl;
      cout << "   Attr display  : " << attrDef->getSize() << endl;
      cout << "   Attr precision: " << attrDef->getPrecision() << endl;
      cout << "   Attr scale    : " << attrDef->getScale() << endl;
      cout << "   Attr update   ? " << attrDef->isUpdatable() << endl;
      cout << "   Attr nullable ? " << attrDef->isNullable() << endl;
      cout << "   Attr queryable? " << attrDef->isQueryable() << endl;
      cout << "" << endl;
      delete attrDef;
     }  // end while
    delete pAttrIter;
   }
  delete pAttrCol;
  delete docDef;
 }  // end while
delete pIter;
}
delete pCol;
```

ImagePlus for OS/390 query syntax

The following example shows the query syntax for ImagePlus for OS/390:

```
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 Content Manager for AS/400

The API classes provided for Content Manager for AS/400 (VisualInfo for AS/400) are very similar to those provided for Content Manager.

Restriction: Content Manager for AS/400 does not support:

- Text Search Engine and QBIC search
- Combined query
- Workbasket and workflow

Listing index classes and attributes

The following example lists entities (index classes) and attributes in Content Manager 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 63 on page 515 shows an example of how to run a query in Content Manager 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 63. Query in Content Manager 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 63. Query in Content Manager 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 63. Query in Content Manager 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 64 on page 519 shows how the components of the object model are related.

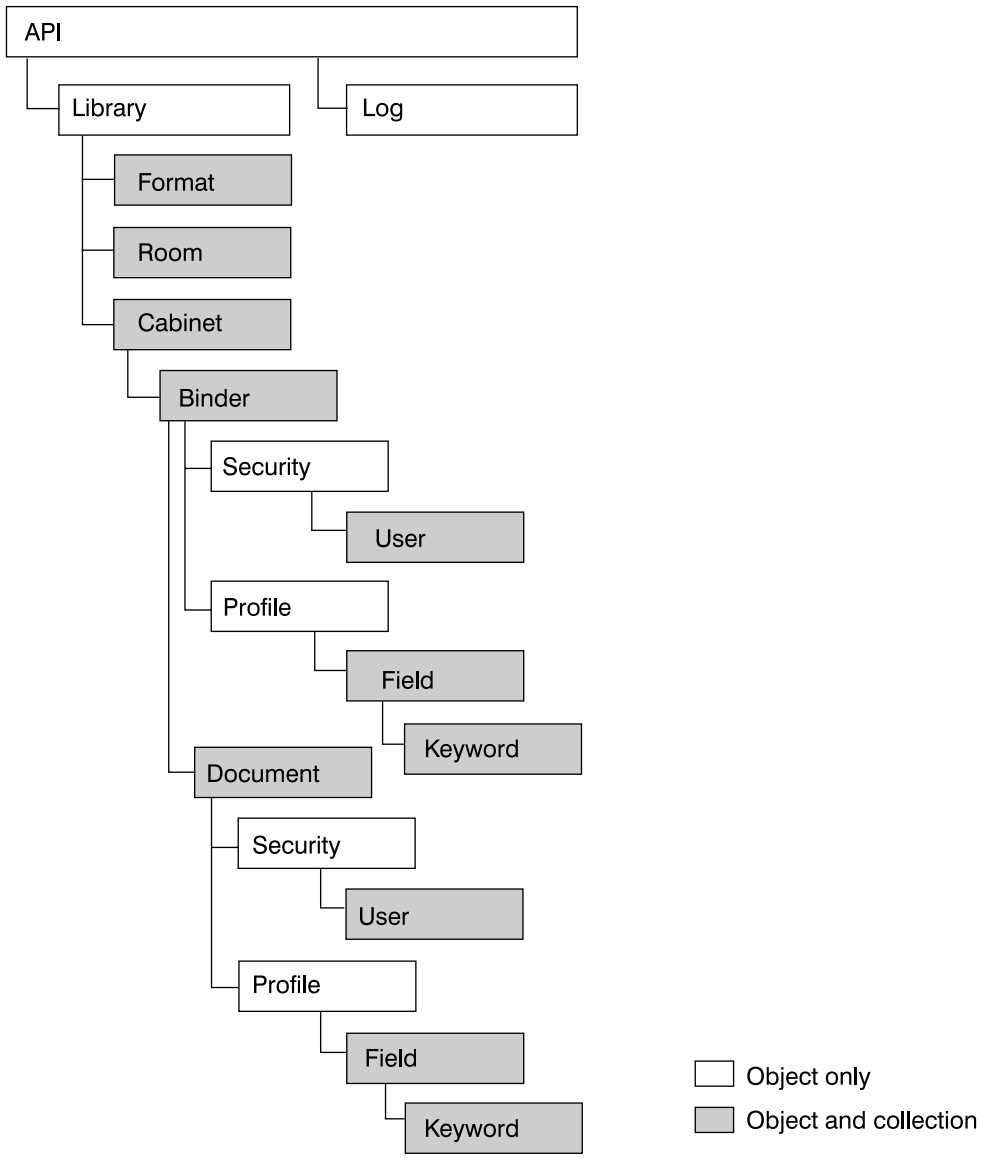

Figure 64. Domino.Doc object model

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 subentities

The following example lists the entities in Domino.Doc. In the example, rooms are the entities, and cabinets, binders and documents are subentities.

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 function to query the Domino.Doc content server. The Domino.Doc content server must be fully text indexed for this function to work efficiently. To test for an index, use the IsFTIndexed property. To create an index, use the UpdateFTIndex function.

The FTSearch function searches all of the documents in a content server—to search documents within a particular view, use the FTSearch function in NotesView; to search documents within a particular document collection, use the FTSearch function in NotesDocumentCollection.

If you do not specify a sort option, documents are sorted by relevance. If you want to sort by date, you do not get relevance scores with the sorted results. If you pass the resulting DocumentCollection to a NotesNewsletter instance, results are sorted by either the document creation date or the relevance score, depending on which sort options you use.

Using query syntax

The syntax rules for a query are in the following list. Use parentheses to override precedence and to group operations.

Plain text

Use plain text to search for a word or phrase as is. Enclose search keywords and symbols 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 Extended Search (ES)

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

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

The EIP classes and APIs for Extended Search provide the ability to perform the following operations:

- Listing ES servers
- Listing databases and fields
- Ability to use Generalized Query Language (GQL) to perform searches
- Ability to retrieve documents

Restriction: ES does not support:

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

All ES features are accessed and controlled by the ES configuration database. Use the configuration database to assign database definitions for data sources to be searched, network addresses, access control information, and other related information.

Listing ES servers

To provide access to multiple ES 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 ES server and the *port number* is the port number defined for accessing the server (for example: `PORT=80`).

Listing entities (databases) and attributes (fields)

When you build a query to search a ES server, you must know the available database and field names. The DKDatastoreDES object provides the `listEntities` function to list the databases and the `listEntityAttrs` function to list the fields for each database. The following example shows how to retrieve a list of databases and their fields.

```
...
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/ES directory.

Using Generalized Query Language (GQL)

ES uses the Generalized Query Language (GQL) to perform searches. Table 52 contains examples of valid GQL expressions.

Table 52. GQL expressions

GQL Expression	Description
"software"	search for documents that contain the word software
(TOKEN:WILD "exec*")	search for documents that contain any word beginning with exec
(AND "software" "IBM")	search for documents that contain both words software and IBM
(START "View" "How")	search for documents in which the View field begins with the word How
(EQ "View" "How Do I?")	search for documents in which the View field contains the exact string How Do I?
(GT "BIRTHDATE" "19330804")	search for documents in which the BIRTHDATE field is greater than August 4, 1933

ES uses the query type DK_DES_GQL_QL_TYPE. This query type has the following syntax:

```
SEARCH=(DATABASE=(db_name | db_name_list | ALL);
                  COND=(GQL expression));
[OPTION=([SEARCHABLE_FIELD=(fd_name, ...);]
                   [RETRIEVABLE_FIELD=(fd_name, ...);]
                   [MAX_RESULTS=maximum_results;]
                   [TIME_LIMIT=time)]
```

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 ES.

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore ES created" << endl;
cout << "connecting to datastore" << endl;
dsDES.connect(libsrv,userid,pw,str);
cout << "datastore connected " << libsrv << " userid - " << userid << endl;

DKString cmd = "SEARCH=(DATABASE=(Notes Help);";
cmd += "COND=((IN \"Subject\" \"your\")));";
cmd += "OPTION=(MAX_RESULTS=2;TIME_LIMIT=10);";

cout << "query string " << cmd << endl;
cout << "executing query" << endl;
```

```
pCur = dsDES.execute(cmd);
cout << "query executed" << endl;
...
```

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

DDO properties in ES

A DDO in ES 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 ES

The persistent identifier (PID) contains specific information about a document. The object type identifies the database where the document was found. A PID is created by using the database name, followed by the | character and the document ID, for example:

```
database name|documentId ()
```

For more information on PIDs see “Understanding persistent identifiers (PID)” on page 14 and “Creating a persistent identifier (PID)” on page 32.

Contents of a ES 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 = (DKPidXDODES*)pXDO->getPid();
cout << "XDO pid database name " << pidXDO->getDatabaseName() << endl;
cout << "XDO pid docId " << pidXDO->getDocId() << endl;
cout << "XDO mimetype " << pXDO->getMimeType() << endl;
((DKBlobDES*)pXDO)->getContentToClientFile("c:\\temp\\temp.html", 1);
}
   break;

         case DKAny::tc_collection :
    {
         pdCol = a;
         cout << strDataName << " collection name : " << strDataName << endl;
         cout << "--------------------------------------" << endl;
         pdIter = pdCol->createIterator();
         ushort b = 0;
         while (pdIter->more() == TRUE)
      {
       b++;
          cout << "  ------------------------------------------------" << endl;
       a = *(pdIter->next());
          switch (a.typeCode())
       {
         case DKAny::tc_string :
   {
          strData = a;
          cout << "attribute name : " << strDataName
                    << " value : " << strData << endl;
    }
         break;
         case DKAny::tc_long :
   {
          long l = a;
          cout << "attribute name : " << strDataName << " value : " << l << endl;
    }
          break;
          case DKAny::tc_double :
    {
          double db = a;
          cout << "attribute name : " << strDataName << " value : " << db << endl;
```

```
    }
            break;
            case DKAny::tc_timestamp :
    {
            DKTimestamp tt = a;
            cout << "attribute name : " << strDataName << " value : "
<< tt.getMonth() << "/" << tt.getDay() << "/" << tt.getYear() << " "
<< tt.getHours() << ":" << tt.getMinutes() << ":" << tt.getSeconds() << endl;
    }
            break;
            case DKAny::tc_dobase :
    {
   pDOBase = a;
   pXDO = (dkXDO*)pDOBase;
   cout << "attribute name : " << strDataName << " value : " << endl;
   pidXDO = (DKPidXDODES*)pXDO->getPid();
   cout << "XDO pid database name " << pidXDO->getDatabaseName() << endl;
   cout << "XDO pid docId " << pidXDO->getDocId() << endl;
   cout << "XDO mimetype " << pXDO->getMimeType() << endl;
   DKString str = "c:\\temp\\temp";
   DKString strT = b;
   str = str + strT + ".html";
   ((DKBlobDES*)pXDO)->getContentToClientFile(str, 1);
    }
     break;
      }
ushort usCount = p->dataPropertyCount(j);
for (ushort k = 1; k <= usCount; k++)
{
a = p->getDataProperty(j, k);
cout << " property " << k << " " << a << endl;
 }
       }

      if (b == 0)
   {
    cout << strDataName << " collection has no elements " << endl;
    }
      cout << "  ------------------------------------------------" << endl;
      break;
  }
   }
  ushort usCount = p->dataPropertyCount(j);
  for (ushort k = 1; k <= usCount; k++)
  {
   a = p->getDataProperty(j, k);
   cout << " property " << k << " " << a << endl;
 }
}
  cnt++;
  delete p;
```

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

Retrieving a document

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

```
DKDatastoreDES dsDES;
dkResultSetCursor* pCur = 0;
cout << "Datastore ES created" << endl;
cout << "connecting to datastore" << endl;
```

```
dsDES.connect(libsrv,userid,pw,str);
    cout << "datastore connected " << libsrv << " userid - " << userid << endl;
...
p = new DKDDO(&dsDES, "");
DKPid pid2;
pid2.setDatastoreType(dsDES.datastoreType());
pid2.setDatastoreName(dsDES.datastoreName());
pid2.setId("Notes Help|215e");
pid2.setObjectType("");
p->setPidObject((DKPid*)&pid2);
p->retrieve();
...
```

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

Retrieving a 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 ES 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 ES created" << endl;
cout << "connecting to datastore" << endl;
dsDES.connect(libsrv,userid,pw,str);
   cout << "datastore connected " << libsrv << " userid - " << userid << endl;
...
cout << "executing retrieve a XDO" << endl;

DKBlobDES* p = new DKBlobDES(&dsDES);
DKPidXDODES pid;
pid.setDocId("D:\\desdoc\\README.html");
pid.setDatabaseName("ES files");
pid.setFieldName("Doc$Content");
pid.setPrimaryId("ES files|D:\\desdoc\\README.html");
p->setPidObject((DKPidXDO*)&pid);

p->retrieve("c:\\temp\\DESReadme.html");

cout << "retrieve executed" << endl;
...
```

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

Associating MIME types with documents

ES does not directly support identification of Multipurpose Internet Mail Extension (MIME) types. However, you must know the MIME type of an XDO that you want to display within a Web browser.

The CMBCC2MIME.INI file is used to determine the MIME type of a document. When a ES query from NotesPump or FileSystem databases returns a BLOB, the CMBCC2MIME.INI file is searched to determine if a MIME type can be assigned to the

BLOB. The default MIME type is `text/html`. A sample file named `cmbcc2mime.ini.samp` is available in the `samples` directory.

Using federated searching in ES

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

Table 53 summarizes the SQL to GQL conversion of the supported comparison and logical operators.

Table 53. 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 OnDemand

Enterprise Information Portal provides a connector and related classes for accessing content on Content Manager OnDemand servers. The OnDemand classes and APIs allow you to:

- Connect to and disconnect from OnDemand servers
- Listing application groups and application group fields
- Listing OnDemand folders
- Querying an application group
- Retrieving entire OnDemand documents or document segments
- 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

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

Representing OnDemand servers and documents

You represent a Content Manager OnDemand content server using a DKDatastoreOD in an Enterprise Information Portal application, and you represent an OnDemand document as a DDO using a DKDDO. OnDemand DDOs contain the following information:

- Document attribute names and their values
- Document data and annotations (represented as DKParts)
- Collection of logical views for a document

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 (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

Note: An EIP administrator must properly define the OnDemand connector by specifying the string in the **Additional Parameters** field on the **Initialization Parameters** page. The string should look like this: `ENTITY_TYPE=TEMPLATES;;` Be sure to include the two semicolons.

Listing information on OnDemand

You can list application groups and folders for OnDemand servers.

Listing application groups

You can list application groups (entities) in OnDemand using the listEntities() function of DKDatastoreOD. The following example illustrates how to use this function:

```
// ----- Show the application groups
// ----- First get the groups
pCol = (DKSequentialCollection*)dsOD.listEntities();
pIter = pCol->createIterator();
int i = 0;
// ---- Process the list
while (pIter->more() == TRUE)
{
  i++;
  agDef = (DKAppGrpDefOD*)((void*)(*pIter->next()));
  strAppGrp = agDef->getName();
  cout << "  app group name[" << i << "]: ==>" << strAppGrp << endl;
  . . .
```

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

```
// ----- Get the attributes for each of the entities(application groups)
pCol2 = (DKSequentialCollection*)dsOD.listEntityAttrs(strAppGrp);
pIter2 = pCol2->createIterator();
int j = 0;
// ----- List the attributes
while (pIter2->more() == TRUE)
{
   j++;
```

```
    attrDef = (DKFieldDefOD*)(void*)(*pIter2->next());
    cout << "    attribute name[" << j << "]: ==>" << attrDef->getName() << endl;
    cout << "      datastore type: " << attrDef->datastoreType() << endl;
    cout << "      attribute of: " << attrDef->getEntityName() << endl;
    cout << "      type: " << attrDef->getType() << endl;
    cout << "      size: " << attrDef->getSize() << endl;
    cout << "      ID: " << attrDef->getId() << endl;
    cout << "      precision: " << attrDef->getPrecision() << endl;
    cout << "      scale: " << attrDef->getScale() << endl;
    cout << "      stringType: " << attrDef->getStringType() << endl;
    cout << "      nULLable: " << attrDef->isNullable() << endl;
    cout << "      queryable: " << attrDef->isQueryable() << endl;
    cout << "      updatable: " << attrDef->isUpdatable() << endl;
    // ----- Clean up the attribute
    delete attrDef;
  }
  cout << "  " << j << " attribute(s) listed for the " << strAppGrp
       << " app group\n" << endl;
  // ----- Clean up the iterators and collections
  if ( pIter2 )
     delete pIter2;
  if ( pCol2 )
     delete pCol2;
   . . .
```

Listing OnDemand folders

You use the listSearchTemplates() function to get a list of the folders for an OnDemand datastore. The following example shows how to list folders in OnDemand:

```
  . . .
  // ----- List the folders
  dsDef = (DKDatastoreDefOD*)dsOD.datastoreDef();
  pCol = (DKSequentialCollection*)dsDef->listSearchTemplates();
  pIter = pCol->createIterator();
  i = 0;
  // ----- Process the list of folders
  while (pIter->more() == TRUE)
  {
     i++;
     folderName = (DKString)(*pIter->next());
     cout << "folder name [" << i << "] - " << folderName << endl;
  }
  // ----- Disconnect
  dsOD.disconnect();
  . . .
```

For more examples of accessing entities and attributes in OnDemand, see the sample applications available in the Cmbroot/Samples/cpp/od directory.

Retrieving an OnDemand document

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

Retrieving a particular document or folder

The following example shows the query used to find particular documents or folders from an OnDemand datastore:

```
dkResultSetCursor* pCur = 0;
  DKNVPair parm0, parm1;
  DKNVPair parm2( DK_CM_PARM_END, 0 );
  DKNVPair parms[3];
  DKString cmd;

  // ----- Set the application Group and create the SQL string
```

```
parm0 = DKNVPair("APPL_GROUP", (DKString)"INSURE  01 ");
parm1 = DKNVPair("MAX_RESULTS", (DKString)10);
cmd = "WHERE POLICY_NUMBER LIKE '324-1443255%' ";
// ----- Set the parameters as an array of DKNVPairs
parms[0] = parm0;
parms[1] = parm1;
parms[2] = parm2;

// ----- Execute the query
pCur = dsOD->execute(cmd, DK_SQL_QL_TYPE, parms);
```

Displaying documents and their parts and attributes

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

```
DKDDO *p = 0;
DKAny a;
 . . .
for (j = 1; j <= numDataItems; j++)
{
   a = p->getData(j);
   strDataName = p->getDataName(j);

   cout << " " << j << ". Attribute Name: " << strDataName << endl;
   cout << "  type: " << p->getDataPropertyByName(j,DK_PROPERTY_TYPE) << endl;
   cout << "  nullable: "
     << p->getDataPropertyByName(j,DK_PROPERTY_NULLABLE) << endl;

   if (strDataName != DKPARTS && strDataName != "DKResourceOD" &&
          strDataName != "DKViewDataOD" && strDataName != "DKFixedViewDataOD")
   {
     cout << "   attribute ID: "
        << p->getDataPropertyByName(j,DK_PROPERTY_ATTRIBUTE_ID) << endl;
   }

   if (a.typeCode() == DKAny::tc_string)
   {
     DKString astring = a;
     cout << "   attribute Value (string): " << astring << endl;
   }
   else if . . .
   {
      // ----- Handle each of the other types
   }
   else if (a.typeCode() != DKAny::tc_null)
   {
     cout << "       Attribute Value (non NULL): " << a << endl;

     if (strDataName == "DKResourceOD" ||
              strDataName == "DKFixedViewDataOD")
     {
       pDO = (dkDataObjectBase*)a;
       if (pDO->protocol() == DK_XDO)
       {
         cout << "         dkXDO object " << endl;
         pXDO = (dkXDO*)pDO;
         pidXDO = (DKPidXDOOD*)pXDO->getPid();
         cout << "    XDO PID string: " << pidXDO->pidString() << endl;
         // ----- Retrieve and open instance handler for an XDO
         pXDO->retrieve();
       }
     }
   }
   else cout << " Attribute Value is NULL" << endl;
```

Tracing

You can trace events with the OnDemand connector. To enable the connector C++ API trace, place the trace ini file, cmbodCtrace.ini, in the root of the C drive (C:\) or in the directory specified in the CMBROOT variable. The default output directory for trace files is C:\Ctrace. To write the trace information elsewhere, edit the trace ini file (cmbodCtrace.ini).

Be sure the path name specified in cmbodtrace.ini is valid and that the line containing CMBODTRACEDIR is not preceded with a # sign. Here is a sample cmbodCtrace.ini file:

```
     #=============================================================
# OnDemand Trace INI file
#
# OnDemand Trace File Directory Name key - CMBODTRACEDIR
#
# The CMBODTRACEDIR key defines the directory where the trace files will
# be written to.  If the directory name does not exist, it will be created.
#
# Please make sure the path name does not point to an existing file name.
# Otherwise, no trace files will be created.
#
# The trace output directory name can be changed to point to a drive
# where more space is available.  But it is recommended not to change
# the trace output directory name in the middle of an active trace
# session.
#
# CMBODTRACESCOPE controls how much trace information to generate.
#
# CMBODTRACESCOPE=ENTRY_EXIT_ONLY
# Trace only the entry and exit of all C++ methods and functions.
#
# If CMBODTRACESCOPE is missing, or set to anything else, a full trace
# is taken.
#
# To disable the trace, add a leading # character in column 1 on
# the CMBODTRACEDIR line.
#
[ODCTRACE]
CMBODTRACEDIR=D:\Ctrace
#CMBODTRACESCOPE=ENTRY_EXIT_ONLY
```

Working with relational databases

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

Connecting to relational databases

To represent a relational database, create a DKDatastore*xx* object, where the *xx* is DB2 for a DB2 database, DJ for DataJoiner, or ODBC for Open Database Connectivity. The following sample connects to the DB2 sample database:

```
try {      DKDatastoreDB2 dsDB2;
      dsDB2.connect("sample", userid, pw);
      . . .
      dsDB2.disconnect();
   }
   catch(DKException &exc). . .
```

Use the database name when connecting.

Connection strings

When connecting to a relational database, you can specify a connection string and pass it as a parameter. If you specify multiple connection strings, separate them with a semi-colon (;). Connection strings can take the following forms:

Connection strings for DB2, DataJoiner, and ODBC:

NATIVECONNECTSTRING=(*native connect string*)

Specifies a native connect string to be passed to the database when connecting. Check the information for your content server to determine the valid native connections strings.

SCHEMA=*schema name*

Specifies the database schema name to be used when running the listEntities, listEntityAttrs, listPrimaryKeyNames, listForeignKeyNames functions.

Configuration strings

You can specify a configuration string and pass it as a parameter to the configuration function 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.

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:

```
try {
  DKDatastoreDB2 dsDB2;
  DKString schema;
  DKSequentialCollection *pCol2 = 0;
  dkIterator *pIter = 0;
  dkIterator *pIter2 = 0;
  DKTableDefDB2 *pEnt = 0;
  DKString strServerName;
  DKString strTable;
  DKColumnDefDB2 *pAttr = 0;
  DKDatastoreDefDB2 *dsDefDB2 = 0;
  DKAny a;
  DKAny *pA = 0;
```

```
  long i = 0;
  long j = 0;

  // ----- Connect to datastore and set value for schema name
     . . .
  // ----- Create a datastore definition and set the schema name
  dsDefDB2 = (DKDatastoreDefDB2 *) dsDB2.datastoreDef();
  if (schema != "")
  {
    dsDefDB2->setSchemaName(schema);
  }

  // ----- Get a list of the entities (tables)
  pCol = (DKSequentialCollection*)((dkCollection*)dsDB2.listEntities());
  pIter = pCol->createIterator();
  i = 0;
  // ----- List the attributes (columns) for each entity (table)
  while (pIter->more() == TRUE)
  {
    i++;
    pEnt = (DKTableDefDB2*)((void*)(*pIter->next()));
    strTable = pEnt->getName();
    cout << "table name [" << i << "] - " << strTable << endl;
    cout << "  list columns for " << strTable << " table" << endl;
    pCol2 =
    (DKSequentialCollection*)((dkCollection*)dsDB2.listEntityAttrs(strTable));
    pIter2 = pCol2->createIterator();
    j = 0;
    while (pIter2->more() == TRUE)
    {
      j++;
      pA = pIter2->next();
      pAttr = (DKColumnDefDB2*) pA->value();
      cout << "    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 << "      primary key   " << pAttr->isPrimaryKey() << endl;
      cout << "      foreign key   " << pAttr->isForeignKey() << endl;
      delete pAttr;
    }
    delete pIter2;
    delete pCol2;
    delete pEnt;
  }
  delete pIter;
  delete pCol;
  dsDB2.disconnect();
}
catch(DKException &exc)
  . . .
```

Refer to TListCatalogDB2.cpp, TListCatalogODBC.cpp, and TListCatalogDJ.cpp in the CMBROOT\Samples\cpp\db2, odbc, and dj directories 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.

```
try {
   DKDatastoreDB2 dsDB2;
   dkResultSetCursor* pCur = 0;
   DKNVPair par[2];
   DKAny anyValue;
   DKString strMax = "5";
   anyValue = strMax;
   par[0].set(DK_CM_PARM_MAX_RESULTS, anyValue);
   par[1].setName(DK_CM_PARM_END);
   // ---- Create a datastore and connect
    . . .
   // ----  Create a query string containing the select
   DKString qstrng = "SELECT * FROM EMPLOYEE";
   // ---- Execute the query
   pCur = dsDB2.execute(qstrng, DK_CM_SQL_QL_TYPE, par);
   // ---- Declarations
   DKDDO *p = 0;
   dkDataObjectBase *pDOBase = 0;
   DKDDO *pDDO = 0;
   dkXDO *pXDO = 0;
   DKAny a;
   ushort j = 0;
   ushort k = 0;
   ushort val = 0;
   ushort cnt = 1;
   DKString strData = "";
   DKString strDataName = "";
   dkCollection* pdCol = 0;
   dkIterator*  pdIter = 0;
   ushort numDataItems = 0;
   DKString strPid;
   DKPid* pid = 0;
   short sVal = 0;
   long lVal = 0;
   while (pCur->isValid())
   {
     p = pCur->fetchNext();
     if (p != 0)
     {
       cout << "==========> " << "Item " << cnt << " <==========" << endl;
       numDataItems = p->dataCount();
       pid = (DKPid*)p->getPidObject();
       strPid = pid->pidString();
       cout << "pid string " << strPid << endl;
       k = p->propertyId(DK_CM_PROPERTY_ITEM_TYPE);
       if (k > 0)
       {
         a = p->getProperty(k);
         val = a;
                   switch (val)
         {
           case DK_CM_DOCUMENT :
           {
             cout << "Item is document " << endl;
             break;
    }
       }
                }
       cout << "Number of Data Items " << numDataItems << endl;
       for (j = 1; j <= numDataItems; j++)
       {
         a = p->getData(j);
         strDataName = p->getDataName(j);
         switch (a.typeCode())
         {
           case DKAny::tc_string :
```

```
      {
               strData = a;
               cout << "attribute name : " << strDataName << " value : "
    << strData << endl;
               break;
      }
           // ---- Handle each type in a similar fashion
           . . .
         }
    }
    // ----- Delete the cursor and disconnect
    if (pCur != 0)
           delete pCur;
    dsDB2.disconnect();
}
catch(DKException &exc)
   . . .
```

Refer to TExecuteDB2.cpp,TExecuteODBC.cpp, and TExecuteDJ.cpp in the CMBROOT\Samples\cpp directories for complete examples.

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 cmbcm816.dll, non-debugged version, and cmbcm816d.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 DKDatastore*xx*, where *xx* identifies the name or type of the specific content server. Table 54 lists the content servers provided in Enterprise Information Portal.

Table 54. Enterprise Information Portal content servers

Server type	Class names
Content Manager	DKDatastoreICM
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

Table 54. Enterprise Information Portal content servers (continued)

ODBC	DKDatastoreODBC
earlier Content Manager	DKDatastoreDL

Common classes in Enterprise Information Portal

dkDDO

The dkDDO class provides a representation and a protocol to define and access an object's data, independent of the object's type. The DDO protocol is implemented as a set of functions to define, add, and access each data item of an object. You can use this protocol to dynamically create an object and get it from the content server regardless of the content server's type.

When implementing a content server, you can utilize schema mapping information, registered in the content server class. The schema maps each individual persistent data item to its underlying representation in the content server.

A DDO has a set of attributes; each attribute has a type, value, and properties associated with it. The DDO itself can have properties that belong to the DDO as a whole. For example, you can map which class to an item in Content Manager 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 functions.

The dkXDO class extends the public interface of dkXDOBase by defining independent content server access, create, retrieve, update, and delete functions. These functions enable an application to store and retrieve the object's data to and from a content server without the existence of an associated DDO class object or stand-alone XDO.

You must set the 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 function, so it can be considered a subclass of the query manager.

The main functions 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 functions.

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 function.

listDataSourceNames()
Returns an array of content server names

getExtension(String)
Gets the dkExtension object from the content server. If the given

extension does not already exist but is supported by the content server, a newly created object is returned, otherwise, a null value is returned.

addExtension(String, dkExtension)
Adds a new extension object (XDO) to this content server

CreateDDO(String,int)
Creates a data object based on the given object type and flag. Create DDO returns a new DKDDO object with all the properties and attributes set. The calling program must provide the attribute values for this data object.

The data object manipulation functions in the dkDatastore interface are:

addObject(dkDataObject)
Adds a new document or older 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 functions 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 functions to access content server information and to create, list, and delete its entities. It maintains a collection of dkEntityDef objects.

Table 55 contains examples of concrete classes for the dkDatastoreDef interface.

Table 55. Concrete classes for dkDatastoreDef

Server type	Class name
Content Manager	DKDatastoreDefICM
OnDemand	DKDatastoreDefOD
Content Manager for AS/400	DKDatastoreDefV4
ImagePlus for OS/390	DKDatastoreDefIP
Domino.Doc	DKDatastoreDefDD
Extended Search	DKDatastoreDefDES

Table 55. Concrete classes for dkDatastoreDef (continued)

DB2 Universal Database	DKDatastoreDefDB2,
DB2 DataJoiner	DKDatastoreDefDJ
ODBC	DKDatastoreDefODBC
earlier Content Manager	DKDatastoreDefDL

The main functions in the dkDatastoreDef interface are:

listEntities()
Lists entities

listEntityAttrs()
Lists entity attributes

addEntity()
Adds an entity

getEntity(*name*)
Gets an entity

Each concrete class can also have its own content server-specific functions with names that are familiar to that content server. For example, the DKDatastoreDefDL class contains these specific functions:

- listIndexClassNames()
- listIndexClasses()

The DKDatastoreDefOD class contains these specific functions:

- listAppGrps()
- listAppGrpNames()

dkEntityDef

The dkEntityDef class defines functions to:

- Create and delete the entity
- Access entity information
- Create and delete attributes

In the dkEntityDef class, all functions that are related to subentities generate a DKUsageError indicating that the default content server does not support subentities. However, if the content server does support this kind of multiple level entity, as does Domino.Doc, for example, the subclass for this content server must implement the proper functions to overwrite the exceptions.

Table 56 contains examples of concrete classes for the dkEntityDef class.

Table 56. Concrete classes for dkEntityDef

Server type	**Class name**
Content Manager	DKIndexClassDefICM
OnDemand	DKAppGrpDefOD
Content Manager for AS/400	DKIndexClassDefV4
ImagePlus for OS/390	DKEntityDefIP
Domino.Doc	DKCabinetDefDD

Table 56. Concrete classes for dkEntityDef (continued)

Extended Search	DKDatabaseDefDES
DB2 Universal Database	DKTableDefDB2, DKColumnDefDB2
DB2 DataJoiner	DKTableDefDJ, DKColumnDefDJ
ODBC	DKTableDefODBC, DKColumnDefODBC
earlier Content Manager	DKIndexClassDefDL

The main functions in the dkEntityDef class are:

ListAttrs()
Lists the entity attributes

getAttr(*String attrName***)**
Gets a specified entity attribute

addAttr(*DKAttrDef***)**
Adds an attribute to an entity

getName()
Gets the name of the entity

setName(*String***)**
Sets the name of the entity

hasSubEntities()
Determines whether the entity contains subentities

getSubEntity(*String***)**
Gets the subentity

addSubEntity(*dkEntityDef***)**
Adds a subentity to the entity

listSubEntities()
Lists the subentities of the entity

removeAttr(*String***)**
Removes a subentity from the entity

add() Adds the entity to the content server

update()
Updates the entity in the content server

retrieve()
Retrieves the entity values from the content server

del() Deletes the entity from the content server

dkAttrDef

The dkAttrDef class defines functions for accessing attribute information and creating and deleting attributes.

Table 57 contains examples of concrete classes for the dkAttrDef class.

Table 57. Concrete classes for dkAttrDef

Server type	**Class name**
Content Manager	DKAttributeDefICM
OnDemand	DKFieldDefOD

Table 57. Concrete classes for dkAttrDef (continued)

Content Manager for AS/400	DKAttrDefV4
ImagePlus for OS/390	DKAttrDefIP
Domino.Doc	DKAttrDefDD
Extended Search	DKFieldDefDES
earlier Content Manager	DKAttributeDefDL

The main functions 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 58 contains examples of concrete classes for the dkServerDef class.

Table 58. Concrete classes for dkServerDef

Server type	**Class name**
Content Manager	DKServerDefICM
OnDemand	DKServerDefOD
Content Manager for AS/400	DKServerDefV4
Domino.Doc	DKServerDefDD
Extended Search	DKServerDefDES
DB2 Universal Database	DKServerDefDB2
DB2 DataJoiner	DKServerDefDJ
ODBC	DKServerDefODBC
earlier Content Manager	DKServerDefDL

The main functions in the dkServerDef class are:

setDatastore(dkDatastore *ds***)**
Sets the reference to the content server object

getDatastore()
Gets the reference to the content server object

getName()
Gets the name of the content server

setName(String *name***)**
Sets the name of the content server

datastoreType()
Gets the content server type

dkResultSetCursor
dkResultSetCursor is a content server cursor in the query result set that you can use to manage a virtual collection of DDO objects. The collection is a query result set. Each element of the collection is not created until the content server retrieves the element.

The main functions in the dkResultSetCursor class are:

isScrollable()
Returns TRUE if the cursor can be scrolled forward and backward

isUpdatable()
Returns TRUE if the cursor can be updated

isValid()
Returns TRUE if the cursor is valid

isBegin()
Returns TRUE if the cursor is positioned at the beginning of the result set

isEnd()
Returns TRUE if the cursor is positioned at the end of the result set

isInBetween()
Returns TRUE if cursor is positioned between data elements in the result set

getPosition()
Gets the current position of the cursor

setPosition(int position, *Object value***)**
Sets the cursor to the specified position

setToNext()
Sets the cursor to point to the next element in the result set

fetchObject()
Retrieves the current element from the result set and returns it as a DDO

fetchNext()
Retrieves the next element from the result set and returns it as a DDO

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 result set handle that is associated with the resultset cursor, by type

Requirement: To use the addObject, deleteObject, and updateObject functions, 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 59 contains examples of concrete classes for the dkBlob class.

Table 59. Concrete classes for dkBlob

Server type	Class name
Content Manager	DKBlobICM
OnDemand	DKBlobOD
VisualInfo for AS/400	DKBlobV4
Content Manager for AS/400	DKBlobIP
Domino.Doc	DKBlobDD
Extended Search	DKBlobDES
DB2 Universal Database	DKBlobDB2
DB2 DataJoiner	DKBlobDJ
ODBC	DKBlobODBC
earlier Content Manager	DKBlobDL

The main functions in the dkBlob class are:

getContent()
Returns a byte array containing the BLOB data of the object

getContentToClientFile(String *afileName, int fileOption***)**
Copies the BLOB data from the object to the specified file

setContent(byte[] *aByteArr***)**
Sets the LOB data for the object with the contents of the byte array

setContentFromClientFile(String *afileName***)**
Replaces the LOB data of the object with the contents of the file *afileName*

add(String afileName)
Adds the content of the specified file to the content server

retrieve(String afileName)
Retrieves the content of the content server into the specified file

update(String afileName)
Updates the object and the content server with the content of the specified file

del(boolean flush)
Deletes the object's data from the content server, if *flush* is TRUE; otherwise the current content is preserved.

concatReplace(dkBlob aBlob), concatReplace(byte[] *aByteArr***)**
Concatenates this object with another dkBlob object or byte array

length()
Returns the length of the LOB content of the object

indexOf(String aString, int startPos), indexOf(dkBlob aBlob, int startPos)
Starting the search at offset start positions, returns the byte offset of the first occurrence of the search argument within this object

subString(int startPos, int length)
Returns a string object that contains a substring of the LOB data of this object

remove(int startPos, int aLength)
Starting at *startPos* for *aLength* bytes, deletes a portion of the LOB data of this object

insert(String *aString***, int startPos), insert(dkBlob aBlob, int** *startPos***)**
Inserts the argument data, following the *startPos* position in the LOB data of the object

open(String *afileName***)**
Unloads the object contents to the file *afileName* and then runs a default file handler

setClassOpenHandler(String aHandler, boolean newSynchronousFlag)
Identifies, by executable program name, the file handler for an entire class. This function also indicates whether to run the handler synchronously or asynchronously for the file object.

setInstanceOpenHandler(String aHandler, boolean newSynchronousFlag)
Identifies, by executable program name, the file handler and indicates whether to run it synchronously or asynchronously for this object

getOpenHandler()
Gets the executable program name of the file handler for an entire class

isOpenSynchronous()
Returns the current synchronization setting for the file handler

dkClob

dkClob is an abstract class that declares a public interface for storing character large object (CLOB) data types, such as documents.

Table 60 contains examples of concrete classes for the dkClob class.

Table 60. Concrete classes for dkClob

Server type	Class name
DB2 Universal Database	DKClobDB2
DB2 DataJoiner	DKClobDJ
ODBC	DKClobODBC

The main functions in the dkClob class are:

open() Open() is a member inherited from dkXDOBase. Open() will be implemented or overridden by concrete subclasses of dkClob.

dkXDO Members: dkXDO& add(), dkXDO retrieve(), dkXDO update(), dkXDO del()
Inherited as protected members from dkXDO. Where necessary, these protected members will be implemented or overridden by concrete subclasses of dkClob.

The following list contains members defined by dkClob:

add(String *afileName***)**
Adds the content of the specified file to the content server

retrieve(String *afileName***)**
Retrieves the content of the content server into the specified file

update(String *afileName***)**
Updates the object and the content server with the content of the specified file

del(DKBoolean flush)
Deletes the object's data from the content server, if *flush* is TRUE; otherwise the current content is preserved.

getContentToClientFile(String *afileName, int fileOption***)**
Copies the CLOB data from the object to the specified file

setContentFromClientFile(String *afileName***)**
Replaces the LOB data of the object with the contents of the file *afileName*

indexOf(String& *aString***, long startPos=1), indexOf(dkClob&** *adkClob***, long startpos=1)**
Starting the search at offset start positions, returns the byte offset of the first occurrence of the search argument within this object,

subString(long startpos, long length)
Returns a string object that contains a substring of the LOB data of this object

remove(long startpos, long *aLength***)**
Starting at *startPos* for *aLength* bytes, deletes a portion of the LOB data of this object.

insert(DKString *aString***, long startpos), insert(dkClob&** *adkClob***, long startpos)**
Inserts the argument data following the *startPos* position in the CLOB data of the object

open(String *afileName***)**
Unloads the object contents to the file *afileName* and then runs a default file handler

setInstanceOpenHandler(String *ahandler***, DKBoolean newSynchronousFlag)**
Identifies, by executable program name, the file handler and indicates whether to run it synchronously or asynchronously for this object

setClassOpenHandler(String *ahandler***, DKBoolean newSynchronousFlag)**
Identifies, by executable program name, the file handler for an entire class. This function also indicates whether to run the handler synchronously or asynchronously for the file object.

getOpenHandler()
Gets the executable program name of the file handler for an entire class

isOpenSynchronous()
Returns the current synchronization setting for the file handler

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

dkDatastoreExt
The dkDatastoreExt class defines the standard datastore extension classes.

Table 61 contains examples of concrete classes for the dkDatastoreExt class.

Table 61. Concrete classes for dkDatastoreExt

Server type	Class name
Content Manager	DKDatastoreExtICM
OnDemand	DKDatastoreExtOD
Content Manager for AS/400	DKDatastoreExtV4
ImagePlus for OS/390	DKDatastoreExtIP
Domino.Doc	DKDatastoreExtDD
Extended Search	DKDatastoreExtDES
DB2 Universal Database	
DB2 DataJoiner	
ODBC	
earlier Content Manager	DKDatastoreExtDL

The main functions in the dkDatastoreExt class are:

getDatastore()
Gets the reference to the owning content server object

setDatastore(dkDatastore ds)
Sets the reference to the owning content server object

isSupported(String functionName)
Determines whether the specified function name is supported by this extension

listFunctions()
Lists all supported function names for the extension

addToFolder(dkDataObject folder, dkDataObject member)
Adds a member to this folder and to the content server

removeFromFolder(dkDataObject folder, dkDataObject member)
Removes a member from this folder and the content server

checkOut(dkDataObject item)
Checks out a document or folder item from the content server. While the item is checked out, you have exclusive updating privileges to the item and other users have read access only.

checkIn(dkDataObject item)
Checks in a document or folder item previously checked out from the content server. By checking in the file, you release all write privileges with this function.

getCommonPrivilege()
Gets the common privilege of a specific content server

isCheckedOut(dkDataObject item)
Determines whether a document or folder item was checked out from the content server

checkedOutUserid(dkDataObject item)
Gets the user ID that checked out the item from the content server

unlockCheckedOut(dkDataObject item)
Unlocks the item from the content server

changePassword (String userId, String oldPwd, String newPwd)
Changes the password on the content server for the specified user ID

moveObject (dkDataObject dataObj, String *entityName***)**
Moves the *entityName* object from one entity to another

retrieveFormOverlay(String id)
Retrieves the form overlay object

DKPidXDO

The DKPidXDO class represents the persistent identification of the BLOB data in the content server.

Table 62 contains examples of concrete classes for the DKPidXDO class.

Table 62. Concrete classes for DKPidXDO

Server type	Class name
Content Manager	

Table 62. Concrete classes for DKPidXDO (continued)

OnDemand	DKPidXDOOD
Content Manager for AS/400	DKPidXDOV4
Content Manager ImagePlus for OS/390	DKPidXDOIP
Domino.Doc	DKPidXDODD
Extended Search	DKPidXDODES
DB2 Universal Database	DKPidXDODB2
DB2 DataJoiner	DKPidXDODJ
ODBC	DKPidXDOODBC
earlier Content Manager	DKPidXDODL

dkUserManagement

The dkUserManagement class represents and processes all of the content server's user management functions.

Table 63 contains examples of concrete classes for the dkUserManagement class.

Table 63. Concrete classes for dkUserManagement

Server type	Class name
Content Manager	DKUserMgmtICM
Content Manager for AS/400	DKUserMgmtV4
ImagePlus for OS/390	DKUserMgmtIP
DB2 Universal Database	
DB2 DataJoiner	
ODBC	
earlier Content Manager	DKUserMgmtDL

DKConstant

All common constants are defined in the DKConstant class. Each content server has its own DKConstant*xx* class for defining constants specific to that content server.

Recommendation: All content servers use the common messages whenever possible.

DKMessageId

All common message IDs are defined in this class. Each content server has its own DKMessageId*xx* class for defining its own message IDs.

Recommendation: All content servers 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.

Chapter 15. Using Enterprise Information Portal workflow in C++

You can use the EIP C++ classes and 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 an instance of DKWorkFlowServicesFed and connect to it. The following example starts workflow services:

```
 // ----- Create the strings for the name of the service, user ID
// -----    and Password
DKString wfsrv = "icmnlsdb";
DKString userid = "cmbadmin";
DKString pw = "password";
// ----- Create a federated datastore
DKDatastoreFed* dsFed = new DKDatastoreFed();
dsFed->connect(wfsrv, userid, pw,"");
//----- Create the workflow service
DKWorkFlowServiceFed* svWF =new DKWorkFlowServiceFed ();
// ----- Set the datastore in the workflow service
svWF->setDatastore(dsFed);
// ----- Connect to the service
svWF->connect (wfsrv, userid, pw,"");
```

When you are finished using the workflow service, you must disconnect by calling the disconnect() and the delete() functions.

```
 svWF->disconnect();
dsFed->disconnect();
delete svWF;
delete dsFed;
```

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, which is a workflow definition defined in the EIP workflow builder.
3. Set the PID 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
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF);
WF->setName("wfl");
// ----- Create an instance of a workflow with the workflow template name
WF->add("WD1");
```

```
// ----- Refresh the workflow object
WF->retrieve();
// ----- Construct the container object for the workflow
DKWorkFlowContainerFed* con = WF.inContainer();
// ----- Retrieve the container data
con->retrieve();
// ----- Add a PID string referring to a content item from Extended Search
con->setPersistentID("45 3 DES4ross10 Notes Help18 15 Help|23fa");
// ----- Assign a priority of 100
con->setPriority(100);
// ----- Update the container
con->update();
// ----- Start the workflow
WF->start(con);
. . .
// ----- When you are done, clean up by deleting the connection and workflow
delete con;
delete WF;
```

Terminating a workflow

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

```
//-----Retrieve the status of the workflow named WF
WF.retrieve();
int state =WF.state();
//-----Check the status and either terminate or delete
if (state ==DKConstantFed.DK_FED_FMC_PS_RUNNING ||
    state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDED ||
    state ==DKConstantFed.DK_FED_FMC_PS_SUSPENDING)
{
    WF.terminate();
}
if (state ==DKConstantFed.DK_FED_FMC_PS_READY ||
    state ==DKConstantFed.DK_FED_FMC_PS_FINISHED||
    state ==DKConstantFed.DK_FED_FMC_PS_TERMINATED)
{
    WF.del();
}
```

Listing all the workflows

You can list all the workflows in a workflow service by using the listWorkFlows() function. The following example lists the name and description of all the workflows in a workflow service referenced by the DKWorkFlowSerivceFed object svWF.

```
// ----- Call the listWorkFlows function
DKSequentialCollection *collWF =
        (DKSequentialCollection*)svWF.listWorkFlows();
DKWorkFlowFed *WF = NULL;
if (collWF != NULL)
{
    dkIterator *iterWF = collWF->createIterator();
    while (iterWF->more())
    {
       WF = (DKWorkFlowFed*)(void*)((*iterWF->next()));
       WF->retrieve();
       cout << "name = " + WF->getName()
  << " description = " << WF->getDescription() << endl;
       delete WF;
    }
    delete iterWF;
}
delete collWF;
```

Suspending a workflow

You can suspend a running workflow with a specific time stamp or can suspend it indefinitely. The following example illustrates how to suspend a workflow until a certain time. If you provide a null DKTimestamp, the workflow is suspended indefinitely.

```
// ----- Construct a DKWorkFlowFed instance
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF, "Test");
WF->retrieve();
// ----- Call the suspend function if the workflow is in
the running state
if (WF->state() == DK_FED_FMC_PS_RUNNING)
{
  // ----- Suspended until 2000-07-27-16.30.00.000000
DKTimestamp* suspension = new DKTimestamp(2000, 7,
27, 16, 30, 0, 0);
  WF->suspend(suspension);
  delete suspension;
}
delete WF;
```

Resuming a workflow

You can resume a suspended workflow by calling the resume() function. The following example shows resuming a suspended workflow:

```
// ----- Construct a DKWorkFlowFed instance
DKWorkFlowFed* WF = new DKWorkFlowFed(svWF, "Test");
WF->retrieve();
// ----- Check whether the workflow is suspended and call resume
if (WF->state() == DK_FED_FMC_PS_SUSPENDED)
{
  WF->resume();
}
delete WF;
```

Listing all the worklists

You can list all the worklists in a workflow service by calling the listWorkLists() function on the workflow service. The following example lists the name and description of all the worklists in a workflow service referenced by the DKWorkFlowServiceFed instance svWF:

```
// ----- Call the listWorkLists function
DKSequentialCollection *collWL =
      (DKSequentialCollection*)svWF.listWorkLists();
DKWorkListFed *WL = NULL;
if (collWL != NULL)
{
   dkIterator *iterWL = collWL->createIterator();
   while (iterWL->more())
   {
      WL = (DKWorkListFed*)(void*)((*iterWL->next()));
      WL->retrieve();
      cout << "name = " << WL->getName() << " description = "
             << WL->getDescription() << endl;
      cout << "Threshold = " << WL->getThreshold() << endl;
      delete WL;
    }
    delete iterWL;
}
delete collWL;
```

Accessing a worklist

You can access a worklist by creating an instance of DKWorkListFed that refers to the worklist you created using system administration client. 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
cout << "worklist name = " << WL->getName() << endl;
cout << "description = " << WL->getDescription() <<
        " owner = " << WL->getOwner() <<
        " filter = " << WL->getFilter() <<
        " threshold = " << WL->getThreshold() <<
        " sort criteria = " << WL->getSortCriteria() << endl;
// -----  Delete the worklist when you are done
delete WL;
```

Accessing work items

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

```
DKSequentialCollection *coll;
      dkIterator *iter;
      DKWorkItemFed* item;
      DKString nodename;
      DKString workflowname;
      // ----- Create a collection and an iterator
      coll = (DKSequentialCollection*)WL->listWorkItems();

      if (coll != NULL)
      {
         iter = coll->createIterator();
         cout << "listWorkItems" << endl;
         // ----- Step through the collections
         while (iter->more ())
         {
           item = (DKWorkItemFed*)((void*)(*iter->next()));

           if (item != NULL)
           {
              //item.retrieve ();
              nodename = item->name();
              workflowname = item->workFlowName();
              cout << "workitem node = " << nodename
                   << "  workflow name = " << workflowname << endl;
              delete item;
           }
         }
         delete iter;
         delete coll;
      }
```

Moving items in the workflow

As a workflow advances, you move work items from one activity to the next by using the checkOut() and checkIn() functions. The following examples illustrates moving work item:

```
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);
delete item;
```

Listing all the workflow templates

You can list all the workflow templates in a workflow service by calling the listWorkFlowTemplates() function. The following example lists the name and description of all the workflow templates in a workflow service referenced by the DKWorkFlowSerivceFed object svWF.

```
// ----- Call the listWorkFlowTemplates function
DKSequentialCollection *collWT =
      (DKSequentialCollection*)svWF.listWorkFlowTemplates();
DKWorkFlowTemplateFed *WT = NULL;

if (collWT != NULL)
{
    dkIterator* iterWT = collWT->createIterator();
    while (iterWT->more())
    {
       WT = (DKWorkFlowTemplateFed*)(void*)((*iterWT->next()));
       WT->retrieve();
       cout << "name = " << WT->name() << " description = "
                << WT->description() << endl;
       delete WT;
    }
    delete iterWT;
 }
 delete collWT;
```

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
San Jose, CA 95141
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM	DisplayWrite	PowerPC
400	e-business	PTX
Advanced Peer-to-Peer Networking	HotMedia	QBIC
AIX	Hummingbird	RS/6000
AIXwindows	ImagePlus	SecureWay
APPN	IMS	SP
AS/400	Micro Channel	VideoCharger
C Set ++	MQSeries	Visual Warehouse
CICS	MVS/ESA	VisualAge
DATABASE 2	NetView	VisualInfo
DataJoiner	OS/2	WebSphere
DB2	OS/390	
DB2 Universal Database	PAL	

Approach, Domino, Lotus, Lotus 1-2-3, Lotus Notes and SmartSuite are trademarks or registered trademarks of the Lotus Development Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines terms and abbreviations specific to this system. Terms shown in *italics* are defined elsewhere in this glossary.

A

abstract class. An object-oriented programming *class* that represents a concept; classes derived from it represent implementations of the concept. You cannot construct an object of an abstract class; that is, it cannot be instantiated.

access control. The process of ensuring that certain functions and stored *objects* can be accessed only by authorized users in authorized ways.

access control list. A list consisting of one or more user IDs or user groups and their associated *privileges.* You use access control lists to control user access to *items* and *objects* in the Content Manager system. You use access control lists to control user access to *search templates* in the Enterprise Information Portal system.

action list. An approved list of the actions, defined by a system administrator or some other *workflow coordinator,* that a user can perform in a *workflow* or document routing process.

ADSM. See *Tivoli® Storage Manager.*

API. See *application programming interface.*

application programming interface (API). A software interface that enables applications to communicate with each other. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by the underlying licensed program.

archive. Persistent storage used for long-term information retention, typically very inexpensive for each stored unit and slow to access, and often in a different geographic location to protect against equipment failures and natural disasters.

attribute. A unit of data that describes a certain characteristic or property (for example, name, address, age, and so forth) of an item, and which can be used to locate that item. An attribute has a type, which indicates the range of information stored by that attribute, and a value, which is within that range. For example, information about a file in a multimedia file system, such as title, running time, or encoding type (MPEG1, H.263, and so forth). For Enterprise Information Portal, see also *federated attribute* and *native attribute.*

attribute group. Convenience grouping of one or more *attributes.* For example, Address might include the attributes Street, City, State, and Zip.

Audio/Video Interleaved (AVI). A RIFF (*Resource Interchange File Format*) file specification that permits audio and video data to be interleaved in a file. The separate tracks can be accessed in alternate chunks for playback or recording while maintaining sequential access on the file device.

AVI. See *Audio/Video Interleaved.*

B

base attributes. A set of indexes that is assigned to each *object*. All Content Manager objects have base *attributes.*

binary large object (BLOB). A sequence of bytes with a size ranging from 0 bytes to 2 gigabytes. This string does not have an associated code page and character set. Image, audio, and video objects are stored in BLOBs.

BLOB. See *binary large object.*

C

cache. A special-purpose buffer, smaller and faster than main storage, used to hold a copy of data that can be accessed frequently. Use of a cache reduces access time, but might increase memory requirements. See also *resource manager cache* and *LAN cache.*

cardinality. The number of rows in a database table.

category. See *item type.*

CGI. See *Common Gateway Interface.*

CGI script. A computer program that runs on a Web server and uses the *Common Gateway Interface (CGI)* to perform tasks that are not usually done by a Web server (for example, database access and form processing). A CGI script is a CGI program that is written in a scripting language such as Perl.

child component. Optional second or lower level of a hierarchical *item type.* Each child component is directly associated with the level above it.

CIF. See *common interchange file.*

CIU. See *common interchange unit.*

class. In object-oriented design or programming, a model or template that can be instantiated to create objects with a common definition and therefore, common properties, operations, and behavior. An object is an instance of a class.

client application. An application written with the Content Manager APIs to customize a user interface. An application written with the object-oriented or Internet APIs to access *content servers* from Enterprise Information Portal.

Client Application for Windows. A complete object management system provided with Content Manager and written with Content Manager APIs. It supports document and folder creation, storage, and presentation, processing, and access control. You can customize it with user exit routines and partially invoke it with APIs.

client/server. In communications, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server.

collection. A group of objects with a similar set of management rules.

combined search. A query that combines one or more of the following types of searches: *parametric,* text, or image.

Common Gateway Interface (CGI). A standard for the exchange of information between a Web server and programs that are external to it. The external programs can be written in any programming language that is supported by the operating system on which the Web server is running. See *CGI script.*

common interchange file (CIF). A file that contains one ImagePlus Interchange Architecture (IPIA) data stream.

common interchange unit (CIU). The independent unit of transfer for a common interchange file (CIF). It is the part of the CIF that identifies the relationship to the receiving database. A CIF can contain multiple CIUs.

component. Generic term for a *root component* or a *child component.*

connection manager. A Content Manager component that helps maintain connections to the library server, rather than starting a new connection for each query. The connection manager has an application programming interface.

connector class. Object-oriented programming *class* that provides standard access to APIs that are native to specific *content servers.*

constructor. In programming languages, a method that has the same name as a class and is used to create and initialize objects of that class.

container. An element of the user interface that holds objects. In the *folder manager,* an *object* that can contain other folders or documents.

content class. See *MIME type.*

content server. A software system that stores multimedia and business data and the related metadata required for users to work with that data. Content Manager and Content Manager ImagePlus for OS/390 are examples of content servers.

cursor. A named control structure used by an application program to point to a specific row within some ordered set of rows. The cursor is used to retrieve rows from the set.

D

data format. See *MIME type.*

datastore. (1) Generic term for a place (such as a database system, file, or directory) where data is stored. (2) In an application program, a virtual representation of a *content server.*

DCA. See *document content architecture.*

DDO. See *dynamic data object.*

destager. A function of the Content Manager *resource manager* that moves objects from the *staging area* to the first step in the object's *migration policy*.

device manager. In a Content Manager system, the interface between the *resource manager* and one or more physical devices.

document. An *item* that can be stored, retrieved, and exchanged among Content Manager systems and users as a separate unit. 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).

document routing process. In Content Manager a sequence of *work steps*, and the rules governing those steps, through which a *document* or *folder* travels while it is being processed.

document type definition (DTD). The rules that specify the structure for a particular class of XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation can be used within the particular class of documents. A DTD is analogous to a database schema in that the DTD completely describes the structure for a particular markup language.

DTD. See *document type definition.*

dynamic data object (DDO). In an application program, a generic representation of a stored object that is used to move that object in to, and out of, storage.

E

element. An *object* that the *list manager* allocates for an application.

extended data object (XDO). In an application program, a generic representation of a stored complex multimedia *object* that is used to move that object in to, and out of, storage. XDOs are most often contained within *DDOs.*

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that was derived from, and is a subset of, SGML. XML omits the more complex and less-used parts of SGML and makes it much easier to write applications to handle document types, author and manage structured information, and transmit and share structured information across diverse computing systems. The use of XML does not require the robust applications and processing that is necessary for SGML. XML is being developed under the auspices of the World Wide Web Consortium (W3C).

F

feature. The visual content information that is stored in the image search server. Also, the visual traits that image search applications use to determine matches. The four *QBIC* features are average color, histogram color, positional color, and texture.

federated attribute. An Enterprise Information Portal metadata category that is mapped to *native attributes* in one or more *content servers.* For example, the federated attribute, `policy number`, can be mapped to an *attribute*, `policy num`, in Content Manager and to an attribute, `policy ID`, in Content Manager ImagePlus for OS/390.

federated collection. A grouping of objects that results from a *federated search.*

federated datastore. Virtual representation of any number of specific *content servers*, such as Content Manager.

federated entity. An Enterprise Information Portal metadata object that is comprised of *federated attributes* and optionally associated with one or more *federated text indexes.*

federated search. A query issued from Enterprise Information Portal that simultaneously searches for data in one or more *content servers*, which can be heterogeneous.

federated text index. An Enterprise Information Portal metadata object that is mapped to one or more *native text indexes* in one or more *content servers.*

file system. In AIX, the method of partitioning a hard drive for storage.

folder. A container used to organize *objects*, which can be other folders or *documents.* A folder can be indexed by *attributes.*

folder manager. The Content Manager model for managing data as online documents and folders. You can use the folder manager APIs as the primary interface between your applications and the Content Manager content servers.

H

handle. A character string that represents an object, and is used to retrieve the object.

history log. A file that keeps a record of activities for a *workflow.*

HTML. See *Hypertext Markup Language.*

Hypertext Markup Language (HTML). A markup language that conforms to the SGML standard and was designed primarily to support the online display of textual and graphical information that includes hypertext links.

I

Image Object Content Architecture (IOCA). A collection of constructs used to interchange and present images.

index. To add or edit the attribute values that identify a specific *item* or *object* so that it can be retrieved later.

index class. See *item type.*

index class subset. In earlier Content Manager, a view of an *index class* that an application uses to store, retrieve, and display folders and objects.

index class view. In earlier Content Manager, the term used in the APIs for *index class subset*.

information mining. The automated process of extracting key information from text (summarization), finding predominant themes in a collection of documents (categorization), and searching for relevant documents using powerful and flexible queries.

inline. In Content Manager, an object that is online and in a drive, but has no active *mounts*. Contrast with *mounted*.

interchange. The capability to import or export an image with its index from one Content Manager ImagePlus for OS/390 system to another ImagePlus system using a *common interchange file* or *common interchange unit*.

IOCA. See *Image Object Content Architecture*.

item. In Content Manager, generic term for an instance of an *item type*. For example, an item might be a *folder*, *document*, video, or image. Generic term for the smallest unit of information that Enterprise Information Portal administers. Each item has an identifier. For example, an item might be a *folder* or a *document*.

item type. A template for defining and later locating like *items*, consisting of a *root component*, zero or more *child components*, and a classification.

item type classification. A categorization within an *item type* that further identifies the *items* of that item type. All items of the same item type have the same item type classification.

Content Manager supplies the following item type classifications: *folder*, *document*, object, video, image, and text; users can also define their own item type classifications.

iterator. A class or construct that you use to step through a collection of objects one at a time.

J

JavaBeans. A platform-independent, software component technology for building reusable Java components called "beans." After they are built, these beans can be made available for use by other software engineers or can be used in Java applications. Using JavaBeans, software engineers can manipulate and assemble beans in a graphical drag-and-drop development environment.

Joint Photographic Experts Group (JPEG). (1) A group that worked to establish the standard for the compression of digitized continuous-tone images. (2) The standard for still pictures developed by this group.

JPEG. See *Joint Photographic Experts Group*.

K

key field. See *attribute*.

L

LAN. See *local area network*.

LAN cache. An area of temporary storage on a local *resource manager* that contains a copy of objects stored on a remote resource manager.

library client. The component of a Content Manager system that provides a low-level programming interface for the library system. The library client includes APIs that are part of the software developer's kit.

library object. See *item*.

library server. The component of a Content Manager system that stores, manages, and handles queries on *items*.

link. A directional relationship between two *items:* the parent and the child. You can use a set of links to model one-to-many associations. Contrast with *reference*.

local area network (LAN). A network in which a set of devices are connected to one another for communication and that can be connected to a larger network.

M

machine-generated data structure (MGDS). (1) An IBM structured data format protocol for passing character data among the various Content Manager ImagePlus for OS/390 programs. (2) Data extracted from an image and put into general data stream (GDS) format.

management class. The term used in the APIs for *migration policy*.

media archiver. A physical device that is used for storing audio and video stream data. The VideoCharger is a type of media archiver.

media server. An AIX-based component of the Content Manager system that is used for storing and accessing video files.

method. In Java design or programming, the software that implements the behavior specified by an operation. Synonymous with member function in C++.

MGDS. See *machine-generated data structure*.

migration. (1) The process of moving data and source from one computer system to another computer system without converting the data, such as when moving to a new operating environment. (2) Installation of a new version or release of a program to replace an earlier version or release.

migration policy. A user-defined schedule for moving *objects* from one *storage class* to the next. It describes the retention and class transition characteristics for a group of objects in a storage hierarchy.

migrator. A function of the *resource manager* that checks *migration policies* and moves objects to the next *storage class* when they are scheduled to move.

MIME type. An Internet standard for identifying the type of object being transferred across the Internet. MIME types include several variants of audio, image, and video. Each object has a MIME type.

Mixed Object Document Content Architecture (MO:DCA). An IBM architecture developed to allow the interchange of object data among applications within the interchange environment and among environments.

Mixed Object Document Content Architecture–Presentation (MO:DCA–P). A subset architecture of MO:DCA that is used as an envelope to contain documents that are sent to the Content Manager ImagePlus for OS/390 workstation for displaying or printing.

MO:DCA. *Mixed Object Document Content Architecture*

MO:DCA–P. *Mixed Object Document Content Architecture—Presentation*

mount. To place a data medium in a position to operate.

mounted. In Content Manager, an object that is online and in a drive, with active *mounts*. Contrast with *inline*.

multimedia. Combining different media elements (text, graphics, audio, still image, video, animation) for display and control from a computer.

multimedia file system. A *file system* that is optimized for the storage and delivery of video and audio.

Multipurpose Internet Mail Extensions (MIME) . See *MIME type.*

N

native attribute. A characteristic of an object that is managed on a specific *content server* and that is specific to that content server. For example, the *key field* `policy num` might be a native attribute in a Content Manager content server, whereas the field `policy ID` might be a native attribute in an Content Manager OnDemand content server.

native entity. An *object* that is managed on a specific *content server* and that is comprised of *native attributes.* For example, Content Manager *index classes* are native entities comprised of Content Manager *key fields.*

native text index. An index of the text *items* that are managed on a specific *content server.* For example, a single text search index on a Content Manager content server.

network table file. A text file that contains the system-specific configuration information for each node in a Content Manager system. Each node in the system must have a network table file that identifies the node and lists the nodes that it needs to connect to.

The name of a network table is FRNOLINT.TBL.

O

object. Any digital content that a user can store, retrieve and manipulate as a single unit, for example, *JPEG* images, MP3 audio, *AVI* video, and a text block from a book.

Object Linking and Embedding (OLE). A Microsoft specification for both linking and embedding applications so that they can be activated from within other applications.

object server. See *resource manager.*

object server cache. See *resource manager cache.*

OLE. See *Object Linking and Embedding.*

overlay. A collection of predefined data such as lines, shading, text, boxes, or logos, that can be merged with variable data on a page during printing.

P

package. A collection of related *classes* and interfaces that provides access protection and namespace management.

parametric search. A query for *objects* that is based on the *properties* of the objects.

part. See *object.*

patron. The term used in the Content Manager APIs for *user*.

persistent identifier (PID). An identifier that uniquely identifies an *object,* regardless of where it is stored. The PID consists of both an item ID and a location.

PID. See *persistent identifier.*

privilege. The right to access a specific *object* in a specific way. Privileges includes rights such as creating, deleting, and selecting objects stored in the system. Privileges are assigned by the administrator.

privilege set. A collection of *privileges* for working with system components and functions. The administrator assigns privilege sets to users (user IDs) and *user groups*.

property. A characteristic of an *object* that describes the object. A property can be changed or modified. Type style is an example of a property.

purger. A function of the *resource manager* that removes *objects* from the system.

Q

QBIC. See *query by image content*.

query by image content (QBIC). A query technology that enables searches based on visual content, called features, rather than plain text. Using QBIC, you can search for objects based on their visual characteristics, such as color and texture.

query string. A character string that specifies the properties and property values for a query. You can create the query string in an application and pass it to the query.

R

rank. An integer value that signifies the relevance of a given part to the results of a query. A higher rank signifies a closer match.

README file. A file that should be viewed before the program associated with it is installed or run. A README file typically contains last-minute product information, installation information, or tips for using the product.

reference. Single direction, one-to-one association between a root or *child component* and another *root component*. Contrast with *link*.

release. To remove suspend criteria from an *item*. A suspended item is released when the criteria have been met, or when a user with proper authority overrides the criteria and manually releases it.

Remote Method Invocation (RMI). A set of APIs that enables distributed programming. An object in one Java Virtual Machine (JVM) can invoke methods on objects in other JVMs.

render. To take data that is not typically image-oriented and depict or display it as an image. In Content Manager, word-processing documents can be rendered as images for display purposes.

Resource Interchange File Format (RIFF) . Used for storing sound or graphics for playback on different types of computer equipment.

resource manager. The component of a Content Manager system that manages *objects*. These objects are referred to by *items* stored on the *library server*.

resource manager cache. The working storage area for the *resource manager*. Also called the *staging area*.

RIFF. See *Resource Interchange File Format*.

RMI server. A server that implements the Java *Remote Method Invocation (RMI)* distributed object model.

root component. The first or only level of a hierarchical *item type*, consisting of related system- and user-defined *attributes*.

S

search criteria. In Content Manager, *attribute* values that are used to retrieve a stored *item*. In Enterprise Information Portal, specific fields that an administrator defines for a *search template* that limit or further define choices available to the *users*.

search template. A form, consisting of *search criteria* designed by an administrator, for a specific type of federated search. The administrator also identifies the *users* and *user groups* who can access each search template.

semantic type. The usage or rules for an *item*. Base, annotation, and note are semantic types supplied by Content Manager; users can also define their own semantic types.

server definition. The characteristics of a specific *content server* that uniquely identify it to Enterprise Information Portal.

server inventory. The comprehensive list of *native entities* and *native attributes* from specified *content servers*.

server type definition. The list of characteristics, as identified by the administrator, required to uniquely identify a custom server of a certain type to Enterprise Information Portal.

SMS. See *system-managed storage*.

staging. The process of moving a stored *object* from an offline or low-priority device back to an online or higher priority device, usually on demand of the system or on request of a user. When a user requests an object stored in permanent storage, a working copy is written to the *staging area*.

staging area. The working storage area for the *resource manager*. Also referred to as *resource manager cache*.

stand-alone system. A preconfigured Content Manager system that installs all of the components of a Content Manager system on a single personal computer.

storage class. Identifies the type of media that an object is stored on. It is not directly associated with a physical location; however, it is directly associated with the *device manager*. Types of storage classes include:

DASD

Fixed Disk

Optical

Stream

Tape

TSM

storage group. Associates a storage system to a storage class.

storage system. A generic term for storage in the Content Manager system. See *TSM volume, media archiver*, and *volume.*

streamed data. Any data sent over a network connection at a specified rate. A stream can be one data type or a combination of types. Data rates, which are expressed in bits per second, vary for different types of streams and networks.

subclass. A *class* that is derived from another class. One or more classes might be between the class and subclass.

superclass. A *class* from which a class is derived. One or more classes might be between the class and superclass.

suspend. To remove an *object* from its *workflow* and define the suspension criteria needed to activate it. Later activating the object enables it to continue processing.

system-managed storage (SMS). The Content Manager approach to storage management. The system determines object placement, and automatically manages object backup, movement, space, and security.

T

table of contents (TOC). The list of *documents* and *folders* that are contained in a folder or *workbasket*. Search results are displayed as a folder table of contents.

thin client. A client that has little or no installed software but has access to software that is managed and delivered by network servers that are attached to it. A thin client is an alternative to a full-function client such as a workstation.

Tivoli Storage Manager (TSM). A *client/server* product that provides storage management and data access services in a heterogeneous environment. It supports various communication methods, provides administrative facilities to manage the backup and storage of files, and provides facilities for scheduling backup operations.

TOC. See *table of contents.*

TSM. See *Tivoli Storage Manager.*

TSM volume. A logical area of storage that is managed by *Tivoli Storage Manager.*

U

uniform resource locator (URL). A sequence of characters that represent information resources on a computer or in a network such as the Internet. This sequence of characters includes the abbreviated name of the protocol used to access the information resource and the information used by the protocol to locate the information resource. For example, in the context of the Internet, these are abbreviated names of some protocols used to access various information resources: http, ftp, gopher, telnet, and news.

user. A person who requires the services of Content Manager. This term generally refers to users of client applications, rather than the developers of applications, who use the Content Manager APIs. In Enterprise Information Portal, anyone who is identified in the Enterprise Information Portal administration program.

user exit. A point in an IBM-supplied program at which a user exit routine can be given control.

user exit routine. A user-written routine that receives control at predefined *user exits.*

user group. A group consisting of one or more defined individual *users*, identified by a single group name.

user mapping. Associating Enterprise Information Portal user IDs and passwords to corresponding user IDs and passwords in one or more content servers. User mapping enables single logon to Enterprise Information Portal and multiple *content servers.*

utility server. A Content Manager component that is used by the database utilities for scheduling purposes. You configure a utility server when you configure a *resource manager* or *library server*. There is one utility server for each resource manager and each library server.

V

volume. A representation of an actual physical storage device or unit on which the objects in your system are stored.

W

wildcard character. A special character such as an asterisk (*) or a question mark (?) that can be used to represent one or more characters. Any character or set of characters can replace a wildcard character.

workbasket. A collection of *documents* or *folders* that are either in process or waiting to be processed. A workbasket definition includes the rules that govern the presentation, status, and security of its contents.

workflow. In earlier Content Manager, a sequence of *workbaskets* through which a *document* or *folder* travels while it is being processed. In Enterprise Information Portal, a sequence of *work steps*, and the rules governing those steps, through which a *work packet*, *document*, or *folder* travels while it is being processed.

For example, `claims approval` would describe the process that an individual insurance claim must follow for approval.

workflow coordinator. In earlier Content Manager workflow, a user who receives notification that a *work item* in the *workflow* has not been processed in some specified time. The user is selected for a specific *user group* or upon creation of the workflow.

workflow state. The status of an entire *workflow*.

work item. In earlier Content Manager workflow and Enterprise Information Portal advanced workflow, any work activity that is active within a *workflow.*

worklist. A collection of *work items*, *documents*, or *folders* that are assigned to a user.

work packet. In Enterprise Information Portal Version 7.1, a collection of *documents* that is routed from one location to another. Users access and work with work packets through *worklists.*

work state. The status of an individual *work item*, *document*, or *folder*.

work step. A discrete point in a *workflow* or *document routing process* through which an individual *work item*, *document*, or *folder* must pass.

X

XDO. See *extended data object.*

XML. See *Extensible Markup Language.*

Index

A

accessing a worklist 230, 556
actions, creating 231
Ad-hoc routing, see Content Manager 137, 464
administration client
  description 4
administration database, description 4
advanced search sample in information mining 365
Annotating documents 309
annotation object, adding 38
application building
  visual beans 257
application building, non-visual beans 238
application programming interfaces (API)
  C++ 265
  Java 23
    architecture 23
    differences from C++ 24
    multiple search 26, 267
    packaging 24
attributes
  Java
    listing 30
  mapping 16
attributes, listing 509, 513, 519

B

beans
  common behaviors 255
  default viewers 254
  external viewers 254
  information mining 333
buffer, adding an XDO 37
building an application
  information mining 333
  visual beans 257
building an application, non-visual beans 238

C

C++ 265
  exceptions 268
cabinet attributes, listing 520
categorization sample in information mining 337
class name terminology 141, 467
client 23
clustering sample in information mining 352
CMBConnection 235
CMBConnectionPool 235
CMBDataManagement 235
CMBDocumentServices 235
CMBDocumentViewer 246
  specifications 253
CMBDocumentViewer *(continued)*
  terminating 253
  using 252
cmbenv81.csh 266
cmbenv81.ksh 266
cmbenv81.sh 25, 266
CMBExceptionHandler 235
CMBFolderViewer 246
  using 251
CMBItemAttributesEditor
  using 254
CMBLogonPanel 246
  using 246
CMBQueryService 235
cmbregist81.bat 26
cmbregist81.sh 26
CMBSchemaManagement 235
CMBSearchResults 235
CMBSearchResultsViewer 246
  using 250
CMBSearchTemplateList 246
  using 248
CMBSearchTemplateViewer 246
  using 249
CMBTraceLog 235
CMBUserManagement 235
CMBWorkFlowDataManagement 236
CMBWorkFlowQueryService 236
code page conversion 267
collections and iterators
  C++ 293
    memory management 295
    sequential collection 294
    sequential iterator 294
    sorting 296
  Java 53
    sequential collection 54
    sequential iterator 54
    Sorting 55
Combined query
  C++ 499
    parametric with text 500
    Programming tips 501
    ranking 501
    using a scope 500
  Java 103, 172, 428
    parametric with text 172
    Programming tips 173
    using a scope 173
common 23
common classes
  Enterprise Information Portal 212
common classes, Enterprise Information Portal 539
common object model 23, 265
configuration strings
  C++ 276
  relational databases 205, 535
connecting to workflow services
  C++ 553
connection strings
  relational databases 205, 535
connectors
  Java
    creating custom 211
connectors, description 4
console sub system, setting 267
constants
  C++ 269
  Java 28
Content Manager
  basics 70, 392
  C++
    working with 391
  checking files in and out 89, 414
  connecting to server 82, 403
  connector APIs 69
  controlling access
    access lists 76, 398
    privileges 75, 396
    user groups 76, 398
  creating a datastore 82, 403
  creating a document 122, 447
  creating attributes 84, 406
  creating folders 91, 415
  creating item types 82, 404
  creating the document management data model 122, 446
  DDOs 81, 403
  deleting a document 124, 449
  document
    creating 122, 447
    deleting 124, 449
    retrieving 124, 449
    updating 123, 448
  document routing 128, 452
    ad-hoc routing 137, 464
    continuing a process 136, 460
    ending a process 136, 460
    getting next work package 462
    resuming a process 136, 461
    suspending a process 136, 460
  ICM connector APIs 80, 402
  library server 69, 391
  library server and resource manager consistency 125, 450
  linking items 93, 417
  managing documents 121, 445
  resource manager 69, 119, 391
  resource manager objects 120, 444
  retrieving a document 124, 449
  retrieving items 88, 413
  routing documents 128, 452
    concepts 128, 453
    process 130, 455
    resume a process 136, 461
    setting up process 131, 456
    starting a process 135, 460
    stopping a process 136, 460
    suspending a process 136, 460
  search query, example 104, 429

Content Manager *(continued)*
searching for items 88, 412
storing XML 49
transactions 125, 450
understanding the query language 98
understanding the search query 98, 423
updating a document 123, 448
work package 462
XML 47
Content Manager for AS/400
C++
entities and attributes 513
query 514
working with 513
Java
attributes 186
entities and attributes 186
index classes 186
parametric query 188
query 187
working with 186
Content Manager for AS/400, working with 186, 513
Content Manager Version 7, see earlier Content Manager 143
Content Manager, earlier, see earlier Content Manager 143
content provider in information mining 370
content server
Java
evaluating a parametric query 59
executing a parametric query 58
content servers 11
C++
connecting 269
executing a parametric query 299
listing 270
options 270
query 297
custom connectors
Java 211
Java
accessing options 29
connecting 28, 29
creating connectors 211
disconnecting 29
listing servers 30
query 56
content servers, specific
C++ 467
Java 141
controller servlet 319, 324
application parameters 325
clean up 324
connection pooling 324
conventions 325
extending 325
JSP sets 324
locale 324
parameters 325, 327
properties file 325, 327
reference 325
request parameters
folders 329
controller servlet *(continued)*
request parameters *(continued)*
general 327
items 328
search 328
sesion management 324
using 324
cs package 24
cursor, result set
Java 65
custom connector
system administration 212
custom connectors
common classes 212
Java
developing 211
custom content servers
C++
developing 538
working with 538

D

data access 11
data definition classes 142
data items
Java 33
databases, listing 523
datastore 11
evaluating a parametric query 59
executing a parametric query 58
datastore registration 17
datastore, evaluating a parametric query 300
datastore, evaluating a text query 61, 302
datastore, executing a text query 60, 301
DB2 Warehouse Manager Information Catalog Manager
connecting 208
entities and attributes 209
Java
query string 210
query 210
running a query 210
working with 208
DDO
adding properties 32
DDO, adding properties 274
DDO, properties 144, 469
DDO, understanding datastores 13
DDO, XDO as a part of 36, 277
DES 193, 522
diagnostic information 27
DKAny
assignment from 292
assignment to 292
destroying 293
display of 293
memory management 291
programming tips 293
type code, getting 292
Typecode 291
using type constructors 292
DKAny, using 291
dkCollection
Java 53
DKConstant 269
DKConstants
Java 28
dkDatastore 213, 540
DKDatastore
custom connectors 211
DKDatastore*xx* 141
DKDatastoreDL
C++ 269
connecting 269
DKDatastoreDL options 270
list schema and schema attributes 271
list servers 270
Java 28
connecting 28
DKDatastoreDL options 29
list schema and schema attributes 30
List servers 30
DKDatastoreTS
C++ 475
connecting 477
DKDatastoreTS options 478
list schema 479
list servers 478
Java 150
connecting 152
DKDatastoreTS options 152
list schema 153
list servers 153
DKDDO 32
DKException (Java) 27
dkFederatedIterator 55
dkIterator 55
DKSequentialCollection
Java 53
dkSort 55
DKTimestamp 228
DKWorkFlowServicesFed
C++ 553
dkWorkFlowUserExit
Java 231
DLL 24, 265
document
Extended Search 525
document viewer
creating 311
generic 311
documents
annotating 309
Extended Search 196
documents, representing 144, 469
Domino.Doc
C++
cabinet attributes, listing 520
entities and attributes 519
query 520
query syntax 521
working with 518
Java
cabinet attributes, listing 191
entities 190
query 191
query syntax 192
searchable entities 191
subentities, listing 190

Domino.Doc *(continued)*
Java *(continued)*
working with 189
Dynamic Data Object (DDO)
C++ 272
adding 273
attribute, DKFOLDER 290
attribute, DKPARTS 288
creating 272
data item values 274
deleting 275
displaying 275
Information, Digital Library 469
Information, Text Search Engine 476
PID 273
properties 274
Java 31
adding 33
attribute properties 34
attribute, DKFOLDER 53
attribute, DKPARTS 52
creating 32
data item values 33
displaying 34
Information, Digital Library 81, 144, 403
Information, Text Search Engine 151
PID 32
properties 34
dynamic data objects 12
dynamic data objects, comparing 14

E

earlier Content Manager
C++
DDO 469
documents 469
folders 469
image queries 495
image search applications 489
object management 470
PID 469
retrieval 474
text search 475
workflow 501
workflow service 501
working with 468
Java
DDO 144
folders 144
handling large objects 143
image searching 159
object management 145
QBIC 159
representing documents 144
text searching 150
workflow 174
workflow service 174
working with 143
storing XML 49
watermarking 505
XML 47
EIP workflow services, see also workflow 227, 553
Enterprise Information Portal database 12
Enterprise Information Portal database infrastructure 211, 538
Enterprise Information Portal, concepts 11
entities
Java
listing 30
entities, listing 509, 519
entity mapping 16
environment
setting
Java 25
environment, setting
C++ 265
Java 24
exception handling 27
DKException 27
exceptions
C++ 268
exits
system administration 22
exporting XML 51
extended data objects 12
Extended Search
C++
DDO 525
document 525
documents 528
Generalized Query Language (GQL) 524
listing entities andattributes 523
PIDs 525
servers 522
working with 522
Java
BLOB 198
databases, listing 194
DDO 196
documents 196
fields, listing 194
Generalized Query Language (GQL) 195
in federated searching 198
MIME types 198
PIDs 196
retrieving a document 198
servers 193
working with 193

F

federated collection
C++
understanding 296
Java 55
understanding 55
federated datastore mapping components
Enterprise Information Portal 17
federated document model 15
federated iterator
C++
understanding 296
Java 55
understanding 55
federated query 15
federated query *(continued)*
Extended Search 198
processing 18
federated query syntax 19
federated schema mapping 17
federated searching
Extended Search 198
understanding 15
federated searching, DES 530
FeServerDefBase 224
fields, listing 523
file, adding and XDO 37
FileNET, see also Panagon Image Services 199
folders, representing 144, 469
folders, updating 147, 473
function, invoking and XDO 280

G

Generalized Query Language 195, 524
GQL 195, 524

H

Help
visual beans 256

I

IBM Enterprise Information Portal for Multiplatforms
components 4
administration client 4
administration database 4
connectors 4
image search 5
sample client application 5
text search 5
workflow builder 5
workflow feature 5
image queries
working with 169
image queries, working with 495
image search
description 5
representing information 169
image search applications 163, 489
image search catalogs
listing 168
image search catalogs, listing 493
image search concepts 160, 487
image search databases
listing 168
image search databases, listing 493
image search engine
indexing an existing XDO 171
using 171
image search engine, using 497
image search features, listing 168, 493
image search servers
listing 168
image search servers, listing 493
image search terms 160, 487
image search, catalogs 160
image search, databases 160

image search, features 161, 488
image search, representing
information 495
ImagePlus for OS/390
C++
entities and attributes 509
query syntax 511
working with 509
Java
attributes, listing 183
entities, listing 183
query syntax 185
working with 183
images by content, searching 159, 486
importing XML 47
index classes, listing 513
informatin mining
tools 376
Information Catalog, working with 208
information extraction sample in
information mining 348
information mining 333
advanced search sample 365
beans 333
building an application 333
catalogs 372
categorization sample 337
clustering sample 352
custom content provider 370
document formats 383
importing documents
sample 357
information extraction sample 348
JSP 382
JSP applications 382
library 372
location of the sample files 336
managing 372
own content provider 370
records and metadata 379
samples 333
searching by category 365
service API 371
summarization sample 344
taxonomies 372
Web Crawler sample 357
Information Mining
description 5
iterators 53

J

jar files 24
Java 23
Java APIs 23
differences 24
Java Server Pages 319
JavaBeans 233
other builders 234
understanding 233
using 233
VisualAge for Java 233
JavaServer Pages (JSP)
and Information Mining beans 245
JSP applications
information mining
samples 382
JSPs 319

L

large objects, handling 143, 468
library 24, 265
listin workflow templates 229
listing workflows 229, 554
listing worklists 229, 555

M

mapping terminology 16
match highlighting 61, 63, 302, 304
media object, adding an XDO 282
media object, deleting and XDO 284
media object, retrieving an XDO 286
member
adding 147
removing 147
member, adding 472
member, removing 472
MIME types,
Extended Search 529
MIME types, Extended Search 198
multimedia data objects 13
multiple search 23, 265

N

non-visual beans 233
building an application 238
configuration 236
understanding events 238
understanding properties 238
working with 235

O

object management
C++ 470
creating 470
deleting 473
updating 471
Java 145
creating 85, 145, 407
deleting 89, 148, 413
updating 87, 146, 410
OnDemand
C++
entities and attributes 531
folders 532
listing application groups 531
listing attributes 531
listing information 531
queries 532
query 532
retrieving documents 532
search template folders 532
servers and documents 531
trace 534
working with 530
Java
annotations 183
asynchronous search 182
OnDemand *(continued)*
Java *(continued)*
displaying attributes 180
displaying documents 180
folders 179, 182
folders in federated searching 182
listing information 179
query 180
retrieving documents 180
servers and documents 178
working with 178
working with folders 181

P

package hierarchy, Java 24
Panagon Image Services
attributes 201
classes 199
DDO 200
document classes 201
documents 200
entities and attributes 201
exception handling 204
messages 204
pages 200
PID 200
query 202
query options 203
query parameters 202
query string 202
searching 202
troubleshooting 204
working with 199
parametric query
C++
evaluating 300
executing 299
formulating 298
formulating multiple criteria 298
Java 26, 57
executing 58
formulating 57
formulating multiple criteria 58
Parametric query 267
parts, updating 147, 472
password mapping 17
persistent identifier 14
PID
Java 32

Q

QBIC 159, 492
connecting 167
queries
Content Manager for AS/400
C++ 514
Java 187
DB2 Warehouse Manager Information
Catalog Manager 210
Domino.Doc
C++ 520
Java 191
Extended Search 195
ImagePlus for OS/390 185

queries *(continued)*
OnDemand
C++ 532
Java 180
Panagon Image Services 202
relational databases
C++ 536
Java 207
query
C++
result set cursor 306
federated 15
processing 18
Query
C++ 297
dkResultSetCursor vs DKResults 298
parametric type 298
text type 300
Java 56
dkResultSetCursor vs DKResults 57
parametric type 57
query object types 56
text type 59
query string
DB2 Warehouse Manager Information Catalog Manager 210
Java 57, 60
Panagon Image Services 202
parametric 57
query syntax
Domino.Doc 192, 521
federated 19
ImagePlus for OS/390 185, 511
Panagon Image Services 202
queryable collection
C++ 307
Queryable collection
C++
evaluating 308
getting results 307
queryable vs refined 308
Java 66
evaluating 67
getting results 66
Programming tips 67
queryable vs refined 67

R

related classes 213, 540
relational databases
attributes, listing 535
C++
entity, listing 535
queries 536
working with 534
configuration strings 535
connection strings 535
connections 534
Java
attributes, listing 206
configuration strings 205
connection strings 205
connections 204
entity, listing 206
relational databases *(continued)*
Java *(continued)*
queries 207
working with 204
Remote Method Invocation (RMI) 26
Result set cursor
C++ 306
creating a collection 307
open and close 306
set and get 306
Java 65
creating a collection 66
open and close 65
set and get 65
resuming a workflow 228, 555
Retrieval
C++ 474
folders 475
parts 474
Java 148
folders 149
parts 149
rights management 505
RMI server 26
cmbregist81.bat 26
cmbregist81.sh 26
starting 26

S

sample client application
description 5
sample files, location
Information Mining 336
samples
watermarking 509
search engines, indexing an existing XDO 498
searchable entities 520
server 23
server, connecting to Content Manager 82, 403
service API
information mining 371
servlet, see also controller servlet 324
setting the environment
C++ 265
Java 24
Settings
C++ 266
Building on AIX 266
Building on NT 267
On AIX 266
On NT 267
Java
Client connect and disconnect 29
Client/Server 23
On AIX 25, 26
On NT 25
Programming tips 24
Setup environment 24
Using sample Java applets and servlet 319
Java application on client 319
Local access 320
Remote access 320
Retrieve servlet 319
shared objects 24, 265
sorting 55
C++ 296
Stand-alone code examples
C++
annotation type 279
from buffer 278
from file 279
search indexed by Text Search Engine 480
Java
search indexed by Text Search Engine 154
starting a workflow 227
C++ 553
storage collection 46, 288
storage collection, adding an XDO 46, 288
storage collection, changing 47, 288
subentities, listing 519
summarization sample in information mining 344
suspending a workflow 228, 555
system administration
customizing 22
user exits 22

T

tag library
configuring 319
connection 320
documents 323
folders 323
items 322
schema 321
search 322
terminating a workflow 228, 554
text query 267
C++
executing 301
formulating 300
formulating multiple indexes 300
getting a particular result item 304
getting each result item 302
Java 26, 59
executing 60
formulating 60
formulating multiple indexes 60
getting a particular result item 63
getting each result item 62
text search
description 5
Text Search Engine
C++
Boolean query 475
Exceptions 268
Free text query 476
GTR query 476
Hybrid query 476
Load and index data 481
parametric queries 268
Programming tip 478
Proximity query 476
text queries 268
Java 150

Text Search Engine *(continued)*
Boolean query 150
Exceptions 27
Free text query 150
GTR query 151
Hybrid query 150
Load and index data 155
MAXPIECE 143
parametric queries 27
Programming tip 152
Proximity query 150
Setting heap size 144, 468
Text Search Engine, text queries 27
text structured document 155, 482
tracing information 268
Java 27
troubleshooting
C++
exceptions 268
Java 27

U

user ID mapping 17

V

viewer toolkit 309
architecture 310
viewing 309
viewing documents 309
visual beans 233
building an application 257
common behaviors 255
connecting 261
Help 256
properties 255
replacing 256
specialized behaviors 256
using in windows 263
working with 245

W

watermarking 505
Web Crawler sample in information mining 357
work items 556
workbasket, understanding 501
workbasket, understanding in earlier Content Manager 174
workflow
C++ 553
accessing worklists 556
connecting to 553
list workflows 554
list worklists 555
moving work items 556
resume 555
starting 553
suspending 555
terminating 554
work itmes 556
worklists 556
Java 227
accessing worklists 230
workflow *(continued)*
Java *(continued)*
actions 231
connecting to 227
list workflow templates 229
list workflows 229
list worklists 229
moving work items 230
resuming 228
starting 227
suspending 228
terminating 228
work items 230
workflow builder
description 5
workflow feature
description 5
workflow service 501
workflow service in earlier Content Manager 174
workflow, understanding 501
workflow, understanding in earlier Content Manager 174
worklist
accessing 230, 556
accessing work items 230, 556
C++ 556
listing 229, 555
moving work items 230, 556

X

XDO
C++ 276
data members 276
DDO, part of 277
indexing 479
PID 276
programming tips 277
stand-alone 278
Java 35
adding an annotation 38
adding from a buffer 37
adding from a file 37
data properties 35
DDO, part of 36
deleting 38
function, invoking an XDO 39
indexing 154
media object, adding 41
media object, deleting 43
media object, retrieving 45
PID 35
Programming tips 35
retrieving 38
stand-alone 37
storage collection, adding 46
storage collection, changing 47
updating 38
XDO, deleting 279
XDO, retrieving 279
XDO, updating 279
XML
dtd for import 48
export 51
import 47
working with 47

Program Number: 5724-B43

Printed in U.S.A.

SC27-1347-00

Spine information:

IBM Content Manager for Multiplatforms/IBM Enterprise Information Portal for Multiplatforms

Workstation Application Programming Guide

Version 8 Release 1